Artificial neural networks in action for an automated cell-type classification of biological neural networks

Eirini Troullinou*†, Grigorios Tsagkatakis†, Spyridon Chavlis†‡,
Gergely Turi§, Wen-Ke Li§, Attila Losonczy§, Panagiotis Tsakalides*†, and Panayiota Poirazi‡
*Department of Computer Science, University of Crete, Heraklion, 70013, Greece
†Institute of Computer Science, Foundation for Research and Technology Hellas, Heraklion, 70013, Greece
‡Institute of Molecular Biology and Biotechnology, Foundation for Research and Technology Hellas, Heraklion, 70013, Greece
§Department of Neuroscience, Columbia University Medical Center, New York, USA

Abstract—In this work we address the problem of neuronal cell-type classification, and we employ a real-world dataset of raw neuronal activity measurements obtained with calcium imaging techniques. While neuronal cell-type classification is a crucial step in understanding the function of neuronal circuits, and thus a systematic classification of neurons is much needed, it still remains a challenge. In recent years, several approaches have been employed for a reliable neuronal cell-type recognition, such as immunohistochemical analysis and feature extraction algorithms based on several characteristics of neuronal cells. These methods, however, are time consuming and demand a lot of human intervention. Moreover, with respect to feature extraction algorithms, it remains unclear what are the best features that define a neuronal cell class. In this work we examine three different deep learning models aiming at an automated neuronal cell-type classification and compare their performance. Experimental analysis demonstrates the efficacy and potent capabilities for each one of the proposed schemes. We demonstrate that automated classification for four types of neurons can be achieved with unprecedented accuracy.

Index Terms—Artificial neural networks, calcium imaging, neuronal cell-type classification

1 Introduction

To understand the function -and dysfunction- of neural circuits we must first identify the subtypes of neurons that comprise these networks, their biophysical and anatomical characteristics as well as their inter-dependencies. Different cell types typically exhibit different anatomy, connectivity and/or biophysical properties which, in turn, influence their specific function and role in pathologies such as epilepsy [1], [2], anxiety disorder [2], [3], the Tourette syndrome [4], autism [5], the Rett syndrome [6] and schizophrenia [7], [8]. The development of a cellular taxonomy will thus facilitate our understanding of both healthy and diseased brain functioning, as most brain pathologies affect specific neuronal types [9], [10].

Despite this pressing need, neuronal classification remains challenging, primarily because the characteristic features of specific neurons and their uniqueness remain largely unknown. Moreover, there are many conceptual and technical challenges associated with the experimental identification of different cell types [11] and little progress has been made in linking cell types to specific brain functions (i.e., association of neuronal types with behavior, computation, and eventually cognition). Traditionally, neuronal cell types are classified using qualitative descriptors, such as the expression of specific proteins in combination with their anatomy/morphology and laminar localization [11], [12], [13]. However, such techniques require expensive and time-consuming experiments and are often limited to just a few neurons with known protein expression profiles and available markers. Given the magnitude and complexity of neuronal classification, and because the manual classification attempts using qualitative descriptors are ill-equipped to deal with big data, high-throughput technologies are demanded.

Recently, quantitative methods using supervised and unsupervised classifiers have been developed for neuronal classification based on morphological, physiological, molecular, and/or electrophysiological characteristics [14], [15], [16], [17]. These methods provide quantitative and unbiased identification of distinct neuronal subtypes when applied to selected datasets. However, obtaining such characteristic features for different cell classes entails laborious and expensive experimentation, often involving several different techniques, which limits the attractiveness of automated classifiers relying on such features.

