Open University of Catalonia

Doctoral Thesis

Gesture Tracking and Neural Activity Segmentation in Head-fixed Behaving Mice by Deep Learning Methods

Author:
Waseem Abbas

Supervisor:
Dr. David Masip Rodo

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Scene Understanding and Artificial Intelligence research group
Network and Information Technologies

July 8, 2020

Declaration of Authorship

I, Waseem Abbas, declare that this thesis titled, “Gesture Tracking and Neural Activity Segmentation in Head-fixed Behaving Mice by Deep Learning Methods” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"*It always seems impossible until it's done.*"

Nelson Mandela

Open University of Catalonia

Abstract

Faculty of Computer Science, Multimedia and Telecommunications
Network and Information Technologies

Doctor of Philosophy

Gesture Tracking and Neural Activity Segmentation in Head-fixed Behaving Mice by Deep Learning Methods

by Waseem Abbas

Abstract

It has long been established that the brain is the most important element in the nervous system which functions as a command and control junction for the nervous. The nervous system then, in turn, regulates and supervises the biological, biochemical, biomechanical and social functions of an organism. The brain itself is made of billions of neural cells, the neurons, which are capable of communicating with each other. These neurons themselves have limited information processing capacity, however, they can form neuronal networks capable of more complex processing. The brain relies on sensory inputs of different kinds (stimuli). The inputs of the sensors are processed by neurons, a response is calculated and is carried upon by different organs. This stimulus-response mechanism offers an interesting research challenge and has been the subject of research for decades. Not only neuroscientists but behavioural scientists are also interested in this area of research because understanding this stimulus-response mechanism can lead to a better understanding of neuro-diseases as well as better behavioural analysis.

The typical setup employed by neuroscientists is to study the response of laboratory animals to a stimulus while recording their neural activity at the same time. With the advent of calcium imaging technology, researchers can now study neural activity at sub-cellular resolutions in vivo. Similarly, recording the behaviour of laboratory animals is also becoming cheaper. Although it is now easier to record behavioural data and neural data, yet this data offers its own set of challenges. The biggest challenge is the annotation of the data due to its sheer volume. A traditional approach is to annotate the data manually, frame by frame. In the case of behavioural data, the manual annotation is done by looking at each frame and tracing the animals while for neural data, the annotation is done by a trained neuroscientist. In this research, we are proposing automated tools based on deep learning which can help behavioural data and neural data. These tools will help neuroscientists annotate and analyze the data they acquire in an automated and reliable way.

We have worked with locomotion data as the behavioural dataset and neural activity data acquired in vivo. The behavioural data is acquired by recording head-fixed mice running on a spherical treadmill from frontal and lateral angles. We have manually annotated this dataset by tracing limbs of the mice in each frame, thus generating locomotion masks. For neural activity data, we have worked with open-source datasets which contain annotations in the form of spatial locations of all the neurons visible in the field of view. For behavioural/locomotion annotation, we trained a two-stage model; the first stage is a 3-dimensional Convolutional Neural Network (3D-CNN) which extracts features from the videos and the second stage is a Long Short Term Memory network (LSTM) which infers if a region belongs to the limb or not. The model we designed is different from the state of the art in the sense that it explicitly encodes the temporal information available in the videos. We trained the models to detect the locations of limbs in each frame. For neural activity detection, we created a dense segmentation model made of Convolutional LSTM blocks which can process spatio-temporal information and trained it to detect and segment neurons visible in the field of view. This model is better than the state of the art in the sense that it is completely end-to-end and requires no human-in-the-loop supervision. We validated our results by comparing them to the ground truth and observed that for both behavioural and neural data, we got results comparable to the state of the art.

Resumen

Se ha establecido desde hace tiempo que el cerebro es el elemento más importante del sistema nervioso que funciona como un centro de mando y control para el sistema nervioso. El sistema nervioso, a su vez, regula y supervisa las funciones biológicas, bioquímicas, biomecánicas y sociales de un organismo. El propio cerebro está formado por miles de millones de células, las neuronas, que son capaces de comunicarse entre sí. Estas neuronas tienen una capacidad individual limitada de procesamiento de información, sin embargo, pueden formar redes neuronales capaces de un procesamiento más complejo. El cerebro depende de entradas sensoriales de diferentes tipos (estímulos). Las entradas de los sensores son procesadas por las neuronas, se calcula una respuesta y esta se ejecuta en los diferentes órganos. Este mecanismo de estímulo-respuesta ofrece un interesante desafío de investigación y ha sido objeto de estudio durante décadas. No sólo los neurocientíficos, sino también los científicos del comportamiento están interesados en esta área de investigación porque la comprensión de este mecanismo de estímulo-respuesta puede conducir a una mejor comprensión de las enfermedades neurológicas, así como a un mejor análisis del comportamiento.

La configuración típica empleada por los neurocientíficos consiste en estudiar la respuesta de los animales de laboratorio a un estímulo y registrar al mismo tiempo su actividad neuronal. Con la llegada de la tecnología de imágenes del calcio, los investigadores pueden ahora estudiar la actividad neuronal a resoluciones subcelulares in vivo. Del mismo modo, el registro del comportamiento de los animales de laboratorio también se está volviendo más asequible. Aunque ahora es más fácil registrar los datos del comportamiento y los datos neuronales,, estos datos ofrecen su propio conjunto de desafíos. El mayor desafío es la anotación de los datos debido a su gran volumen. Un enfoque tradicional es anotar los datos manualmente, fotograma a fotograma. En el caso de los datos sobre el comportamiento, la anotación manual se hace mirando cada fotograma y rastreando los animales, mientras que para los datos neuronales, la anotación la hace un neurocientífico capacitado. En esta investigación, proponemos herramientas automatizadas basadas en el aprendizaje profundo que pueden ayudar a procesar los datos de comportamiento y los datos neuronales.

Hemos trabajado con datos de comportamiento derivados del movimiento del roedor y datos de actividad neuronal adquiridos en ambos casos in vivo. Los datos de comportamiento se adquieren registrando ratones con la cabeza fija que corren en una cinta esférica desde ángulos frontales y laterales. Hemos anotado manualmente este conjunto de datos trazando las extremidades de los ratones en cada fotograma, generando así máscaras de locomoción. Para los datos de actividad neuronal, hemos trabajado con conjuntos de datos de código abierto que contienen anotaciones en forma de ubicaciones espaciales de todas las neuronas observables en el campo de visión. Para la anotación del comportamiento/locomoción, entrenamos un modelo de dos etapas; la primera etapa es una red neuronal convolucional tridimensional (3D-CNN) que extrae características de los vídeos y la segunda etapa es una red de memoria a largo y corto plazo (LSTM) que infiere si una región pertenece al miembro o no. El modelo que diseñamos es diferente del estado del arte en el sentido de que codifica explícitamente la información temporal disponible en los videos. Entrenamos a los modelos para detectar la ubicación de los miembros en cada fotograma. Para la detección de la actividad neuronal, creamos un modelo de segmentación densa basada en bloques LSTM convolucionales que pueden procesar información espacio-temporal y lo entrenamos para detectar y segmentar las neuronas visibles en el campo de visión. Este modelo mejora el estado del arte en el sentido de que se entrena mediante un paradigma de principio a fin y no requiere supervisión humana en el bucle. Validamos nuestros resultados comparándolos con los datos anotados y observamos que tanto para los datos conductuales como para los neuronales, obtuvimos resultados comparables al estado del arte.

Resum

S'ha establert des de fa temps que el cervell és l'element més important de el sistema nerviós que funciona com una centre de comandament i control pel sistema nerviós. El sistema nerviós, al seu torn, regula i supervisa les funcions biològiques, bioquímiques, biomecàniques i socials d'un organisme. El propicervell està format per milers de milions de cèl·lules neuronals, les neurones, que són capacesde comunicar-se entre si. Aquestes neurones tenen una capacitat limitada de processament d'informació, però, poden formar xarxes neuronals capaces d'un processament més complex. El cervell depèn d'entrades sensorials de diferents tipus (estímuls). Les entrades dels sensors són processades per les neurones, es calcula una resposta i s'executa alsdiferents òrgans. Aquest mecanisme d'estímul-resposta ofereix un interessant desafiament de recerca i ha estat objecte de recerca durant dècades. No només els neurocientífics, sinó també els científics del comportament estan interessats en aquesta àrea d'investigació perquè la comprensió d'aquest mecanisme d'estímul-resposta pot conduir a una millor comprensió de les malalties neurològiques, així com a un millor anàlisi del comportament.

La configuració típica emprada pels neurocientífics consisteix a estudiar la resposta dels animals de laboratori a un estímul i registrar al mateix temps la seva activitat neuronal. Amb l'arribada de la tecnologia d'imatge basades en calci, els investigadors poden ara estudiar l'activitat neuronal a resolucions subcel·lulars in vivo. De la mateixa manera, el registre del comportamentl'comportament dels animals de laboratori també ha esdevingut molt més assequible. Tot i que ara és més fàcil registrar les dades del comportamenti les dades neuronals, aquestes dades ofereixen el seu propi conjunt de reptes. El major desafiament és l'anotació de les dades degut al seu gran volum. Un enfocament tradicional és anotar les dades manualment, fotograma a fotograma. En el cas de les dades sobre el comportament, l'anotació manual es fa mirant cada fotograma i rastrejant els animals, mentre que per a les dades neuronals, l'anotació la fa un neurocientífic capacitat. En aquesta investigació, proposem eines automatitzades basades en l'aprenentatge profund que poden ajudar a modelar les dades de comportament i les dades neuronals.

Hem treballat amb dades de comportament derivades del moviment del rosegador i dades d'activitat neuronal adquirides in vivo. Les dades de comportament s'adquireixen registrant ratolins amb el cap fix que corren en una cinta esfèrica des d'angles frontals i laterals. Hem anotat manualment aquest conjunt de dades traçant les extremitats dels ratolins en cada fotograma, generant així màscares de locomoció. Per a les dades d'activitat neuronal, hem treballat amb conjunts de dades de codi obert que contenen anotacions en forma d'ubicacions espacials de totes les neurones visibles en el camp de visió. Per a l'anotació de comportament / locomoció, entrenem un model de dues etapes; la primera etapa és una xarxa neuronal convolucional tridimensional (3D-CNN) que extreu característiques dels vídeos i la segona etapa és una xarxa de memòria a llarg i curt termini (LSTM) que infereix si una regió pertany al membre o no. El model que vam dissenyar és diferent de l'estat de l'art en el sentit que codifica explícitament la informació temporal disponible en els vídeos. Entrenem als models per detectar la ubicació dels membres en cada fotograma. Per a la detecció de l'activitat neuronal, vam crear un model de segmentació densa que consta de blocs LSTM convolucionals que poden processar informació espaciotemporal i l' entrenem per detectar i segmentar les neurones observables en el camp de visió. Aquest model millora l'estat de l'art en el sentit que s'entrena seguint un paradigma de principi a fi i no requereix supervisió humana en el bucle. Validem els nostres resultats comparant-los amb les dades annotades i observem que tant per a les dades conductuals com per als neuronals, obtenim resultats comparables a l'estat de l'art.

Acknowledgements

It will be an understatement to say that I alone made this thesis possible. First, all praise goes to Allah for bestowing me with the opportunity, energy, will and courage to undertake this thesis. Then, there is a multitude of people who supported me in different capacities and the wonderful city of Barcelona which made my stay welcoming and pleasant. First of all, I would like to acknowledge the immense contribution of my supervisor, Dr. David Masip and thank him for the support and guidance he provided. Without him, this thesis wouldn't have been possible. Instead of his demanding and hectic role as the director of the doctoral school, he made sure that I got the resources, time and guidance I needed. Second, I would also like to acknowledge the contributions of our co-author Dr. Andrea Giovannucci in the form of datasets, guidance and moral support. I would like to thank my family who provided emotional support and helped me stay focused. I would especially like to acknowledge the sacrifices my wife, Arfa Ali, made so I can do this thesis. She stood by me like a rock through thick and thin. Then I would like to thank the wonderful friends I made here, Samia Oukemeni, Leila Mohammadi, Pilar Gomez Del Ray, Ronak Kosti and Marta Fondo. They made my PhD journey pleasant and rewarding and over time, became my second family. I would also like to thank the staff of the doctoral school, especially Agnes Requena and Marga Franco. From outside of the UOC community, I would also like to thank Mr. Shahzad Hanif who helped me settle in Barcelona when I first moved here and knew no one. I would also like to acknowledge the everlasting impact Mr. Haji Ahmad Raza, my mentor and a person to look up to, made on my life, my career choices and my upbringing. Moreover, I would like to acknowledge Mr. Mehtab Bangash for pushing me towards a research career. Last but not the least, I am thankful that I got a chance to do my thesis in this wonderful city and amazing country.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN	**C**onvolutionl **Neural N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory (network)
CLSTM or ConvLSTM	**C**onvolutional **L**ong **S**hort **T**erm **M**emory (network)
SNR	**S**ignal (to) **N**oise **R**atio
ROI	**R**egion **O**f **I**nterest
GECI	**G**enetically **E**ncoded **C**alcium **I**ndicators
ROC	**R**eceiver **O**perating **C**haracteristic (cruve)
SLIC	**S**imple **L**inear **S**terative **C**lustering

This thesis is dedicated to my family, especially my late mother, my father, my wife and my toddler.

Chapter 1

Introduction

The human brain is an amazing organ. With roughly 86 billion neural cells (neurons) and around 100 trillion neural connections (synapses)(Herculano-Houzel, 2009), it easily outshines any computing machine of similar size and will possibly do so for many generations to come. Contrary to common wisdom, the number of neurons and synapses do not increase with age but the opposite happens in a process called *synaptic pruning* (Paolicelli et al., 2011). In *synaptic pruning*, the overall number of neurons and synapses is reduced almost by half as certain neuron clusters are replaced with smaller but more complex and efficient clusters. As *synaptic pruning* suggests, the human brain is optimized for efficient learning and cognition; which raises the question; how exactly does that happen? How exactly do the billions of neurons and trillions of synapses come together to give rise to cognition? How do individual neurons learn to connect and communicate? How do neurons in different parts of the brain learn to perform specific cognitive tasks? All these questions are asked and ascertained in computational neuroscience; a more specialized discipline of neuroscience which focuses on understanding the computational model of the brain.

The term "computational neuroscience" was coined by Eric L. Schwartz in 1985, at a conference to provide a review of a field, which until that point was referred to by a variety of names, such as Neural modelling, Brain theory, and Neural Networks. Earlier in the 1950s and 60s, two neuroscientists namely Professor David Hubel and Professor Tortson Wiesel conducted a series of experiments, popularly referred to as Hubel & Wiesel experiments which laid the foundations of modern computational neuroscience. In one experiment, done in 1959, they inserted a microelectrode into the primary visual cortex of an anaesthetized cat. They then exposed the cat to patterns of light and dark on a screen. They found that some neurons fired more frequently when the cats were exposed to lines at one angle, while others responded more to another angle. Some of these neurons responded to light patterns and dark patterns differently. Hubel and Wiesel called these neurons simple cells (Hubel and Wiesel, 2004). Other neurons, which they termed complex cells, detected edges regardless of their spatial position in the receptive field of the neuron. These neurons were also sensitive to motion in certain directions (Hubel, 1995). These studies showed how the visual system constructs complex representations of visual information from simple stimulus features (Goldstein, 2001).

Hubel and Wiesel research also demonstrated that to understand the computation model of the brain, it needs to be studied on a cellular level and to understand how individual neurons learn to form connections, the associated stimulus (or the response the neural clusters being studied are controlling) also need to be modelled and quantified. According to Suzana Herculano-Houzel (Herculano-Houzel, 2009), the human brain is a scaled-up model of a primate brain and it can also be linearly correlated with the brains of smaller animals. Therefore, the cellular dynamics of the human brain can be understood by studying animal brains since studying an animal brain (mice brain for instance) is easier than studying a human brain. For this thesis, we will focus on the mice in a laboratory setting.

The thesis will focus on developing computer vision tools to automate the analysis of large scale data from two linked sources: (i) behavioural data from the mice locomotion and (ii) neuron segmentation from calcium images acquired in vivo. Since we have established that to understand the cellular dynamics of the brain, we need to understand the associated stimulus (or response they are controlling), we will focus on quantifying locomotion of the mice as a physical response and to understand the cellular dynamics in the mice brain, we will focus on quantifying and modelling of the activity patterns of individual neural cells (neurons) in time-lapse videos of the mice brain at sub-cellular resolution.

1.1 Quantifying the physical response: the locomotion and (or) gesture tracking (limbs detection) (pose estimation)

Locomotion is a good "stimulus and response" option to quantify to understand the neural dynamics since it is easy to model and represents a good example of neural activity controlling a physical action. Also, the easy availability of affordable computing and sensing methods has made it easier to record milliseconds resolution videos of behaving mice and monitor their neuronal activity at the same time. A general trend is to record videos of the mice in a controlled but uninterrupted environment for extended periods (Sousa, Almeida, and Wotjak, 2006). One of the commonly used controlled environments is to make the mice walk on a spherical or cylindrical treadmill with their heads fixed. This allows the mice to walk freely on the treadmill with a minimal movement of their torso while reducing the stress associated with head-fixation. The video cameras are usually placed either beside, above, below or in front of the mice. The acquired video data are annotated manually, frame by frame. As increasingly large volumes of data are generated, manual annotation becomes impractical for two reasons; the impossibility to scale up to hundreds of thousands of frames and the lack of reproducibility. Indeed, high-quality research calls for reliable automatic methods to replace manual labelling of animal behaviour.

Some of the commonly used methods for limb annotation either use specific hardware for motion tracking or model the statistical properties of the background and the animal (background subtraction) (Abbas and Masip, 2019). Hardware-based approaches are usually difficult to reproduce in new scenarios. And background modelling is vulnerable to noisy images and moderate periods with lack of significant motion. Detecting only individual body parts of a moving animal becomes also challenging. Besides, due to motion and the change of perspective, limbs do not appear rigid throughout the video. They present relevant deformations from frame to frame. In these cases, traditional object recognition methods do not perform well. Robust appearance-based methods solve these shortcomings by learning a classifier Mathis et al., 2018b focused on diverse sets of image patches with the body part to be tracked. Recently, this problem has been approached from a pose estimation perspective as well (Mathis et al., 2018a; Arac et al., 2019; Graving et al., 2019; Mathis et al., 2018b). In pose estimation approaches, the animal/mouse pose is modelled by a set of posture points which are inferred in every video frame on a frame-by-frame basis. These approaches themselves are built upon the pose estimation methods for humans (Insafutdinov et al., 2016). In practice, these approaches work very well yet they lack one key aspect. All of the pose estimation approaches discard temporal information from the video itself. We have attempted to solve this shortcoming by proposing a pipeline which encodes the temporal information present in the video to infer limbs.

In the first contribution of this thesis, we develop a new automated limb annotation algorithm that considers both the appearance of the body parts and the temporal dynamics to infer the limb segmentation. Contrary to the state of the art methods for limb segmentation (see section 2.1) which rely on frame-by-frame annotation, we propose a pipeline which is specifically targeted towards encoding the Spatio-temporal information present in the videos. We propose to consider both the local appearance of the limbs and their location in previous frames to infer segmentation. We have validated the approach on videos of freely behaving head-fixed mice in two settings (see section 2.1.1). Experimental results show promising tracking accuracies even when only a small portion of training data is used. We also assess our proposal against two state-of-the-art limb tracking algorithms, and show significant improvements, especially in noisy recordings. Moreover, we also validate that when Spatio-temporal information in videos is properly encoded, tracking can be achieved even in noisy datasets.

1.2 Neural activity segmentation

To understand how the cellular dynamics inside a brain evolve, we need to understand the behaviour of individual neural cells (neurons). For this purpose, we need to be able to record the activity of individual neurons. This is difficult to achieve with traditional brain imaging techniques (Aine, 1995; Gazzaniga, 2006). To solve this issue, fluorescent calcium imaging was introduced to record neuronal networks at sub-cellular resolution in vivo (Stosiek et al., 2003a; Stosiek et al., 2003b; Grienberger and Konnerth, 2012). The basic principle of calcium imaging is to record the changing concentrations of so-called calcium indicators in a cellular body with the help of fluorescence microscopy. In the following sub-sections, we introduce the calcium indicators and fluorescence microscopy.

1.2.1 Two-photon Fluorescence Calcium Imaging

Two-photon fluorescence calcium imaging technique is a revolutionary biological microscopy method which makes the imaging of a single biological cell possible in vivo. It is the combination of two equally revolutionary technologies; calcium imaging and two-photon fluorescence imaging. These technologies are explained below.

Calcium (Ca^{2+}) imaging

Ca^{2+} is an intracellular messenger regulating multiple cellular functions such as cellular excitability, secretion, contraction and gene expression. Calcium ions were recognized to be essential for regulating biological processes by Sydney Ringer, who found that "lime salt" is necessary to maintain the contractions of an isolated frog heart (Ringer, 1883). Therefore, understanding the spatial and temporal dynamics of Ca^{2+} signals in cells and tissues can offer invaluable insights about cellular structures. In the 1970s, Roger Tsien and his colleagues pioneered the chemical synthesis of organic Ca^{2+} indicators. This made direct monitoring of cellular Ca^{2+} signals possible (Tsien, 1980; Tsien, 1981; Grynkiewicz, Poenie, and Tsien, 1985). Early studies focused on measuring Ca^{2+} signals in vitro. Since the true evolution of neuronal networks in stimulus-response situations cannot be studied in vitro, it is essential to record and analyse Ca^{2+} signals in vivo. Fortunately, advances in optical and computing technologies yielded powerful microscope systems that enabled the realization of this goal. By combining appropriate Ca^{2+} indicators with appropriate optical imaging techniques, cellular Ca^{2+} signals have been recorded with a high degree of spatial and temporal resolution. This combination gave rise to the widely popular fluorescent calcium imaging technique which has been extensively used to record neuronal networks with sub-cellular resolutions.

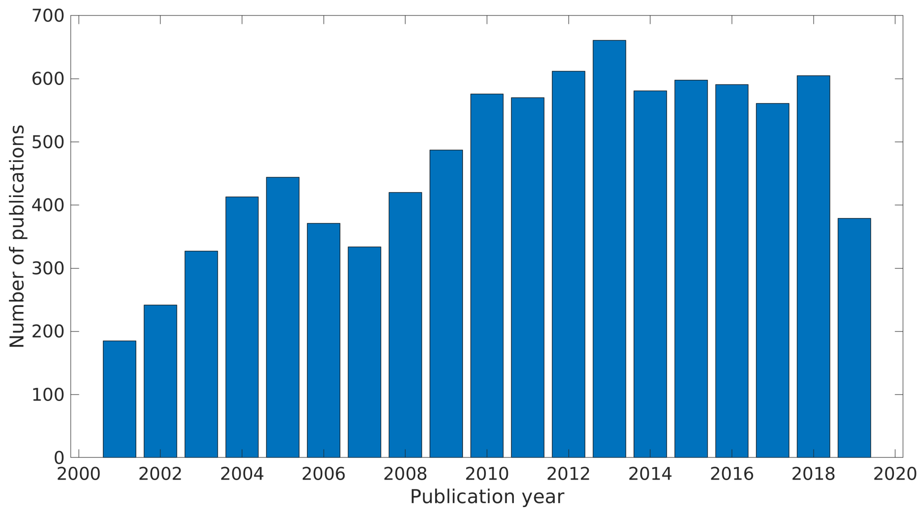
Fluorescence microscopy

Fluorescence microscopy aims to reveal only the (fluorescent) objects of interest in an otherwise black background (Lichtman and Conchello, 2005; Denk, Strickler, and Webb, 1990a). It requires that the objects of interest exhibit fluorescence; which is the emission of light that occurs within nanoseconds after the absorption of light of shorter wavelength. By filtering the exciting light and letting the emitted fluorescence through, it is possible to see only the fluorescent objects. Therefore, in some cases, certain molecules which exhibit fluorescence (called the fluorophores) are introduced into the biological tissues being imaged.

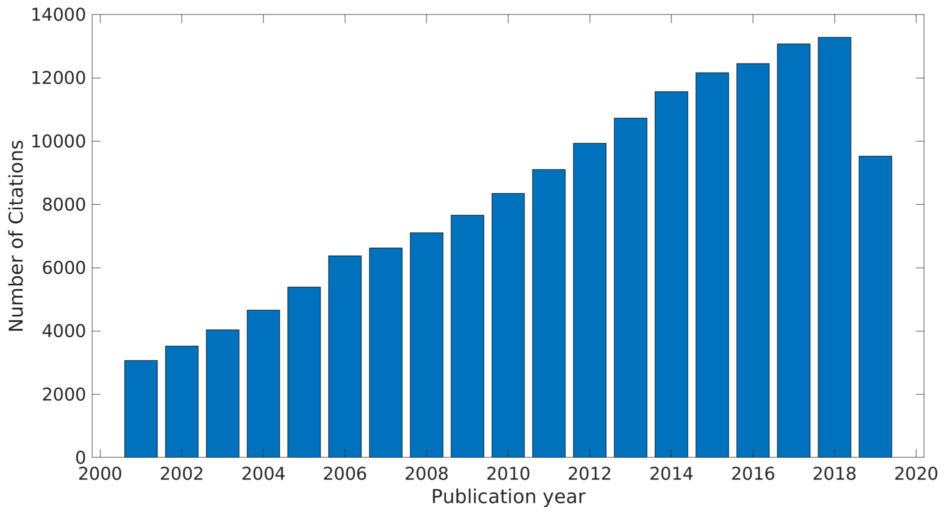
Fluorescence calcium imaging

Fluorescence Ca^{2+} imaging is a type of calcium imaging extensively used for monitoring neuronal activity. It exploits the fact that in living cells, most of the depolarizing electrical signals are associated with Ca^{2+} influx which can be attributed to the activation of one or more of the numerous types of voltage-gated Ca^{2+} channels (Tsien and Tsien, 1990; Berridge, Lipp, and Bootman, 2000). These Ca^{2+} signals constitute the elementary forms of neuronal communication (Neher, 1998; Südhof, 2000; Yuste and Denk, 1995; Kovalchuk et al., 2000). Moreover, the Ca^{2+} signalling also plays an important role in the induction of memory neuronal plasticity (Chittajallu, Alford, and Collingridge, 1998).

The development of two-photon fluorescence Ca^{2+} imaging has made the imaging of neurons located up to 500 μm below the cortical surface possible (Svoboda et al., 1997). Imaging in vivo has been mostly restricted to a single neuron at a given time and, therefore, has not been used for monitoring neuronal networks. For in vitro experiments, there is a simple method for loading cells with Ca^{2+} indicator dyes. Slices or cell cultures are incubated in external saline containing the membrane-permeant acetoxymethyl (AM) ester form of the dye (Tsien, 1981). When combined with two-photon Ca^{2+} imaging, this method allows simultaneous monitoring of individual neurons located up to 200 μm below the surface of a slice (Garaschuk et al., 2000). To make in vivo observation of neuronal networks possible, a new breed of calcium indicators and loading methods have gained popularity. The calcium indicators are genetically engineered (called genetically encoded calcium indicators, GECIs) from green fluorescent protein-like (GFP-like). Since the GECIs can be targeted towards specific cell types, they can be used to observe neuronal networks in vivo (Stosiek et al., 2003a; Miyawaki et al., 1997; Persechini, Lynch, and Romoser, 1997; Subach et al., 2019). Over the past two decades, two-photon fluorescent calcium imaging has seen steady growth in terms of acceptance in the research community. Fig. 1.1 shows this growth in terms of the number of research articles on the subject and number of times those articles were cited in other works.



(A) Number of publications

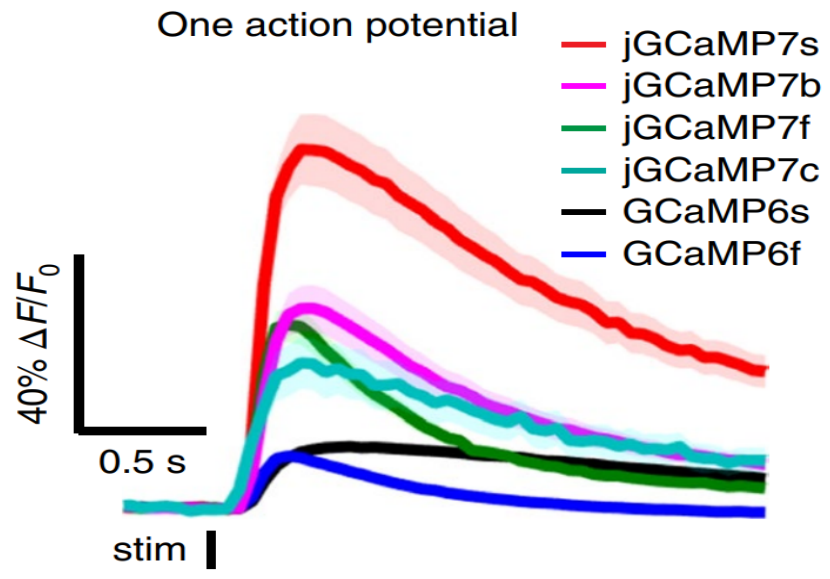


(B) Number of citations

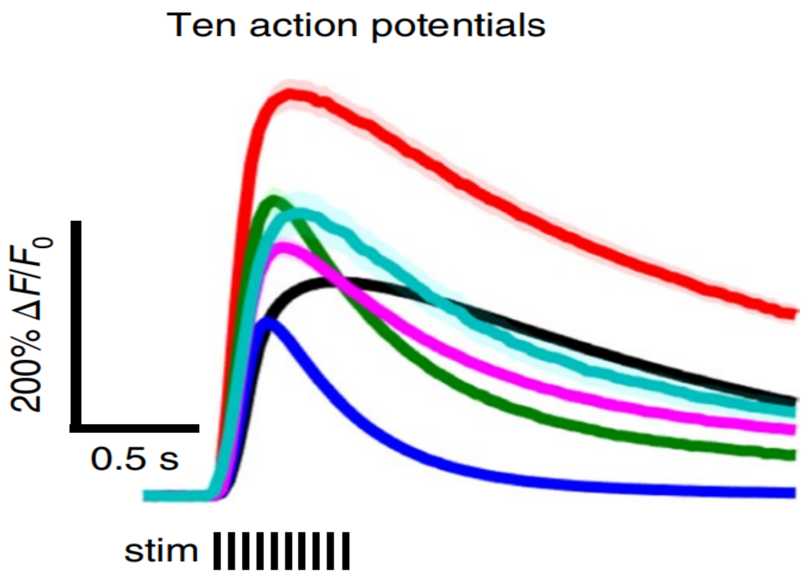
FIGURE 1.1: The steady gain in acceptance and popularity of fluorescence calcium imaging. (A) Although the number of publications tends to vary over years, the general trend is that more and more works are being published in the field. (B) The story with the number of citations of works on fluorescence calcium imaging is completely different. There is a strong upward trend in the number of citations over the years which shows high acceptance of the imaging technology among various fields. Source: *Web Of Science* 2019

1.2.2 Neural activity segmentation

The calcium indicators introduced in the earlier sections have a specific response pattern to calcium transient events. When a calcium transient happens, the GECIs respond and their fluorescence rises to a certain value within a very short time (short rise time). Then, the fluorescence takes a longer time to go down to its equilibrium value (long decay time). The many different variants of GECIs available might have different rise and decay times, but all of them follow the pattern illustrated in Fig. 1.2.



(A) One action potential



(B) Ten action potentials

Figure 1.2: This figure shows the rise and decay times of different GECIs, jGCaMP7s, jGCaMP7f, jGCaMP7b, jGCaMP7c, GCaMP6s, GCaMPf (Figures are taken from *jGCaMP7 | Janelia Research Campus* 2020). (A) shows the rise and decay of the six GECIs when they are excited by one calcium transient. (B) shows the rise and decay of the same GECIs when they are excited by a train of 10 calcium transients.

The recent advancements in two-photon calcium imaging have sufficient Signal to Noise Ratio (SNR) that single neural cells can be easily separated from the background. In the meantime, two-photon calcium imaging has also enabled the long term study of neuronal population activity during learning and behaviour (Peron, Chen, and Svoboda, 2015). The size of a typical neuron is quite small. For example, the size of a typical cortical neuron in a mouse brain is 10-20 μm. Therefore, the minimum spatial resolution of calcium imaging should be at least 5 μm. When a brain region is scanned with this resolution over a long time, it produces enormous amounts of imaging data. This poses a considerable signal processing problem.

The data is in the form of time-lapse image stacks since the typical sampling rate of calcium images is 5-10 frames per second. Individual neurons appear as small flashing light bulbs in these image stacks; going on and off depending upon the calcium transients and neural firing. To decode spiking activity from imaging data, the first step is to accurately detect regions of interest (ROIs), which may be cell bodies, neurites or combinations of the two. The specific calcium indicator used also affects both a cell's resting fluorescence and its apparent shape. For example, some GECIs are excluded from the nucleus and therefore produce fluorescent "doughnuts". Moreover, imaging data is contaminated with measurement noise and movement artefacts. Therefore, a signal processing pipeline is necessary which can remove the measurement noise and motion artefacts and can detect ROIs.

The crude approach towards this problem is manual segmentation of calcium imaging datasets. A person goes through the whole of the dataset, frame by frame and identifies ROIs. While manual segmentation offers the flexibility to use complex selection criteria, it is neither reproducible nor scalable. No two people will agree on the exact position, shape and size of the ROIs and the process becomes impractical once the dataset size becomes large. Therefore, an automated method is needed to segment ROIs (neurons) from the background.

To detect neurons in calcium imaging stacks, the GECIs dynamics, neurons spatial organization and their firing patterns need to be taken into account. These factors give rise to unique challenges while designing an automated neuron detection pipeline. As mentioned earlier, the choice of GECI can lead to toroid (doughnut) shaped neurons. Moreover, neurons might appear overlapped when observed from the experimenter's point of view. Also, some neurons might fire more often than the rest. These challenges make the mathematical modelling of the neurons close to impossible. Machine learning especially supervised learning offers a possible solution to these problems. Since in supervised learning, the inherent data distribution and model is learned from the data itself, there is no need to develop the mathematical model explicitly. Generally, an expert neuroscientist annotates all the detectable neurons in a calcium imaging stack manually and then this annotated stack is used to train a supervised algorithm to detect neurons in calcium imaging stacks acquired in similar settings. In some cases, neurons might be annotated by using biological markers (Apthorpe et al., 2016; Valmianski et al., 2010). Other approaches are also reported in the literature which either models the Spatio-temporal dynamics of neurons by matrix factorization approaches (Maruyama et al., 2014; Mukamel, Nimmerjahn, and Schnitzer, 2009) or by using summary statistics (Ohki et al., 2005; Kaifosh et al., 2014) or by modelling the expected shape and size of the neurons (Smith and Häusser, 2010).

