Deep learning techniques for geometric matching of microscopy images of *C. elegans* brains

Richie Yat-tsai WAN, Pedro BEDMAR-LÓPEZ, Longlai QIU
richie.wan@epfl.ch, pedro.bedmarlopez@epfl.ch, longlai.qiu@epfl.ch
CS-433 - Ecole Polytechnique Fédérale de Lausanne
Project hosted by Prof. Sahand Rahi, LPBS, Institute of Physics, EPFL
December 17, 2020

***Abstract*—Deep Learning techniques have been at the forefront of great progress in Computer Vision tasks, and achieve state of the art on problems such as classification, object segmentation, object detection, among many others. One of the components that make the success of deep learning are convolutional neural networks (CNN). In this report, we propose a model to geometrically match a source image to a target, applied to fluorescence microscopy images of *C. elegans* brains. First, we designed a pipeline to extract 3D images from microscopy data and generate a large training dataset using meaningful data augmentation. Secondly, we adapt a CNN used for geometric matching, initially trained on the Proposal Flow dataset. The adapted network, named WORMBRAIN, instead matches frames of microscopy images of that have undergone deformation, rotation, translation. We added our own architecture for both feature extraction, feature regression, and show that it performs at least as well as the base network. Finally, we align frames extracted from a video.**

I. INTRODUCTION

Computer vision (CV) and pattern recognition (PR) tasks can be applied to datasets pertaining to everyday objects such as CIFAR10, ImageNet. One of the factor in DL techniques' success for CV is the use of CNNs, pioneered with the invention of LeNet5 [1] to identify handwritten digits and AlexNet [2] applied to ImageNet among others. DL techniques for CV can also be used in more specialized applications as in the biological or medical field, such as microscopy data. One important techniques to study biological processes is the characterization of cell activity or phenotype by using fluorescent markers. For example, one can label a gene using a fluorescent protein, and use it to track a cell type, or activity such as cell signaling [3].

Traditionally, the image processing and analysis can be done manually using techniques like thresholding, edge detection, etc. But identifying cells of interest manually over thousands of image frames can prove to be time-consuming, require human intervention and results may be vary depending on the user. On top of that, in biological applications, repeat experiments typically generate large amount of data.

DL can take advantage of this amount of data, whether it is for supervised or unsupervised learning. In CNNs, the network can effectively learn the weights of their kernels in all layers by optimizing a loss over epochs. This yields a network that is tailored to a task such as identifying cells in an image.

In CV, the underlying task is referred to as cell segmentation. Many methods have been developed for automatic cell detection [4], [5] as well as cell segmentation [6], [7]. Nevertheless, cell segmentation still poses many challenges [8], due to different factors such as varying fluorescent intensities, the 3D nature of cells requiring to choose the optimal Z-plane to achieve the best focus, or differences in imaging hardware setups and protocols such as lenses or exposure time.

In the case of this project, the cells in question are not fixed and may move, roll, translate, deform across frames, creating another obstacle for analysis. A subset of neurons are marked with red fluorescent proteins to serve as landmarks, while green nuclear GCaMP has varying intensities with respect to the concentration of intracellular calcium.

Thus, the aim of this project is to geometrically match and align cells of interests after deformation across frames. To do so, we build upon the work of Rocco et al. [9], who used a CNN for geometrical matching. The concepts relating to [9] will be expanded upon in section II. An example of the movement across frames as well as alignment is showcased in figure 9.

II. BASIS FOR OUR WORK

In this section, we expand on the idea behind [9]'s CNNGEOMETRIC network. Their base implementation can be found at https://github.com/ignacio-rocco/cnngeometric_pytorch. All figures in this section are taken or adapted from [9].

A. Architecture

Their proposed architecture can be separated in three modules and illustrated below in figure 1.

- 1. A CNN Feature Extraction (FE) layer
- 2. A CNN Feature Correlation (FC) layer
- 3. A CNN and linear layer for Feature Regression (FR)

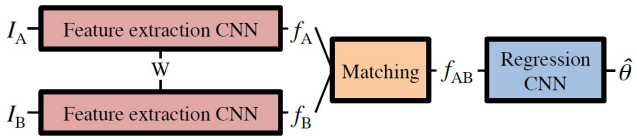

Fig. 1. The architecture proposed by Rocco et al., adapted from their 2017 paper. The FE CNN shares weights for I_A, I_B, and outputs the estimated parameters for a given geometric transformation $\hat{\theta}$.