In this work, we aim at the automated cell-type recognition using a real-world dataset and relying on a single data type, namely timeseries of calcium (Ca^{2+}) activity signals of neurons. Specifically, we use timeseries describing the activity of four neuronal cell-types in the CA1 subregion of the hippocampus collected from behaving animals. The dataset consists of the excitatory pyramidal cells (PY) and three GABAergic interneuronal subtypes, namely parvalbumin-

positive (PV), somatostatin-positive (SOM) and vasoactive intestinal polypeptide-positive (VIP) cells. Neuronal activity is measured using Ca^{2+} imaging, which is a powerful technique for monitoring the activity of distinct neurons in brain tissue *in vivo* [18] and is currently the most popular recording technique for behaving animals [19], [20], [21]. Here, we examine the potential of replacing existing approaches (i.e., feature extraction-based algorithms as well as immunohistochemical analysis methods) with fast, reliable, cell-type classification, which is based on Ca^{2+} imaging recordings using state-of-the-art Deep Learning (DL) architectures for timeseries analysis. Towards this goal, we consider the raw fluorescence signal without any preprocessing. To our knowledge, this is the first successful attempt to classify cell-types based solely on the raw Ca^{2+} activity signal.

Since their origin in 1946, Artificial Neural Networks (ANN) have been used in many real-world applications, such as speech [22] and handwriting recognition [23], audio-visual speech enhancement [24], video dynamic detection [25] and in applications of emerging areas such as, real time sign-language translation [26] and smart cities [27]. The rise of the "golden age" of DL is attributed to their ability to derive important characteristics from the raw data by learning hierarchical representations encoding different levels of abstraction, thus replacing the conventional approaches that heavily rely on the definition of efficient feature extractors, a non-trivial and very challenging task. The recent success of DL has been facilitated by their capacity to harness big data and the cutting-edge hardware technologies that are exploited by these networks.

With the advent of DL, various approaches and models for timeseries analysis and forecast have been developed too. For different fields and applications, suitable algorithms and models vary depending on the nature and purpose of the data. For the task of neuronal cell-type classification considered in this work, we employ three types of network architectures, namely 1-Dimensional Convolutional Neural Networks (1D-CNNs), Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs), which are widely used in timeseries analysis. The motivation behind using the 1D-CNN model is that this architecture can effectively exploit temporal locality, while the other two architectures are ideal for revealing long-term dependencies in the Ca^{2+} activity data.

The remainder of the paper is organized as follows: In section II, we report the motivation and contribution of our study as well as the related, prior work on neuronal cell type classification. In Section III, we describe and analyze the proposed approaches. Experimental results are presented in Section IV and conclusions are drawn in Section V.

2 Objectives, Related Work and Contribution

Our study investigates whether the selected DL methods (i.e., 1D-CNN, RNN and LSTM) can solve the problem of neuronal cell-type classification when utilizing as input a single signal type, namely the raw Ca^{2+} activity signal of neurons recorded in the CA1 area of the hippocampus from behaving mice. We focus on a four-class problem, consisting of PY cells and three GABAergic interneurons (IN) subtypes (PV, SOM and VIP cells).

2.1 Related Work

Timeseries analysis using DL architectures is among the most active areas of research in machine learning. DL architectures like CNN, RNN and LSTM have been used for a multitude of different tasks including the analysis of measurements from wearable sensors , modeling networks traffic for cybersecurity, and high frequency trading in finance among others [28]. For the case of biological data analysis, application of DL and traditional machine learning methods are presented next.

Cell-type classification has been extensively studied in the past decades, however it was primarily aimed at discriminating between healthy and malignant cells [29], [30], [31], [32]. Hence, little progress has been made in the automated classification of neuronal cell-types. For example, it is widely known that the two major neuronal types of the mammalian brain are the pyramidal cells and the GABAergic IN. While both PY and IN vary substantially in their anatomy and biophysical properties across brain areas, a clear classification of the different cell types has yet to be achieved. INs in particular come in many different types and shapes [33], even within the same brain region. Specifically, the hippocampus has over 20 different types of INs [34]. In this section we review several qualitative as well as quantitative approaches, whose ultimate goal is to automate neuronal cell-type classification.