As mentioned earlier, the supervised machine learning pipeline is trained on an annotated calcium imaging stack. The annotation is in the form of a binary mask where all the regions containing neurons are represented by a high value (1) and non-neuron regions (the background) are represented by a low value (0). There are two general approaches to do this. In one approach, the machine learning algorithm is trained on summary images to produce the final inference of the neurons (see section 2.2.2 for details on summary images, Giovannucci et al., 2017b; Klibisz et al., 2017). This approach has two significant shortcomings; one, the temporal information is lost while generating the summary images from the imaging stacks and two, it cannot distinguish between spatially overlapped neurons. The second approach is to train the algorithm over short segments of the image stack since processing the whole stack in one go is computationally impractical. Moreover, the spatially overlapped neurons are active at different times, therefore, their spatial footprint can be learned by training the algorithm on short temporal segments of the image stacks. However, this approach also faces two significant problems. One, we need to produce target masks for the short temporal windows of the image stacks and two, we still cannot be sure that overlapping neurons won't be active at the same time in one window. Moreover, the temporal masks need to be aggregated to get one final mask in the post-processing stage.

The second contribution of this thesis mitigates these problems. , we have proposed an end-to-end training pipeline which processes the whole stack without the need for summary images or temporal windowing. This end-to-end pipeline is based on a novel variant of popular U-net architecture (Ronneberger, Fischer, and Brox, 2015; Li et al., 2018), modified for processing spatial sequences. The pipeline can be broken down in three hierarchical stages. Stage one consists of the basic processing elements of the U-Net. We have designed a novel encoder cell based on Convolutional LSTM (ConvLSTM) (Xingjian et al., 2015) and decoder based on the same ConvLSTM cells. In the second hierarchy, we have used these novel encoder and decoder cells to create the modified U-Net architecture (see section 4.1.5 for details). The U-Net can be trained to segment a temporal window of the video or the whole video. In the third stage, we have formulated an end-to-end training pipeline for the modified U-Net. As mentioned earlier, the current state of the art neuron segmentation methods, which do not discard temporal information, rely on segmenting neurons in temporal segments of the videos and then aggregating them in the end to create the final mask. The image stack is processed in three stages by three copies of the same neural networks with slightly different parameters in a hierarchical manner. First, we divide the video into short temporal windows of equal length. Then the first neural network processes those segments one by one, producing masks. Once we process the same number of segments as the segment's length, the generated masks are temporally aggregated to create the second hierarchical temporal segment. The second level hierarchical segments are processed by the second level hierarchical neural network to produce masks. The process is repeated until we generate the same number of second hierarchical level masks as the mask length. We repeat this process until we have processed the whole video (see 4.1.4). By this approach, we eliminate the need to generate temporal masks for the temporal segments. This approach also helps us avoid the trouble of choosing the correct length of temporal windows to minimize the overlapping neuronal noise.

1.3 Thesis Outline

In this section, we discuss the outline of the thesis, with a summary of the chapters. At the end of this section, we outline our main contributions.

The principal goal of this thesis is to demonstrate and validate the usefulness of temporal information in gesture tracking (we will interchangeably use the terms gesture tracking or pose estimation or limb detection) and neural activity segmentation. In working towards this goal, we analyzed and curated two kinds of datasets; the first one consists of videos of head-fixed behaving mice running on a spherical treadmill. These videos were acquired both from frontal and lateral views. The second kind of dataset consists of time-lapse image stacks of neuronal networks of mouse brains acquired by two-photon fluorescent imaging in vivo. To analyze the gesture tracking dataset, we developed a stacked architecture; consisting of two kinds of neural networks. For neural activity segmentation, we developed the first of its kind neural network which tracked neural activity by processing the whole video at once without discarding any temporal information.

Chapter 1 gives a concise introduction to the need and nature of gesture tracking and how it is related to "stimulus and response" neuronal network analysis. It also introduces the two-photon fluorescent calcium imaging modality, the kind of datasets it produces and the unique challenges it poses. We also provide an overview of the neural activity detection and segmentation in this context;

Chapter 2 (see chapter 2 for details) covers the state of the art research being done on gesture tracking and neural activity segmentation. It is to be noted that in the literature review, we have only focused on approaches which are based on machine learning. Therefore, the literature review is complete in the context of machine learning while not thorough in a general context. For a comprehensive overview of the literature on gesture tracking, please refer to our review paper published in Sensors (Abbas and Masip, 2019). We prepared a comprehensive literature review on neural activity segmentation (covering non machine learning approaches as well), currently under review.

Chapter 3 (see chapter 3 for details) covers the gesture tracking dataset, the problem statement, our proposed approach, experiments and results.

Chapter 4 (see chapter 4 for details) discusses the neural activity segmentation problem, the Neurofinder challenge (*neurofinder*) benchmark, the dataset, our proposed approach, experiments and results. We also discuss the building blocks of our proposed approach in detail.

Chapter 5 (see chapter 5 for details) is an account of our conclusions, our thoughts and observation on gesture tracking and neural activity segmentation. It also outlines the potential impact of this research, its shortcoming and how they might be corrected in future research.

Main contributions of this thesis are summarized below.

- Curation of completely new datasets for gesture tracking. We have manually annotated 4 videos for gesture tracking in mice by tracking limb boundaries in each video frame. For neural activity segmentation, we have also created a frame by frame mask of all the active neurons in two calcium imaging stacks. Although for our proposed approach, we do not need frame by frame masks, yet these masks might be of tremendous use to other approaches.

- A stacked architecture for gesture tracking in head-fixed behaving mice. The first stage consists of a CNN which acts as a feature extractor while the second stage is an LSTM which infers if a given superpixel is part of the limb or not.

- A novel variant of U-Net built around a ConvLSTM cell which is capable of processing spatio-temporal sequences. We have designed a novel training algorithm around this U-Net capable of end-to-end training on calcium imaging stacks without the need of any frame by frame temporal masks.

- Empirical experiments to demonstrate the effectiveness of temporal information in gesture tracking and usefulness of end-to-end training in neural activity segmentation.

Chapter 2

Literature review

Due to the nature of the problem at hand, studying neural activity in conjunction with physical activity needs the test subjects (mice/rodents in this case) to be studied in a confined setting. A typical setup includes a closed environment (either a room or a box), video cameras/acquisition hardware, the mouse, mini-size calcium imaging hardware mounted on the mouse's head, control systems, and processing hardware. Depending upon the quality of neural activity recording setup, the mouse can either be restrained or freely behaving. There might be just a single camera or multiple video cameras recording the physical activity from different angles.

We can divide the task at hand into two sub-problems i.e. gesture tracking and neural activity tracking.

2.1 Gesture tracking

In principle, gesture tracking for neuroscience applications is not different from general-purpose gesture tracking; therefore, all the gesture tracking techniques can be applied to it in one way or the other. Although the general idea is the same, the environment for such type of gesture tracking is more confined than general-purpose tracking. Broadly speaking, gesture tracking in rodents/small animals can be divided into four main categories; hardware-based tracking, background modeling and subtraction based approaches, statistical approaches and machine-learning-based approaches. The hardware-based tracking pipelines mostly or solely rely on specialized hardware to track rodents' limbs, tail, head or whiskers. The last three approaches employ specialized hardware to varying extents. In this thesis, we will focus on machine-learning-based approaches.

Machine-learning-based tracking can further be subdivided into two main approaches. In one approach, the problem is dealt with as an object detection task while in the other approach, it is dealt with as a pose estimation task. Both the approaches have their advantages and disadvantages and both of them are intended to extract different versions of the same information. For instance, if someone is interested in the general gait of the rodents, pose estimation approach is more appropriate to use while if someone is interested in motion patterns, object detection/motion segmentation is preferred.

Dankert et al. (Dankert et al., 2009) proposed a machine vision-based automated micro-movements tracking in flies. Although this article does not directly deal with rodents, the detection and tracking algorithms used for legs and wings can be used for legs motion detection in rodents too. Videos of a pair of male and female flies are recorded for 30 minutes in a controlled environment. Wingbeat and legs motion data is manually annotated for lunging, chasing, courtship and aggression. The data analysis consists of four stages. In the first stage, the Foreground image F_I is computed by dividing the original image I by $(\mu I + 3\sigma_I)$ (F_I values in false-colours). In the second stage, the fly body is localized by fitting a Gaussian mixture model (Bishop, 2006) (GMM) with three Gaussians; background, other parts, and body to the histogram of F_I values (grey curve) using the Expectation-Maximization (EM) algorithm (Bishop, 2006). First (top) and final (bottom) iterations of the GMM-EM optimization. All pixels with brightness values greater than a threshold are assigned to the body and are fitted with an ellipse. In the third stage, the full fly is detected by segmenting the complete fly from the background, with body parts and wings (Otsu, 1979). In the fourth stage, the head and abdomen are resolved by dividing the fly along the minor axis of the body ellipsoid and comparing the brightness-value distribution of both halves. The approach is built upon proven statistical models. It can handle instrument noise. Since it has a learning element, the more data it sees, the better it gets. However, since the learning is not end-to-end, and the data processing pipeline is complex and need an in-depth understanding of statistical theory, therefore it is hard to work with for a neuroscientist. Since it has a background modelling component, therefore it also suffers from the inherent weaknesses of background subtraction models i.e. sensitivity to sudden changes in the environment.

Palmer et al. (Palmér et al., 2014) proposed a paw-tracking algorithm for mice when they grab food and can be used for gesture tracking as well. They developed the algorithm by treating it as a pose estimation problem. They model each digit (a limb such as a finger) as a combination of three phalanges (bones). Each bone is modelled by an ellipsoid. For 4 digits, there are a total of 12 ellipsoids. The palm is modelled by an additional ellipse. The forearm is also modelled as an ellipsoid while the nose is modelled as an elliptic paraboloid. The paw is modelled using 16 parameters for the digits (four degrees of freedom per digit), four constant vectors representing the metacarpal bones and 6 parameters for position and rotation of the palm of the paw. Furthermore, the forearm is assumed to be fixated at the wrist and can rotate along all three axes in space. This amounts to a total of 22 parameters. In each frame, these ellipsoids are projected in such a way that they best represent the edges. The best projection of ellipsoids is found by optimization and is considered a paw. They haven't reported any quantitative results. This approach is very useful if the gesture tracking problem is treated as pose estimation with a temporal context. Since the approach treats gesture tracking as a pose-estimation problem, it opens the possibility of using state of the art pose-estimation methods in gesture tracking. However, the computational cost is high for real-time deployment without graphical accelerators.

In (Palmér et al., 2012), Palmer et al. extended their work from (Palmér et al., 2014). The basic idea is the same. It models the paw made of different parts. Four digits (fingers), each digit having 3 phalanges (bones). Each phalange is modelled by an ellipsoid, so there is a total of 12 ellipsoids for the phalanges plus an additional one for the palm. In this paper, the movement of the 13 ellipsoids is modelled by vectors with 19 degrees of freedom, unlike 22 from (Palmér et al., 2014). The solution hypothesis is searched not simultaneously, but in stages to reduce the number of calculations. This is done by creating a different number of hypotheses for every joint of every digit and then finding the optimum hypotheses.

A. Giovannucci et al. (Giovannucci et al., 2018) proposed an optical flow and cascade learners-based approach for tracking of head and limb movements in head-fixed mice walking/running on a spherical/cylindrical treadmill. Unlike other approaches, only one camera installed from a lateral field of view was used for limb tracking and one camera installed in front of the mouse was used for whisker tracking. They calculated dense optical flow fields in a frame-to-frame method for whisker tracking. The estimated optical flow fields were used to train dictionary learning algorithms for motion detection in whiskers. They annotated 4217 frames for limb detection and 1053 frames for tail detection and then used them to train Haar-Cascades classifiers for both the cases. They have reported a high correlation of 0.78 ± 0.15 for whiskers and 0.85 ± 0.01 for hind limb. The proposed hardware solution in the paper is low cost and easy to implement. The tracking approach is also computationally not demanding and can be run in real-time. They, however, did not deal with the micro-patterns in motion dynamics which can be best captured with the inclusion of temporal context to the tracking approach. Moreover, an accurate estimation of flow fields either takes too much time or requires graphical processing units. Also, the Haar-cascades based method cannot deal with occlusions.

Mathis et al. (Mathis et al., 2018a) introduced a user-defined body-parts tracking method based on deep learning called DeepLabCut. The body-part (which can be either limbs or tail or head) is built on top of the human pose estimation pipeline, the DeeperCut (Insafutdinov et al., 2016). The DeepLabCut employs feature detectors of DeeperCut to build user-defined body part detectors in laboratory animals. The training procedure is standard, a user manually annotates all the limbs/tail/body parts in some of the video frames which are used to fine-tune the DeeperCut feature detectors. Then another prediction layer predicts the pose of the animal by labelling the body parts in question. The authors have reported an error of 4.17 ± 0.32 pixels on test data. The reported architecture is remarkable since it is a general-purpose architecture and can be modified to track another body part relatively easily. Also, since the architecture is built upon the existing state of the art deep networks for pose estimation, it is easy to train and it inherits all the strengths of parent deep networks. However, the reported pipeline can have a problem in the case of occlusions.

In DeepBehavior, the authors have proposed an open-source behavioural analysis toolbox built on top of existing validated approaches (Arac et al., 2019). The toolbox contains routines for gesture tracking, 3D kinematics analysis for humans and rodents and behavioural analysis for rodents. The toolbox is built on top of three existing and validated convolutional neural networks architecture named Tensorbox (Russell91, 2018), YOLOv3 (Redmon et al., 2016), and Openpose (Cao et al., 2018). For 3D kinematics tracking, the toolbox needs a stereo system with a properly calibrated camera. They recommend using Tensorbox if only one test subject needs to be tracked, YOLOv3 if multiple test subjects need to be tracked and Openpose if human subjects need to be tracked. They have initialized the networks by using models trained on ImageNet and fine-tuned them with custom datasets. The authors have not provided paw tracking results for rodents. This toolbox is a good example of using gesture and pose tracking approaches developed and tested for humans to be used for rodents and small animals. Since the system is built upon the existing state of the art pose estimation architectures, it inherits their strengths and weaknesses. For instance, Openpose can have a hard time identifying a pose it has not seen. It also does not know how to tell two subjects apart, therefore it can try to impose one pose upon two test animals in a situation in which one animal is partially occluded by the other. Also, it can face difficulties in estimating the pose of animals at an angle.

Pereira et.al. proposed an animal pose estimation pipeline called LEAP (**L**EAP **E**stimates **A**nimal **P**ose) (Pereira et al., 2019). It is built on top of a human pose estimation pipeline, employing a fully convolutional architecture. The architecture learns a mapping from raw images to a set of confidence maps (heatmaps). These maps can be interpreted as the 2-d probability distribution centred at the spatial coordinates of each body part within the image. The network consists of 15 layers of repeated convolutions and pooling. They tested their model on videos of Drosophila fruit fly, using only 1500 frames for training. They model the pose by 32 points (4 points for each of 6 legs, two for wingtips, three points for abdomen and three points for the head). The network was trained to produce heatmaps for all the 32 points. They report that the network can be deployed at 185 frames (192×192 resolution) per second on a GPU. To evaluate the performance, they calculated Euclidean distance between the coordinates of a ground truth pose point and predicted pose point. The report a maximum pixel error of 1.67 for the training set. In the testing phase, the network was able to achieve error <2.5 pixels in 87% of the tested frames.

Graving. et. al. proposed a general-purpose open-source framework called "DeepPoseKit" for pose estimation in animals, confined or otherwise (Graving et al., 2019). They proposed two models to efficiently and accurately solve the pose estimation problems of animals. They model the pose of an animal with 32 points and employ a confidence graph to illustrate the 32 point pose. Therefore, the confidence graph has 32 layers, one layer representing one pose point. They proposed two models for pose estimation. One model, called Stacked Densenet, implements a novel variant of classical hourglass architecture proposed by Jegou arranged in a stacked hourglass configuration. The Stacked DenseNet consists of an initial 7×7 convolutional layer with stride 2, to efficiently downsample the input resolution. This is followed by a stack of densely-connected hourglasses with intermediate supervision applied at the output of each hourglass. It uses 1×1 convolution to inexpensively compress the number of feature maps before each 3×3 convolution as well as when downsampling and upsampling. The second model is a modified version of the Stacked Hourglass model from (Newell, Yang, and Deng, 2016) It includes hyperparameters for changing the number of filters in each convolutional block to constrain the number of parameters. It uses a block size of 64 filters (64 filters per 3×3 convolution) with a bottleneck factor of 2 ($64/2 = 32$ filters per 1×1 bottleneck block). For the Stacked DenseNet model, they used a growth rate of 48 (48 filters per 3×3 convolution), a bottleneck factor of 1 ($1\times$ growth rate = 48 filters per 1×1 bottleneck block), and a compression factor of 0.5 (feature maps compressed with 1×1 convolution to 0.5m when upsampling and downsampling, where m is the number of feature maps). The authors have reported that both of their models, Stacked DenseNet and Stacked Hourglass outperforms LEAP. They report that by reducing the number of hyperparameters from 26 million to 0.5 million, the average pose prediction error increased only by 0.5 pixels. They further report that as compared to LEAP, their models produced less variance in prediction, indicating that they don't produce extreme predictions as often as LEAP.

2.1.1 Data acquisition environment and dataset description

Animal Experiments

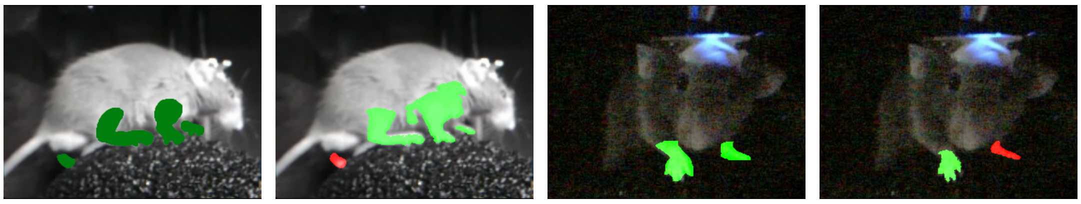
The gesture dataset was cordially provided by Dr Andrea Giovannucci working at Center for Computational Biology, Flatiron Institute, Simons Foundation, New York, NY, USA at the time. Experimental procedures were carried out as approved by the Princeton University Institutional Animal Care and Use Committee and performed following the animal welfare guidelines of the National Institutes of Health. The same preparations we employed in our behavioural analysis were also used for some imaging experiments. For a complete description of the surgical and behavioural preparations refer to (Giovannucci et al., 2018; Giovannucci et al., 2017a). 4 males 12- to 16-week-old C57BL/6J mice (Jackson Laboratory) were used, housed in the reversed light cycle. Mice underwent anaesthesia (isoflurane, 5% induction, 1.0-2.5%

maintenance) and a custom-made two-piece head plate (Dombeck et al., 2010) was attached to the animal's head. A 3mm or 5mm-wide craniotomy was drilled over the paranormal area of cerebellar lobule VI. After 15 hrs delay for animal recovery the top plate was removed for delivery of AAV1.Syn.GCaMP6f.WPRE.SV40 [Penn Vector Core, lot AV-1-PV2822] virus (Giovannucci et al., 2017a). All animals were placed back in their home cage for 2 weeks of recovery.

Animals were first habituated to a cylindrical or spherical treadmill that rotates along a single axis for repeated intervals over 5 days of incremental exposure. After habituation, animals were exposed to a variety of stimuli. Some of the movies were taken from animals that were undergoing eyeblink conditioning (Giovannucci et al., 2017a). Training consisted of repeated pairings of two stimuli, either whisker-puff/eye-puff or light/eye-puff, separated by intervals of 250 or 440ms respectively. This training often induced movements in an otherwise-still mouse. The stimuli consisted of (i) a periorbital air puff (10-20psi, 30ms in duration, delivered via a plastic needle placed 5mm from the cornea and pointed at it, (ii) a flash of light (400nm, 500ms), or (iii) an air puff to whisker vibrissae (2-3psi, 800ms).

Dataset

The pipeline for limbs detection and tracking is tested on two types of video data, frontal videos and lateral videos. The frontal videos are recorded by a camera installed in front of a mouse running on a spherical treadmill while the lateral videos are recorded by a camera trained at the right side of the mouse. The mice are running freely on a spherical treadmill with their heads fixed, so their heads cannot show significant movements while their torsos are free to move. Both types of videos are acquired at a frame rate of 120 frames per second and have a resolution of 240×320. The frontal videos have a higher amount of salt and pepper noise as well as equipment noise than the lateral videos. Each of the frontal videos is 239 frames(2 seconds) long while each of the lateral videos is 767 frames(6.39 seconds) long. Typical frames from both frontal and lateral videos are shown in Fig. 2.1c and 2.1a.



(A) A sample frame with limbs annotated from a lateral video.

(B) Detection results overlaid on original frame.

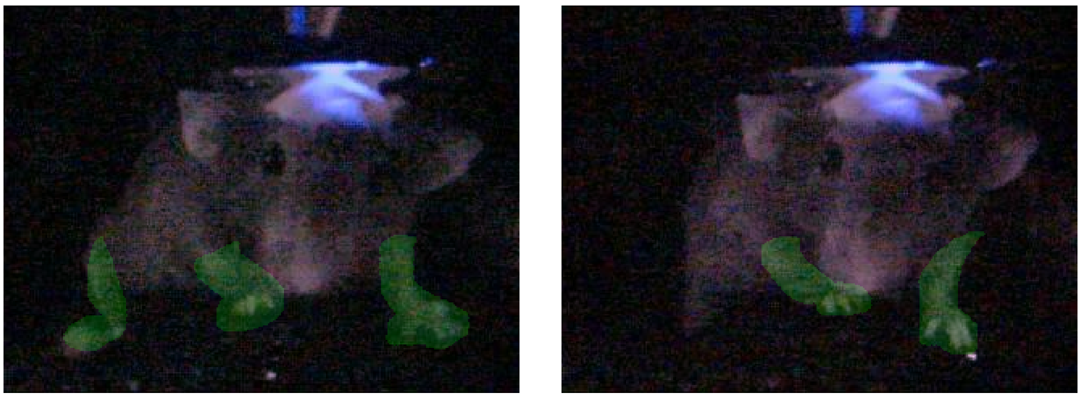
(C) A sample frame with limbs annotated from a frontal video.

(D) Predicted limbs overlaid on a sample frame of the frontal video.

FIGURE 2.1: Sample frames from lateral videos (a,b) and frontal videos (c,d). In (b,d) the areas overlaid by the green mask are correct predictions while the areas overlaid by a red mask are missed limbs.

In the frontal videos, one limb (hind left limb) is not visible at all in the video frames, so in best case scenario, only three limbs (front left, front right and hind right) are visible in a video frame. In some cases, even two or three limbs are occluded. This is illustrated in Fig. 2.2.

In lateral videos, due to the camera angle, all four of the limbs can be visible (best case). There are some video frames where only three limbs are visible and in extreme cases, only two are visible. In some cases, the front right and hind right limb

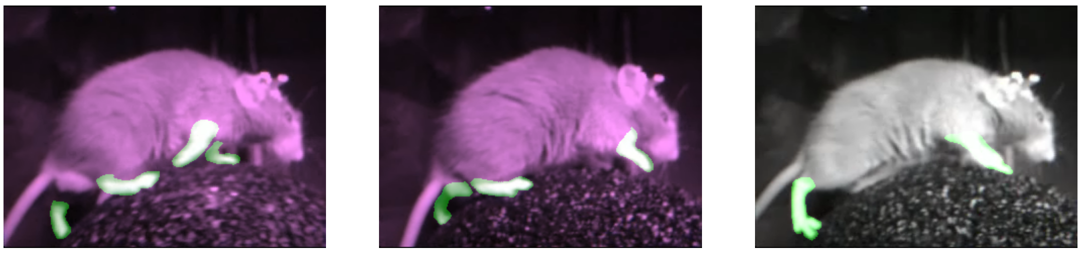


(A) A sample video frame from frontal videos with three limbs visible (Best case).

(B) A sample video frame from frontal videos with only two limbs visible. The other two are occluded due to posture of the mouse and camera angle

FIGURE 2.2: Sample frames from frontal videos showing different cases.

(these two limbs are more or less always visible) might overlap which might affect the detectors training. These cases are shown in Fig. 2.3.



(A) A sample video frame from frontal videos with all the four limbs visible (Best case).

(B) A sample video frame from lateral videos with only one limb occluded.

(C) A sample video frame from lateral videos with only two limbs visible. The other two are occluded due to posture of the mouse and camera angle

FIGURE 2.3: Sample frames from lateral videos showing different cases. The light purple color is due to false coloring to highlight the limbs, the original videos are gray scale.

Quality of the videos (Signal to Noise Ratio (SNR))

In order to evaluate the quality of the videos, we calculated the average value of SNR across all video frames. There are multiple methods reported in literature for calculating SNR of an image. Three commonly used methods are expressed by equations 2.1, 2.2 and 2.3.

$$SNR = 20log(\frac{\mu_{signal}}{\mu_{background}}) \tag{2.1}$$

$$SNR = 20log(\frac{\mu_{signal}}{\sigma_{signal}}) \tag{2.2}$$

$$SNR = 20log(\frac{\mu_{signal}}{\sigma_{background}}) \tag{2.3}$$

where μ_{signal} represents the average value of the signal, $\mu_{background}$ represents the average value of the background (noise), σ_{signal} represents the standard deviation of the signal. For better understanding, $\sigma_{background}$ represents the standard deviation of the background. To better evaluate the SNR, we considered two scenarios for considering which region constitutes the signal and which region constitutes the background.

- Only limbs are considered as the signal and everything else as the background.
- The limbs and the mouse body are considered as signal and everything else as the background

The SNR values in both scenarios calculated by equations 2.1, 2.2 and 2.3 are given in Tables 2.1 and 2.2.

TABLE 2.1: Average SNR of frontal videos

	Equation 2.1	Equation 2.2	Equation 2.3
SNR Scenario 1 (db)	-2.9	1.5	-1.4
SNR Scenario 2 (dB)	4.8	5.0	2.6

TABLE 2.2: Average SNR of lateral videos

	Equation 2.1	Equation 2.2	Equation 2.3
SNR Scenario 1 (db)	5.2	5.1	5.4
SNR Scenario 2 (dB)	18.6	8.8	13.6

Tables 2.1 and 2.2 show that regardless of the SNR equation or scenario, the quality of lateral video is much higher than the quality of frontal videos.

2.2 Neural activity segmentation

The arrival of calcium imaging technology has made it easier to study neural activity at cellular levels (Stosiek et al., 2003a). With the advantage of sub-cellular resolutions, calcium imaging comes with its own challenges (Yang and St-Pierre, 2016). The main challenge is the non-uniform, non-Gaussian noise in calcium imaging videos which originates from the way calcium transients work, equipment noise and biological matter. The second challenge is the non-uniform decay cycle of different calcium indicators. The third challenge is the 2D nature of calcium imaging since it only captures calcium transients in 2D if two neurons occupy similar XY coordinates but different z-coordinates, they might appear overlapped. To address the above-stated challenges, there are three main approaches for neural activity segmentation reported in the literature.

One approach tries to solve neural activity segmentation problem by image and signal processing methods. When analyzed by the naked eye, the calcium activity in neurons appears in the form of blobs or doughnuts flashing on and off at different time intervals. This approach first collapses the time dimension of calcium imaging videos to create what are called "summary images". Standard image and signal processing techniques are then applied to the summary images to segment neurons from the background.

The second approach tries to solve this problem by factorizing the calcium imaging videos in temporal and spatial components by matrix factorization (Maruyama et al., 2014; Mukamel, Nimmerjahn, and Schnitzer, 2009). The underlying hypothesis is that since calcium imaging videos capture the spatial coordinates of the neurons as well as their temporal activity, the activity every pixel of the video can be factorized in spatial and temporal components. The spatial components refer to the location of a neuron in the summary images while the temporal components refer to when a specific neuron was active. There is no agreed-upon way for matrix factorization as different researchers have proposed different methods.

The third approach treats it as a machine learning task. In this approach, machine learning models are trained to infer the number and position of each neuron in calcium imaging video. The standard approach is to feed calcium imaging video or summary images to the pipeline and train it to infer a binary mask for neuron locations. The inferred binary mask is then used for localizing the activity of every neuron present in the mask. For this thesis, we will consider machine learning-based approaches only.

Valmianski et. al. (Valmianski et al., 2010) proposed a multi-staged classification pipeline to determine which pixel belongs to a neuron or otherwise. First two stages consist of Robust-boost classifiers (Schapire et al., 1998). The first stage classifier classifies every pixel into neuron and non-neuron classes based on 8 statistical features. A median filter is then used to remove small regions falsely identified as neurons. The second stage classifier is another Robust-Boost classifier which classifies if a candidate neuron is a neuron or false positive based on six morphological features. The output probability map is then thresholded and neurons are segmented using Connected Thresholds. The authors have tested their approach on 64 different datasets; resulting from 16 different regions imaged over four trials each four mice. Each data set consisted of 200 frames at a resolution of 256×256 pixels. They have reported a test error of 7% and ROC curve maxing at 0.97.

Xu. et. al. proposed a CNN architecture with a graph regularization term which can utilize unlabeled data along with labelled data for neuron segmentation (Xu et al., 2016). First, they over-segment a neuron image into superpixels (Achanta et al., 2012). The superpixels are then fed to a CNN which minimizes a composite loss function over the labelled and unlabeled superpixels. The CNN is composed of two convolutional layers with 6 and 50 filters each. The loss function over labelled superpixels is represented by a softmax loss while the loss function over unlabeled superpixels is represented by a graph-regularized loss. The graph regularized loss function is formulated on the assumption that neighbouring elements in a graph will most likely share the same label. Therefore, they perform label propagation from labelled superpixels onto unlabeled superpixels by Gaussian Field Harmonic Functions (Zhu, Ghahramani, and Lafferty, 2003). They repeat the segmentation process for all the images in a volume. Finally, they perform neuron segmentation in 3D volume by global association method (Zhang, Li, and Nevatia, 2008). They have validated their approach on a private 1-photon data set. They manually annotated 2000 neuron regions spread over 1000 images (which form a 3D volume when stacked up), with half of them used for training and the remaining used for testing. Along with 2000 manually annotated regions, they include 70,000 more unlabelled regions (superpixels) which include 4000 neurons. They have reported an F1 score of 0.96

when they use as less as 40 images for training and an F1 score of 0.99 when they use 1000 images for training.

Apthorpe. et.al.(Apthorpe et al., 2016) proposed a 3D (referred to as (2+1)D) convolutional network-based neurons segmentation approach for calcium imaging data. It accepts an input image stack containing T time slices. There are four 10×10 convolutional layers, a max-pooling overall time slices and two 1×1 bottleneck fully connected layers. The output layer is configured to yield two 2D grayscale images as output, which together represent the softmax probability of each pixel being inside an ROI centroid. The dimensions of the input image stack are $37 \times 37 \times T$, therefore to cover the whole spatial domain, the window is made to slide in two dimensions over the input image stack to produce an output pixel for every location of the window fully contained within the image bounds. The network was trained on Two-photon calcium imaging data gathered from both the primary visual cortex (V1) and medial entorhinal cortex (MEC) from awake-behaving mice. Human experts annotated ROIs using the ImageJ Cell Magic Wand Tool (*Cell Magic Wand - omicX* 2020), which automatically generates a region of interest (ROI) based on a single mouse click. The human experts found 4006 neurons in the V1 dataset with an average of 148 neurons per image series and 538 neurons in the MEC dataset with an average of 54 neurons per image series. They have used the F1 score to evaluate the network performance and reported F1 = 0.71.

Wen et. al. (Wen et al., 2018) proposed a 3D-UNet (Ronneberger, Fischer, and Brox, 2015) based neuron segmentation approach on the calcium imaging data acquired from nematode "Caenorhabditis elgans". The UNet accepts a $160 \times 160 \times 16$ image stack and produces a binary mask of the same dimension, with every voxel either represents a neuron or background. The 3D UNet consists of 3 encoding layers, one middle layer and 3 decoding layers. 1st encoding layer consists of 2 convolutional operations with 8 and 16 filters, second and third also consists of 2 convolutional operations with 16,32 and 32,64 filters. Each encoding layer is followed by 2×2 downsampling. The middle layer has 2 convolutional operations with 64 filters each. The output of each encoding layer is concatenated with the input of the decoding layer at a similar level. The decoding layer consists of one convolutional operation and one concatenation with 8, 16 and 32 filters respectively. The output probability score for each voxel is thresholded at 0.5 and then neurons are segmented by Gaussian Blurring and watershed segmentation. Original 3D image stack are $512 \times 1024 \times 28$, therefore they divided the initial raw images into $160 \times 160 \times 16$ sized smaller images. They have used one 3D stack for training and reported that the pipeline tracked 98% of neurons in the test data. They have also reported that the pipeline was able to track 91% of the neurons even when artificial noise was added.

Wang et. al. (Wang et al., 2019) proposed a two-staged neuron detection pipeline. In the first stage, the raw image stack is first denoised by taking a moving average along the temporal axis. Then they normalize the stack using local maxima and minima along the temporal axis. After normalization, they generate a reference image by taking the maximum value along the temporal axis and an average image by taking the average value along the temporal axis. They binarize the reference image using the Local Adaptive Thresholding Algorithm (LATA)(Singh et al., 2012). After binarization, they use the watershed algorithm on the binary and reference image to create an intermediate segmentation mask. To separate neurons from non-neuron blobs in the segmentation mask, they train a three-layered CNN. First and second layers consist of a convolutional layer followed by a max-pooling layer while the third layer is a fully connected layer generating two outputs, 1 for neurons and 0 for otherwise. They have evaluated their pipeline on 9 manually annotated image stacks and reported F1 score of 0.85 ± 0.01. They haven't reported the performance of their pipeline on any benchmarks, so it is hard to compare their pipeline against other state-of-the-art approaches.

Peterson et. al. (Petersen, Simon, and Witten, 2018) proposed a dictionary-learning based approach, called SCALPEL, for neuron detection in calcium imaging data. First, they binarize every frame in the image stack after standardization. Then they use connected thresholds (Sonka, Hlavac, and Boyle, 2014) method to segment candidate neurons from the background. After segmentation, they filter candidate neurons which are smaller or larger than an expected size range. Then they generate a dissimilarity matrix based on temporal and spatial properties of candidate neurons in which each element represents how dissimilar a candidate neuron is compared to another candidate neuron. Using the dissimilarity matrix, they cluster similar candidate neurons together by hierarchical clustering approach (Hastie et al., 2005). In the final step, they optionally filter the refined dictionary elements, arguing that clusters with a larger number of members are more likely to be true neurons. They have validated their pipeline on three calcium videos/image stacks. The first video is a one-photon video, collected by the lab of Ilana Witten (`https://wittenlab.org/`) at the Princeton Neuroscience Institute, has 3000 frames of size 205×226 pixels sampled at 10 Hz. They have only reported the successful detection of neurons, but no precision, recall or F1 scores. The second and third videos are two-photon videos from Allen Brain Observatory (`http://observatory.brain-map.org/visualcoding`), containing 105,698 and 105,710 frames respectively of size 512×512. Once again, they have reported successful detection of quite a large number of neurons but no precision, recall or F1 scores.