A pair of images goes through the FE layer which extracts relevant features. These are then matched in the FC layer and go through the FR layer. At the end, the output are $\hat{\theta}$, the set of parameters estimated for a given geometric transformation from I_A to I_B. Two geometric transformations have been implemented by the authors, namely affine transformation (AFF) and thin-plate spline transformation (TPS).

1) FEATURE EXTRACTION*:* From a pair of images I_A, I_B, the FE is used to output the feature-maps f_A, f_B. This process is done by a Siamese network [10], [11] that apply the same weights on both inputs. This helps extracting similar features, while removing issues with variation during the first stage across different feature extractors. In their PyTorch implementation, Rocco et al. used either VGG16 [12] and Resnet101 [13] as FEs. The output feature-maps are then L2-normalized on a per-feature basis.

2) FEATURE CORRELATION*:* The authors match the feature-maps by computing a correlation map C_{AB}. Effectively, each pixel of feature-map is matched to each pixel through a scalar product described in equation 1 below, while the result is illustrated in figure 2.

$$c_{AB}(i,j,k) = \mathrm{f_B}(i,j)^T \mathrm{f_A}(i_k, j_k) \tag{1}$$

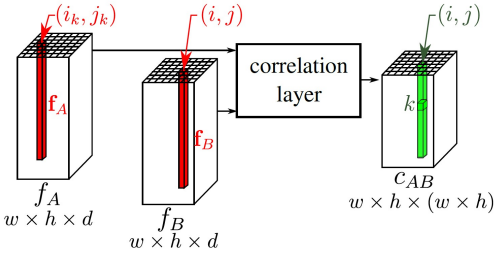

Fig. 2. Correlation layer.

3) FEATURE REGRESSION*:* The correlation map C_{AB} is then L2-normalized and fed through the FR layer. This consists of a simple 2 layers CNN, followed by a single fully connected layer with no activation function to mimic linear regression and estimate the parameters for a given geometric transformation. In the previous layer, dense per-feature matching allows to retain information across large distances from the feature-maps, while local information is retained after passing through a small CNN. The regression layer is regularized through means of BatchNorm as described in figure 3.

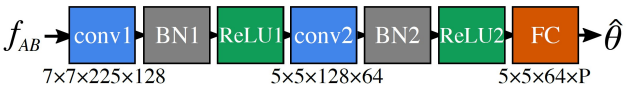

Fig. 3. Feature regression module used by Rocco et al. The output dimension P may be equal to 6 for AFF transformation, or 18 for TPS transformation.

4) Two-stage model: Finally, the authors propose a two-stage model. For a pair of images I_A, I_B, they first estimate the parameters for an affine transformation for coarse matching $\hat{\theta}_{Aff}$. Using $\hat{\theta}_{Aff}$, they warp the original image I_A and they generate another pair from which they estimate the parameters of TPS $\hat{\theta}_{TPS}$ transformation for fine matching. As a result, the total image is first warped using $\hat{\theta}_{Aff}$ followed by $\hat{\theta}_{TPS}$.

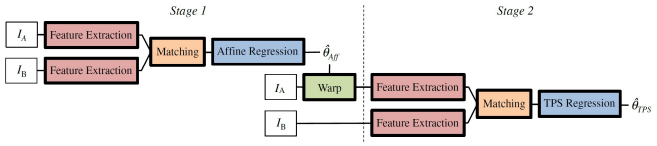

Fig. 4. Two stages model for coarse then fine alignment. The first stage estimates parameters for AFF transformation, while the second estimates the parameters for TPS transformation.

B. Training

For training, image pairs are generated synthetically: a θ_{GT} ground truth transformation is applied to I_A, leading to I_B. These image pairs are used to train the model, a process that consists in reducing the grid loss function error, i.e. minimizing the distance between $\hat{\theta}$ and θ_{GT}, the MSE between the transformed grid points and defined in Equation 2 where g_i is the coordinate (x_i, y_i) of the uniform grid. The parameters are optimized using Adam, and takes about 2 hours to train over 40 epochs using a NVIDIA GeForce RTX 2060.

$$L(\hat{\theta}, \theta_{GT}) = \frac{1}{N} \sum_{i=1}^{N} d(\tau_{\hat{\theta}}(g_i), \tau_{\theta_{GT}}(g_i))^2 \tag{2}$$

III. DATA

The laboratory we work with has provided us with images of a worm's brain in the form of a video and two .h5 files. These images contain noise and are not perfectly focused, because the worm moves inside the microscope.

A. Data extraction

The .h5 files are organized as a file system, where there are folders called groups, and within them files called frames. These frames are multidimensional arrays that represent, in our case, 3D images with the following dimensions depending on the file:

- epfl3.h5: 1715 frames with shape [1,112,112,32], green channel only.
- more_data.h5: 575 frames with shape [2,512,512,35], green and red channel.