To aid ongoing efforts towards IN classification, to facilitate the exchange of information and to build a foundation for future progress in the field, the Petilla Interneuron Nomenclature Group (PING), proposed a standardized nomenclature of IN properties [17] by defining qualitative descriptors (i.e., key features) that can be used for their identification. Such features are morphological, molecular, physiological and biophysical properties of these cells. Based on these qualitative descriptors, Zeng et al' [11] reviewed high-throughput classification methods, such as light microscopy, electron microscopy, optical imaging of electrical activity and molecular profiling, which enable the collection of morphological, physiological and molecular data from large numbers of neurons.

Guerra et al' [14] explored the utilization of supervised and unsupervised classification algorithms to distinguish INs from PY, based solely on their morphological features. They used a database of 128 PY and 199 INs from mouse neocortex, and for each cell, 65 morphological features were measured, creating a data matrix. Their main finding was that supervised classification methods outperformed unsupervised algorithms (hierarchical clustering), and thus the latter approach is not as effective as supervised classification when distinguishing between the aforementioned cell types. Eventually, they showed that the selection of subsets of distinguishing features enhanced the classification accuracy for both sets of algorithms.

Vasques et al' [15] used 43 morphological features as predictors and showed the results of applying supervised and unsupervised classification techniques in order to distinguish neuronal cell-types. More specifically, they assessed

and compared the accuracy of different classification algorithms trained on 430 digitally reconstructed neurons and classified them according to layer and/or m-type (i.e morphology) with young and/or adult developmental state. Their findings regarding the superiority of supervised algorithms against unsupervised coincide with those of study [14].

DeFelipe et al' [35] proposed a classification scheme based on 6 axonal features, which are considered as a representative subset of axonal morphological properties that could be suitable for IN classification. They also designed and deployed an interactive web-based system to empirically test the level of agreement among 42 experts in assigning the 6 features to individual cortical INs. More specifically, experienced neuroscientists were asked to ascribe the categories they considered most appropriate to each neuron (there were 6 features and 21 categories in total, and thus, based on each feature, the neuron would be ascribed to one out of the 2 or more categories that corresponded to the specific feature). For some of the proposed features, there were high levels of observed agreement between experts in the classification of neurons, while for other features there was a low level of inter-expert agreement. The ultimate goal of their experiment was to build a model that could classify a neuron on the basis of its morphological characteristics and more specifically, in terms of the 6 features defined in their study. They built 6 classifiers (one per feature) and assigned to each neuron the category that received the highest number of votes. As in the case of experts' agreement analysis, some features yielded high performance accuracy, while for some other features the accuracy was much lower.

Overall, these studies highlight the current state-of-the-art in cell-type classification in a feature-based manner. However, to our knowledge, there is no prior work on neuronal cell-type classification that relies solely on the Ca^{2+} activity of neurons recorded *in vivo* from behaving animals. As such, a major novelty of this work is the application of DL-based time-series analysis methods in cell classification.

2.2 Our contribution

In this work, we employ and compare the performance of 1D-CNN, RNN, and LSTM models on the task of neuronal cell-type classification based solely on the raw Ca^{2+} signals measured in different types of neurons while mice perform a goal oriented task [36]. A comparative research analysis on the specific application using the aforementioned models is missing from the existing literature, and to the best of our knowledge, this is the first time that algorithmic models use raw Ca^{2+} signals to classify different neuronal types *in vivo*. As discussed in the Related Work section, most of the algorithmic approaches use morphological features of neurons, with numerous limitations: (1) the geometry of individual neurons varies significantly within the same class, (2) different techniques are used to extract morphologies (e.g., imaging, histology, and reconstruction techniques), which introduce variability in the measured characteristics, and finally (3) there is high inter-laboratory variability [37].

Our work utilizes Ca^{2+} imaging, which is currently the most widely used technique for recording the activity of neurons in the behaving animal. It allows the simultaneous recording of tens to hundreds of cells without causing any damage to the neural tissue of interest, as opposed to more invasive, electrode-based recording methods (e.g., tetrodes, octrodes and silicon probes). Compared to these methods, Ca^{2+} imaging is also more stable, as the same neurons can be recorded over time periods that extend from days to months. Nevertheless, due to its slow kinetics, the signal produced is not ideal for resolving single spikes and makes the inference of neuronal cell-types non-trivial. Thus, cell-type classification relies mostly on the expression of specific molecular markers [11], which however are limited to just a small number of classes, many of which comprise of several sub-types of INs. Moreover, such an approach is laborious, expensive and requires the use of many different transgenic animal lines.