Aleksander et. al. (Klibisz et al., 2017) proposed a standard UNet (Ronneberger, Fischer, and Brox, 2015) based neurons segmentation pipeline. They create a single mean summary image by taking the average value across the temporal dimension. This operation flattens the temporal dimension and converts 3D stack into a 2D image. Then they apply a UNet on 128×128 patch of the summary image to segment the neurons. The UNet is composed of an input layer, three encoding layers, one middle layer, three decoding layers and an output layer. The input layer has two 3×3 convolutional operations with 32 filters each. First encoding layer consists of two 3×3 convolutional layers with 64 filters each and a 0.25 dropout layer. The second encoding layer consists of two 3×3 convolutional layers with 128 filters each and a 0.5 dropout layer. The third encoding layer consists of two 3×3 convolutional layers with 256 filters each and a 0.5 dropout layer. The Middle encoding layer consists of two 3×3 convolutional layers with 512 filters each and a 0.5 dropout layer. Each decoding layer consists of concatenation, two 3×3 convolutions, a 3×3 deconvolution and dropout. The third decoding layer has 256,256 and 128 filters in each of its two convolutional and one decovolutional layer, the second decoding layer has 128,128 and 64 filters in each of its two convolutional and one decovolutional layer, and the first decoding layer has 64,64 and 32 filters in each of its two convolutional and one decovolutional layer. The output layer has a concatenation, two 3×3 convolutions with 32 filters each and output softmax layer. They have validated their approach on the Neuronfinder challenge dataset (`http://neurofinder.codeneuro.org/`). They trained their network on the top 75% of the summary images and validated it on bottom 25%. In 2017, their pipeline stood third on Neurofinder challenge with an F1 score of 0.59 ± 0.16.

Soltanian-Zadeh et. al. proposed spatio-temporal neural network architecture for segmenting active neurons in calcium imaging data (Soltanian-Zadeh et al., 2019). Their architecture is based on DensVNet (Gibson et al., 2018). Like other popular fully CNNs for semantic segmentation of medical images e.g., UNet (Ronneberger, Fischer, and Brox, 2015) and VNet (Milletari, Navab, and Ahmadi, 2016), DenseVNet is composed of encoding layers, decoding layers, and skip connection components. Each encoder stage of DenseVNet is a dense feature stack. The input to each convolutional layer of the stack is the concatenated outputs from all preceding layers of the stack. The authors made the following two modifications to DenseVNet: (i) they changed the last convolutional layer of DenseVNet to have 10 output channels instead of the number of classes and (ii) they added a temporal max-pooling layer to the upsampled features, followed by a 2D convolutional layer with 10 3×3 kernels, and a final convolutional layer with two 3×3 kernels to the output of DenseVNet. They optimized the network using the Dice-loss objective function. They validated their pipeline on a subset of Allen Brain dataset (http://observatory.brain-map.org/visualcoding) and Neurofinder Challenge dataset (http://neurofinder.codeneuro.org/). The Neurofinder challenge dataset was improved by having it analyzed by two expert human annotators. They compared their approach with four other state of the art neuron segmentation approaches namely CaImAn (Giovannucci et al., 2019), Suite2P (Pachitariu et al., 2017), HNCcorr (Spaen et al., 2019), and UNet2DS (Klibisz et al., 2017). They have reported a superior true positive detection performance as compared to the states approaches at lower PSNR values; with their approach detecting 80% true positives at PSNR = 10 while the mentioned approaches achieved 65%, 52%, 47% and 41% respectively. This approach achieved an average F1 score of 0.69 on Neurofinder challenge.

2.2.1 Data acquisition

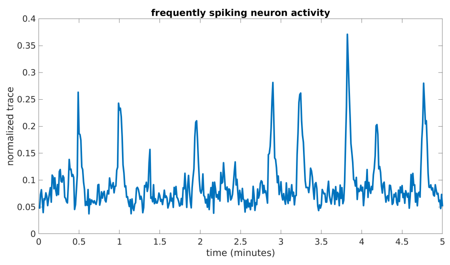
For benchmarking purposes, we have validated our approaches on the datasets provided by Neurofinder challenge (http://neurofinder.codeneuro.org/). The challenge was launched by a collective of neuroscientists called **CodeNeuro** (http://codeneuro.org/). The collective strives to bring neuroscience and data science together. This particular challenge was launched to encourage other researchers to develop neuron segmentation pipelines. The **CodeNeuro** have provided time-lapse images of mice brains in the form of image stacks and a JSON file containing the spatial coordinates of all the neurons present in the field of view. The task of this challenge is to locate each neuron in the image stack.

The datasets are acquired using two-photon calcium imaging techniques. The challenge consists of 19 (prefixed by **00, 01, 02, 03 and 04**) training and 9 testing datasets acquired and annotated by four different laboratories. These datasets are diverse: They reported activity from different cortical and subcortical brain regions and varied in imaging conditions such as excitation power and frame rate. The ground truth labels (JSON files containing spatial coordinates) were available for the training sets, while they were held out for the test set. The first dataset (numbered by a prefix **00** and **03** in the challenge) segmented neurons using fluorescently labelled anatomical markers, while others were either manually marked or curated with a semiautomatic method. Based on the calcium indicators used in the data collection process and the activation dynamics of neurons, following observations need to be taken into account while analyzing the datasets.

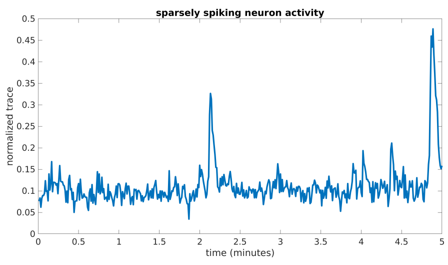
- Different calcium indicators have a different set time, decay time and decay profile. Therefore, if the decay time of a calcium indicator is larger than the silent time of a neuron, that particular neuron will appear to be always active.

- Due to the 2D nature of calcium imaging, two neurons which are not in the same z-plane might appear overlapped in the x-y plane.

To elaborate on the above-mentioned observations, we will use summary images, ground truth and calcium activity traces over time (see Appendix A for details). Fig. 2.4 shows the calcium traces of two neurons, one neuron spiking frequency is higher enough that the calcium indicator doesn't get enough time to decay while the other neurons' spiking frequency is comparatively lower.



(A) Frequently spiking neuron



(B) Sparsely spiking neuron

FIGURE 2.4: Two different spiking behaviours of neurons. (A) The neuron is spiking frequently. In some cases, the time between two spikes is smaller than the decay time of the calcium indicator (Build up to first large spike; the smaller peaks before also represent spikes) while in other cases, a neuron might spike sparsely, thus giving ample time to the calcium indicator to return to equilibrium (B)

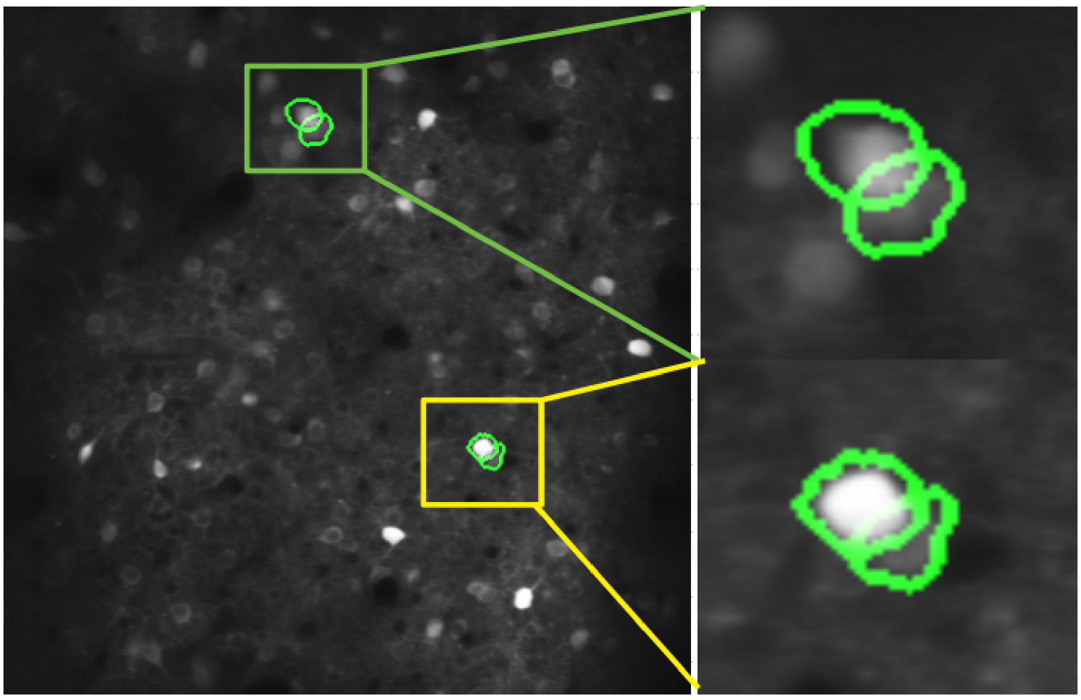


FIGURE 2.5: Overlapping neurons. The left panel shows a summary image (mean image across the temporal dimension) while the right panels show an enlarged view of the neurons overlapped in z-domain. Although the calcium imaging technique takes images of a very thin slice of the brain, yet two neurons might likely overlap in the XY plane (signified by width and height in these images) while having a different z-coordinate.

Fig. 2.6 compares two neurons based on their average activity. Since some of the neurons spike more frequently than the others while some neurons barely spike, their activity averaged over time is significantly different as illustrated in Fig. 2.5. Moreover, the calcium indicators distribution is not uniform throughout the brain regions being images, therefore some neurons appear inherently brighter than the rest.

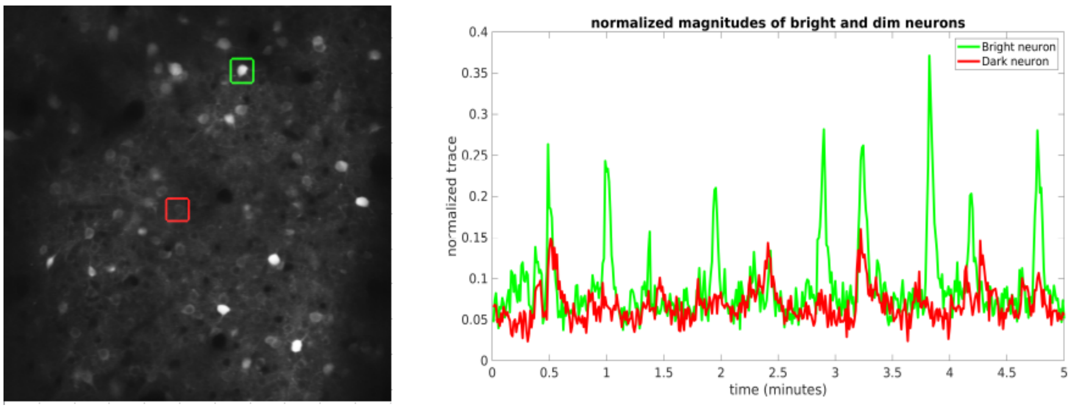


FIGURE 2.6: This figure shows the temporal activity of two neurons; neuron one (bounded by a green square in the left panel) appears much brighter than neuron 2 (bounded by a red box in the left panel) for a variety of reasons (see Appendix A for details) while the right panel contains their fluorescence traces (signified by corresponding colours) plotted against time. It is worth noting that the bright neuron is visible in the summary image while the darker neuron is hardly visible in the summary image.

Because of the inadequacies in the calcium imaging, some areas in the field of view appear darker than the rest. This is illustrated in Fig. 2.7.

2.2.2 Summary images

The image stacks acquired by two-photon calcium imaging cannot be visualized by standard image visualization tools since they have a temporal dimension; therefore, researchers often use summary images. A standard summary image is obtained by applying a specific function across the temporal dimension which collapses the temporal dimension and gives a 2D image representation of the stacks. Based on the collapsing function used, we can divide the summary images into the following two categories.

Statistical summary images

The statistical summary images are obtained when pixels in the 2D image are replaced by a statistical measurement of all the pixels on that specific location across temporal domain (Klibisz et al., 2017). Widely used statistical summary images include the mean image, standard deviation image and max image. For instance, when calculating a mean summary image, the value at a specific spatial location is calculated by taking the mean of all pixels on that specific location across the temporal dimension.

Correlation summary images

The correlation summary images are obtained by finding the mean temporal correlation of a pixel with its neighbours (Giovannucci et al., 2017b). For instance, to

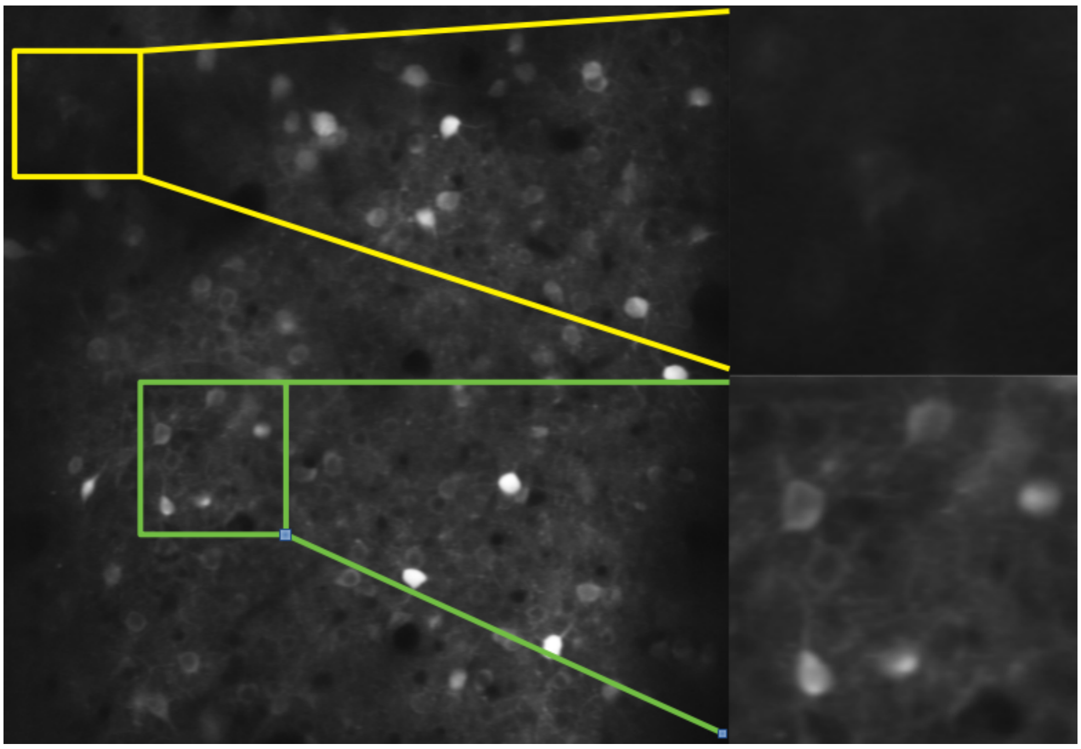


FIGURE 2.7: Bright and dark regions. The left panel shows a summary image in which two regions are bounded by two boxes; the yellow bounding box is for the darker region while the green bounding box is for the brighter region. There is a visible difference between the two regions as evident in their enlarged view on the right side.

find a pixel value on a specific location in the correlation image, first, the temporal correlation of the pixel on that location is calculated with its immediate neighbours (4 or 8 neighbours). Then mean of the temporal correlations is placed as the pixel value at that specific location.

Chapter 3

Gesture tracking

In this chapter, we discuss our proposed deep learning pipeline and related concepts used for gesture tracking in mice. At the end of the chapter, we present the performance of our pipeline.

3.1 Methods

3.1.1 Problem statement

Gesture tracking can be approached in two manners; one, using pose estimation methods and two, by segmenting each limb of the mouse in every frame. We will use the latter approach. Therefore, we postulate that for proper gesture tracking, we need to track all the limbs, head, tail and whiskers of the mouse. This chapter focuses on the limb parts. For successful limbs tracking, we need to consider the following.

- The shape of the limbs is deformable i.e. the limbs might appear to change shape from frame to frame.
- The limbs might be occluded in some frames
- Since limbs are part of the larger torso, therefore the motion of the torso should also be taken into consideration. The limbs might not move or move very slowly in some frames.

Therefore, to successfully track limbs in every situation, we have to take both its deformable shape and how it moves into consideration.

3.1.2 A Haar cascades based approach

In this approach, we try to locate every limb in a frame by training Haar cascades to detect limb-like shapes (Giovannucci et al., 2018). It requires training with a moderate quantity of labelled samples, and operates at faster than realtime speeds. In labelled samples, we draw bounding boxes around all visible limbs. The classification algorithm is composed of two steps (Fig. 3.1(a)–(c)): (i) Feature extraction using Haar filters on all the possible rectangular sub-windows present in the Region of Interest (sliding windows search); and (ii) efficient classification of each sub-window as limb/non-limb using a cascade of Adaboost classifiers. We computed the Haar-like features as described in Viola and Jones (Viola and Jones, 2001). Haar filters convolve the image with rectangular functions, and have been extensively used in computer vision tasks such as face detection (Viola and Jones, 2001) and object recognition (Lienhart and Maydt, 2002). The computation of the Haar descriptors can be significantly sped up using the integral image (Ehsan et al., 2015). Convolutions with rectangular Haar filters can be expressed in terms of sums of rectangular areas of pixels and can be reduced to three fixed-point operations (two subtractions and one addition) on the integral image regardless of the size of the filter. The resulting features are used to train a cascade of classifier ensembles. The Adaboost (Freund and Schapire, 1997) algorithm is used to train a robust combination of weak decision stump classifiers on the Haar features within a sliding window (Fig. 3.1(c)). Each classifier is trained to discard a large number of subwindows at early stages of the cascade, and the final classifiers are specialized to process the most difficult examples. At the detection phase, only a few sub-windows pass through all layers of the cascade, and each sub-window is classified as limb/no limb or tail/no tail.

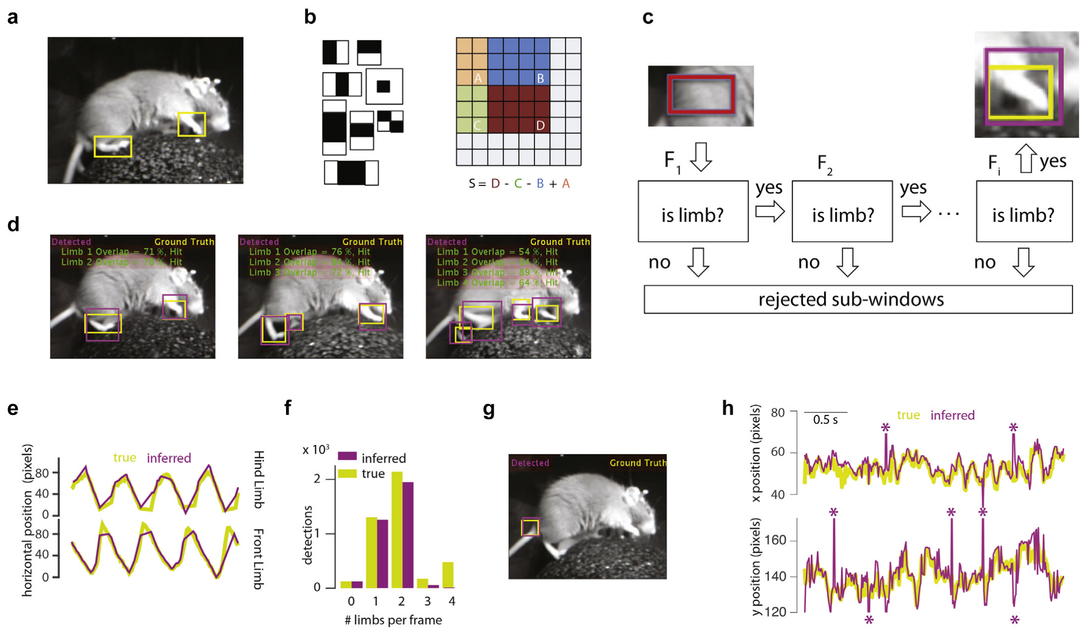


Figure 3.1: Block-Loc. (a) Illustration of the mouse with two examples of manually labelled paws (yellow squares). (b) Left: Convolution of a bank of rectangular filters (Haar filters, left). This step is optimized using the integral image, which takes advantage of the rectangular structure of the Haar filters. The integral image (right) is constructed in such a way that each pixel (i, j) in the integral image is the result of adding all the image pixels from the previous pixels in both coordinates. The convolution with a rectangular filter becomes a simple addition of four numbers. (c) Cascade of boosted classifiers. All the cropped windows are extracted from the image following a sliding-windows approach and the Haar features are computed. Then the sub-windows are processed in a cascade of classifiers that discard the vast majority of crops at early stages. (d–f) Empirical evaluation of Block-Loc. (d) Examples of overlaid detected and manually identified limbs in the case of 2 (left), 3 (middle) and 4 (right) detected limbs. (e) Left: Overlaid ground truth (yellow) and inferred (purple) horizontal position for hindlimb (top) and forelimb (bottom). The horizontal axis represents time. (f) Histogram describing the number of limbs detected per frame. In purple the detected and in yellow the ground truth. (f) Examples of overlaid detected and manually identified tail location. (h) Overlaid ground truth (yellow) and inferred (purple) horizontal (top) and vertical (bottom) positions for the tail. The purple asterisk indicates time points where the detected location was substantially far from the ground truth.

3.1.3 Gesture tracking by limbs segmentation: CNN and LSTM based motion tubes

Notation

In Table 3.1 we summarize the notation employed in this dissertation. As a general rule, we use (blackboard) boldface capital letters to denote matrices, (normal) capital letters to denote scalars, boldface small letters to denote vectors and normal small letters to denote indices. To be consistent, $\mathbb{I}^{W \times H \times T}$ denotes the whole video (or image stack), $\mathbb{I}_i^{W \times H}$ denotes ith frame of the video (or image stack), $\mathbb{I}_{i,j}^{s:Ws \times Hs}$ denotes j^{th} superpixel of ith frame. We will consistently use i for frame index in a video and j for j^{th} superpixel in a video frame. If an index appears in small brackets, it corresponds to the exact locations of the matrix elements. For example, $I(x_k, y_k, i)$ corresponds to a pixel (element) in I at kth x and kth y location in frame i.

Method overview

This approach is based on the intuition that a limb can be told apart from its surroundings by its shape and how it moves. Figure 3.2 summarizes the three main steps of the proposed method. Given a video $\mathbb{I}^{W \times H \times T}$ consisting of T frames:

1. For each frame, we compute the superpixels of the image. For each superpixel, we look D_{cnn} frames into the past, we find the closest matches for the superpixel on the time axis, and we stack them in chronological order to construct what we name ***motion tubes*** $\mathbb{M}_{i,j}$ (a tensor of size $Ws \times Hs \times D_{cnn}$).

2. We extract appearance features from each motion tube using a trained CNN as a feature extractor. It will produce a feature vector of length Lf for each motion tube. The CNN parameters are previously learned with a training set containing motion tubes from limb and non-limb regions (see section 3.1.3).

3. We stack sets of D_{lstm} features extracted from the CNNs to construct the ***motion sequences*** $\mathbb{X}_{i,j}$ (size $D_{lstm} \times Lf$). The motion sequences are used to train an LSTM that performs the image segmentation taking into account the temporal coherence of the motion tubes.

In the following subsections, we detail the methods we used to build the motion tubes, extract the feature vectors and obtain the LSTM output for each sequence.

Motion tubes

Motion tubes are defined as the 3D structures with the temporal history of groups of similar pixels that move similarly through time. We first reduce the complexity of a frame by grouping all the pixels into superpixels and then track their progression through time. A superpixel refers to a polygonal part of an image, larger than a normal pixel, that is rendered in a similar colour and brightness (*Superpixel, Empirical Studies and Applications* 0202; Stutz, Hermans, and Leibe, 2018). To overcome the computational cost of over-segmenting a frame into superpixels, we used the efficient SLIC (Simple Linear Iterative Clustering) method proposed by Achanta et. al. (Achanta et al., 2012). (refer to Appendix A for more details).

Superpixels (especially the ones located in mice skin) may have very similar appearance regardless of their state of motion. To effectively use superpixels, we introduce temporal context to them, treating time as the third dimension. Besides, superpixels differ in shape and size. In order to find where a superpixel is located

TABLE 3.1: Notations used in this chapter

Notation	Meaning
D_{cnn}	Depth of CNN input (motion tube)
D_{lstm}	Depth of LSTM sequence
Ws, Hs	width and height of a superpixel window
W, H, T	Width, height and duration of the video
N	Number of superpixels in one video frame
$\mathbf{I}$	Video consisting on T frames (size $W \times H \times T$)
$\mathbf{I}_i$	Frame at the i-th time step (size $W \times H$)
$\mathbf{F}$	Optical flow tensor (size $W \times H \times 2 \times T - 1$)
$\mathbf{F}_i$	Optical flow associated with I_i (size $W \times H \times 2$)
$\mathbf{I}_{i,j}$	j-th superpixel of I_i (size $Ws \times Hs$)
$\mathbf{M}_{i,j}$ (size $Ws \times Hs \times D_{cnn}$)	Motion tube tensor associated with j-th superpixel of I_i
$\mathbf{X}_{i,j}$ (size $D_{lstm} \times L_f$)	Motion sequence associated with j-th superpixel of I_i
L_f	length of feature vector extracted by CNN

in the next frame (motion path), we propose to use optical flow tracking (Horn and Schunck, 1981a) (refer to Appendix B for more details). Notice that the shape of the superpixels forming the motion tube is deformable. To make the method computationally tractable, we used a fixed-sized window to construct the tube. Also, temporal window slices from superpixels of the motion tube (along the depth dimension) may not necessarily refer to the same fixed spatial locations in successive frames, as body parts migrate along with the image coordinates. Fig. 3.3 shows an example of a superpixel tracked backwards through time. The depth of the tube (D_{cnn}) is controlled by the user and can be varied to change the extent of temporal context captured by the tube.

More formally, let's assume we are dealing with a video with W, H and T dimensions (width, height and number of frames). Row elements are denoted by $\boldsymbol{r}$, column elements are denoted by $\boldsymbol{c}$ while time dimension is denoted by $\boldsymbol{t}$. A frame I can be represented by: 3.1

$$\mathbb{I}i = \mathbb{I}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{t}_i) \tag{3.1}$$

The SLIC algorithm clusters all the points in $\mathbb{I}_i^{W \times H}$ in N superpixels. The superpixels generated from the current frame are not guaranteed to be present on the same spatial coordinates in the next (or previous) frame. Also, a direct temporal link between superpixels generated from the current frame to superpixels generated in the next (or previous) frame cannot be established. However, the apparent motion of superpixel centroids can give a rough estimate for its closest relatives in the next (or previous) frame if we assume that there aren't any abrupt changes in luminosity, shape and position of the objects in successive frames. We establish a temporal link between the superpixel in the current frame and its closest match in the previous or next frame and we stack them up to generate a 3D tube.

To generate the motion tubes of depth D_{cnn} for the N superpixels per each frame $i \in 1...T$, we proceed as follows:

1. We compute the optical flow from the 3D structure $\mathbb{I}$ (size $W \times H \times T$) representing the video (being $\mathbb{I}_i$ the *ith* frame). We store it in a flow field $\mathbb{F}$

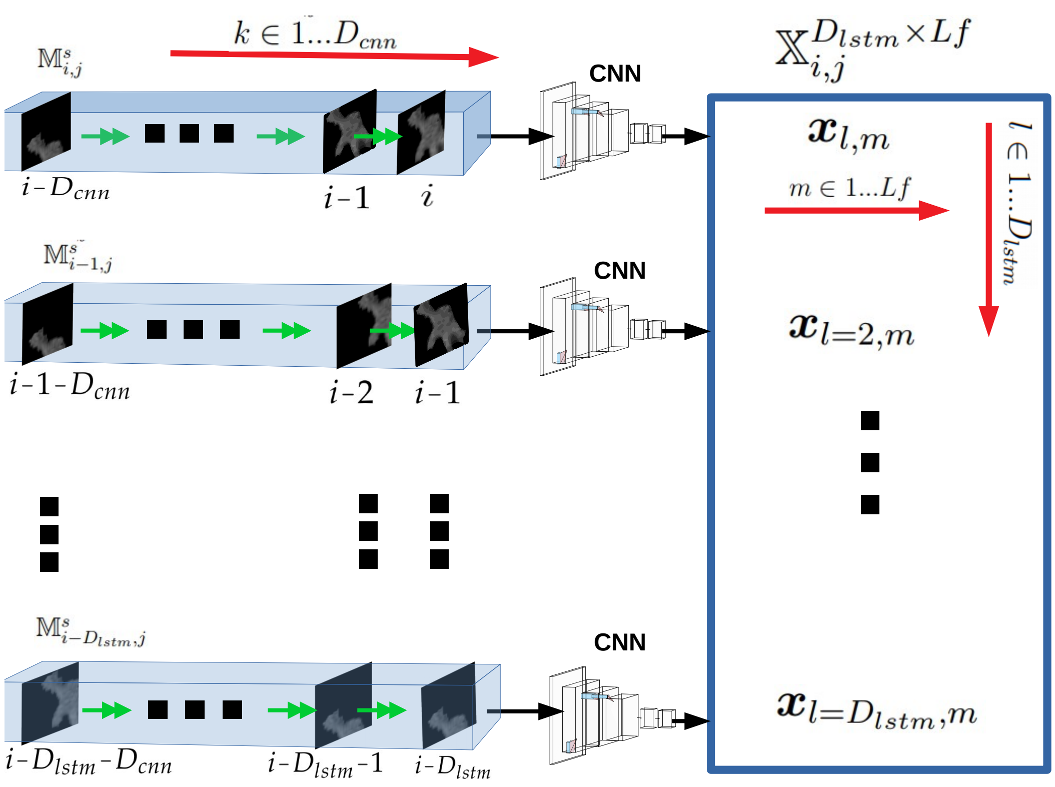


FIGURE 3.2: Master Flowchart: The image slice in the tubes $\mathbb{M}^s_{i,j}$ represents a specific superpixel and the corresponding fixed window which encloses it in frame i and its various matches tracked through time. Double headed arrow represents the flow of time, so superpixel at time $i-1$ was the predecessor of superpixel at position i. The shaded 3D structures on the left represent the 3D motion tubes formed by stacking D_{cnn} superpixels together chronologically while the blue-bordered matrix on the right represents the motion sequence. An enlarged view of a motion tube with its constituent superpixel shaded in green is shown in Fig. 3.3. The individual feature vector extracted from the tube (formed by a specific superpixel number j from frame number i) is represented by a row vector of length Lf as $\boldsymbol{x}_{l,m} = [\boldsymbol{x}_{l,1}, \boldsymbol{x}_{l,2}, ..., \boldsymbol{x}_{l,Lf}]$. D_{cnn} represents the depth of the motion tube and D_{lstm} represents the length of the motion sequence which is given as input to the LSTM. A single motion sequence is formed by stacking feature vectors extracted from D_{lstm} motion tubes by the CNNs in the correct chronological sequence.

size($W \times H \times 2 \times T - 1$). $\mathbb{F}_i$ (size $W \times H \times 2$) represents the optical flow associated with frame i.

2. We compute the centroid $\mathbb{I}^s_{i,j}$ of each j^{th} superpixel $\mathbb{I}^{s:Ws \times Hs}_{i,j}$ at frame i.

3. We estimate the coordinates of the centroid on the previous frame $i-1$ using the current coordinates and the optical flow vector.

4. We compute the distance from the estimated centroid $\mathbb{I}^s_{i-1,j}$ and all the superpixels in the previous frame.

5. We add the window of fixed size $Ws \times Hs$ centered on the closest centroid at frame $i-1$ to the motion tube $\mathbb{M}^{Ws \times Hs \times D_{cnn}}_{i,j}$.

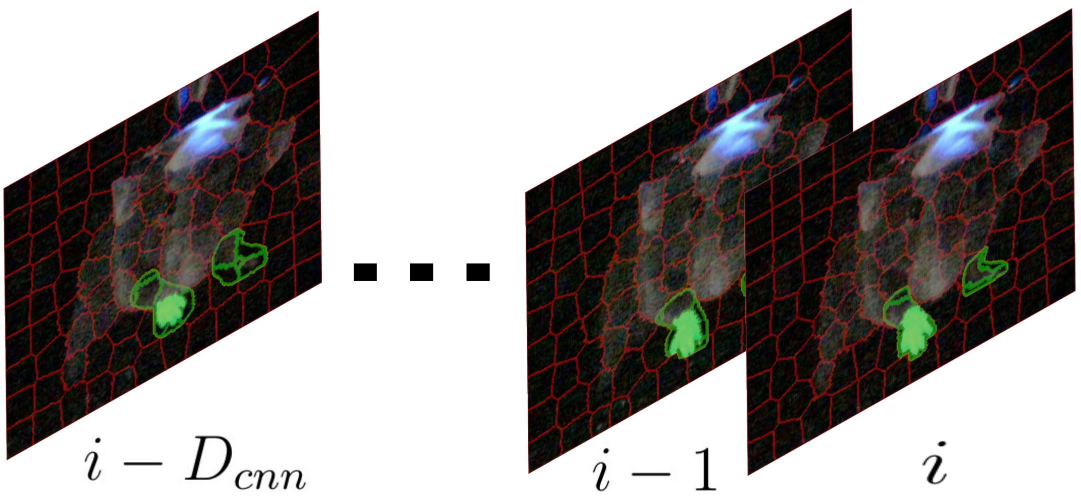


FIGURE 3.3: Enlarged view of how the constituent superpixel of a motion tube is tracked backwards through time. The superpixel shaded in green color is forming the tube whose closest relatives are tracked backward in time.

Notice that the procedure of locating the closest match of superpixels in previous frames allows us to handle situations where two superpixels converge in time (typically because the limb regions shrink as movement dynamics evolve). The algorithm 1 details the main steps of the procedure that outputs N motion tubes for each frame.