To train the model we will use more_data.h5, as the images in the other file have too little resolution. Our model works with 2D images with 3 channels, so we transform each frame inside the .h5 file into a 2D image by applying max projection on each channel (see figure 5) and add an empty blue channel.

We use the video frames for testing once the model is trained. Each frame of the video is aligned with the previous frame or a target frame, thus stabilizing it.

B. Data augmentation

Given the circumvent the small amount of data available for training (572 images), we opted to use data augmentation in a way relevant to our task and generate more training data. The microscopy data we have may sometimes be noisy. The

Fig. 5. **frame_0** extracted from the .h5 file. Left shows stacked channels, middle shows the red channel, and right the green channel.

cells of interests are often rotated, translated, and naturally deformed (shear along the X and Y axis). Thus, we wanted to artificially generate data that mimic those conditions.

To do so, we used the library IMGAUG which can be found at https://imgaug.readthedocs.io/en/latest/. We've chosen IMGAUG because of it's ease of use as well as the many random parameters it allows. We have chosen the following **methods** for transformation:

- **ShearX** combined with **ShearY** for stretching along the X and Y axis, with degrees of shearing sampled uniformly in $(-10, 10)^2$, once in each dimension.
- **Rotate** with degrees of rotation chosen with equal probability $p = \frac{1}{4}$ in $\{-30, -15, 15, 30\}$
- **PiecewiseAffine** for distorsion, scale sampled in range $(0.01, 0.06)$
- **Affine** for translation and zooming, scale sampled uniformly in range $(1.1, 1.2)^2$, once for each dimension.
- **AdditiveGaussianNoise**, centered at 0 and amount chosen randomly in scale $(0, \frac{1}{40} \times 255)$

Using **SomeOf**, we randomly choose 2, 3 or 4 transformations with equal probability $p = \frac{1}{3}$ to be applied on a given batch of images. In total, we augment the training images tenfold, yielding us a total of 5720 of augmented images + 572 original images. From those, we split 80% of it for training set and the remaining 20% for the validation set.

IV. Models

As we felt the correlation layer is the most important contribution from [9]'s CNNGEOMETRIC MODEL, we decided to leave that untouched, and rather tune the FE and FR.

A. Baseline

As mentioned in section II, the baseline network uses either VGG16 or Resnet101 with a FR described in 3.

B. WormBrain

For our custom architecture from scratch, we've implemented both a FE and two FRs, keeping the feature correlator from [9]. The FE was based on a simple convolutional network, using the same kernel size across layers, with increasing number of channels. To mitigate the effect of vanishing gradients due to the deep architecture and many black pixels from the frames, we opted to use the activation LeakyReLU, as well as BatchNorm2d to regularize. This architecture is shown in figure 6.

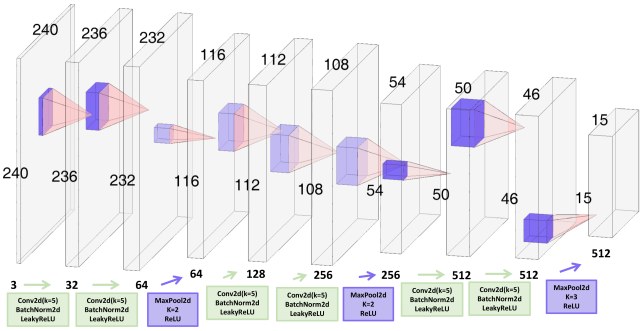

Fig. 6. WORMBRAIN feature extractor and operations.

For the FR, initially, we'd found the prediction layer to be too simple. While the base FR yielded good results for validation loss, when attempting to align images, the results were qualitatively not satisfactory. Instead, we've first tried using a small MLP of 5 layers with number of hidden units decreasing by half in each layer (diagram not shown). Each layer is also regularized using BatchNorm1d and DropOut to make the network better at generalization and encourage the network to learn weights that are more independent of the other hidden units.

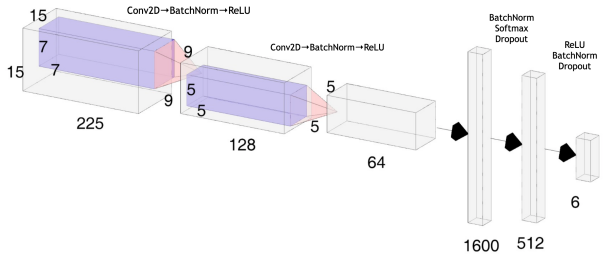

Fig. 7. SIMPLER Feature regression module.