To address these limitations, our work makes the following contributions to the problem of automatic neuronal cell-type classification:

- We present a comparative research analysis of the 1D-CNN, RNN and LSTM models, in the domain of timeseries analysis for the task of neuronal cell-type classification, where such an analysis is missing from the existing literature.
- To the best of our knowledge, this is the first time that algorithmic models use raw Ca^{2+} signals to classify different neuronal types. The models are based solely on the Ca^{2+} imaging signatures of the neuronal types, unlike existing post-hoc techniques.
- Our proposed approach (i.e., DL models based only on Ca^{2+} signals) is feature-independent and can thus replace feature-extraction-based methods.
- Performance accuracy is very high, suggesting that a gradual substitution of immunohistochemical analysis could be potentially considered, as such analyses are laborious, time-consuming and very expensive.

3 Proposed Approaches

To assess whether four cell types (PY, PV, SOM and VIP), whose activity is described with timeseries of raw Ca^{2+} imaging data can be correctly classified, we employ the following classification models and compare their performance:

- 1-Dimensional Convolutional Neural Networks (1D-CNN)
- Recurrent Neural Networks (RNN)
- Long Short-Term Memory Networks (LSTM)

3.1 Preliminary Concepts

In this subsection we discuss the theoretical background of the proposed approaches.

3.1.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a class of deep neural networks (DNN), most commonly applied to imaging applications that consists of input, output and hidden layers of nodes along with their respective connections that encode the learnable weights of the network. CNNs are regularized versions of traditional Multilayer Perceptrons

(MLPs), which usually refer to fully connected networks, i.e., each node in one layer connects to all nodes in the next layer. This characteristic of MLPs makes them prone to overfitting during training. Hence, one of the distinguishing property of CNNs compared to MLPs is the local connectivity among nodes. When dealing with high-dimensional inputs such as Ca^{2+} imaging timeseries data, it is impractical to connect nodes in one layer with all nodes in the previous volume because such a network architecture does not take the overall structure of the data into account.

CNNs exploit local correlations by enforcing a sparse local connectivity pattern between neurons of adjacent layers, i.e., each node is connected to only a small region of the input signal. Namely, in a convolutional layer, nodes receive input from only a restricted subarea of the previous layer and this input area of a node is called its receptive field. Another distinguishing feature of CNNs is the shared weights. In CNNs, each filter is replicated across the entire receptive field. These replicated units share the same parameterization (weight vector and bias) and form a feature map, i.e., all nodes in a given convolutional layer respond to the same feature within their specific receptive field. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting a property of translation invariance.

3.1.2 Typical Architecture of a 1-Dimensional CNN

A typical 1D-CNN architecture, as shown in Fig. 1(c) is formed by a stack of distinct layers that transform the input volume into an output volume (holding the class scores) through a differentiable function. The core building block of a CNN is the convolutional layer, whose parameters' consist of a set of learnable filters (or kernels), which have a small receptive field. Given an input vector $x \in R^{1\times N}$ and a trainable filter $f \in R^{1\times K}$, the convolution of the two entities results in an output vector $c \in R^{1\times M}$, where $M = N - K + 1$. The value of M may vary based on the stride of the operation of convolution, with bigger strides leading to smaller outputs.