To reduce the dimensionality of the data and to extract meaningful appearance features, we use a 4-layer Convolutional Neural Network (CNN) (Abbas, Masip, and Giovannucci, 2020). We conjecture that since we train the CNN to learn how to discriminate between limb/non-limb categories from motion tubes, the features extracted will be more descriptive than handcrafted features. The input to the first convolutional layer is a 3D motion tube $\mathbb{M}^{Ws \times Hs \times D_{cnn}}$ where Ws and Hs represents spatial dimensions while D_{cnn} represents tube depth. The output of the CNN is a feature vector $\boldsymbol{x}_1$ obtained from the last layer (fully connected), of length L_f. Table 3.2 details the parameters for each layer used in this paper.

Motion sequences

Motion sequences integrate several temporally consecutive motion tubes to learn a sequential classifier that performs the image segmentation for limb tracking. For a given superpixel j in frame i, we extract one dimensional motion tube vector $\boldsymbol{x}_1$. Then we do the same process for the closest match of this superpixel in the previous frame and extract another one-dimensional feature vector $\boldsymbol{x}_2$. We repeat the process for the previous D_{lstm} frames. Finally we stack all these one dimensional feature vectors to form the motion sequence $\mathbb{X}$ (size $D_{lstm} \times Lf$) $= [\boldsymbol{x}_1; \boldsymbol{x}_2; ...; \boldsymbol{x}_{D_{lstm}}]$. We compute one motion sequence per superpixel and we train an LSTM Neural Network (Hochreiter and Schmidhuber, 1997a) to predict the probability of the sequence of belonging to a limb (see Appendix A).

Training data for the CNN and LSTM regressors

We use a CNN for the feature extraction and an LSTM for the image segmentation. In both cases, we train these models to provide a probability of the superpixel of being limb /non-limb. We use a mean squared error loss for the regression task.

Algorithm 1 Motion tubes construction

```
Result: Motion tubes for superpixel $I^s_{i,j}$
$k = 0$; M = { };
if $i < D_{cnn}$ then
    ▷ Not traversed the minimum number of frames yet while Number of available
    frames is less than required ($i < D_{cnn}$) do
        Append $\mathbb{I}_i$ to $\mathbb{I}$ at position $i$;
        Append $\mathbb{F}_i$ to $\mathbb{F}$ at position $i$;
    end
else
    Start from 1st superpixel ($j = 0$)
    while we have available superpixels ($j < N$) do
        Start from current frame i ($k = 0$)
        while we haven't traversed backwards $D_{cnn}$ frames ($k < D_{cnn}$) do
            if current frame i (k==0) then
                Place $\mathbb{I}^s_{i,j}$ at last location of the tube
            else
                Find centroid of superpixel $\mathbb{I}^s_{i,j}$ and store in $(x_{s:j}, y_{s:j})$.

                Project it onto frame $\mathbb{I}_{i-k}$ using optical flow as $(x_{s:j*}, y_{s:j*}) =$
                $(x_{s:j}, y_{s:j}) + \mathbb{F}_{i-k}(x = x_{s:j}, y = y_{s:j})$.

                Find the distance between the centroid of the projected superpixel
                and centroids of all superpixels in the previous frame and store it in
                $D_{s:}$. The closest relative of superpixel with centroid $(x_{s:j}, y_{s:j})$ is the
                one returned by
                $I^s_{j*} = argmin(D_{s:})$.

                Append $\mathbb{I}^s_{i,j*}$ at location $D_{cnn} - k$ of motion tube $\mathbb{M}^s_{i-k,j}$.
            end
            $k = k + 1$;
        end
        $j = j + 1$
    end
end
```

Training data are manually annotated in binary terms (limbs/non-limb) at a pixel level (segmentation mask). Nevertheless, the training algorithm requires a unique probabilistic value for all the pixels forming a superpixel. The training data is generated by manually annotating limbs in video frames. The limbs (moving or still) are labelled accurately by tracing their boundaries and then extracting a mask.

To generate a target label for a superpixel $I^s_{i,j}(x,y)$, the corresponding superpixel with same spatial coordinates x, y is extracted from the labelled frame. Let's say $x_{s:j}, y_{s:j}$ correspond to the x and y coordinates of all the pixels of $I^s_{i,j}$, the label $Y_{s:j}$ (label for superpixel j) is determined by Eq. 3.2 and 3.3.

$$Y_{s:j*} = \frac{\{(x,y)|x \in x_{s:j}, y \in y_{s:j} \wedge I(x_{s:j}, y_{s:j}) = 1\}}{\{(x,y)|x \in x_{s:j}, y \in y_{s:j}\}} \tag{3.2}$$

$$Y_{s:j} = \begin{cases} 0 & Y^s_{j*} <= 0.1 \\ Y_{s:j*} & Y^s_{j*} > 0.1 \end{cases} \tag{3.3}$$

The 'limb-ness' score is then associated with every pixel within such a superpixel. These new pixel values are used to generate a dense segmentation frame with the same width and height as the original frame. A simple thresholding on such segmentation frames produces a mask (Eq. 3.3). Such a mask can be compared to the manually annotated ground truth for evaluation.

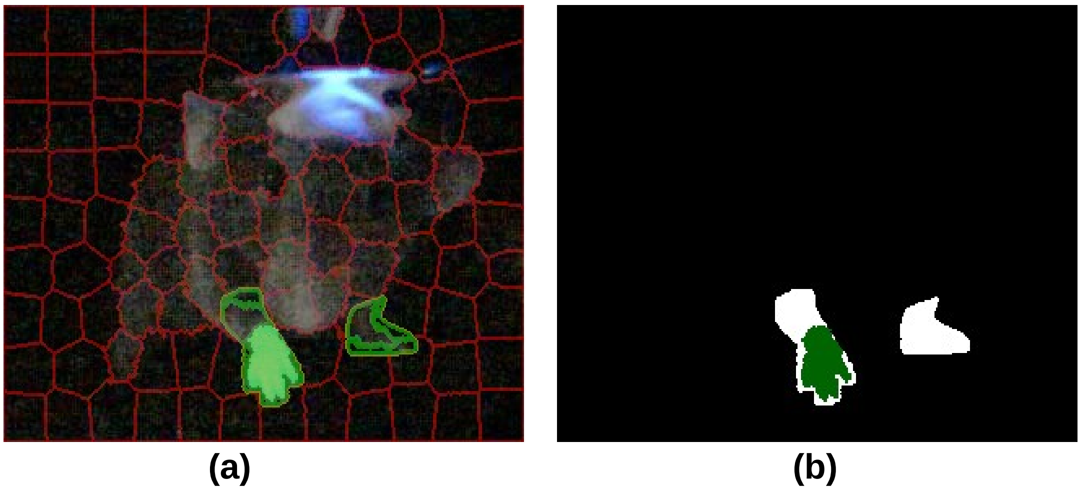


FIGURE 3.4: Example of how a label is given to a superpixel. First, the original frame is over-segmented into superpixels. To find out the target label for a superpixel, shaded in green in (a), the corresponding superpixel, also shaded in green in (b) is extracted and then evaluated according to Eq. 3.2 and 3.3

3.2 Experiments

3.2.1 Annotation

Two frontal videos (478 frames) and two lateral videos (1534 frames) are annotated by three human annotators independently. The degree of agreement between the three human annotators is higher than 95% on average, therefore the annotation is reliable to be used for training and testing purposes. The limbs (moving or still) are labeled accurately by tracing their boundaries and then extracting a mask.

To generate a target for a a motion tube created by algorithm 1 for, say, superpixel $I^S_{1,j}$, the corresponding superpixel with same spatial positions as that of $I^S_{i,j}$ is extracted from the labeled frame and the masks are then generated according to Eq. 3.2 and 3.3.

3.2.2 Motion tubes generation

We have experimented with a tube depth of 9 frames (75 ms). The size of the superpixels is controlled by the number of superpixels N in a frame. To include some spatial context to the individual slice of the tube, we have included an image patch of size 61 with the corresponding superpixel at its centre. Each superpixel slice is resized to 41×41 to keep the size uniform throughout. So for a tube of depth $D_{cnn} = 9$ frames, its size will be $41 \times 41 \times 9$. The tubes can only be generated once have traversed at least D_{cnn} frames of the video, so with a video length of 239 frames, $D_{cnn} = 9$, $N = 100$ and size of a superpixel patch being 41×41, we will have a total of 230×100 tubes, and the resultant data will have the dimension of $41 \times 41 \times 9 \times 23000$.

3.2.3 Building motion sequences from features extracted by CNN

A 20 layered CNN is trained on motion tubes with an input layer, 4 hybrid layers of convolutional and pooling operations, a dropout layer with a dropout ratio of 0.5, a fully connected layer and a regression layer. The hybrid layers consist of a convolutional layer, a ReLu activation layer, a batch normalization layer and an average pooling layer with parameter values given in Table 3.2. An illustration of CNN is shown in Fig. 3.5

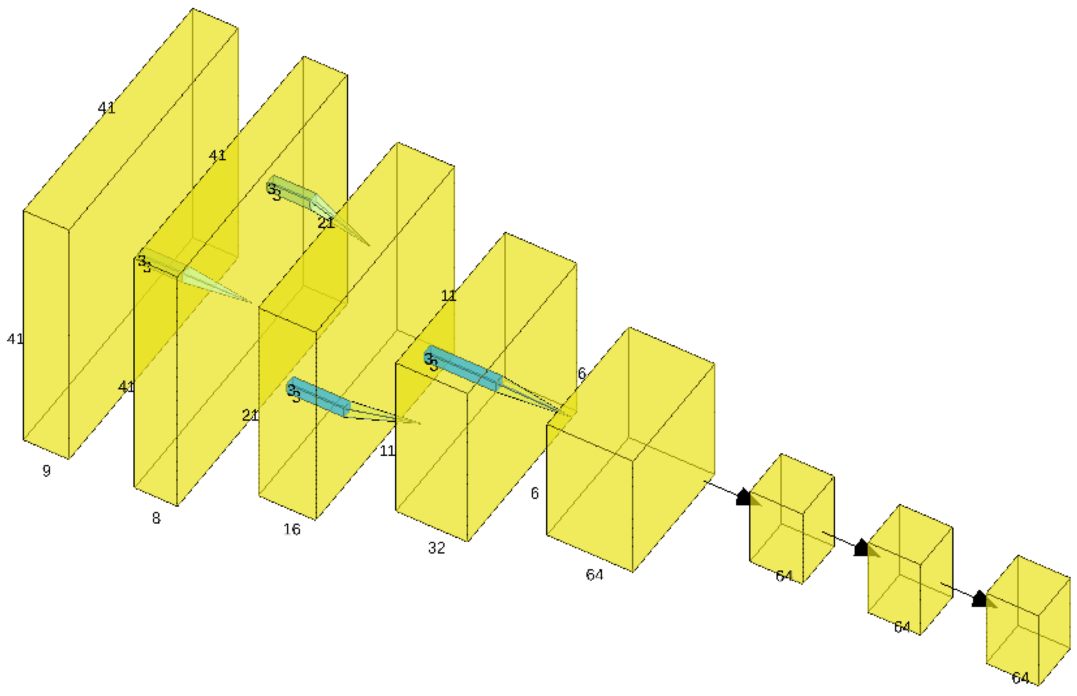


FIGURE 3.5: Illustration of the CNN employed for features extraction. It has four layers with 8, 16, 32 and 64 filters respectively. Each layer has 3×3 convolution followed by a batch normalization, ReLu activation and average pooling.

TABLE 3.2: Parameter values for the layers

	Layer 1	**Layer 2**	**Layer 3**	**Layer 4**
Number of filters	8	16	32	64
Filter size	3	3	3	3
Stride of pooling	2	2	2	2

We train the CNN as a feature extractor for later stages. The CNN is trained to predict the amount of "limbness" of a superpixel (see Eq. 3.2). Once the CNN is trained, we apply it on a motion tube and extract the output generated by the fully connected layer as the feature vector. Since the fully connected layer size is 64, the superpixel is represented by a vector of 64 features which are learned, not handcrafted. Since the motion tubes, width and height are dictated by the associated superpixels and their depth behaviour is dictated by the temporal history of the superpixels, we can loosely call the extracted features as spatiotemporal features.

3.2.4 Limbs detection by LSTM

Once the features are generated by CNN from motion tubes, they are used to create motion sequences for LSTMs (see algorithm 1). We generated 15 features-vectors-deep sequences. The length of sequence depth $= 15$ was chosen empirically. Therefore, a superpixel is represented by 15 spatio-temporal feature vectors of length 64 each, stacked together in a temporal sequence in correct chronological order.

We generated motion sequences for all the superpixels. So for a video of 239 frames, with a motion tube depth of 9 frames and motion sequence depth of 15 frames, the first motion sequence will be available after 24 frames which will leave $239 - 24 = 215$ frames. With 100 superpixels in each frame, feature length of 64 and motion sequence depth of 15, we will end up with a training data of $15 \times 64 \times 21500$ motion sequences.

A 6 layered LSTM is trained with the generated motion sequence data which have following architectures.

- An input layer
- An lstm layer with 64 neurons generating sequence as an output
- An lstm layer with 32 neurons generating sequence as an output
- A bi-lstm layer with 16 neurons generating a vector as an output
- A fully connected layer
- A regression layer

3.2.5 Experimental settings

Experiment 1 (setting 1)

In this setting, we trained the proposed pipeline on one video and tested it on the other video (767 frames for training and 767 frames for testing or 6.3 seconds of video for training and 6.3 seconds for testing).

Experiment 2 (setting 2)

In setting 2, we trained the proposed pipeline on 30% of available frames and tested it on the remaining 70% frames (460 frames for training and 1074 frames for testing or 3.8 seconds of video for training and 8.9 seconds for testing).

Experiment 3 (setting 3)

In setting 2, we trained the proposed pipeline on 10% of available frames and tested it on the remaining 90% frames (154 frames for training and 1380 frames for testing or 1.3 seconds of video for training and 11.5 seconds for testing).

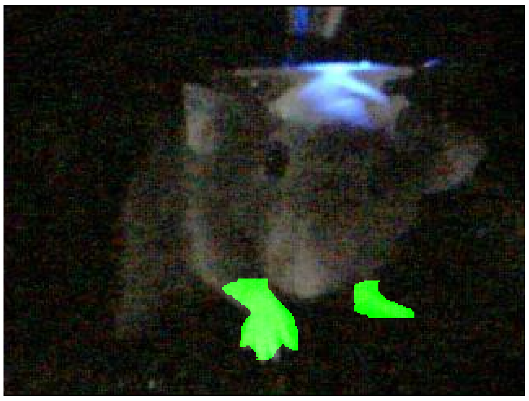
3.3 Results

The primary goal of gesture tracking is to detect all four limbs (in some cases, detect tail and head too) in all of the video frames. If there are occlusions, the pipeline should be able to avoid false positives and when the limb reappears after some time, the pipeline should be able to detect it right away (avoid false negatives). In this section, we present the results of the proposed pipeline on the dataset discussed in 2.1.1. We present both qualitative (visual) and quantitative results. For qualitative results, we present figures illustrating successful detection, false positives, false negatives and outliers. For quantitative results, we present the standard performance metrics for detection problems.

3.3.1 Qualitative results

Success cases

As discussed in earlier chapters, the apparent shape and size of the limbs are deformable because of locomotion. Therefore, successful detection depends upon the generalization capability of the pipeline. In Fig. 3.6 and 3.7, we show that the proposed pipeline has a good generalization capability in different conditions.


(A) A sample frame with limbs annotated from a frontal video

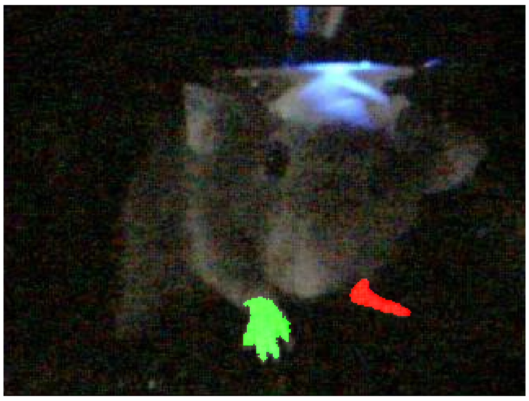
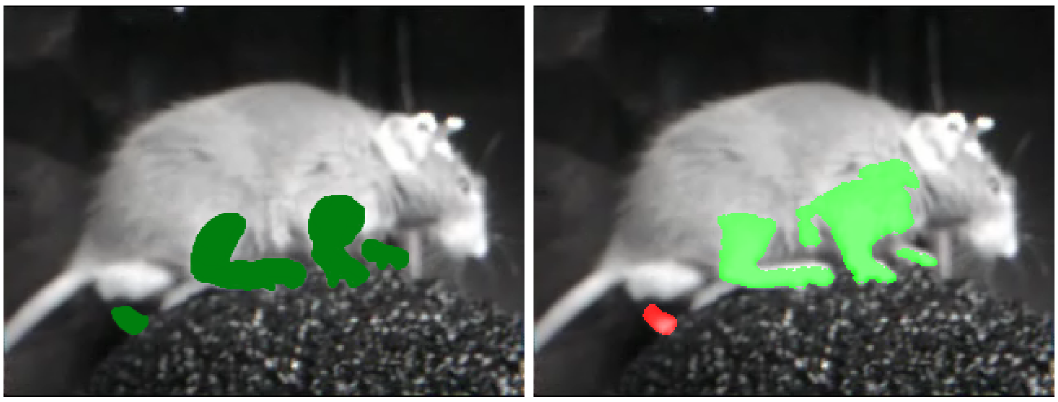

(B) Predicted limbs overlaid on a sample frame of frontal video

FIGURE 3.6: Sample frames from frontal videos. In (b), the areas overlaid by green mask are correct predictions while the areas overlaid by a red mask are missed limbs



(A) A sample frame with limbs annotated from a lateral video (B) A sample frame from a lateral video with detection results overlaid on original frame

FIGURE 3.7: Sample frames from lateral videos. In (b), the areas overlaid by green mask are correct predictions while the areas overlaid by a red mask are missed limbs

Failure cases

Since the successful detection of limbs depends upon a variety of factors (superpixel size, illumination, occlusion, presence of other body parts which might look or move like limbs etc.), therefore, the pipeline is not always able to infer correct results. In some cases, it might miss to detect a limb (false negative) or give high probability to superpixels not part of the limbs (false positives). In Fig. 3.8 and 3.9, we have shown some failure cases.

3.3.2 Quantitative results

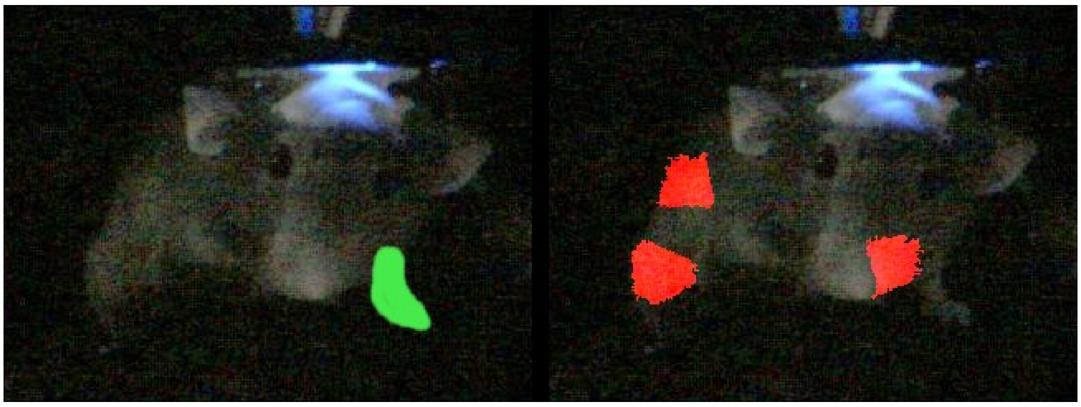
For quantitative analysis, we calculated Jaccard Index (Levandowsky and Winter, 1971), precision, recall and detection accuracy according to the following formulae. Let's say in the video frame I_i, the limb is represented by mask I_i^L with same size as I_i, the detected limb is represented by mask $I_i^{L^*}$,then Jaccard Index can be defined by Eq. 3.4

$$J_i = \frac{I_i^L \cap I_i^{L^*}}{I_i^L \cup I_i^{L^*}} \tag{3.4}$$

The Jaccard Index helps us decide if a limb is present or absent. If the Jaccard Index is higher than a threshold (0.5 in this case), we conclude we have detected the limb, and vice versa.

To understand the performance in terms of precision, recall and accuracy, we first define the true positives and negatives. A true positive (TP) refers to a case when a limb is present in both the actual frame and predicted result, true negative (TN) refers to a case when a limb is absent from both original frame and predicted result, false positive (FP) refers to a case when a limb is not present in the original frame but it is still detected, false negative (FN) refers to a case when a limb is present in the original frame while missing from the predicted result. With these definitions of TP, TN, FP and FN, we define precision, recall and accuracy in Eq. 3.5, 3.6 and 3.7.

$$Precision = \frac{TP}{TP + FP} \tag{3.5}$$


(A) Failure case 1, false positive

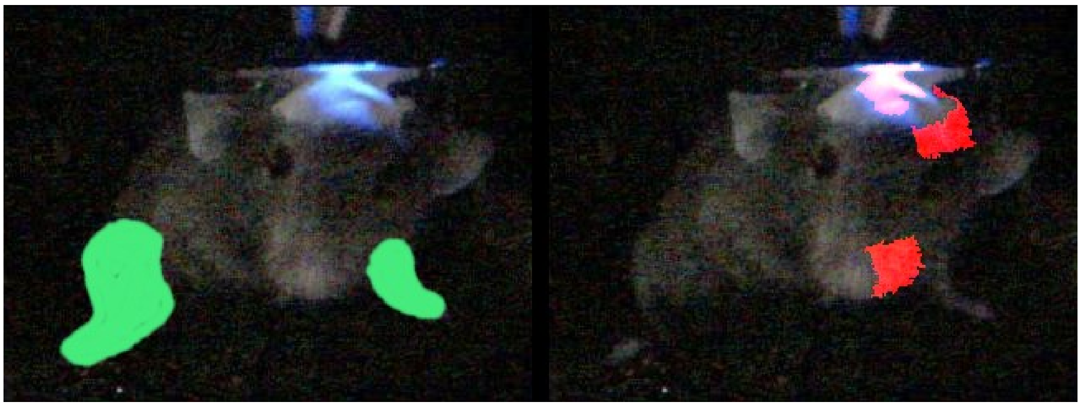

(B) Failure case 2, false negative

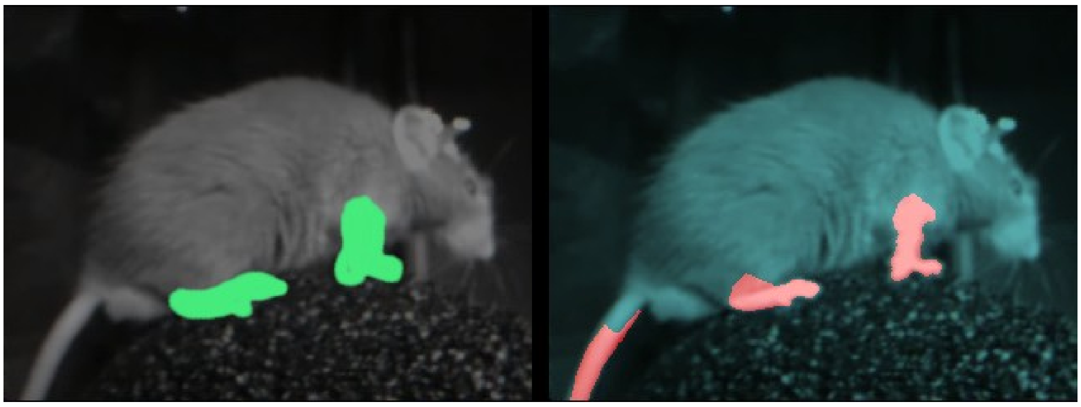
FIGURE 3.8: Sample failure cases. Sample failure cases are shown along with the original frame. In the left panel, we have the original frame and in the right panel, we have the prediction overlaid on the original frame. (a) shows a false positive while (b) shows a false negative.

$$Recall = \frac{TP}{TP + FN} \tag{3.6}$$

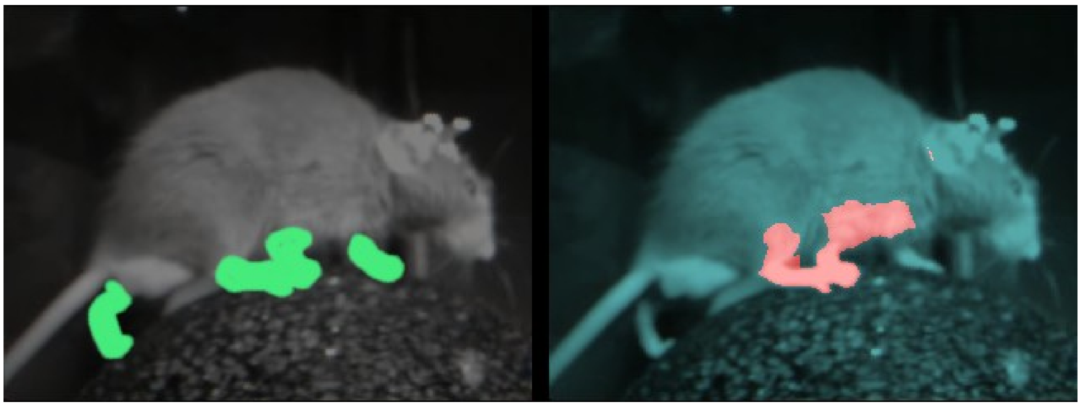
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.7}$$

3.3.3 Lateral videos

Lateral videos have low noise ratio and the limbs are visible and identifiable in most of the frames. As seen in Fig. 2.3, the limbs can be told apart from the background easily. To understand the impact of the number of frames used for training the pipeline, we conducted experiments with lateral videos according to the settings described in 3.2.5. The results for these three settings are summarized in Tables 3.3,3.4 and 3.5



(A) Failure case 1, false positive



(B) Failure case 2, false negative

FIGURE 3.9: Sample failure cases for lateral videos. Sample failure cases are shown along with the original frame. In the left panel, we have the original frame and in the right panel, we have the prediction overlaid on the original frame. (a) shows a false positive while (b) shows a false negative.

TABLE 3.3: Performance measure for lateral videos in setting 1

	Precision	Recall	Accuracy
Front left limb	88.2%	38.8%	81.2%
Front right limb	98.0%	98.5%	96.6%
Hind right limb	96.8%	98.3%	95.3%

3.3.4 Frontal Videos

Compared to lateral videos, frontal videos contain high amounts of noise (both salt and pepper and equipment noise). The limbs are not always identifiable from the body or background due to heavy noise. Therefore, the detection and tracking performance is worse than that for lateral videos. A sample annotated frame is shown in Fig. 2.2.

The quantitative results for setting 1, 2 and 3 (see section 3.2.5) are shown in tables 3.6, 3.7 and 3.8.

Table 3.4: Performance measure for lateral videos in setting 2

	Precision	Recall	Accuracy
Front left limb	86.9%	40.2%	81.0%
Front right limb	96.6%	91.3%	88.5%
Hind right limb	97.6%	96.2%	95.0%

Table 3.5: Performance measure for lateral videos in setting 3

	Precision	Recall	Accuracy
Front left limb	77.4%	38.7%	77.0%
Front right limb	97.8%	94.8%	92.8%
Hind right limb	94.7%	95.0%	90.1%

Table 3.6: Performance measure for frontal videos in setting 1

	Precision	Recall	Accuracy
Front left limb	91.5%	71.4 %	79.1%
Front right limb	92.9%	91.7%	88.2%
Hind right limb	89.1%	70.7%	80.5%

Table 3.7: Performance measure for frontal videos in setting 2

	Precision	Recall	Accuracy
Front left limb	95.7%	71.0%	77.8%
Front right limb	93.8%	87.9%	86.3%
Hind right limb	92.1%	72.5%	82.2%

Table 3.8: Performance measure for frontal videos in setting 3

	Precision	Recall	Accuracy
Front left limb	94.3%	68.3%	74.3%
Front right limb	93.4%	87.6%	85.1%
Hind right limb	91.8%	67.4%	78.0%

As evident from Tables 3.3,3.4,3.5, 3.6,3.7 and 3.8, the detection performance does drop as we reduce the number of training samples, however, the drop in performance is not as high as the drop in number of samples (A maximum drop of 8.1% in accuracy for lateral videos while a maximum drop of 4.2% in accuracy for lateral videos against a 60% drop of number of training samples). The reason behind this phenomenon is that since the mouse is head fixed, therefore, from the camera perspective, the limbs are moving in a (rough) cyclic fashion. So if we can train the network with enough samples that it covers one motion cycle, the network should be able to learn properly. Moreover, since fewer frames are enough to identify the apparent changes in the shape of the limbs, the pipeline can learn despite the small number of frames.

To evaluate the tracking performance of the proposed pipeline, we tracked the centroids of the front left limb in both manually annotated (Ground Truth) and predicted frames. We compared them qualitatively by plotting the annotated and actual tracks side by side and then for compared them quantitatively by finding the mean distance between manually annotated and predicted positions (centroids) of the front left limb according to equation 3.8.

$$D_i^p = \sqrt{(x_G^c - x_P^c)^2 + (y_G^c - y_P^c)^2} \tag{3.8}$$

where x_G^c and y_G^c are the coordinates of the front left limb coordinates in ground truth frames and x_P^c and y_P^c are the coordinates of the front left limb coordinates in predicted frames.
The manually annotated and predicted limbs in frontal videos are separated by a mean distance of **3.84 pixels**. Tracks (plots of centroids as a function of position) of the left frontal limb from a frontal video are shown in Fig. 3.10.

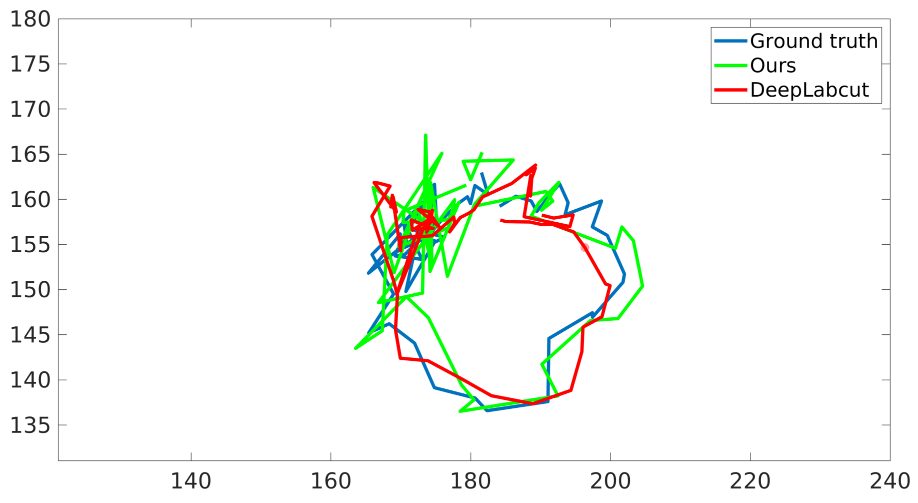


FIGURE 3.10: Actual path and path predicted by our approach and DeepLabcut (Mathis et al., 2018a) of front left limb in a frontal video.

To compare the effectiveness of our approach, we compared it on tracking performance to the Haar cascades based approach proposed by (Giovannucci et al., 2018) because they have used the same lateral videos as we have. We plotted the tracks of limbs extracted from the annotated video along with tracks of predicted limbs and calculated the distance between centroids of manually annotated limbs and predicted limbs. We found out that the manually annotated limbs and the limbs predicted by the proposed approach are separated by **2.24 pixels** on average in lateral videos. The same separation grows to **9.5 pixels** on average when limbs are predicted by Haar cascades. The tracks of a right hind limb from a lateral video are shown in Fig. 3.11 and 3.12.

The average distance between (defined by Eq. 3.8) the centroids of a detected limb and ground truth for all of the trajectories shown in Fig. 3.10 and 3.11 reflects the average tracking accuracy of the pipelines. Table. 3.9 summarizes this average distance for DeepLabcut, Deepposekit and the proposed pipeline.

From the qualitative and quantitative results, we can draw the following conclusions.

- Learning based approaches perform better than unsupervised statistical approaches.

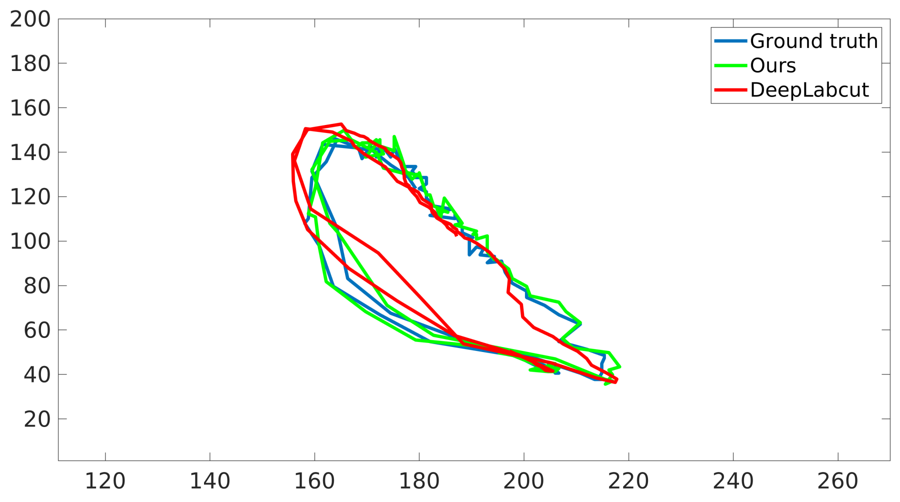


FIGURE 3.11: Actual path and path predicted by our approach and DeepLabcut (Mathis et al., 2018a) of hind right limb in a lateral video. The path is calculated by finding centroids of hind right limb in each frame and then plotting its y coordinate against its x coordinate.