This approach was not satisfactory, as shown by figure 8 and table I. We hypothesized that our model was too deep and did not extract any more useful information through the multiple layers. Using another approach, we decided to first use Softmax as the first activation after the convolutional layers. The idea behind this is that it will normalize all the features to a probability density of range (0,1). In doing so, it mimics a generalization of a multinomial logistic regression where the task is to predict the parameters of the geometric transformation. On top of that, we made the network more shallow while keeping the regularization elements such as BatchNorm and DropOut of 20%. This yielded the FR shown in figure 7.

V. Results and Discussion

Given the task, evaluation was strictly done using grid-loss on the validation set, as we do not have a real test-set per se. Probability of Correct Keypoints would have been a good metric, as used in [9], but would have required manual annotation of many image frames.

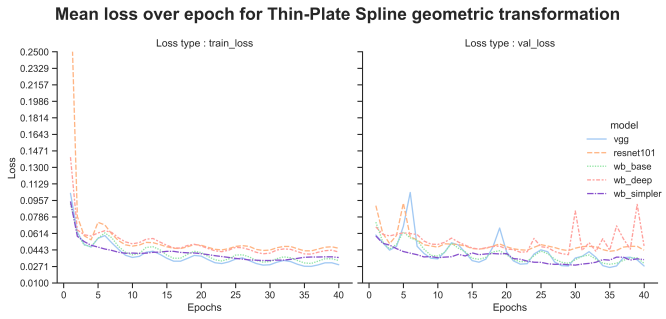

Fig. 8. Training and Validation loss for TPS transformation.

A. Training and validation losses

As estimation for TPS proved to be more challenging than AFF (Higher grid loss on average, data not shown), all the results we show focus on TPS. Grid-losses on validation sets as well as hyperparameters used during training are summarized in I and evolution of both training and validation loss per epochs are shown in 8. Qualitatively, we notice a cyclic behaviour w.r.t. the loss, which could be generated by the CosineAnnealingLR scheduler used in training. The best validation loss reached by our custom architecture (WB+Simpler) manages to achieve results close to the best baseline model (VGG16+Base) with a difference of 9.17%, while the loss over epochs appears to be more stable overall.

TABLE I
BEST VALIDATION LOSS FOR ALL MODELS USING MSE-GRID LOSS FOR THIN-PLATE SPLINE TRANSFORMATION.
γ:LEARNING RATE, **MOM** : MOMENTUM USED IN BATCHNORM

FE + FR	γ	Mom.	Epochs	Best Val. loss
VGG16 + Base	0.001	0.9	40	0.0260
Resnet101 + Base	0.001	0.9	40	0.0428
WB + Base	0.00366	0.92	40	0.0294
WB + Deep	0.003667	0.93	40	0.0393
WB + Simpler	0.005667	0.93	40	0.0285

B. Alignment

Qualitatively, we have tried to align images using the base architecture as well as our architecture (WB+Simpler). We have implemented two ways to align frames : the first one requires specifying a target frame, whereas the second one aligns a frame to the previous one. Please e-mail at richie.wan@epfl.ch for a video demo. It is hard to quantify the actual performance alignment without having the Key Points, but visually, we do notice that our best model (WB+Simpler) tended to yield better results than when we used Resnet101 as a basis.

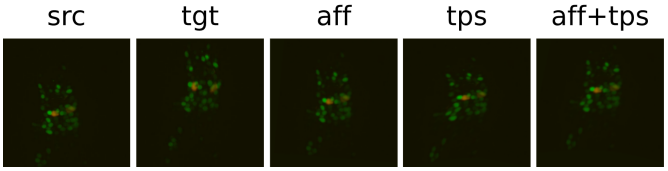

Fig. 9. From left to right : SRC : source image, TGT : target image, AFF, TPS, AFF+TPS : source image after alignment to target image using AFF, TPS, AFF+TPS

VI. CONCLUSION

In summary, we first introduce a pipeline to extract and use slices of 3D images of dimensions (C,W,H,Z) as training data. Then, we adapt the architecture described in [9], using a custom architecture and show that we were able to obtain validation grid losses results at least as good as the baseline network of II re-trained on our dataset. Using our pretrained weights, we can, in a single run, apply the whole pipeline needed for alignment, from data extraction to frames alignment to video generation. For example, our script can take as input either an .h5 file, or a video, or a folder of images, specifying the target frame in the process. It will then save the aligned frames as well as an "aligned.mp4" video.