The trainable parameters of the network, i.e., the filter and the bias are initialized randomly, but as the network is trained using the backpropagation learning algorithm, they are optimized and are able to capture important features from the given inputs. In order to construct a reliable network that will be able to capture complex and abstract features from the input data, we need to build a deep architecture comprised with more than one convolutional layer. As we go through layers in deep architectures, the features not only increase in number (depth size) but also in complexity. In other words, the network builds a hierarchical representation of the input, as the first layer represents the input in terms of elementary features and the deeper we go, the more abstract features can be recognized from the layers. The capture the complex features requires also to introduce some non-linearity in our system. Thus, non-linear functions are interjected between adjacent convolutional layers, and as a result a two-layer CNN can be proven to be a universal approximator [38], while the identity function does not satisfy this property and generally, when multiple layers use the identity function, the entire network is equivalent to a single-layer model. Typical choices for the non-linear function, also known as activation function, include the logistic (sigmoid) function, the hyperbolic tangent (tanh), the Rectified Linear Unit (ReLU) and its variations [39].

Another important concept of CNNs is the pooling layer, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling, among which max pooling is the most common. Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind pooling in CNNs. The pooling layer progressively reduces the size of the representation, the number of parameters, the memory footprint and the amount of computation in the network, and hence also controls overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture.

Eventually, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via the Fully Connected layers (FC), commonly referred to as as dense layers. As its name implies, nodes in a fully connected layer have connections to all nodes in the previous layer leading to a very dense connectivity structure. Essentially, when the FC is inserted at the end of the architecture, it looks at the output of the previous layer, which represents the activation maps of high level features and determines which features are mostly correlated to a particular class.

As a final classification step, we use the softmax activation function, which extends the idea of logistic regression into a multi-class world. That is, softmax assigns decimal probabilities to each class in a multi-class problem using the following equation:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \qquad \text{for i=1,...,C} \tag{1}$$

where x is the input of the fully connected layer and C is the total number of the distinct classes related to the problem at hand. This probabilistic approach renders possible to quantify the level of confidence for each estimation and provides a lucid view on what has been misconstrued in the case of misclassification.

3.1.3 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a kind of neural network that specializes in processing sequences and has shown promising results in many Natural Language Processing (NLP) tasks [40]. The idea behind RNNs is the usage of sequential information, and they are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a memory component, which captures information about what has been calculated thus far.

The RNN model in our proposed method receives an input vector x and gives an output vector o, which in our case are the input timeseries and the cell-type label, respectively. The diagram in Fig. 1(a) shows an RNN architecture being unrolled (or unfolded) into a full network, which means that we write out the network for the complete sequence.

At timestep t, $x_t \in R^{1\times d}$ is the input entry, where d is the dimensionality of that entry and h_t is the hidden