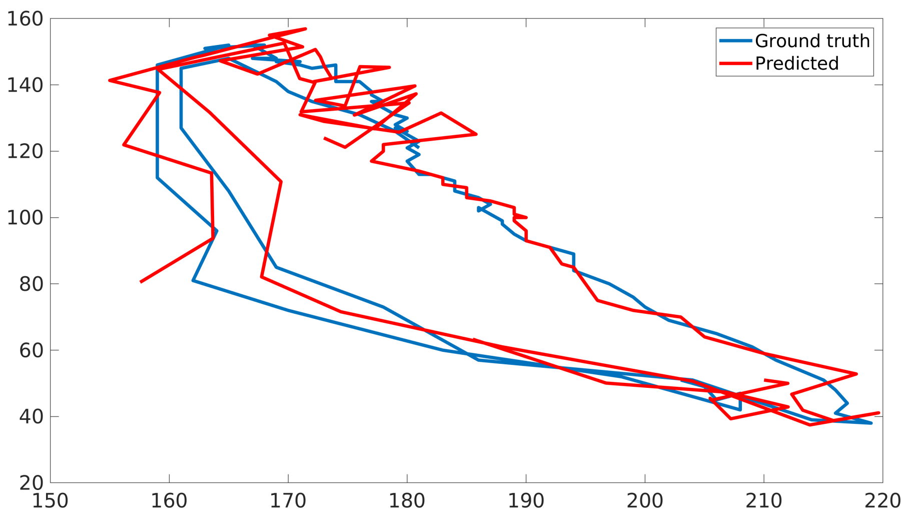


FIGURE 3.12: Actual path and path of hind right limb predicted by Haar cascades as proposed by (Giovannucci et al., 2018) in a lateral video. The path is calculated by finding centroids of hind right limb in each frame and then plotting y coordinate against its x coordinate.

- The proposed pipeline has comparable and sometimes superior performance than state of the art in term of tracking accuracy.
- The proposed pipeline is superior to the current state of the art in terms of its explicit consideration of the temporal nature of the problem.
- The proposed pipeline can be optimized to be less data hungry and more efficient.

TABLE 3.9: Tracking mean squared error of the trajectories in the frontal and lateral settings.

Method	**Frontal**	**Lateral**
Deeplabcut	5.5	6.35
Deepposekit (StackedHourglass)	3.97	5.39
Deepposekit (StackedDensenet)	4.74	5.63
Motion Tubes	2.24	3.84

Chapter 4

Neural activity (neurons) segmentation

As mentioned in earlier chapters, action potentials cause transient changes in the intracellular concentration of calcium ions which are detected by observing the fluorescence of calcium indicator molecules, typically using two-photon microscopy in the mammalian brain (Denk, Strickler, and Webb, 1990b). When a single image plane is scanned repeatedly, we get a time series of 2D neural activity images. This is effectively a video in which neurons blink whenever they are active (Chen et al., 2013).

In the traditional workflow for extracting neural activities from the video, a human expert manually annotates the regions of interest (ROIs) corresponding to individual neurons (Dombeck et al., 2007). Within each ROI, pixel values are summed for each frame of the video, which yields the calcium signal of the corresponding neuron versus time. A subsequent step may deconvolve the temporal filtering of the intracellular calcium dynamics for an estimate of neural activity with better time resolution. The traditional workflow has the deficiency that manual annotation becomes laborious and time consuming for very large datasets. Furthermore, manual annotation does not demix the signals from spatially overlapping neurons.

4.1 Methods

4.1.1 Problem statement

To trace neuron activity in the 3D image stacks (neuron activity videos) produced by calcium imaging, the following challenges need to be taken into consideration.

- The temporal dynamics (rise time, decay time, decay constant) of different calcium indicators are different, therefore, a unified framework to define temporal profiles of different calcium indicators cannot be defined.
- When a single image plane (x,y) is recorded over an extended time, two neurons which are located at different z-coordinates might appear spatially overlapped (partially or fully). They can only be told apart when they are active at different times.
- The many different neurons do not share the same activation pattern, some might fire more often than others.
- Due to the way calcium indicators work, they might accumulate near cell borders in some cases instead of getting distributed uniformly across the neuron cell body, thus creating a toroid shape instead of solid disc shape in fluorescence images.

4.1.2 Spatio-temporal U-Net (STUNet)

U-Net is a convolutional neural network architecture designed for medical image segmentation (Ronneberger, Fischer, and Brox, 2015). It is based on an encoder-decoder architecture with one arm encoding input images into compact representations and the other arm unfolds that representation into dense segmentation maps. The successive stages of encoder arm compress (downsample) the input image into compact feature maps while the successive decoding stages upsample the encoded feature maps. The encoder arm intermediate outputs are copied to the decoder arm which then guides the upsampling of feature maps. This architecture has delivered many breakthroughs in medical image segmentation (Ronneberger, Fischer, and Brox, 2015; Li et al., 2018; Klibisz et al., 2017).

Despite its performance on dense segmentation of medical images, U-Net architecture was primarily designed for 2D image segmentation. We have designed a new U-Net inspired architecture for neuron segmentation in 3D image stacks which not only exploits the segmentation power of U-Net but it also does not discard the information contained in the temporal activity of neurons. We have designed a modular U-Net; a network which can be tweaked to process 3D spatio-temporal stacks of different temporal dimensions by including or excluding some layers. The U-Net consists of encoding layers, spatio-temporal downsampling layers, middle layer, decoding layers, spatial upsampling layers and an output layer. In the following sections, we explain the internal architecture of encoding, decoding, spatio-temporal downsampling, middle, spatial upsampling and output layers and how the connectivity works. We also illustrate and explain two variants of the U-Net. The encoding and decoding layers are based on the relatively new architecture of LSTM which implements an LSTM pipeline for images. This LSTM architecture is called Convolutional LSTM (which we will refer to as ConvLSTM or CLSTM) and is explained below.

Convolutional LSTM (ConvLSTM)

Introduced by Xingjian. et al. (Xingjian et al., 2015), convolutional LSTM is a hybrid of convolutional operations and LSTM network. ConvLSTM is an extension of Fully Connected LSTM (FC-LSTM) (Sainath et al., 2015) architecture to have convolutional structures in both the input-to-state and state-to-state transitions. The FC-LSTM, though powerful for handling temporal correlation, contains too much redundancy for spatial data. The ConvLSTM mitigates this problem by replacing point operations in input-to-state and state-to-state transition by convolutions. An LSTM cell processes sequential data by updating it's hidden states through which information flow is regulated by gates (see Appendix A for details). Input gate regulates the inputs, forget gate regulates how much information flows between state transitions and output gate regulates the outputs. The architecture of a typical LSTM cell is shown in Fig. 4.1.

For a ConvLSTM cell, the input sequence $\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_t$, cell outputs $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_t$, hidden states $\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_t$, and gates i_t,f_t,o_t of the ConvLSTM are 3D tensors whose last two dimensions are spatial dimensions (rows and columns). For better visualization, let's assume the ConvLSTM cell is a three dimensional grid with x and y dimensions representing the spatial information associated with the image sequences and z dimension representing vectors associated with the inputs and hidden states. It is to be noted that if only one pixel is considered in the image sequence, the input sequence it forms and the hidden states it will produce can be represented by one dimension temporal vectors. Therefore, the inputs and states of a pixel can be perceived as vectors standing on a spatial grid. The ConvLSTM determines the future

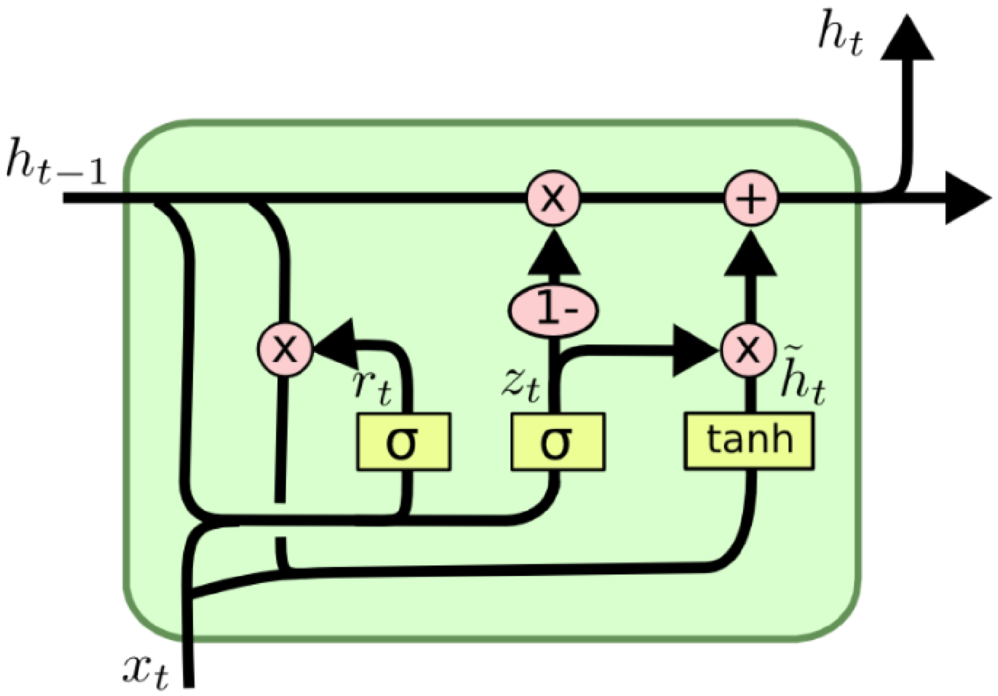


FIGURE 4.1: Illustration of the inner structure of an LSTM. Source: *Understanding LSTM Networks – colah's blog* 2020

state of a certain cell in the grid by the inputs and past states of its local neighbors. This can easily be achieved by using a convolution operator in the state-to-state and input-to-state transitions (see Fig. 4.2). The key equations of ConvLSTM, modified from the state equations of FC-LSTM (see Appendix A) are shown in Eq. (4.1) below, where '*' denotes the convolution operator, '∘', denotes the Hadamard product and σ denotes non-linear activation function:

$$\begin{aligned}
i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_i) \\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \\
\mathcal{H}_t &= o_t \circ tanh(\mathcal{C}_t)
\end{aligned} \tag{4.1}$$

If the states are viewed as the hidden representations of moving objects, a ConvLSTM with a larger transitional kernel should be able to capture faster motions and vice versa. Usually, all the states of the LSTM are initialized to zero which corresponds to "total ignorance" of the future. Similarly, zero-padding of the hidden states sets the state of the outside world to zero and assumes no prior knowledge.

4.1.3 Layers

Encoding Layer

The encoding layer is built upon ConvLSTM layers and normal convolutional layers. The architecture of an encoding layer is shown in Fig. 4.3. As shown in Fig. 4.3, the encoding layers accept one 3D tensor as input and have 4 outputs, two 3D tensors (one 3D tensor can be fed as the input of next encoding stage while the other 3D

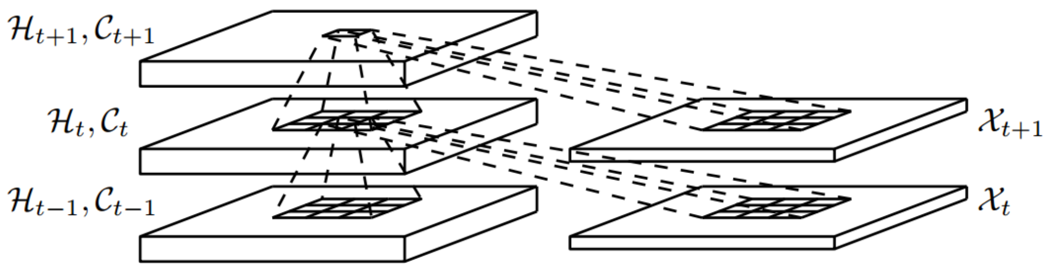

$\mathcal{H}_{t+1}, \mathcal{C}_{t+1}$
$\mathcal{H}_t, \mathcal{C}_t$
$\mathcal{H}_{t-1}, \mathcal{C}_{t-1}$
$\mathcal{X}_{t+1}$
$\mathcal{X}_t$

FIGURE 4.2: Architecture of a ConvLSTM cell.

tensor is copied to the decoding stage of a similar level), a 2D tensor and hidden states of the ConvLSTM. The 3D input tensor is fed into a ConvLSTM whose 3D output is copied to the decoding stage. The 3D output then passes into two $3 \times 3 \times 3$ convolutional stages (the number of filters is decided by the user) followed by 3D downsampling. Finally, the downsampled 3D tensor is passed through a batch-norm and reLu activation layer (see Appendix B for more details on reLu activation). The first hidden state of the ConvLSTM is passed through two 3×3 convolutional stages followed by 2D downsampling, batch-norm and reLu activation. The downsampled 2D output is copied to a similar level decoding stage after the reLu activation. All the hidden states are copied to a similar level decoding stage as well.

The encoding layer has two processing streams; one processes the 3D input in 3D (3D convolutions and 3D pooling) while the other processes it in 2D (2D convolutions and 2D pooling). The split between 3D and 2D occurs at the ConvLSTM which forwards its 3D output to the 3D arm while the first 2D hidden state is forwarded to the 2D processing arm. The intuition behind simultaneous 2D and 3D processing is that the 3D arm will learn information from temporal and spatial domains while the 2D arm will learn a better spatial representation of the input 3D tensor. Since the encoding layer accepts a 3D tensor and has a downsampled 3D tensor (along with a downsampled 2D tensor, the original 3D output of the ConvLSTM, and hidden states of the ConvLSTM), multiple encoding layers can be stacked on top of each other to form the encoding arm of the spatio-temporal U-Net. This layer can be summarized as follows.

- It accepts a 3D input.
- It processes the 3D input by first passing it through a convolutional LSTM and then through 2D convolutional layers, so it produces 3 kinds of outputs; 3D, 2D and LSTM states.
- It produces both 3D and 2D downsampled outputs. So this layer acts as a spatio-temporal downsampling layer and can be used for downsampling both in spatial and temporal domains.

Decoding Layer

The decoding layer is also built around a ConvLSTM at the beginning of the layer. A decoding layer accepts 4 inputs, a 3D input, initial hidden states (copied from the encoding layer at a similar level), a 2D tensor copied from the encoding layer (output of the 2D processing arm of the encoding layer) and another 2D tensor which is the output of the previous decoding layer. A standard decoding layer is shown in Fig. 4.4.

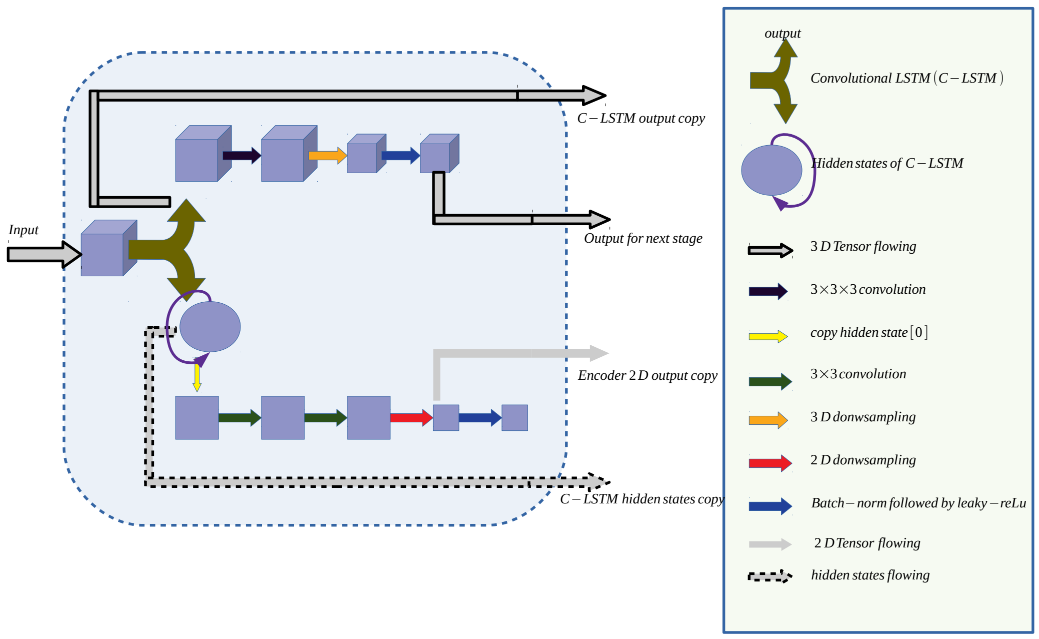


FIGURE 4.3: The encoding layer is built around a ConvLSTM cell. It accepts a 3D input (a video segment or a spatio-temporal image stack) as an input. First, the 3D inputs are processed by a ConvLSTM cell; we copy the hidden states and 3D output of the ConvLSTM to be used as inputs in a decoding layer at a similar level. The 3D output is then passed through a $3 \times 3 \times 3$ convolutional layer followed by 3D downsampling, batch-norm and leaky-relu. The output of this operation is then fed into the next encoding layer. Simultaneously, the first hidden state (a 2D tensor) is taken and fed into two 3×3 convolutional layers followed by 2D downsampling, batch norm and leaky-relu. The output of this operation is then copied to the decoding layer at a similar level. Therefore, two data-streams are running inside an encoding layer, a 3D data stream which processes the input both in temporal and spatial domains (to learn spatio-temporal features) while another 2D data stream processes the input in 2D (spatial domain only).

As shown in this figure, the 3D tensor and hidden states copied from the encoding layer initialize the ConvLSTM. The first hidden state of this ConvLSTM is passed through a 3×3 convolutional stage. The 2D tensor copied from the encoding stage and output of previous decoding stage is concatenated and then passed through an upsampling and deconvolutional layer. The resultant 2D tensor and the output of 3×3 convolutional stage are concatenated and passed through another 3×3 convolutional layer. Finally, the output is passed through a batch-norm and reLu activation layer resulting in a final upsampled 2D tensor. This upsampled 2D tensor can be fed to the next stage decoding layer or the output layer. This layer can be summarized as follows.

- It accepts a 3D input, LSTM states and a 2D input.
- It passes the 3D input and LSTM states through a CLSTM.

- The 2D input is upsampled and added to one of the output states of the CLSTM.
- This layer produces a spatially upsampled output, so given the LSTM states, this layer can be used a spatial upsampling layer.

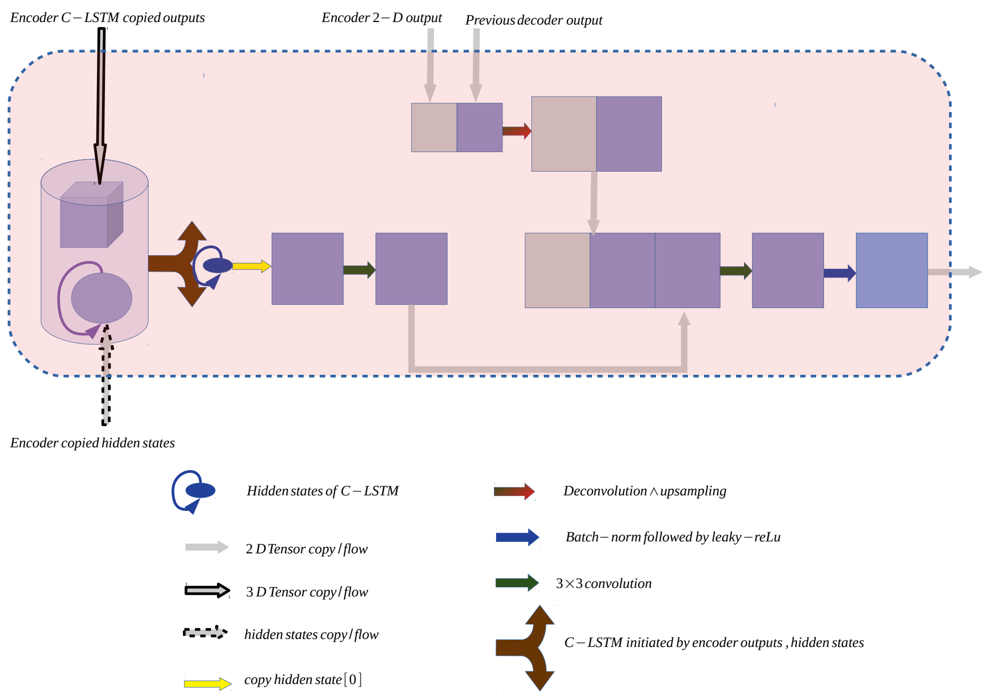

Encoder C − LSTM copied outputs
Encoder 2 − D output
Previous decoder output
Encoder copied hidden states
Hidden states of C − LSTM
2 D Tensor copy / flow
3 D Tensor copy / flow
hidden states copy / flow
copy hidden state [0]
Deconvolution ∧ upsampling
Batch − norm followed by leaky − reLu
3×3 convolution
C − LSTM initiated by encoder outputs , hidden states

FIGURE 4.4: The decoding layer is also built around a ConvLSTM cell but in this case, the ConvLSTM cell is initialized by outputs and states copied from the encoding layer at a similar level. Once the ConvLSTM cell is initialized, its first hidden state is copied and passed through a 3×3 convolutional layer. The 2D output copied from encoding layer at a similar level is concatenated with the 2D output of earlier decoding layer and then passed through deconvolution and upsampling stage. The upsampled 2D tensor is then concatenated with the first hidden state of the ConvLSTM and passed through another 3×3 convolutional layer. The 2D output of this convolutional layer is passed through a batch-norm and leaky-relu layer and then fed into the next decoding layer.

Middle (Bottleneck) Layer

The bottleneck layer is composed of 2 3×3 convolutional layers followed by a batch-norm and leaky reLu activation layer. The middle/bottleneck layer simply adds another layer of feature extraction in the spatial domain. The input to the bottleneck layer is a 2D tensor and the output is another 2D tensor with the same dimensions. This 2D output tensor is fed to the last decoding layer. A standard middle layer is shown in Fig. 4.5.

Spatio-Temporal Downsampling Layer

This layer is specifically designed to downsample stacks in temporal dimension as well as spatial dimensions. It consists of 3D convolution followed by downsampling

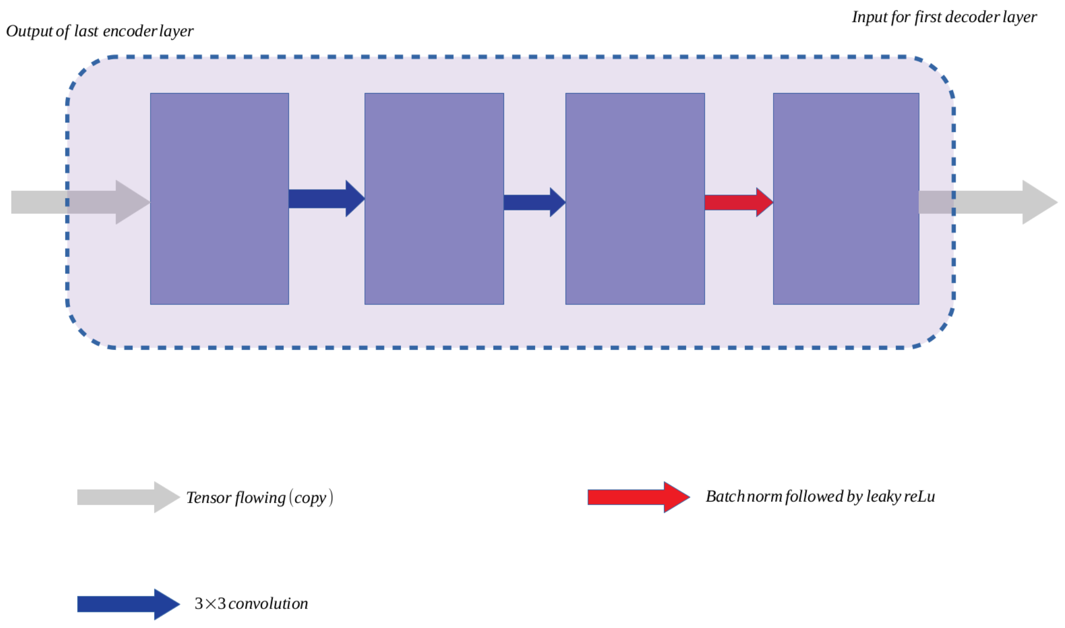

Output of last encoder layer
Input for first decoder layer
Tensor flowing (copy)
Batchnorm followed by leaky reLu
3×3 convolution

FIGURE 4.5: The middle layer consists of two 3×3 convolutional layers followed by the batch norm and leaky-relu. This layer simply acts as a bridge between the last encoding layer and first decoding layer. The output layer consists of a 1×1 convolutional layer which maps the outputs of the network onto the desired number of labels in the predicted mask.

in temporal dimension and batch normalization. The 3D convolution makes sure that information loss from downsampling the temporal dimension does not affect the learning ability of the network. Although every encoding layer also has a temporal downsampling block, inserting more encoding block can either overload the memory or the spatial dimensions of the input stack might not allow it since the encoding layer also has a spatial downsampling block. The temporal downsampling layer solves both of these problems. This layer can be summarized as follows.

- It accepts a 3D input.
- It can perform downsampling either in the spatial domain or temporal domain or both.
- Given a 3D input, this layer can be used for downsampling both in spatial and/or temporal domain

The block diagram of a temporal downsampling layer is shown in Fig. 4.6

Spatial Upsampling Layer

This layer is designed to upsample images in the spatial domain. If there are memory and processing power constraints, this layer can be used instead of the decoding layer to upsample the feature maps in the spatial domain. It receives two inputs; one is the feature map from the corresponding level in the encoding arm of the network copied directly and the other one is the feature map coming from the preceding layer in the decoding arm. The feature map coming from the encoding arm passes through a 1×1 convolution, the feature map from the previous decoding layer passes through a 3×3 convolution and then they are added. This is followed by a transpose convolutional

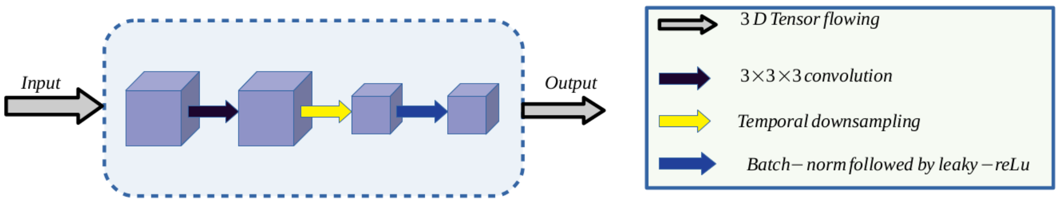


FIGURE 4.6: The temporal downsampling layer consists of one 3×3×3 convolutional layer followed a $(2, 1, 1)$ downsampling layer which only downsamples in the temporal domain (In our case, the first dimension is the temporal dimension). This is followed by a batch norm and leaky-relu layer. This layer downsamples the input 3D spatio-temporal stack only in the temporal domain while keeping the learned feature space relatively constant by including 3D convolution before downsampling.

block/upsampling block supported by a 3×3 convolution. The convolution is then followed by a reLu activation and batch norm. This layer can be summarized as follows.

- It accepts a 2D and a 3D input.
- It generates a 2D tensor from the 3D input by collapsing the temporal dimension and then add it to the 2D input.
- It upsamples the tensor in the spatial domain.
- This layer can be used as a general-purpose upsampling layer with more learning power than a standard upsampling layer.

The block diagram of a spatial upsampling layer is shown in Fig. 4.7

Output Layer

The output layer transforms the output of the final decoding layer in the required shape. Depending upon the number of filters employed, the output of the final decoding layer might have multiple layers while we might need just two output layers or three output layers depending on the choice of masks. Therefore, the output layer is composed of one 1×1 convolution with the number of filters equal to the number of required classes. It is followed by sigmoid activation. A typical output layer is shown in Fig. 4.8.

4.1.4 End to end training

As mentioned in 4.1.1, the neurons which are spatially overlapped in x,y dimensions but non overlapped in z dimension pose a challenge due to the nature of imaging modality since they appear overlapped in the acquired image stacks when viewed through a summary image. The only way to tell them apart is to observe their temporal activity. Although they are spatially overlapped, yet they might not be active at the same time. The simple way to incorporate this information for neurons segmentation is to split the image stacks into multiple smaller stacks along the temporal dimension. By doing so, we increase the chances of creating temporal windows where the overlapped neurons are active at different times. But to train our network on this dataset, we need to create a target mask for each of these windows

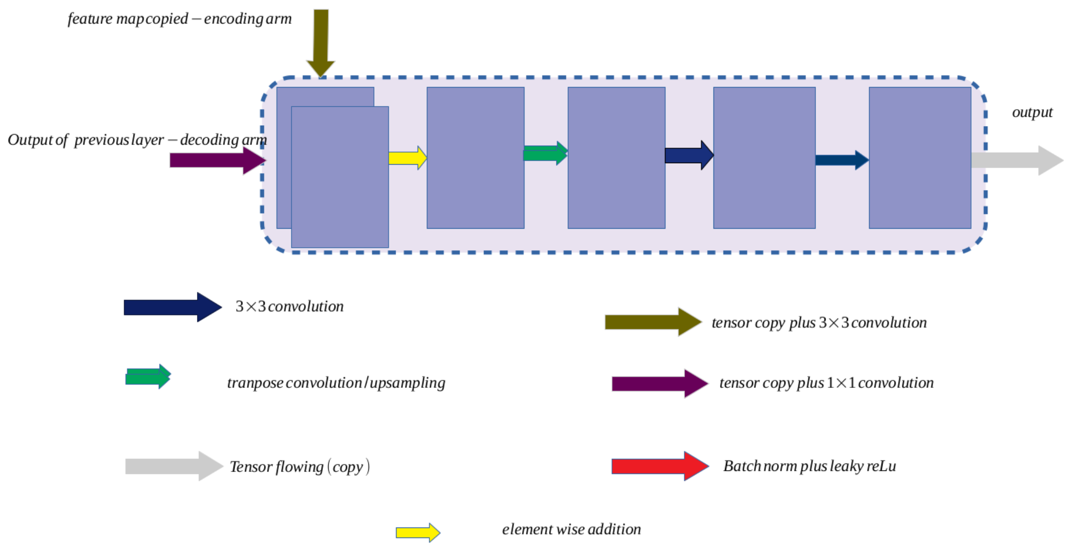

feature map copied – encoding arm
Output of previous layer – decoding arm
output
3×3 convolution
tensor copy plus 3×3 convolution
tranpose convolution / upsampling
tensor copy plus 1×1 convolution
Tensor flowing (copy)
Batch norm plus leaky reLu
element wise addition

FIGURE 4.7: The spatial upsampling layer consists of two convolutional layers (one 1×1 convolutional layer to match feature map of 2D input from the previous decoding layer and a 3×3 convolutional layer applied on the input coming from the encoding arm. The encoding arm input is first averaged on temporal axis), one transpose convolutional layer followed by a 3×3 convolution, the temporal downsampling layer consists of one $3 \times 3 \times 3$ convolutional layer. This is followed by a batch norm and leaky-relu layer.

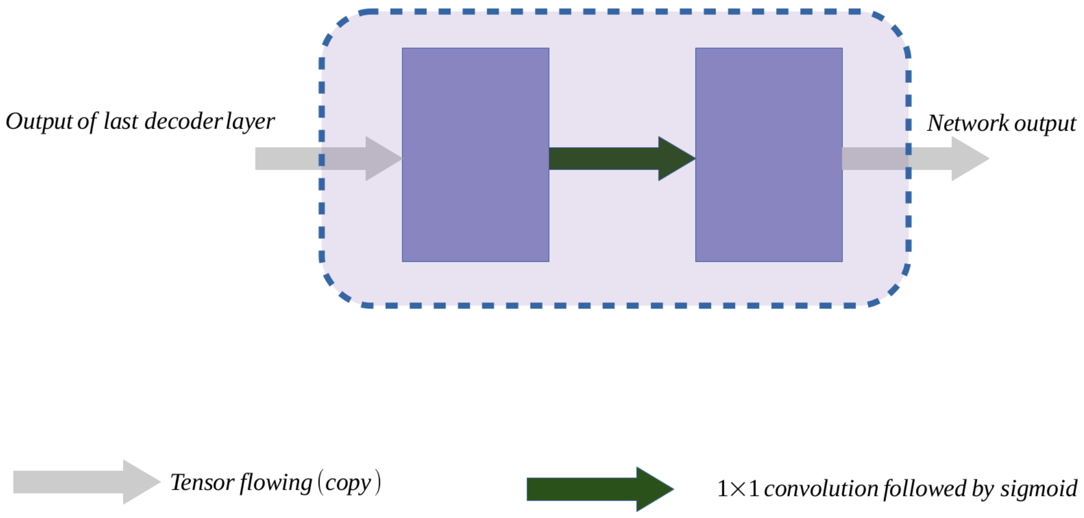

Output of last decoder layer
Network output
Tensor flowing (copy)
1×1 convolution followed by sigmoid

FIGURE 4.8: The output layer.

which is time-consuming. There are automated ways of creating the temporal masks (Soltanian-Zadeh et al., 2019) but those masks are not guaranteed to be accurate. We have designed an end-to-end pipeline which deals with both of these problems. We have made the following assumptions:

- All the neurons in the field of view will fire at least once when we consider a large portion of the stack (say 90%).
- We know the spatial coordinates of all the neurons at training time, not just the combined binary masks of all the neurons.

With the above-mentioned assumptions in mind, we propose two processing pipelines.

4.1.5 Hierarchical Training

For this training, we use the network illustrated in Fig. 4.9. The training pipeline is created in steps. For the sake of simplicity, let's assume we have 512 frames in the input stack to process.
Step 1: First 64 frames are divided into 8 stacks, with each stack containing 8 frames.
Step 2: The network is repeatedly applied to the 8 stacks, resulting in 8 output masks. These masks are concatenated in correct chronological order to create a stack of 8 masks.
Step 3: Steps 1 and 2 are repeated for frames 65 to 128, then 129 to 192 until 512. By the end of step 3, we have 8 stacks of output masks with each stack containing 8 output masks.
Step 4: Once again, we apply the network to the stacks obtained in step 3 repeatedly, resulting in 8 new masks. These masks are again concatenated in correct chronological order to create a final stack of 8.
Step 5: Apply the network once again to the final stack to get the final mask. In the training stage, this mask is used for error backpropagation while at the inference stage, this mask is used as the final output. The 5-steps pipeline is illustrated in Fig. 4.10.

4.1.6 Single Stage Training

For single-stage training, we either increase the number of layers in the network illustrated in Fig. 4.9 or we introduce temporal downsampling layers, depending on the computing capacity available and the temporal depth of the stacks. In our case, we introduced temporal downsampling layers in the network so it can be trained on the same hardware. With a total of 6 temporal downsampling operations (3 encoding layers and 3 temporal downsampling layers), the network can process spatio-temporal stacks consisting of 64 frames ($2^6 = 64$). The network is shown in Fig. 4.11.

4.2 Mask generation for an end to end training

Even if the second assumption we made in 4.1.4 is true and we have the spatial coordinates of all the neurons, we still have to define a proper target mask with the following properties.

- The mask should be dense and should be of the same spatial dimensions as the input stack.
- The target mask should include all neurons. It should not be formed by deleting the overlapping neurons.
- The probability of overlap between binary masks of two individual neurons should be as small as possible.
- We should not make any assumption about the number of neurons in the stack.