A. Future steps

Currently, while every stage of the network is trainable, the loss is only computed with the output of the final stage, with respect to the parameters predicted compared to the ground truth. The gradient flowing backwards all the way to the feature extractor, meaning all three stages of the network train on the loss on outputted parameters. If we have the necessary data, one can improve the training by adding an auxiliary loss at the output of the first stage. For example, if we have cell segmentation maps available (e.g. using another network [14] to generate them), we can better train the network by jointly training the FE to identify and segment cells, or regressing bounding boxes and computing a loss for the features maps f_A, f_B. This way, the network will learn to extract better features with respect to cell segmentation, in the sense that it will better detect useful edges and which part should then be deformed together and aligned from I_A to I_B.

Beyond the FE, we can also imagine having a FC metric, and a secondary auxiliary loss, for example cosine similarity to be trained jointly. In the end, this yields :

$$\begin{aligned} L_{total} =& \alpha L_1(\hat{seg}, seg_{GT}) \\ &+ \beta L_2(cor, cos) + \gamma L_3(\hat{\theta}, \theta_{GT}) \end{aligned} \tag{3}$$

Where α, β, γ are weights in range $]0, 1]$ for the auxiliary loss of each sub-module and the first argument of each loss is the output of that module, and the second argument is the target value on which we compute the loss. Where L_3 is the grid-loss described in eq.2 and functions for L_1, L_2 can take various forms, such as MSE loss, CrossEntropyLoss or bounding box regression loss.

Beyond this, our network currently takes input of shape (3, 240, 240). When using images that are larger, frames are simply resized using bi-linear as implemented by Rocco et al. One key improvement would be to tune the feature extractor, correlator and regressor dimensions to make it even better suited to our datasets, as the frames are of dimension (512,512) and only really contain two color channels and an artificial (empty) blue channel filled with zeros.

REFERENCES

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, May 2017.

[3] W.-X. Jiang, X. Dong, J. Jiang, Y.-H. Yang, J. Yang, Y.-B. Lu, S.-H. Fang, E.-Q. Wei, C. Tang, and W.-P. Zhang, "Specific cell surface labeling of GPCRs using split GFP," *Scientific Reports*, vol. 6, p. 20568, Feb. 2016.

[4] H. Jiang, S. Li, W. Liu, H. Zheng, J. Liu, and Y. Zhang, "Geometry-Aware Cell Detection with Deep Learning," *mSystems*, vol. 5, pp. e00840–19, /msystems/5/1/msys.00840–19.atom, Feb. 2020.

[5] Y. Xie, F. Xing, X. Shi, X. Kong, H. Su, and L. Yang, "Efficient and robust cell detection: A structured regression approach," *Medical Image Analysis*, vol. 44, pp. 245–254, Feb. 2018.

[6] S. Dimopoulos, C. E. Mayer, F. Rudolf, and J. Stelling, "Accurate cell segmentation in microscopy images using membrane patterns," *Bioinformatics*, vol. 30, pp. 2644–2651, Sept. 2014.

[7] N. O. Loewke, S. Pai, C. Cordeiro, D. Black, B. L. King, C. H. Contag, B. Chen, T. M. Baer, and O. Solgaard, "Automated Cell Segmentation for Quantitative Phase Microscopy," *IEEE transactions on medical imaging*, vol. 37, no. 4, pp. 929–940, 2018.

[8] H. Peng, "Bioimage informatics: a new area of engineering biology," *Bioinformatics*, vol. 24, pp. 1827–1836, Sept. 2008.

[9] I. Rocco, R. Arandjelović, and J. Sivic, "Convolutional neural network architecture for geometric matching," *arXiv:1703.05593 [cs]*, Apr. 2017. arXiv: 1703.05593 [cs].

[10] D. Chicco, "Siamese Neural Networks: An Overview," in *Artificial Neural Networks* (H. Cartwright, ed.), vol. 2190, pp. 73–94, New York, NY: Springer US, 2021. Series Title: Methods in Molecular Biology.

[11] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-Convolutional Siamese Networks for Object Tracking," *arXiv:1606.09549 [cs]*, Sept. 2016. arXiv: 1606.09549.

[12] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015. arXiv: 1409.1556.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

[14] N. Dietler, M. Minder, V. Gligorovski, A. M. Economou, D. A. H. Lucien Joly, A. Sadeghi, C. H. Michael Chan, M. Koziński, M. Weigert, A.-F. Bitbol, and S. J. Rahi, "YeaZ: A convolutional neural network for highly accurate, label-free segmentation of yeast microscopy images," preprint, Bioinformatics, May 2020.