By keeping the above-mentioned assumptions in mind, we decided to experiment with different target masks explained in the following subsections.

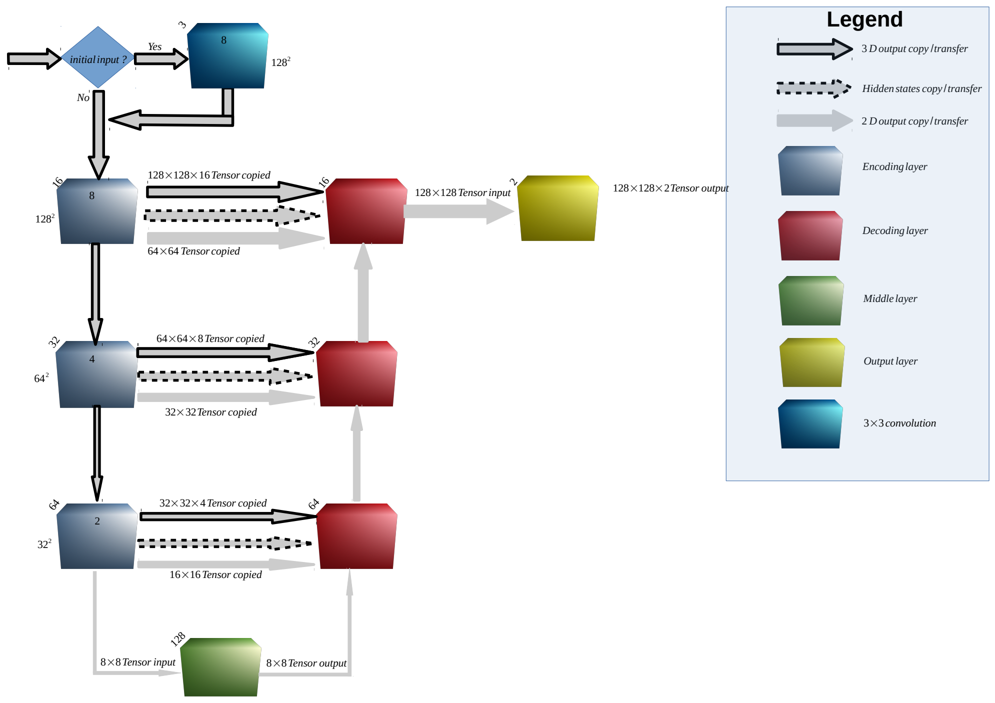

initial input ?
Yes
No
3
8
128^2
16
8
128^2
$128\times128\times16$ Tensor copied
64×64 Tensor copied
16
128×128 Tensor input
2
$128\times128\times2$ Tensor output
32
4
64^2
$64\times64\times8$ Tensor copied
32×32 Tensor copied
32
64
2
32^2
$32\times32\times4$ Tensor copied
16×16 Tensor copied
64
128
8×8 Tensor input
8×8 Tensor output
Legend
3 D output copy / transfer
Hidden states copy / transfer
2 D output copy / transfer
Encoding layer
Decoding layer
Middle layer
Output layer
3×3 convolution

FIGURE 4.9: The architecture of proposed U-Net used in end-to-end training. The U-Net has three encoding layers, three decoding layers, a middle (bottleneck) layer, an output layer (Explained in 4.1.3) and a logical layer used to determine if the input stack is from initial steps or intermediate steps. The signal flows from top to bottom in encoding layers while it flows bottom to top in decoding layers. For better visualization, we have named all the layers (encoding and decoding) from top to bottom although the signal flows in reverse order in decoding layers (from layer 3 to layer 1, not the other way around). This way, we can visualize that three tensors are copied from the first encoding layer to first decoding layer and so on. The bottleneck layer transforms one of the outputs of the 3rd encoding layer to be fed at one of the inputs of the 3rd decoding layer and the output layer transforms the signal into the desired shape. The logical layer determines if the input sequence is coming from the original stack or if it was created by intermediate steps of the pipeline. Since the original stack is grayscale, therefore, there the frames have 1 colour channel. The logical layer applies a 3×3 convolutional with three filters to transform the input into three colour channels.

4.2.1 4 layered target mask with overlapping neurons in different layers

The principle behind this target mask is to separate the overlapping neurons by placing them in different layers under the assumption that no more than three neurons can overlap in a stack. This assumption is made to limit the number of layers in the target mask to four, three for the neurons and one for the background. The mask is created in the following steps.

1. Create a matrix with the same spatial dimensions as the input stack and four layers.

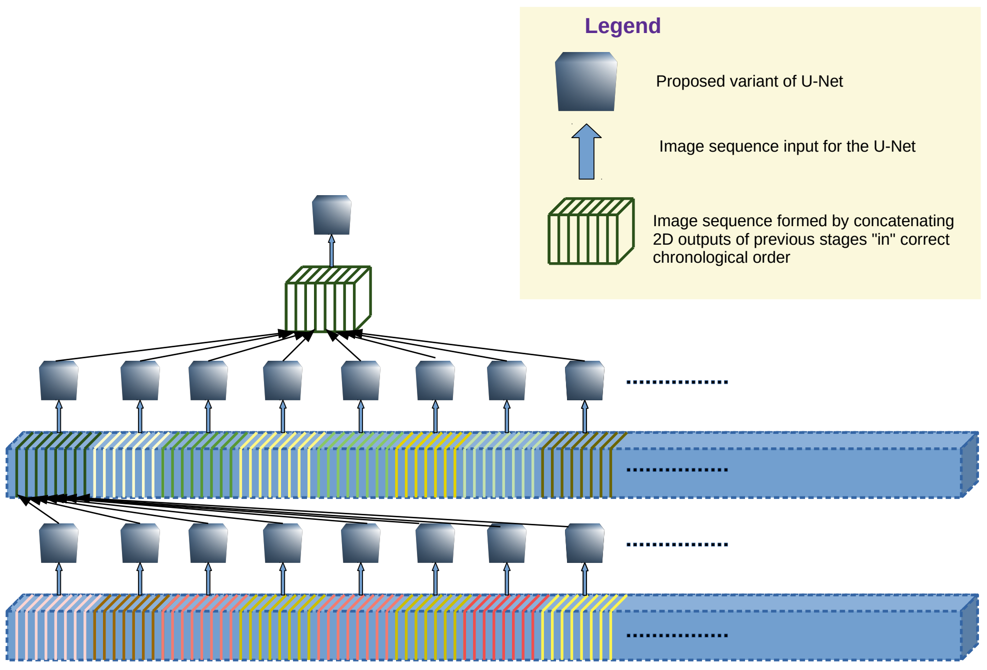

Legend
Proposed variant of U-Net
Image sequence input for the U-Net
Image sequence formed by concatenating 2D outputs of previous stages "in" correct chronological order

FIGURE 4.10: Graphical illustration of end to end training pipeline. The original stack is represented by the long tube in the bottom. The group of coloured lines (8 in a group) slicing the stack represent individual frames in the stack. One coloured group goes into the network as input and produces one output (frame) represented by a line slicing through the mini-stack. When this process is repeated 8 times, 8 create a coloured-lines group (another sequence) in the mini stack. The mini stack contains 8 such groups. These coloured-lines groups (signifying the sequences of step 3 in 4.1.4) are again fed into the network repeatedly. The outputs are then again concatenated in the final cube. The final cube signifies the sequence formed by the outputs of step 4 in 4.1.4. Finally, the network is applied once again to the cube to arrive at the final output mask.

2. Select a neuron randomly. Place its binary mask in the first layer.

3. Select another neuron randomly. Check if it has any overlap with the first layer of the mask. If there is no overlap, place its binary mask in the first layer. If there is an overlap, check if it has an overlap with the second layer of the mask as well. If there is no overlap with the second layer, place its binary mask in the second layer of the target mask. If there is an overlap with the second layer as well, check if it has an overlap with the third layer. If it has no overlap in the third layer, place its binary masks in the third layer, otherwise, go to step 4.

4. By now, we know that the neuron in question has an overlap with all the three layers. Find the intersection over the union of current neuron with all the layers. Then place it in the layer with the smallest value of intersection over the union.

5. Repeat step 2 to 4 for all the remaining neurons.

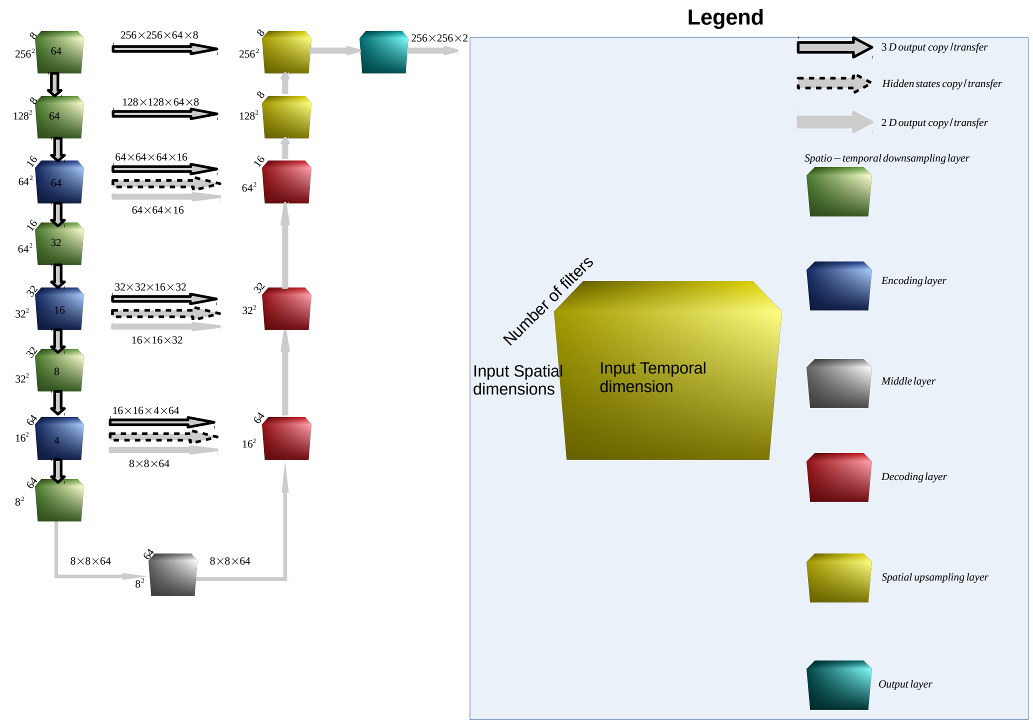


FIGURE 4.11: The architecture of proposed single-stage U-Net used in end-to-end training. The U-Net has two spatial downsampling layers, three encoding layers, three decoding layers, three temporal downsampling layers (embedded in the encoding arm after each encoding block), two spatial upsampling layers, a middle (bottleneck) layer and an output layer (Explained in 4.1.3). The first two layers on the encoding arm are spatial downsampling layers, then three pairs of encoding and temporal sampling layers. The decoding arm has one decoding layer at the same level as the corresponding encoding layer in the encoding arm and two spatial upsampling layers at the same level as the spatial downsampling layers of the encoding arm. Encoding layers included, there are a total of 6 temporal downsampling steps (3 from temporal downsampling layers and 3 from encoding layer) thus downsampling temporal dimension from 64 to 1. The three encoding layers downsample the spatial dimensions from 256×256 to 8×8 in the encoding arm. Therefore, this network accepts mini-stacks of size $256 \times 256 \times 64$ and returns an output of $256 \times 256 \times number_of_required_labels$

6. Check the number of neurons in each layer after step 5. Move neurons between layers in such a way that there are more or less equal numbers of neurons in each layer and the overlap is as small as possible.

7. Create a background layer by taking a complement of the first three layers.

A sub-window from the 4 layered target mask is shown in Fig. 4.12

4.2.2 3 layered target mask with overlapping regions in one layer

The principle behind this target mask is to identify those regions in which neurons are overlapped and put them in a separate layer. This mask has a layer for neurons, a layer for the overlapping regions and a background layer. It is created in the following steps.

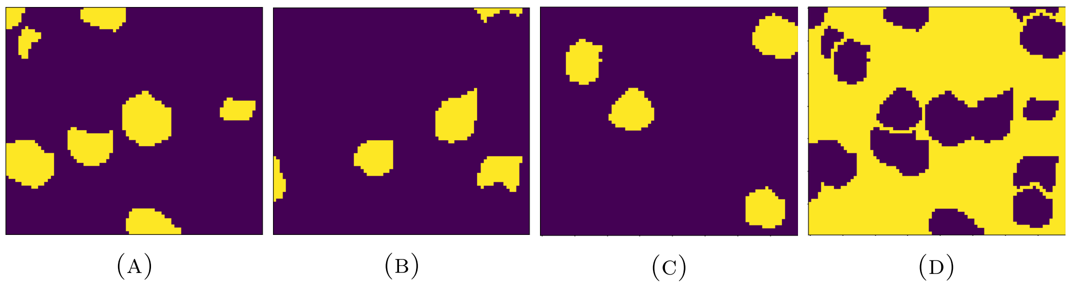


Figure 4.12: Four layered mask: (A), (B) and (C) show masks of randomly placed neurons in first three layers minimizing the chance of overlap, (D) shows the mask for background class created by taking compliment of first three layers

1. Create a matrix with the same spatial dimensions as the input stack and three layers.
2. Select a neuron randomly. Place its binary mask in the first layer.
3. Select another neuron randomly. Check if it has any overlap with the first layer of the mask. If there is an overlap, find the spatial coordinates of the overlap and place the overlap in the second layer.
4. Repeat step 2 and 3 for all the remaining neurons.
5. Create a background layer by taking a compliment of the first two layers.

A sub-window from the 3 layered target mask is shown in the Fig. 4.13

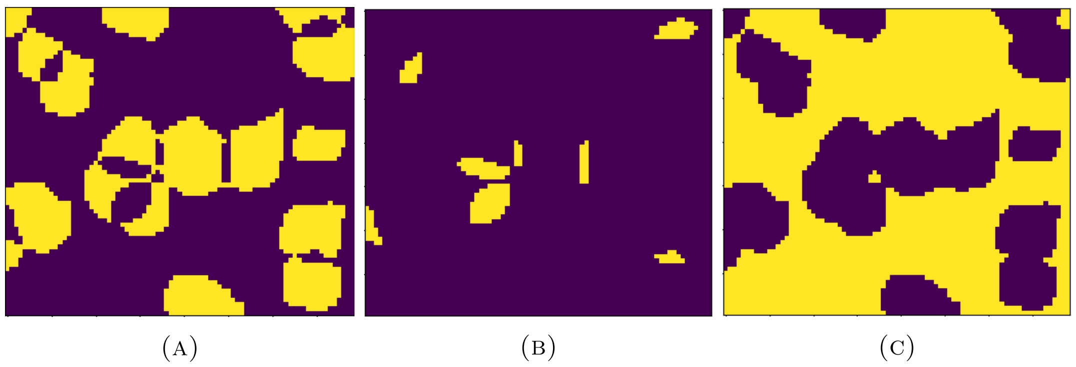


Figure 4.13: Three layered mask: (A) shows the mask of all neurons excluding the overlapping regions (B) shows the mask for all the overlapping regions (overlap class) and (C) shows the mask for background class created by taking compliment of first two layers

4.2.3 Gaussian masks

The target mask introduced in 4.2.1 suffers from the effects of randomness. Since the neurons are randomly placed in the first three layers, it is not guaranteed that a neuron will always go to a specific layer each time the mask is created for a specific stack although the stack and physical location of the neurons remain constant. This, in turn, can confuse the network about which layer it should put the detected neurons in. The target mask introduced in 4.2.2 suffers from the effects of class imbalance. Usually, the overlapping regions are much smaller in size and numbers than the actual neurons. The disparity is even bigger when compared to the background class. This, in turn, can skew the network learning against the overlapping class. Moreover, since the overlapping regions for different neurons have very different expression profiles, the network might struggle with consistent learning of these regions.

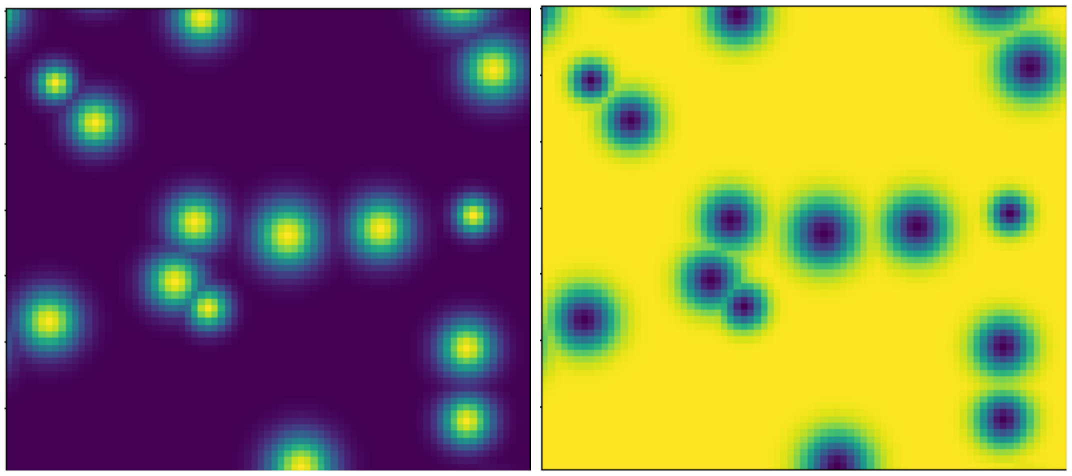
To mitigate these problems, we have introduced target masks based on a Gaussian kernel placed at the centre of each neuron. We have proposed two masks based on Gaussian kernels.

Circular Gaussian kernels

To create this mask, we place a circular Gaussian kernel at the centre of each neuron in such a way that its value is 1 at the centre and decrease in an exponential manner as we travel radially away from the centre. The rate of decay is controlled by the variance of distances computed over a circle with a radius big enough to inscribe the neuron. This way, we get a decaying exponential function with a maximum value at the centre of the neuron and minimum at the boundaries. If two neurons overlap each other, the maximum of two masks is placed in the overlapping regions. Finally, we create a second layer by subtracting the first layer from 1. Let's assume $x_d(i)$ represents the distance of a pixel i to the closest edge in the mask and x_d represents a set of all such distances, the Gaussian counterpart $x_{dg}(i)$ can be calculated by Eq. 4.2.

$$\begin{aligned} x_{dg}(i) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_d}{\sigma}\right)^2\right) \\ x_{dg}(i) &= \frac{x_{dg}(i)}{max(x_d(i))} \\ x_{dg}(i) &= 1 - x_{dg}(i) \end{aligned} \tag{4.2}$$

A sub-window from the circular Gaussian kernels target mask is shown in the Fig. 4.14



(A) (B)

FIGURE 4.14: Probability mask produced by placing circular Gaussian kernels at the center of each neuron: (A) shows the Gaussian mask of all neurons and (B) shows the mask for background class created by subtracting first layer from 1

Gaussian kernels based on distance transform

The Gaussian kernels proposed in the earlier section might not necessarily capture the complete structural details of the neurons and might struggle to accommodate for the structural difference among neurons. We have proposed Gaussian Kernels based on distance transform to reduce the effect of this phenomenon. This mask is created in the following steps.

1. Create a matrix with the same spatial dimensions as the input stack and two layers.

2. Select a neuron randomly. Create it's binary mask and then apply the distance transform. The distance transform will replace all the pixels in the mask with the distances to the closest boundary.

3. Apply a Gaussian kernel to the distance transform with the variance of distances controlling the decay of Gaussian as it moves from the center to the edges. Let's assume $x_d(i)$ represents the distance of a pixel i to the closest edge in the mask and x_d represents a set of all such distances, the Gaussian counterpart $x_{dg}(i)$ can be calculated by eq. 4.3.

$$\begin{aligned} x_{dg}(i) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_d(i)}{\sigma}\right)^2\right) \\ x_{dg}(i) &= \frac{x_{dg}(i)}{max(x_d)} \end{aligned} \tag{4.3}$$

4. Repeat step 2 and 3 for all the remaining neurons.

5. Create a background layer by subtracting the first layer from 1.

A sub-window from the distance transform based Gaussian kernels target mask is shown in the Fig. 4.15.

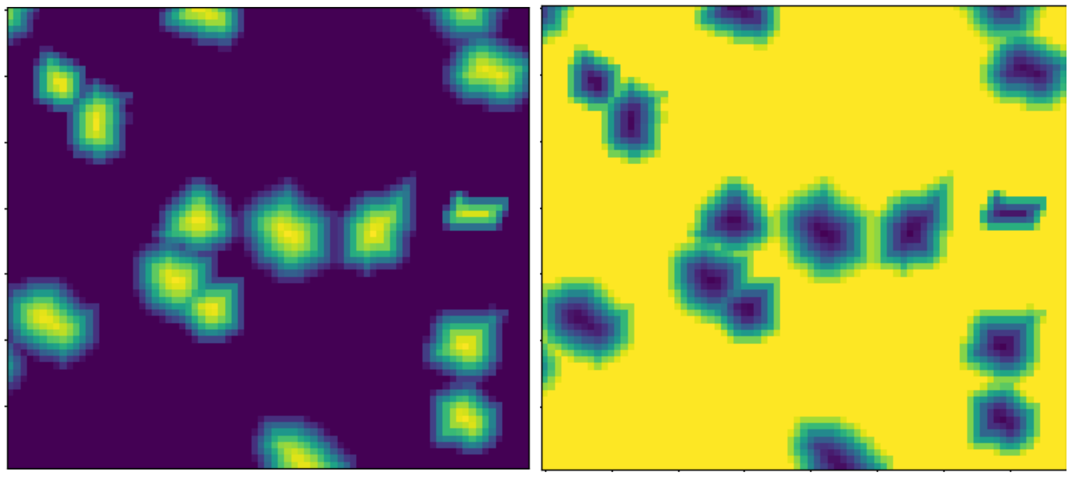

(A) (B)

FIGURE 4.15: Probability mask produced by placing Gaussian kernels produced by distance transform the location of each neuron: (A) shows the Gaussian mask of all neurons and (B) shows the mask for background class created by subtracting first layer from 1.

To create this mask, first, we place a circular Gaussian kernel at the centre of each neuron in such a way that its value is 1 at the centre and decrease in an exponential manner as we travel radially away from the centre. The rate of decay is controlled by the variance of distances computed over a circle with a radius big enough to inscribe the neuron. This way, we get a decaying exponential function with a maximum value at the centre of the neuron and minimum at the boundaries. If two neurons overlap each other, the maximum of two masks is placed in the overlapping regions.

4.3 Dataset and training data generation

4.3.1 Dataset

We have tested our pipelines on the Neurofinder Community Benchmark. The Neurofinder community benchmark CodeNeuro (2016) is an initiative of the CodeNeuro collective of neuroscientists that encourages software tool development for neuroscience research (*neurofinder*; *codeneuro/neurofinder*; Spaen et al., 2019). The Neurofinder benchmark aims to provide a collection of datasets with ground truth labels for benchmarking the performance of cell detection algorithms. The benchmark consists of 28 motion-corrected calcium-imaging movies provided by four different laboratories. Datasets are annotated manually or based on anatomic markers. They differ in recording frequency, length of the movie, magnification, signal-to-noise ratio, and in the brain region that was recorded. The datasets are split into two groups: training datasets and test datasets. The 18 training datasets are provided together with reference annotations, whereas the reference annotations for the 9 test datasets are not disclosed. The test datasets and their undisclosed annotations are used by the Neurofinder benchmark to provide an unbiased evaluation of the performance of the algorithms. The characteristics of the test datasets are listed in 4.1.

TABLE 4.1: Characteristics of the test datasets of the Neurofinder benchmark and their corresponding training datasets

Name	Source	Resolution	Length (sec)	Frequency (Hz)	Brain region
00.00	Svoboda Lab	512×512	438	7.00	vS1
00.01	Svoboda Lab	512×512	458	7.00	vS1
01.00	Hausser Lab	512×512	300	7.50	v1
01.01	Hausser Lab	512×512	667	7.50	v1
02.00	Svoboda Lab	512×512	1000	8.00	vS1
02.01	Svoboda Lab	512×512	1000	8.00	vS1
03.00	Losonczy Lab	512×512	300	7.50	dHPC CA1
04.00	Harvey Lab	512×512	444	6.75	PPC
04.01	Harvey Lab	512×512	1000	3.00	PPC

The reference annotations for a specific training dataset are in the form of a JSON file which contains the information about the spatial location of every annotated neuron in the dataset. We use this information to create the training masks explained in 4.2. All datasets can be downloaded directly from the Neurofinder benchmark for cell identification (*neurofinder*).

4.3.2 Training data generation

For the sake of consistency, we will represent the whole video stack by $\mathbf{I}^{W \times H \times T} = f(x, y, t)$ as a function of spatial dimensions x, y and temporal dimension t. The superscript in $\mathbf{I}^{W \times H \times T}$ denotes the width, height and length of the spatio temporal stack. Following this notation, a frame is denoted by $\mathbf{I}(x, y, t_k)$ and a pixel at location (x_i, y_j) and time step k is denoted by $\mathbf{I}(x_i, y_i, t_k)$.

Preprocessing

We first downsample the calcium imaging stacks in the temporal domain by a moving average with an overlap between successive downsampling steps. After temporal downsampling, we apply homomorphic filtering to correct non-uniform background illumination (Oppenheim, Schafer, and Stockham, 1968; Pitas and Venetsanopoulos, 1990; Soltanian-Zadeh et al., 2019).

Training data generation for hierarchical training

The hierarchical pipeline processes the stacks in three stages; each stage accepts a mini-stack of 8 frames. The first stage processes 8 mini-stacks whose outputs are then concatenated to generate input for stage two and so on. Therefore, for all the three stages, the minimum number of frames in $512(8^3)$. Let's denote the input tensor to the network by $\mathbf{X}^{t_h \times w \times h \times temp}$ where t_h represents the temporal dimensions of input mini-stacks (t_h is 8 in this case), w, h represent the width and height of the mini-stacks and *temp* represents a temporary dimension which holds the mini-stacks for hierarchical stages. Because of memory consumption problems, we fixed w, h to be 64 each, therefore, $\mathbf{X}^{t_h \times w \times h \times temp} = \mathbf{X}^{8 \times 64 \times 64 \times 64}$ (Since the input stack is 512 frames long, which is divided in mini-stacks of 8 frames, the temp dimension has to be 64 to hold all of the mini-stacks). Similarly, the target masks will be $\mathbf{Y}^{w \times h \times layers} = \mathbf{Y}^{64 \times 64 \times layers}$ where *layers* represents the number of layers in the target mask which depends upon the choice of target masks explained in 4.2. Depending on the choice of dataset and the values of downsampling, the final number of frames in a downsampled stack cannot be lower than 512. We performed our training with the datasets contributed by Svoboda and Hausser labs because those datasets are acquired at the same frame rates. The details of downsampled datasets are shown in table 4.2.

TABLE 4.2: Details of downsampled datasets

Name	Original frames	Downsampling factor	Overlap	Downsampled frames
00.00	3066	7	2	613
00.01	3206	7	2	641
01.01	2250	7	3	561
04.00	2997	7	2	599

We generate training samples for an epoch by following steps.

1. Calculate the number of extra frames available T_{extra} in the downsampled stack by subtracting 512 from the number of frames in the downsampled stack.

2. Choose a random index between 1 and T_{extra} (say T_{rand}) and pick all the frames from T_{rand} to T_{rand}+512 and store them in a tensor ${X_{in}}^{512 \times 512 \times 512}$ ex expressed in eq. 4.4
$$X_{in} = f(X, Y, t), t \in T_{extra} + \{1, 2, ..., 512\} \tag{4.4}$$

3. Create a tensor $X^{8 \times 64 \times 64 \times 64}$

4. Generate the target mask.

5. Pick a neuron from the annotation. Place a window of 64×64 with the neuron at it's centre.

6. Apply this window to the first 8 frames of the stack and put the resulting mini-mask in the first *temp* dimension of X.

7. Repeat step 6 for all the frames in the stack, processing the next 8 frames in each iteration and placing the resultant mini-stacks in the next *temp* dimension of X.

8. Apply the window to the target mask as well. This way, the target mask for this mini-stack will have at least one neuron.

9. Repeat this process for all the neurons available in the target annotations.

This way, we can generate as many training samples as we want since, for each new training sample, we will pick a random index between 1 and T_{extra} resulting in a slightly different mini-stack. This can be made even more robust by random flipping or rotation of the mini-stacks and their target masks. At inference time, we simply split X_{in} in smaller mini-stacks with the same dimensions as X by sliding a window of the same size over it.

Training data generation for single stage training

The single-stage pipeline processes the whole stack in one step. Therefore, if there are no processing or memory constraints, the stacks can be fed directly into the pipeline. As explained in 4.1.6, the pipeline can process stacks of 64 frames. Let's denote the input tensor to network by $\mathbf{X}^{t_h \times w \times h}$ where t_h represents the temporal dimensions of input mini-stacks (t_h is 64 in this case), w, h represent the width and height of the mini-stacks. Because of memory consumption problems, we fixed w, h to be 256 each, therefore, $\mathbf{X}^{t_h \times w \times h} = \mathbf{X}^{64 \times 256 \times 256}$. Similarly, the target masks will be $\mathbf{Y}^{w \times h \times layers} = \mathbf{Y}^{256 \times 256 \times layers}$ where $layers$ represents the number of layers in the target mask which depends upon the choice of target masks explained in 4.2. We generate training samples for an epoch by following steps.

1. Calculate the number of extra frames available T_{extra} in the downsampled stack $\mathbf{I}$ by subtracting 64 from the number of frames in downsampled $\mathbf{I}$.

2. Choose a random index between 1 and T_{extra} (say T_{rand}) and pick all the frames from T_{rand} to $T_{rand} + 64$ and store them in a tensor ${X_{in}}^{64 \times 512 \times 512}$ according to eq. (4.4)

3. Create a tensor $X^{64 \times 256 \times 256}$

4. Generate the target mask.

5. Pick a neuron from the annotation. Place a window of 256×256 with the neuron at it's centre.

6. Apply this window to all the frames in X_{in} put the resulting mini-mask in X (with dimensions flipped, now the temporal dimension is represented by the first dimension instead of the last).

7. Apply the window to the target mask as well. This way, the target mask for this mini-stack will have at least one neuron.

8. Repeat this process for all the neurons available in the target annotations.

The process explained above will generate as many mini-stacks as the neurons in the target annotation. If the whole process is repeated, it will generate another batch of slightly different mini-stack. Random flipping and rotation can be performed on the resultant mini-stacks to make them as diverse as possible and reduce the chance of over-fitting.

Data generation for single stage testing and inference

We cannot use the data generation pipeline presented in the previous section at testing and inference time because this pipeline generates mini-stacks from an input image stack without respecting the spatial position of that mini-stack in the original stack. At testing and inference time, we need to keep the spatial cohesion of mini-stacks intact. For instance, the previous pipeline picks up a neuron randomly and creates a window around it without considering where this neuron is located. At inference time, we don't have the location of neurons, so we cannot centre the window around one. To get around this problem, we slide a window on the spatial dimension of the stack and introduce an overlap. For example, for stack with spatial dimensions of 512×512, mini-stack size of 256×256 and overlap of 50%, the first mini-stack will be created by sub-sampling the input stack from row 0 to 256 and column 0 to 256. For the next mini-stack, we slide horizontally by 128 to let a 50% overlap, so the next mini-stack will be created by sub-sampling the input stack from row 0 to 256 and column 128 to 384. We repeat this process until we run out of mini-stacks. For this specific case, we can slide 3 steps horizontally and 3 steps vertically, so a total of 9 mini-stacks with spatial dimensions of 256×256 can be created. These mini-stacks are given to the network as inputs in proper order which generates an output for each mini-stack. To stitch the mini-stacks back together, we assign a weight matrix to the output of every stack. The weight matrix of a mini-stack will be a matrix of ones if this specific mini-stack was created without an overlap. But if there is an overlap involved, weights of the overlapping regions will be an average of the number of overlap steps involved. The weight matrix for this setting is shown in Fig. 4.16.

4.4 Experiments

We have two training frameworks explained in 4.1.4 and four proposed target masks as explained in 4.2. Therefore, we can design 8 experiments under these conditions which are described below.

4.4.1 Post processing for Gaussian masks

The Gaussian masks are tailored to favour the nucleus region of the neurons, so the predicted output masks may not necessarily represent the true shape of the neurons. For Distance transform-based Gaussian masks, this problem is solved by morphological dilation by a small number of pixels. Since predicted output masks for circular Gaussian might be circular instead of representing the true shape of the neuron, we pass only the masked area to another copy of the same network which is trained with only one neuron present and everything other than neurons is masked. Along with some constraints (size of the neuron and location of the neuron), this step gives us

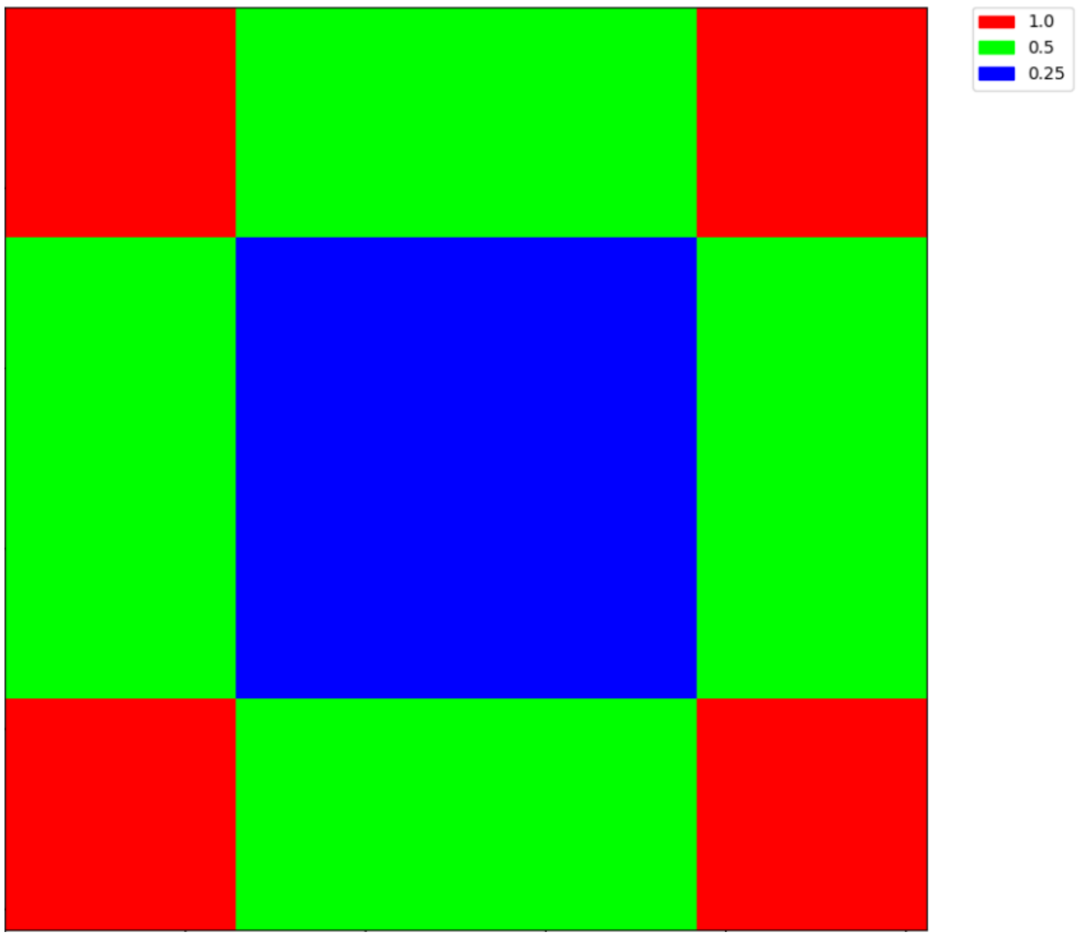


FIGURE 4.16: Weight matrix: Spatial dimensions of input stack = 512 × 512. Spatial dimensions of required mini-stacks = 256 × 256. Overlap = 128

boundaries for each of the detected neurons. It is to be noted that the window centred around the detected neuron is different from the window centred around neurons in training stages because in this case, only a specific area of the window is visible to the network instead of the whole window.

4.4.2 Hierarchical and single-stage training experiments

Let's denote the hierarchical training protocol by H (see Section 4.1.4, Fig. 4.9 and Fig. 4.10), single stage training by S (see Section 4.1.6 and Fig. 4.11), 4 layered mask by $M_{4layers}$, 3 layered mask by $M_{3layers}$, circular Gaussian mask by M_{gcirc} and distance Gaussian by M_{gdist} (see Section 4.2 for details). The experiments created by iterating through the two training pipelines and four masks are listed in table 4.3

TABLE 4.3: Different experiments resulting from combining the two training frameworks with four choices of target masks

$H + M_{4layers}$	$H + M_{3layers}$	$H + M_{gcirc}$	$H + M_{gdist}$
$S + M_{4layers}$	$S + M_{3layers}$	$S + M_{gcirc}$	$S + M_{gdist}$

For our best network, we use the following training parameters.
Learning rate: We used an exponentially decaying leaning rate starting from 0.001 with an exponential decay rate of 0.998 after every 400 epochs.
Optimizer: Adam (Kingma and Ba, 2014)
Number of training epochs: 3000
Loss function: Pixel-wise Kullback–Leibler divergence (Kullback and Leibler, 1951; Van Erven and Harremos, 2014)

4.4.3 Evaluation of the results

We will evaluate the results of the network, both qualitatively and quantitatively. For qualitative evaluation, we will compare the output predicted masks with the original target masks by placing them side by side. Moreover, we will also impose the boundaries of the predicted and real masks on summary images of the input stacks to see how much they agree. Finally, we will compare the visuals of some selected individual neurons to see the prediction performance in finer details. For quantitative performance evaluation, we have used the comparison app provided by the Neurofinder challenge (*codeneuro/neurofinder*) organizers. We have used this app to validate our results locally for those datasets which are annotated. We will validate our results against the published benchmark for those datasets whose annotations are not released by the organizers. This app compares predicted neuron masks to ground truth neuron masks and calculates the following metrics.

Precision

Precision is the percentage of all the positively detected neurons which are neurons to all positively detected neurons. This is a measure of how accurate the positive detection is in discarding false positives. Precision can be calculated by Eq. 4.5

$$Precision = \frac{TP}{TP + FP} \tag{4.5}$$

where TP and FP correspond to true positives and false positives.

Recall

The recall is the percentage of all the positively detected neurons which are neurons to all neurons in the ground truth. This is a measure of how accurate the positive detection is in avoiding false negatives. Recall can be calculated by Eq. 4.6

$$Recall = \frac{TP}{TP + FN} \tag{4.6}$$

where TP and FN correspond to true positives and false negatives.

F1-score

F1-score is the harmonic mean of precision and recall and can be described by Eq. 4.7

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.7}$$

4.4.4 Results

Network training

We got the best results when we used single stage training coupled with Gaussian masks (see Section 4.2). We trained the network illustrated in Fig. 4.11 for 3000 epochs and saved the training status after every 60 epochs. We used the dataset **02.00** from table 4.1 as test data and the remaining three datasets (**01.00, 01.01, 04.00**) as training data. The details of the downsampling are listed in table 4.4.

TABLE 4.4: Details of downsampled datasets

Name	Original frames	Downsampling factor	Overlap	Downsampled frames
01.00	2250	30	5	89
01.01	5502	30	0	165
03.00	2250	30	5	89
04.00	2997	30	5	119

The network appears to reduce the loss rapidly in the beginning and then starts to plateau. We observed that although the loss decreases very slowly in later stages of the training, yet there is a significant change in the visual quality of the predicted masks as we train the network for more epochs. The epoch loss is illustrated in Fig. 4.17

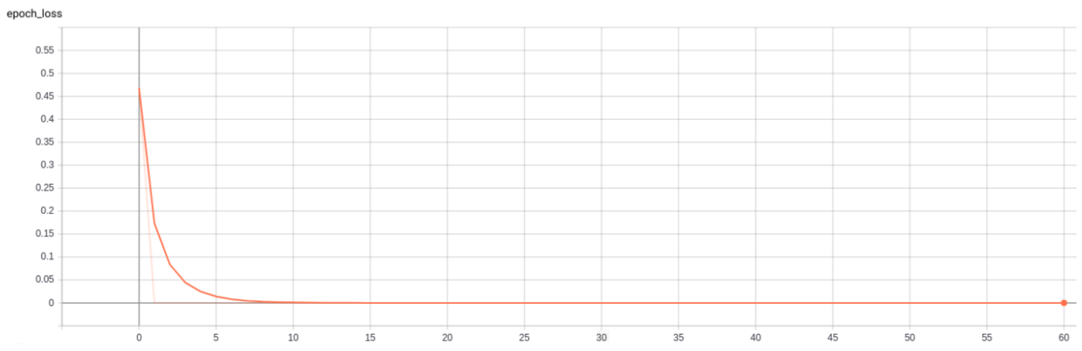


FIGURE 4.17: Epoch loss visualized after saving the epoch loss value after every 50 epochs. One step on the x-axis indicate that the network has been trained for 50 epochs

Qualitative results

In this section, we will compare the visual outputs of the trained network. It is to be noted that when the Gaussian masks are converted to binary masks, the size of a neuron might be smaller in the converted binary mask than the original neuron mask. This is because when we are creating a Gaussian mask, we are making the edges of the neuron weaker than its centre by assigning it a smaller probability which is inversely proportional to the distance of the edges from the centre as illustrated in Fig. 4.18.

To compare the ground truth and predicted binary masks visually, we have placed them side by side in Fig. 4.19. Fig. 4.20 shows the same two Gaussian masks

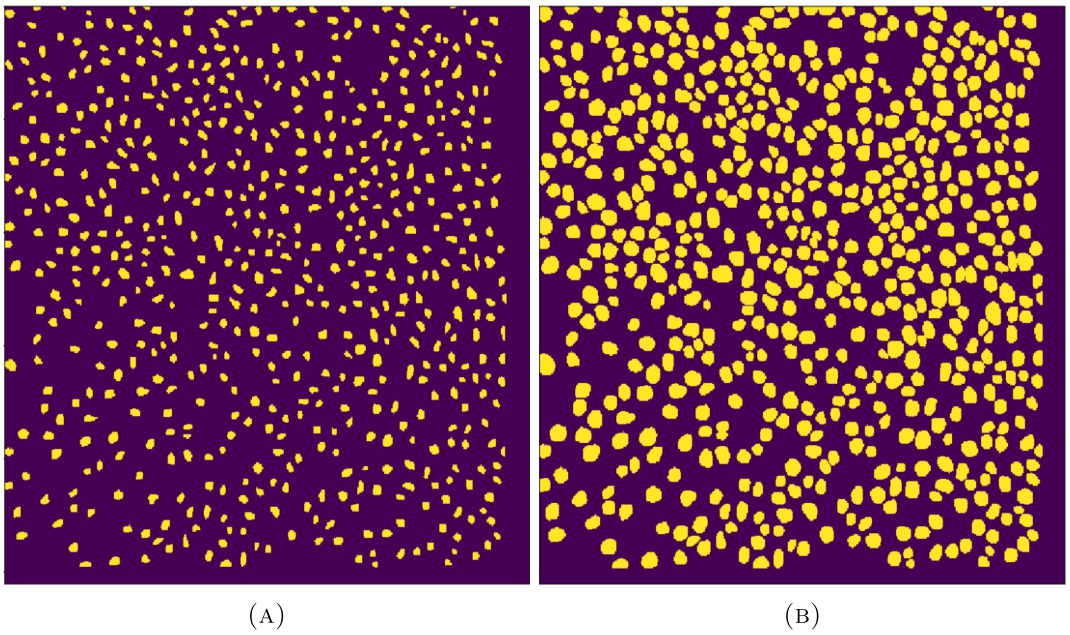


FIGURE 4.18: Ground truth masks for test data: (A) shows the ground truth mask converted from the Gaussian distance mask (B) shows the actual ground truth mask. The difference in scale is due to the decaying nature of Gaussian kernels used to represent the neurons.

converted to binary masks and Fig. 4.21 shows the binary masks zoomed on specific regions for better comparison.

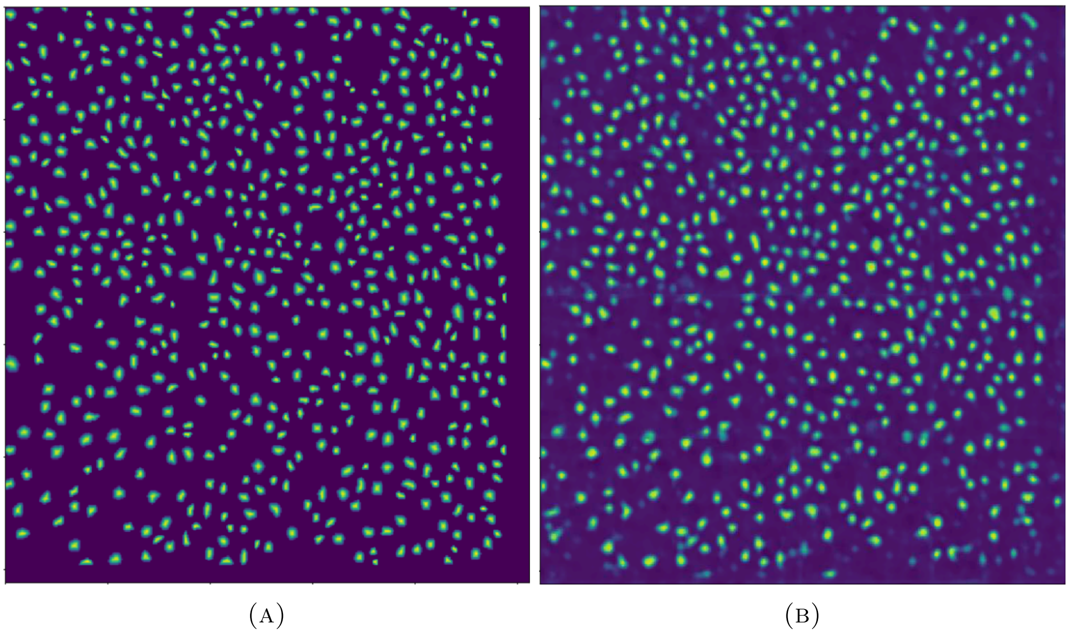


FIGURE 4.19: Ground truth Gaussian mask for test data compared to predicted Gaussian mask for test data: (A) Ground truth Gaussian distance mask (B) Predicted Gaussian distance mask

In order to visualize the comparative boundaries of the neurons, we overlaid the boundaries of neurons in the ground truth and predicted masks and visualized them with different colors. We also indicated the neurons which are present in the ground

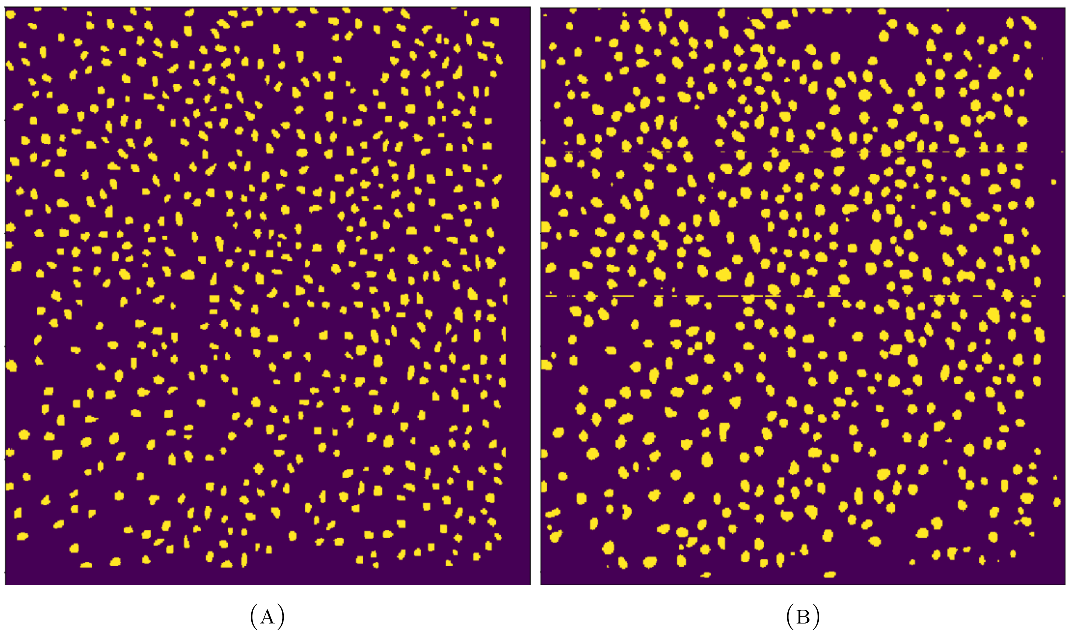


FIGURE 4.20: Binary masks extracted from Fig. 4.19: (A) Ground truth (B) predicted

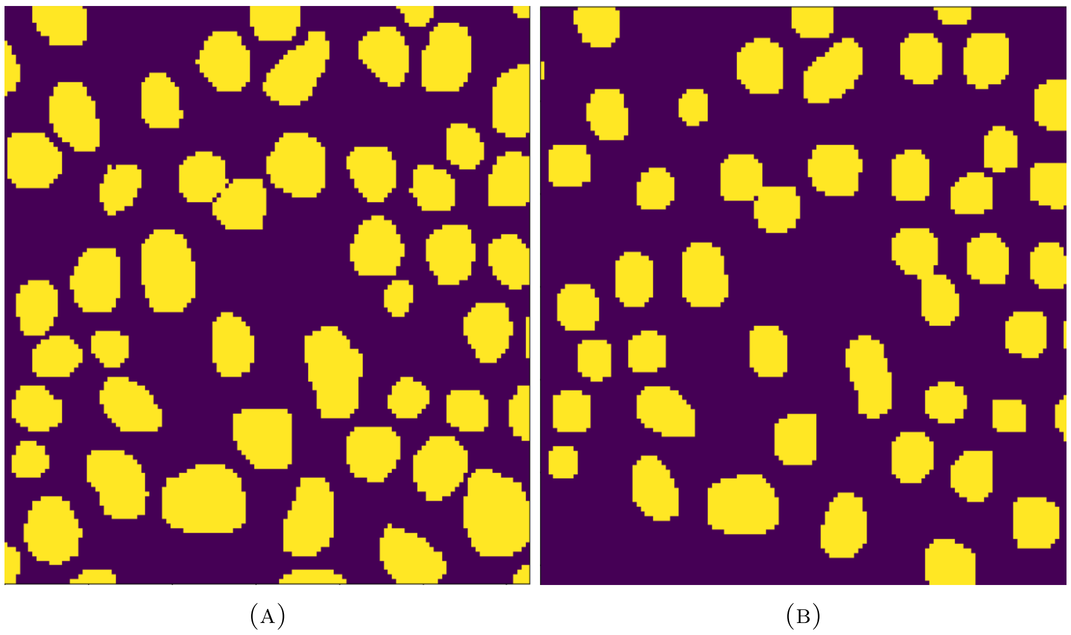


FIGURE 4.21: Zoomed masks from Fig. 4.20: (A) Ground truth (B) Predicted

truth but missing in the predicted masks (false negatives). Fig. 4.22 shows the boundaries overlaid on the neurons shown in Fig. 4.21.

In order to visualize the temporal activity of detected neurons, we averaged the neurons in the spatial domain while keeping the temporal domain intact. This resulted in an activity trace for the neuron over time. We observed that for most of the detected neurons, the activity traces were very similar to the activity traces of ground truth neurons. This is illustrated in Fig. 4.24.

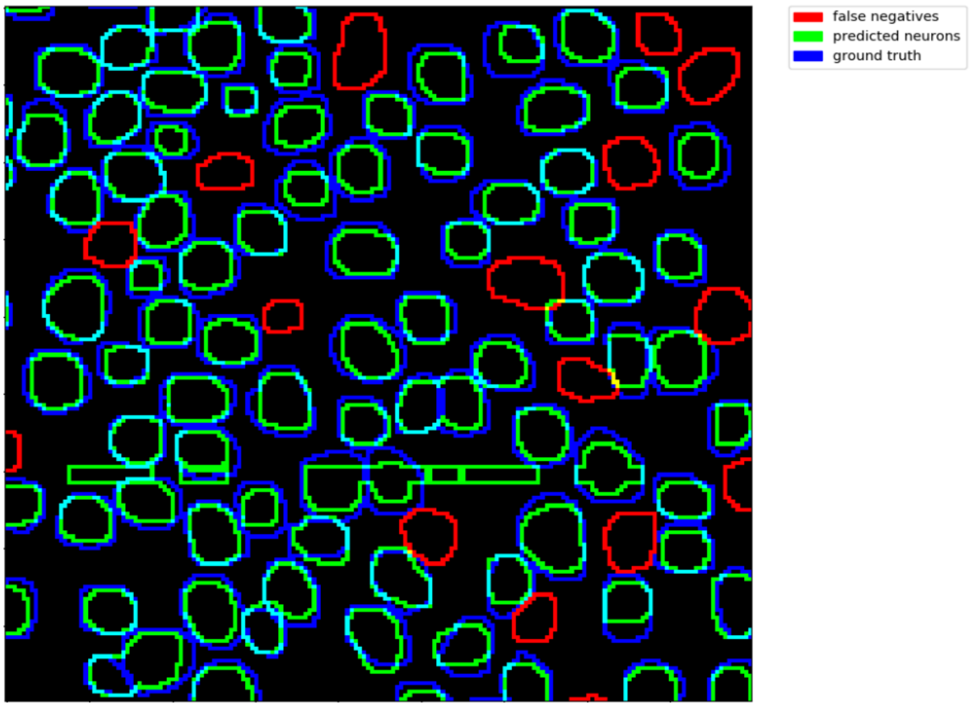


FIGURE 4.22: Failure case: False negatives. Boundaries of neurons from Fig. 4.21: Green represents the neurons detected by the network, blue represents the neuron from ground truth and red represents the neurons missed by the network.

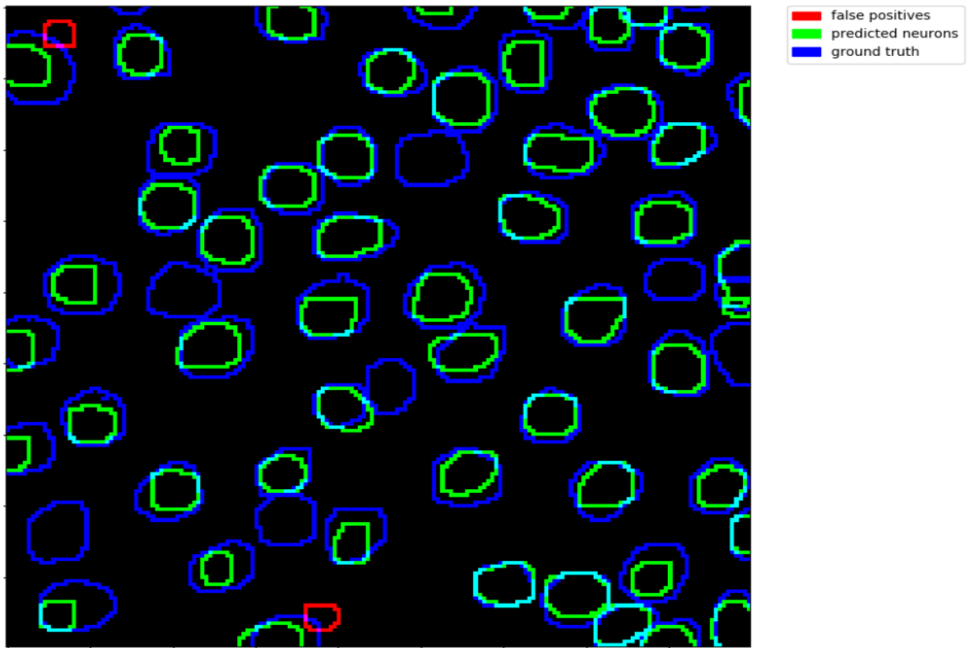
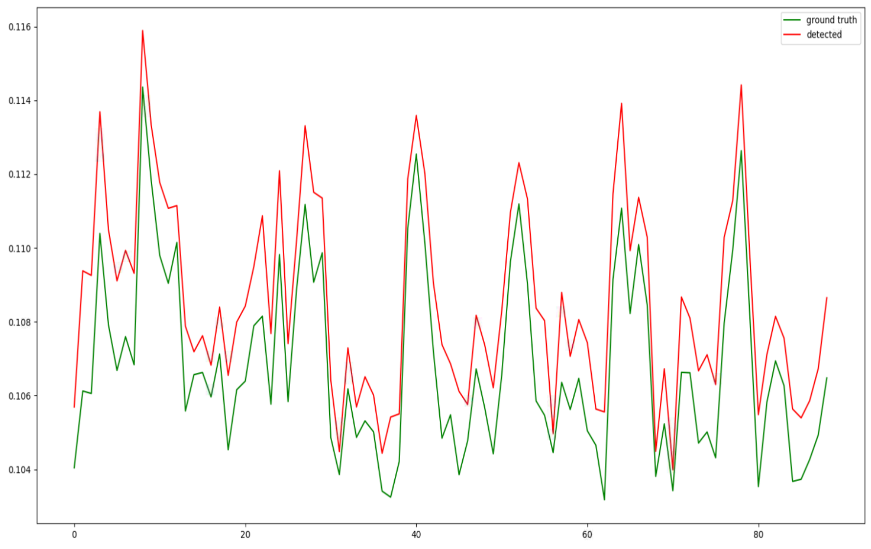
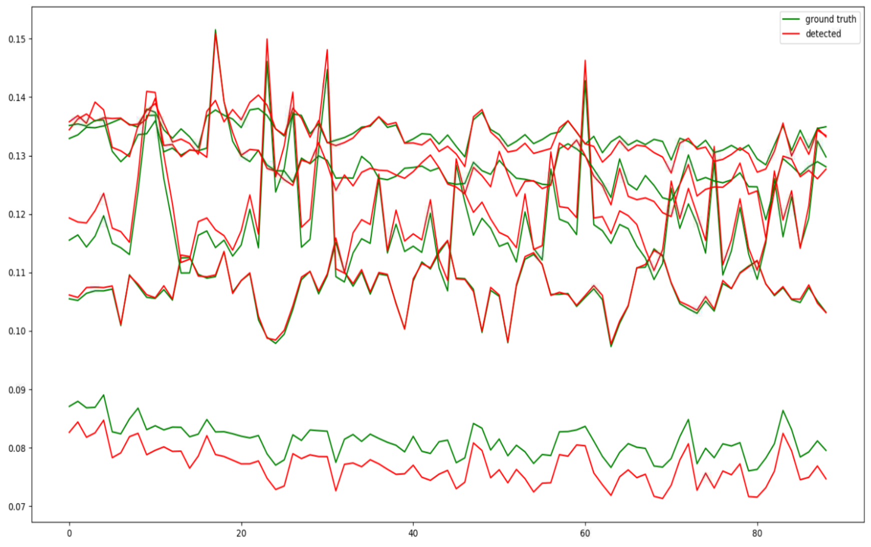


FIGURE 4.23: Failure case: False positives. Boundaries of neurons from Fig. 4.21: Green represents the neurons detected by the network, blue represents the neuron from ground truth and red represents the neurons predicted by the network but missing from ground truth (false positives). Note that there are multiple ground truth boundaries without any predicted boundaries, they are false negatives but omitted from this illustration since they are illustrated in Fig. 4.22



(A)



(B)

FIGURE 4.24: Neural activity traces of neurons: (A) Neural activity trace of a single neuron (B) Neural activity traces of multiple neurons visualized simultaneously

Quantitative results

For the datasets described above, we ran the pipeline multiple times with slightly different settings to get a good picture of the network performance. For each setting, we trained the network slightly more and then tested it. Since the network loss enters a plateau after a while (see Fig. 4.17), we assume that the network achieves a baseline performance after a certain number of epochs. Treating that network as a baseline, we train it further and then evaluate its performance. For validation, the network is trained on three stacks and tested on one which is not included in the training data. Table 4.5 shows the settings and results for the trained network proposed in 4.1.6.

Table 4.6 illustrate the statistics of the network performance.

To benchmark the results, we compared the performance of our pipeline to two other approaches which are benchmarked against the Neurofinder challenge. To keep the comparison fair, we tested both of those approaches on the same test data we

TABLE 4.5: Quantitative results

number of training epochs	precision	recall	F1
19,000	0.73	0.68	0.70
20,000	0.80	0.70	0.74
21,000	0.82	0.75	0.78
22,000	0.92	0.82	0.87
23,000	0.90	0.85	0.87
24,000	0.98	0.81	0.88
25,000	0.99	0.89	0.93

TABLE 4.6: Statistics of quantitative results

.	precision	recall	F1
Minimum	0.73	0.68	0.70
Maximum	0.99	0.89	0.93
Mean	0.88	0.78	0.82
Std	0.09	0.07	0.08

used in our analysis and trained one of them on the same data as we used to train our pipeline. One of the approaches is called 'HNCCorr' (Spaen et al., 2019). This is an unsupervised approach which segments neurons pixel by pixel by searching and grouping all those neurons which show similar temporal behaviour. This is achieved by temporal correlation of pixels with their neighbourhoods. The other approach is called 'UNet2DS' and it employs a standard UNet applied on 2D summary images of the neuronal stacks (Klibisz et al., 2017) (see Section 2.2.2 for details on summary images). Table 4.7 shows the comparison of our approach to the mentioned approaches. We chose these approaches for comparison because they are among the top-performing in state of the art and their results are comparatively easier to reproduce and there is open-source code available for them. Moreover, we chose HNCCorr as a representative of the unsupervised approaches and UNet2DS for supervised approaches. We chose UNet2DS because it is built around the same architecture as STUNet.

TABLE 4.7: Comparison of our approach to the state of the art

.	precision	recall	F1
HNCCorr	0.84	0.42	0.56
UNet2DS	0.85	0.72	0.78
STUNet (ours)	**0.88**	**0.78**	**0.82**

From the qualitative and quantitative results, we can draw the following conclusions.

- The proposed approach performs better than the state of the art unsupervised approaches.

- Proposed approach also performs better than supervised state of the art approaches in a test setting, although this need to be further validated on other open source datasets.
- The main strength of the proposed approach is its tandem operation of 2D and 3D information streams, which make it an appropriate candidate for spatiotempral segmentation.

Chapter 5

Discussion

In this research, we have proposed deep-learning-based solutions for behaviour annotation and neural activity annotation which can help behavioural and computational neuroscientists establish a causal link between behaviour and neural activity. Since we were dealing with two very different aspects of essentially one problem, the causal relationship between behaviours and neural activity, we had to establish two different pipelines to deal with those separately. The first aspect of this problem is how to quantify behaviours and the second aspect is how to detect and quantify neural activity. We decided to treat them as separate but similar problems. For behavioural annotation, we selected to work with locomotion data of head-fixed mice running on a spherical treadmill which were being recorded from frontal and lateral angles (please refer to section 2.1.1 for details about experiment settings). The dataset consists of RGB videos in different lighting conditions with a mouse in the field of view. Part of this dataset was manually annotated to generate training data. The annotation was done by tracing limbs boundaries in each frame and the proposed pipeline was trained to detect the limbs in each frame based on the previous segments of the videos. Since the detection was done by analysing previous segments of the video in a spatio-temporal manner, this problem can also be treated as spatio-temporal localization. For neural activity detection, we decided to test the proposed pipeline on an open benchmark, the Neurofinder challenge (see section 4.3.2 and refer to *codeneuro/neurofinder* for more details). This dataset consists of spatio-temporal stacks of calcium imaging data contributed by different laboratories. The training dataset contains information about all of the neurons visible in the field of view. The neuron's spatial annotation is done automatically for some of the videos using anatomical markers while manually for others. The core challenge is to infer the spatial position of each neuron and separate them from each other, therefore, it can also be treated as a spatio-temporal localization.

5.1 Gesture Tracking

Based on the spatio-temporal nature of the data at hand, we have proposed a deep learning-based spatio-temporal solution to annotate the behavioural data. In a typical behavioural neuroscience data set, researchers aim to identify limbs in every frame. We started from the widely popular approaches of object detection: the Haar cascades. Although Haar cascades gave us good results on one dataset (lateral video), it failed to produce promising results on frontal videos. So we moved towards a deep-learning-based solution. Conventional techniques rely on a frame by frame image segmentation or object detection. Our approach is based on the notion that limbs are regions of the frames which feature specific and learnable spatio-temporal characteristics.

We defined the notion of motion tubes and motion sequences, that use compact representations (superpixels) to simultaneously extract appearance and temporal features on videos. In other words, we treat the limbs as a collection of superpixels which moves coherently. Under this assumption, we track the superpixels in time and build a 3D tube representing the journey of these superpixels. We used features learned by training a CNN to classify if a constructed temporal tube belongs to a limb or not. We extract a feature vector for each temporal tube and then stack feature vectors on the temporal axis to get a temporal representation of the features extracted from successive temporal tubes. The resulting feature sequence is essentially a temporal sequence (we refer to it as a motion sequence) and we train an LSTM to classify this sequence into the limb and non-limb regions. Since we create the temporal tubes and subsequently the motion sequence by tracking the journey of superpixels in time, when we classify a motion sequence, we can essentially segment part of the limbs.

We obtained promising results on two different acquisition conditions (lateral and frontal videos) and under different noise patterns. We have developed this approach under the assumption that the animal, in this case, the mouse, is head-fixed. Therefore, for our approach to work in freely moving animals, additional steps need to be included. Moreover, we have also compared our work to the state of the art tracking and pose estimation approaches and concluded that our pipeline produces results at par with state of the art.

We have published three papers which describe parts of our gesture tracking work. The first paper summarizes our work on Haar cascade (Giovannucci et al., 2018). The second paper is a comprehensive literature review of gesture tracking/pose estimation methods for laboratory animals/mice (Abbas and Masip, 2019). The third paper covers our work on the deep-learning-based pipeline for limbs segmentation/tracking (Abbas, Masip, and Giovannucci, 2020).

5.2 Neural activity detection/segmentation

As mentioned earlier, although gesture/activity tracking by limbs segmentation and neural activity detection/segmentation are primarily different tasks, yet both share similarities since both of them are done by analyzing spatio-temporal data. Therefore some of the concepts from gesture tracking can also be imported into neural activity detection. For instance, we can/did exploit the sequence learning ability of LSTMs to learn temporal sequences. In the case of gesture tracking, we used LSTMs to annotate motion sequences if they were created from limbs or non-limbs superpixels while in the case of neural activity detection, we used a special kind of LSTMs to learn binary masks of neurons from spatio-temporal calcium imaging data. The LSTMs we employed are referred to as convolutional LSTMs and unlike conventional LSTMs, ConvLSTMs can learn the representation of image sequences (videos). Because of the increased interest in neural activity detection/segmentation due to the Neurofinder challenge, there are other promising approaches as well which have been proposed by other researchers (see section 2.2). Some of the recent approaches are based on deep models, especially the UNet and DenseNet, however, these approaches are lacking in two key areas; one, they need temporal masks for neurons so they are not end-to-end and two, they do not expressly treat the problem as a spatio-temporal problem. We have addressed these two problems by building on top of the classical UNet architecture by replacing its blocks by custom blocks. The custom blocks are based on ConvLSTMs and they process the stacks in two streams, a 2D stream which learns a spatial representation of the data and a 3D stream which learns the 3D spatio-temporal structure of the data. The 2D stream is built by employing standard convolutional, deconvolutional and sampling blocks while the 3D stream relies on ConvLSTMs. We address the problem of end to end training by proposing a novel target mask pipeline which creates a Gaussian instead of a binary ROI for each neuron. The Gaussian of each neuron is strong at the centre and becomes weak at the edges, so even if we have overlapping binary masks for two neurons, they can be easily separated by their Gaussian masks. This allows us to train the proposed network in an end-to-end manner and eliminate the need to create temporal masks of the neurons. Therefore, all this network needs is an input stack of calcium imaging data and the locations of neurons it is supposed to learn.

Preliminary results of this network are very promising (see section 4.4.4). The network generates good neuron masks which are very similar to the ground truth masks in most of the cases. We have some instances of false negatives while very few instances of false positives, therefore we can conclude that the network does not annotate wrong regions as neurons, although it might miss annotating the neurons correctly sometimes. Moreover, the network can also separate neurons which appear overlapped in the 3rd dimension. This network still needs to be refined and validated on more datasets but we can safely conclude that the proposed network has a proven ability to separate neurons from the background. This part of our research is under review for publication.

5.3 Future work

The idea which was driving this research was to pave the way for finding a causal relationship between physical activity (gestures/limbs movements) and neural activity. This was out of scope for the research goals we set for this thesis but we have suggestions to expand this work into that direction. We believe the following research directions can be interesting to explore in future work.

- Refine the gesture tracking pipeline by pursing the following directions.
  - Replace the two-stage pipeline (the CNN and the LSTM) by a single network, preferably a ConvLSTM.
  - Create better training data set by using more data and better masks for limbs.
  - Explore domain adaptation for better tracking i.e. a well known tracking dataset (Ruffieux et al., 2014) can be used as the source domain and gesture tracking for mice dataset can be treated as the target domain.
- Refine the neural activity segmentation pipeline by pursing the following directions.
  - Refine the pipeline for more optimized computation.
  - Validate the pipeline on more datasets.
  - Explore more robust target masks.
  - Explore variable temporal domain network. For instance, the network in its current form can only process calcium imaging stack with a fixed number of frames (temporal steps). This way, the pipeline can be used to process arbitrarily long stacks without worrying about losing neurons.
- Validate the neural activity segmentation pipeline against published benchmarks.

- Create a biological neuronal network of the neurons segmented by the pipeline.
- Establish a causal relationship between the biological neuronal network and gesture tracking.
- Create a pipeline which can learn to do both the gesture tracking and neural activity detection/segmentation simultaneously.
- If synchronized behavioural and neuronal data becomes available, explore the joint learning of both tasks in a self-supervised manner, at large scale, with no need for human annotations.

Chapter 6

Contributions

This chapter provides a summary of our research contributions.

6.1 1st paper

The first paper summarizes our work on Haar cascades (Giovannucci et al., 2018).

6.1.1 title

Andrea Giovannucci, EA Pnevmatikakis, B Deverett, T Pereira, J Fondriest, MJ Brady, SS-H Wang, W Abbas, P Pares, and David Masip. "Automated gesture tracking in head-fixed mice" Journal of neuroscience methods, Vol. 300, 15 April 2018, Pp 184-195. ISI JCR IMPACT FACTOR: 2.554 (2016). **2nd quartile**.. Biochemical Research Methods.

6.1.2 Abstract

Background

The preparation consisting of a head-fixed mouse on a spherical or cylindrical treadmill offers unique advantages in a variety of experimental contexts. Head fixation provides the mechanical stability necessary for optical and electrophysiological recordings and stimulation. Additionally, it can be combined with virtual environments such as T-mazes, enabling these types of recording during diverse behaviors.

New method

In this paper we present a low-cost, easy-to-build acquisition system, along with scalable computational methods to quantitatively measure behavior (locomotion and paws, whiskers, and tail motion patterns) in head-fixed mice locomoting on cylindrical or spherical treadmills.Existing methods: Several custom supervised and unsupervised methods have been developed for measuring behavior in mice. However, to date there is no low-cost, turn-key, general-purpose, and scalable system for acquiring and quantifying behavior in mice.

Results

We benchmark our algorithms against ground truth data generated either by manual labeling or by simpler methods of feature extraction. We demonstrate that our algorithms achieve good performance,both in supervised and unsupervised settings. Conclusions: We present a low-cost suite of tools for behavioral quantification, which serve as valuable complements to recording and stimulation technologies being developed for the head-fixed mouse preparation.

6.2 2nd paper

The second paper is a comprehensive literature review of gesture tracking/pose estimation methods for laboratory animals/mice (Abbas and Masip, 2019).

6.2.1 Title

Abbas, W., Masip, D. Computer Methods for Automatic Locomotion and Gesture Tracking in Mice and Small Animals for Neuroscience Applications: A Survey. Sensors, 19(15), 3274. (2019). ISI JCR IMPACT FACTOR: 3.031 (2018). **1st quartile**. Instruments and instrumentation.

6.2.2 Abstract

Neuroscience has traditionally relied on manually observing laboratory animals in controlled environments. Researchers usually record animals behaving freely or in a restrained manner and then annotate the data manually. The manual annotation is not desirable for three reasons;(i) it is time-consuming,(ii) it is prone to human errors, and (iii) no two human annotators will 100% agree on annotation, therefore, it is not reproducible. Consequently, automated annotation for such data has gained traction because it is efficient and replicable. Usually, the automatic annotation of neuroscience data relies on computer vision and machine learning techniques. In this article, we have covered most of the approaches taken by researchers for locomotion and gesture tracking of specific laboratory animals, ie rodents. We have divided these papers into categories based upon the hardware they use and the software approach they take. We have also summarized their strengths and weaknesses

6.3 3rd paper

The third paper covers our work on the deep learning based pipeline for limbs segmentation/tracking (Abbas, Masip, and Giovannucci, 2020).

6.3.1 Title

Abbas, W., Masip, D., & Giovannucci, A. (2020). Limbs detection and tracking of head-fixed mice for behavioral phenotyping using motion tubes and deep learning. IEEE Access, 8, 37891-37901. SI JCR IMPACT FACTOR: 4.098 (2018). **1st quartile**.

6.3.2 Abstract

The broad accessibility of affordable and reliable recording equipment and its relative ease of use has enabled neuroscientists to record large amounts of neurophysiological and behavioral data. Given that most of this raw data is unlabeled, great effort is required to adapt it for behavioral phenotyping or signal extraction, for behavioral and neurophysiological data, respectively. Traditional methods for labeling datasets rely on human annotators which is a resource and time intensive process, which often produces data that is prone to reproducibility errors. Here, we propose a deep learning-based image segmentation framework to automatically extract and label limb movements from movies capturing frontal and lateral views of head-fixed mice. The method decomposes the image into elemental regions (superpixels) with similar appearance and concordant dynamics and stacks them following their partial temporal trajectory. These 3D descriptors (referred as motion cues) are used to train a deep convolutional neural network (CNN). We use the features extracted at the last fully connected layer of the network for training a Long Short Term Memory (LSTM) network that introduces spatio-temporal coherence to the limb segmentation. We tested the pipeline in two video acquisition settings. In the first, the camera is installed on the right side of the mouse (lateral setting). In the second, the camera is installed facing the mouse directly (frontal setting). We also investigated the effect of the noise present in the videos and the amount of training data needed, and we found that reducing the number of training samples does not result in a drop of more than 5% in detection accuracy even when as little as 10% of the available data is used for training.

Appendix A

Long Short Term Memory (LSTM) Network

Long Short-Term Memory neural networks are a special kind of recurrent neural networks (RNN) designed to overcome these error back-flow problems Hochreiter and Schmidhuber, 1997a. RNNs are specifically designed for learning sequences with some form of memory to remember its state from previous examples. They have loops, allowing information to persist. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. The loops can be unraveled to form sequences which make RNNs incredibly useful for sequence learning. This usefulness comes at a cost. As the memory span grows, the back propagated error increases. An LSTM is designed to overcome these error backflow problems. It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, in-compressible input sequences, without loss of short time lag capabilities. The key to LSTMs is the cell state, which can be visualized as a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. The LSTMs have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed of a sigmoid neural net layer and a point wise multiplication operation. The major innovation of LSTM is its memory cell c_t which essentially acts as an accumulator of the state information. The cell is accessed, written and cleared by several self-parameterized controlling gates. Every time a new input comes, its information will be accumulated to the cell if the input gate it is activated. Also, the past cell status c_{t-1} could be "forgotten" in this process if the forget gate f_t is on. Whether the latest cell output c_t will be propagated to the final state ht is further controlled by the output gate o_t. The architecture of an LSTM cell is shown in Fig. A.1

One advantage of using the memory cell and gates to control information flow is that the gradient will be trapped in the cell (also known as constant error carousels (Hochreiter and Schmidhuber, 1997b)) and be prevented from vanishing too quickly, which is a critical problem for the vanilla RNN model (LeCun, Bengio, and Hinton, 2015; Hochreiter and Schmidhuber, 1997b; Pascanu, Mikolov, and Bengio, 2013).

A.1 Fully connected LSTM (FC-LSTM)

FC-LSTM may be seen as a multivariate version of LSTM where the input, cell output and states are all 1D vectors. The key equations are shown in Eq. A.1, where '$\circ$'

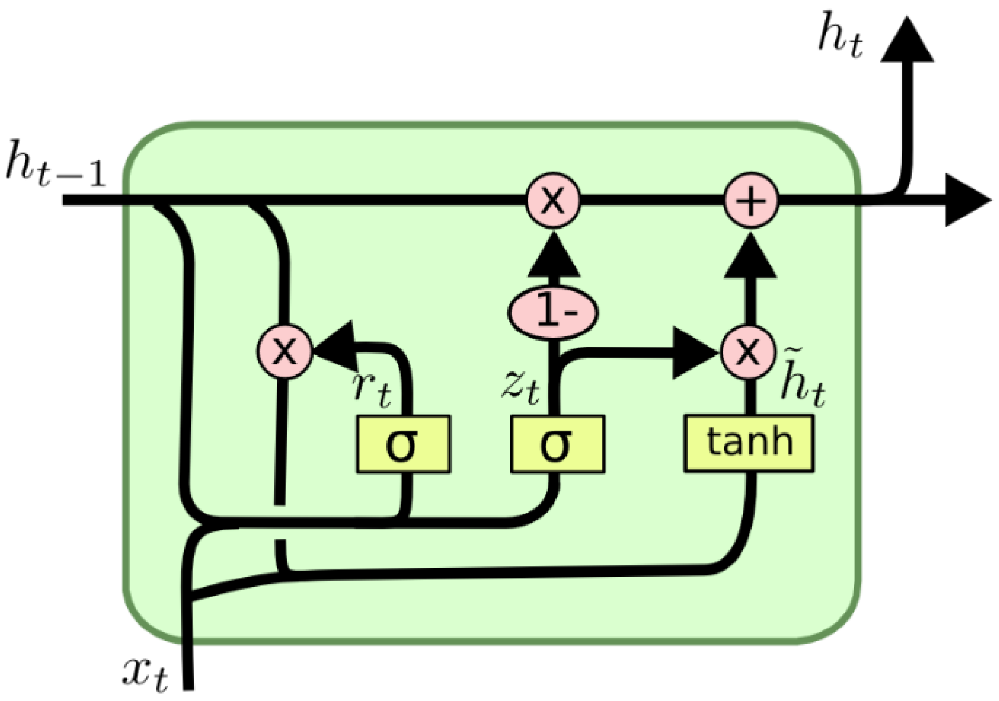


FIGURE A.1: Illustration of the inner structure of an LSTM. Source: *Understanding LSTM Networks – colah's blog* 2020

denotes the Hadamard product and σ represents any non-linear activation function:

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \\ c_t &= f_t \circ c_{t-1} + i_t \circ tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \\ h_t &= o_t \circ tanh(c_t) \end{aligned} \tag{A.1}$$

Appendix B

Supplemental information for Chapter 4

B.1 Leaky reLu Activation

A rectifier is an activation function defined by the following equation

$$f(x) = x^{+} = argmax(0, x) \tag{B.1}$$

where x is the input. This is also alternatively known as a ramp function.

This activation function was first introduced by Hahnloser et al. in (Hahnloser and Seung, 2001; Hahnloser et al., 2000). Glorot et. al. demonstrated for the first time in 2011 that this activation function can better train deeper networks (Glorot, Bordes, and Bengio, 2011). When a small, positive gradient is allowed even when the unit is not active, the typical reLu activation becomes leaky reLu, represented by the following equation.

$$f(x) = \begin{cases} x & if\ x > 0 \\ 0.01x & otherwise \end{cases} \tag{B.2}$$

The leaky relu activation function is illustrated in Fig. B.1

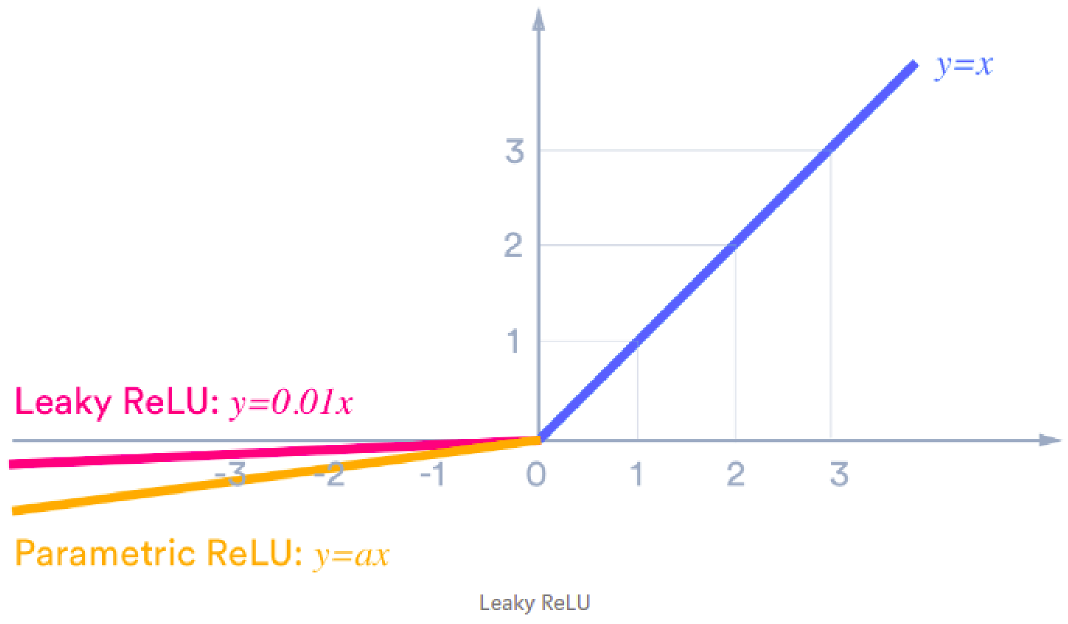


FIGURE B.1: Illustration of Leaky ReLu activation function. Source: *Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning* 2020

Appendix C

Simple Linear Iterative Clustering (SLIC) Algorithm

SLIC stands for Simple Linear Iterative Clustering. This algorithm clusters pixels in the combined five-dimensional color and image plane space (labxy) to efficiently generate compact, nearly uniform superpixels where [lab] is the pixel color vector in CIELAB color space and xy is the pixel position. The spatial distances need to be normalized so the Euclidean distance can be used in this 5D space because the maximum possible distance between two colors in the CIELAB space is limited whereas the spatial distance in the xy plane depends on the image size. SLIC takes as input a desired number of approximately equally-sized superpixels K. At the onset of the algorithm, K superpixel cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]$ are chosen with k= [1,K] at regular grid intervals S. Since the spatial extent of any superpixel is approximately S^2(the approximate area of a super-pixel), it is safely assumed that pixels that are associated with this cluster center lie within a $2S \times 2S$ area around the superpixel center on the xy plane. The normalized distance measure (D_s) to be used in the 5D space is defined as :

$$D_s = d_{lab} + (m/S) * d_{xy} \tag{C.1}$$

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2 + (x_k - x_i)^2 + (y_k - y_i)^2} \tag{C.2}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{C.3}$$

The variable m is introduced in D_s to control the compactness of a superpixel. The greater the value of m, the more compact the cluster.

First, K regularly spaced cluster centers are sampled and moved to seed locations corresponding to the lowest gradient position in a 3×3. Image gradients are computed as:

$$G(x, y) = \|I(x+1, y) - I(x-1, y)\|^2 + \|I(x, y+1) - I(x, y-1)\|^2 \tag{C.4}$$

where I(x,y) is the lab vector corresponding to the pixel at position (x,y), and $\|.\|$ is the L2 norm. Each pixel assigned to the nearest cluster center whose search area overlaps the pixel. After pixels association to the nearest cluster center, the cluster center is updated. This process is repeated until there is no significant change in cluster membership.

Appendix D

Optical Flow

Optic flow or optical flow is defined as the apparent change of structured light in the image due to a relative motion between the camera and the scene. According to Helmholtz: "My belief too is that it is mainly by variations of the retinal image due to bodily movements that one-eyed persons are able to form correct apperceptions of the material shapes of their surroundings." (Helmholtz, 2013)

According to Gibson: "Analytically, this total transformation of the array appears to mean that the elements of this texture are displaced, the elements being considered as spots. Introspectively, the field is everywhere alive with motion when the observer moves." (Gibson, 1966)

According to Horn: "The apparent motion of brightness patterns observed when a camera is moving relative to the objects being imaged is called optical flow." (Horn and Schunck, 1981b; Horn, Klaus, and Horn, 1986)

Let's assume that a video or an image sequence is described by a gray-value function f(x,y,t), where x and y denote the spatial Cartesian coordinates and t the time or temporal dimension. We are interested in the local, dense vector field (u(x,y,t),v(x,y,t)) to be estimated from that video/image sequence. The optical flow methods try to calculate the motion between two image frames in the form of dense fields which are taken at times t and $t + \delta t$ at every voxel position. These methods are based on local Taylor series approximations of the image signal as they use partial derivatives with respect to the spatial and temporal coordinates. Optical flow is computed under **brightness constancy constraint** which can be expressed as follows.

$$I(x, y, t) = I(x + \delta x, y + \delta y, t) \tag{D.1}$$

If we assume that the pixel movement between two adjacent frames is small, the Taylor series can be approximated as follows.

$$I(x + \delta x, y + \delta y, t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t \tag{D.2}$$

Ingnoring higher order terms of the Taylor series, we can linearize the above aquation as follows.

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0 \tag{D.3}$$

$$\frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} = 0 \tag{D.4}$$

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{D.5}$$

where V_x, V_y are the x and y components of the optical flow and ∂ represents the partial derivative. Summarizing the above equation.

$$I_x V_x + I_y V_y = -I_t \tag{D.6}$$

$$\nabla . \vec{V} = -I_t \tag{D.7}$$

We have two unknowns in this equation, so it cannot be solved straight away. Therefore, multiple researchers have proposed different approaches to estimate the flow fields (Beauchemin and Barron, 1995).

Bibliography

Abbas, Waseem and David Masip (2019). “Computer Methods for Automatic Locomotion and Gesture Tracking in Mice and Small Animals for Neuroscience Applications: A Survey”. In: *Sensors* 19.15, p. 3274.

Abbas, Waseem, David Masip, and Andrea Giovannucci (2020). “Limbs detection and tracking of head-fixed mice for behavioral phenotyping using motion tubes and deep learning”. In: *IEEE Access* 8, pp. 37891–37901.

Achanta, Radhakrishna et al. (2012). “SLIC superpixels compared to state-of-the-art superpixel methods”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.11, pp. 2274–2282.

Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning (2020). https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e. (Accessed on 06/21/2020).

Aine, Cheryl J (1995). “A conceptual overview and critique of functional neuroimaging techniques in humans: I. MRI/FMRI and PET.” In: *Critical reviews in neurobiology* 9.2-3, pp. 229–309.

Apthorpe, Noah et al. (2016). “Automatic neuron detection in calcium imaging data using convolutional networks”. In: *Advances in Neural Information Processing Systems*, pp. 3270–3278.

Arac, Ahmet et al. (2019). “DeepBehavior: A deep learning toolbox for automated analysis of animal and human behavior imaging data”. In: *Frontiers in systems neuroscience* 13, p. 20.

Beauchemin, Steven S. and John L. Barron (1995). “The computation of optical flow”. In: *ACM computing surveys (CSUR)* 27.3, pp. 433–466.

Berridge, Michael J, Peter Lipp, and Martin D Bootman (2000). “The versatility and universality of calcium signalling”. In: *Nature reviews Molecular cell biology* 1.1, p. 11.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Cao, Zhe et al. (2018). “OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields”. In: *arXiv preprint arXiv:1812.08008*.

Cell Magic Wand - omicX (2020). https://omictools.com/cell-magic-wand-tool. (Accessed on 06/10/2020).

Chen, Tsai-Wen et al. (2013). “Ultrasensitive fluorescent proteins for imaging neuronal activity”. In: *Nature* 499.7458, p. 295.

Chittajallu, R, S Alford, and GL Collingridge (1998). “Ca2+ and synaptic plasticity”. In: *Cell calcium* 24.5-6, pp. 377–385.

Codeneuro. *codeneuro/neurofinder*. URL: https://github.com/codeneuro/neurofinder.

Dankert, Heiko et al. (2009). “Automated monitoring and analysis of social behavior in Drosophila”. In: *Nature methods* 6.4, p. 297.

Denk, Winfried, James H Strickler, and Watt W Webb (1990a). “Two-photon laser scanning fluorescence microscopy”. In: *Science* 248.4951, pp. 73–76.

— (1990b). “Two-photon laser scanning fluorescence microscopy”. In: *Science* 248.4951, pp. 73–76.

Dombeck, Daniel A et al. (2007). “Imaging large-scale neural activity with cellular resolution in awake, mobile mice”. In: *Neuron* 56.1, pp. 43–57.

Dombeck, Daniel A et al. (2010). “Functional imaging of hippocampal place cells at cellular resolution during virtual navigation”. In: *Nature neuroscience* 13.11, p. 1433.

Ehsan, Shoaib et al. (2015). “Integral images: efficient algorithms for their computation and storage in resource-constrained embedded vision systems”. In: *Sensors* 15.7, pp. 16804–16830.

Freund, Yoav and Robert E Schapire (1997). “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of computer and system sciences* 55.1, pp. 119–139.

Garaschuk, Olga et al. (2000). “Large-scale oscillatory calcium waves in the immature cortex”. In: *Nature neuroscience* 3.5, p. 452.

Gazzaniga, Michael S et al. (2006). *Handbook of functional neuroimaging of cognition.* Mit Press.

Gibson, Eli et al. (2018). “Automatic multi-organ segmentation on abdominal CT with dense v-networks”. In: *IEEE transactions on medical imaging* 37.8, pp. 1822–1834.

Gibson, James Jerome (1966). “The senses considered as perceptual systems.” In:

Giovannucci, Andrea et al. (2017a). “Cerebellar granule cells acquire a widespread predictive feedback signal during motor learning”. In: *Nature neuroscience* 20.5, p. 727.

Giovannucci, Andrea et al. (2017b). “Onacid: Online analysis of calcium imaging data in real time”. In: *Advances in Neural Information Processing Systems*, pp. 2381–2391.

Giovannucci, Andrea et al. (2018). “Automated gesture tracking in head-fixed mice”. In: *Journal of neuroscience methods* 300, pp. 184–195.

Giovannucci, Andrea et al. (2019). “CaImAn an open source tool for scalable calcium imaging data analysis”. In: *Elife* 8, e38173.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). “Deep sparse rectifier neural networks”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323.

Goldstein, E (2001). *Bruce: Sensation and Perception.*

Graving, Jacob M et al. (2019). “Fast and robust animal pose estimation”. In: *bioRxiv*, p. 620245.

Grienberger, Christine and Arthur Konnerth (2012). “Imaging calcium in neurons”. In: *Neuron* 73.5, pp. 862–885.

Grynkiewicz, Grzegorz, Martin Poenie, and Roger Y Tsien (1985). “A new generation of Ca2+ indicators with greatly improved fluorescence properties.” In: *Journal of biological chemistry* 260.6, pp. 3440–3450.

Hahnloser, Richard HR and H Sebastian Seung (2001). “Permitted and forbidden sets in symmetric threshold-linear networks”. In: *Advances in neural information processing systems*, pp. 217–223.

Hahnloser, Richard HR et al. (2000). “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. In: *Nature* 405.6789, p. 947.

Hastie, Trevor et al. (2005). “The elements of statistical learning: data mining, inference and prediction”. In: *The Mathematical Intelligencer* 27.2, pp. 83–85.

Helmholtz, Hermann von (2013). *Treatise on physiological optics.* Vol. 3. Courier Corporation.

Herculano-Houzel, Suzana (2009). “The human brain in numbers: a linearly scaled-up primate brain”. In: *Frontiers in human neuroscience* 3, p. 31.

Hochreiter, Sepp and Jürgen Schmidhuber (1997a). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.

— (1997b). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.

Horn, Berthold, Berthold Klaus, and Paul Horn (1986). *Robot vision.* MIT press.

Horn, Berthold KP and Brian G Schunck (1981a). “Determining optical flow”. In: *Artificial intelligence* 17.1-3, pp. 185–203.

— (1981b). “Determining optical flow”. In: *Techniques and Applications of Image Understanding.* Vol. 281. International Society for Optics and Photonics, pp. 319–331.

Hubel, David H (1995). *Eye, brain, and vision.* Scientific American Library/Scientific American Books.

Hubel, David H and Torsten N Wiesel (2004). *Brain and visual perception: the story of a 25-year collaboration.* Oxford University Press.

Insafutdinov, Eldar et al. (2016). “Deepercut: A deeper, stronger, and faster multi-person pose estimation model”. In: *European Conference on Computer Vision.* Springer, pp. 34–50.

jGCaMP7 | Janelia Research Campus (2020). https://www.janelia.org/open-science/jgcamp7. (Accessed on 06/10/2020).

Kaifosh, Patrick et al. (2014). “SIMA: Python software for analysis of dynamic fluorescence imaging data”. In: *Frontiers in neuroinformatics* 8, p. 80.

Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980.*

Klibisz, Aleksander et al. (2017). “Fast, simple calcium imaging segmentation with fully convolutional networks”. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support.* Springer, pp. 285–293.

Kovalchuk, Yury et al. (2000). “NMDA receptor-mediated subthreshold Ca2+ signals in spines of hippocampal neurons”. In: *Journal of Neuroscience* 20.5, pp. 1791–1799.

Kullback, Solomon and Richard A Leibler (1951). “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1, pp. 79–86.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.

Levandowsky, Michael and David Winter (1971). “Distance between sets”. In: *Nature* 234.5323, pp. 34–35.

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

Lichtman, Jeff W and José-Angel Conchello (2005). “Fluorescence microscopy”. In: *Nature methods* 2.12, p. 910.

Lienhart, Rainer and Jochen Maydt (2002). “An extended set of haar-like features for rapid object detection”. In: *Proceedings. international conference on image processing.* Vol. 1. IEEE, pp. I–I.

Maruyama, Ryuichi et al. (2014). “Detecting cells using non-negative matrix factorization on calcium imaging data”. In: *Neural Networks* 55, pp. 11–19.

Mathis, Alexander et al. (2018a). *DeepLabCut: markerless pose estimation of user-defined body parts with deep learning.* Tech. rep. Nature Publishing Group.

Mathis, Alexander et al. (2018b). “Markerless tracking of user-defined features with deep learning”. In: *arXiv preprint arXiv:1804.03142.*

Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi (2016). “V-net: Fully convolutional neural networks for volumetric medical image segmentation”. In: *2016 Fourth International Conference on 3D Vision (3DV).* IEEE, pp. 565–571.

Miyawaki, Atsushi et al. (1997). "Fluorescent indicators for Ca 2+ based on green fluorescent proteins and calmodulin". In: *Nature* 388.6645, p. 882.

Mukamel, Eran A, Axel Nimmerjahn, and Mark J Schnitzer (2009). "Automated analysis of cellular signals from large-scale calcium imaging data". In: *Neuron* 63.6, pp. 747–760.

Neher, Erwin (1998). "Vesicle pools and Ca2+ microdomains: new tools for understanding their roles in neurotransmitter release". In: *Neuron* 20.3, pp. 389–399.

neurofinder. URL: http://neurofinder.codeneuro.org/.

Newell, Alejandro, Kaiyu Yang, and Jia Deng (2016). "Stacked hourglass networks for human pose estimation". In: *European conference on computer vision*. Springer, pp. 483–499.

Ohki, Kenichi et al. (2005). "Functional imaging with cellular resolution reveals precise micro-architecture in visual cortex". In: *Nature* 433.7026, p. 597.

Oppenheim, A van, Ronald Schafer, and Thomas Stockham (1968). "Nonlinear filtering of multiplied and convolved signals". In: *IEEE transactions on audio and electroacoustics* 16.3, pp. 437–466.

Otsu, Nobuyuki (1979). "A threshold selection method from gray-level histograms". In: *IEEE transactions on systems, man, and cybernetics* 9.1, pp. 62–66.

Pachitariu, Marius et al. (2017). "Suite2p: beyond 10,000 neurons with standard two-photon microscopy". In: *Biorxiv*, p. 061507.

Palmér, Tobias et al. (2012). "A system for automated tracking of motor components in neurophysiological research". In: *Journal of neuroscience methods* 205.2, pp. 334–344.

Palmér, Tobias et al. (2014). "Rat Paw Tracking for Detailed Motion Analysis". In: *Visual observation and analysis of Vertebrate And Insect Behavior 2014*.

Paolicelli, Rosa C et al. (2011). "Synaptic pruning by microglia is necessary for normal brain development". In: *science* 333.6048, pp. 1456–1458.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*, pp. 1310–1318.

Pereira, Talmo D et al. (2019). "Fast animal pose estimation using deep neural networks". In: *Nature methods* 16.1, p. 117.

Peron, Simon, Tsai-Wen Chen, and Karel Svoboda (2015). "Comprehensive imaging of cortical networks". In: *Current opinion in neurobiology* 32, pp. 115–123.

Persechini, Anthony, Jennifer A Lynch, and Valerie A Romoser (1997). "Novel fluorescent indicator proteins for monitoring free intracellular Ca2+". In: *Cell calcium* 22.3, pp. 209–216.

Petersen, Ashley, Noah Simon, and Daniela Witten (2018). "Scalpel: Extracting neurons from calcium imaging data". In: *The annals of applied statistics* 12.4, p. 2430.

Pitas, I. and A. N. Venetsanopoulos (1990). "Homomorphic Filters". In: *Nonlinear Digital Filters: Principles and Applications*. Boston, MA: Springer US, pp. 217–243. ISBN: 978-1-4757-6017-0. DOI: 10.1007/978-1-4757-6017-0_7. URL: https://doi.org/10.1007/978-1-4757-6017-0_7.

Redmon, Joseph et al. (2016). "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Ringer, Sydney (1883). "A further contribution regarding the influence of the different constituents of the blood on the contraction of the heart". In: *The Journal of physiology* 4.1, pp. 29–42.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on*

Medical image computing and computer-assisted intervention. Springer, pp. 234–241.

Ruffieux, Simon et al. (2014). "A survey of datasets for human gesture recognition". In: *International Conference on Human-Computer Interaction*. Springer, pp. 337–348.

Russell91 (2018). *Russell91/TensorBox*. URL: `https://github.com/Russell91/TensorBox`.

Sainath, Tara N et al. (2015). "Convolutional, long short-term memory, fully connected deep neural networks". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 4580–4584.

Schapire, Robert E et al. (1998). "Boosting the margin: A new explanation for the effectiveness of voting methods". In: *The annals of statistics* 26.5, pp. 1651–1686.

Singh, T Romen et al. (2012). "A new local adaptive thresholding technique in binarization". In: *arXiv preprint arXiv:1201.5227*.

Smith, Spencer L and Michael Häusser (2010). "Parallel processing of visual space by neighboring neurons in mouse visual cortex". In: *Nature neuroscience* 13.9, p. 1144.

Soltanian-Zadeh, Somayyeh et al. (2019). "Fast and robust active neuron segmentation in two-photon calcium imaging using spatiotemporal deep learning". In: *Proceedings of the National Academy of Sciences* 116.17, pp. 8554–8563.

Sonka, Milan, Vaclav Hlavac, and Roger Boyle (2014). *Image processing, analysis, and machine vision*. Cengage Learning.

Sousa, N, OFX Almeida, and CT Wotjak (2006). "A hitchhiker's guide to behavioral analysis in laboratory rodents". In: *Genes, Brain and Behavior* 5, pp. 5–24.

Spaen, Quico et al. (2019). "HNCcorr: A novel combinatorial approach for cell identification in calcium-imaging movies". In: *eNeuro* 6.2.

Stosiek, Christoph et al. (2003a). "In vivo two-photon calcium imaging of neuronal networks". In: *Proceedings of the National Academy of Sciences* 100.12, pp. 7319–7324.

— (2003b). "In vivo two-photon calcium imaging of neuronal networks". In: *Proceedings of the National Academy of Sciences* 100.12, pp. 7319–7324.

Stutz, David, Alexander Hermans, and Bastian Leibe (2018). "Superpixels: An evaluation of the state-of-the-art". In: *Computer Vision and Image Understanding* 166, pp. 1–27.

Subach, Oksana M et al. (2019). "Near-infrared genetically encoded positive calcium indicator based on GAF-FP bacterial phytochrome". In: *International journal of molecular sciences* 20.14, p. 3488.

Südhof, Thomas C (2000). "The synaptic vesiclecycle revisited". In: *Neuron* 28.2, pp. 317–320.

Superpixel, Empirical Studies and Applications (0202). `https://ttic.uchicago.edu/~xren/research/superpixel/`. (Accessed on 06/10/2020).

Svoboda, Karel et al. (1997). "In vivo dendritic calcium dynamics in neocortical pyramidal neurons". In: *Nature* 385.6612, p. 161.

Tsien, Richard W and Roger Y Tsien (1990). "Calcium channels, stores, and oscillations". In: *Annual review of cell biology* 6.1, pp. 715–760.

Tsien, Roger Y (1980). "New calcium indicators and buffers with high selectivity against magnesium and protons: design, synthesis, and properties of prototype structures". In: *Biochemistry* 19.11, pp. 2396–2404.

— (1981). "A non-disruptive technique for loading calcium buffers and indicators into cells". In: *Nature* 290.5806, p. 527.

Understanding LSTM Networks – colah's blog (2020). https://colah.github.io/posts/2015-08-Understanding-LSTMs/. (Accessed on 06/21/2020).

Valmianski, Ilya et al. (2010). "Automatic identification of fluorescently labeled brain cells for rapid functional imaging". In: *Journal of neurophysiology* 104.3, pp. 1803–1811.

Van Erven, Tim and Peter Harremos (2014). "Rényi divergence and Kullback-Leibler divergence". In: *IEEE Transactions on Information Theory* 60.7, pp. 3797–3820.

Viola, Paul, Michael Jones, et al. (2001). "Rapid object detection using a boosted cascade of simple features". In: *CVPR (1)* 1.511-518, p. 3.

Wang, Yangzhen et al. (2019). "Efficient implementation of convolutional neural networks in the data processing of two-photon in vivo imaging". In: *Bioinformatics*.

Web Of Science (2019). URL: www.webofknowledge.com.

Wen, Chentao et al. (2018). "Deep-learning-based flexible pipeline for segmenting and tracking cells in 3D image time series for whole brain imaging". In: *bioRxiv*, p. 385567.

Xingjian, SHI et al. (2015). "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*, pp. 802–810.

Xu, Kun et al. (2016). "Neuron segmentation based on CNN with semi-supervised regularization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 20–28.

Yang, Helen H and François St-Pierre (2016). "Genetically encoded voltage indicators: opportunities and challenges". In: *Journal of Neuroscience* 36.39, pp. 9977–9989.

Yuste, Rafael and Winfried Denk (1995). "Dendritic spines as basic functional units of neuronal integration". In: *Nature* 375.6533, p. 682.

Zhang, Li, Yuan Li, and Ramakant Nevatia (2008). "Global data association for multi-object tracking using network flows". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 1–8.

Zhu, Xiaojin, Zoubin Ghahramani, and John D Lafferty (2003). "Semi-supervised learning using gaussian fields and harmonic functions". In: *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919.