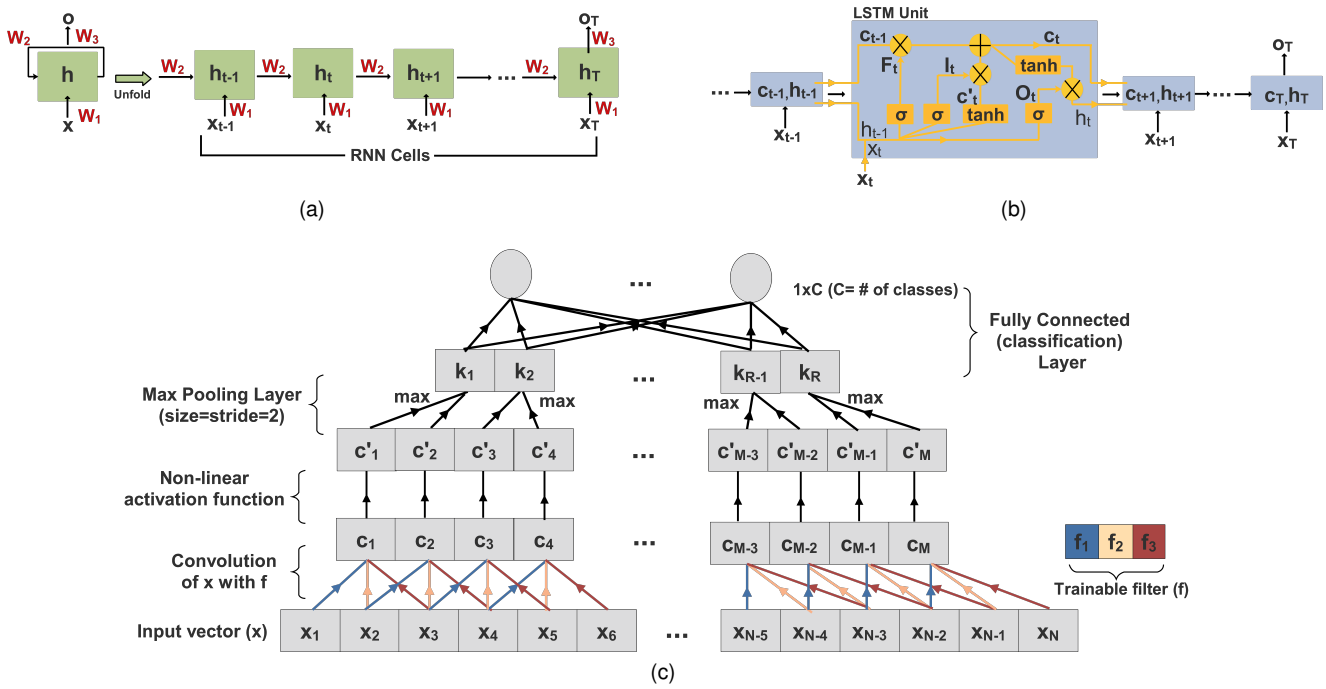

Fig. 1. Proposed Deep Learning architectures for the neuronal cell-type classification: (a) RNN architecture: An RNN layer unfolded in T RNN Cells, where T is the total number of timesteps. Every cell receives at timestep t, the current value x_t and the previous hidden state h_{t-1} value as inputs and, in our case, only the last cell outputs a vector o_T, which represents the 4 distinct classes of our problem. (b) LSTM architecture: An LSTM layer unfolded in T LSTM units. At timestep t the gates I, F and O calculate their activations (i.e. I_t, F_t and O_t respectively) considering the current value x_t and the activation of the memory cell at the previous timestep c^{t-1}. Circles containing the X symbol represent an element-wise multiplication between its inputs. The rectangles containing a σ symbol represent the application of the sigmoid differentiable function. At the final timestep T the last LSTM unit outputs the vector o_T with the 4 classes of the problem. (c) 1D-CNN architecture: The input vector x is convolved with a trainable filter f (stride equal to 1) resulting to a vector c, to which a non-linear activation function is applied, resulting to another vector c' with the same size. A max pooling layer of size 2 is also applied to c', in order to down-sample the input representation reducing its dimensionality. The number of the output nodes C equals to the number of the classes (4 in our case).

state at timestep t. Hidden state h_t can be considered as the memory component of the network, given that it captures information about what happened in all previous timesteps. It is calculated based on the following equation and utilizes the previous hidden state as well as the input at the current timestep t:

$$h_t = f_1(W_1 x_t + W_2 h_{t-1} + b_h) \tag{2}$$

where function f_1 is usually non-linear, such as tanh or ReLU, W_1 and W_2 are the weight matrices used for both $x_t \rightarrow h_t$ and $h_{t-1} \rightarrow h_t$ links, respectively, whereas b_h is the bias term added when calculating h_t. Hidden state h_{-1}, which is required for the calculation of the first hidden state, is typically initialized to zeroes. At the final timestep T, a dense layer calculates, as shown in Fig. 1(a), the output o_T, given by the following equation:

$$o_T = f_2(W_3 h_T + b_o) \tag{3}$$

where f_2 is usually a softmax activation function (as in the case of 1D-CNNs, and generally when the task is a multi-class classification problem), which takes the logits of the dense layer and turns them into probabilities, and b_o is the added bias. Note that depending on the task, Fig. 1(a) could have an output o_t at each timestep t, but for our application, which outputs a cell-type label given an input timeseries, only one output is calculated at the final timestep. Moreover, unlike a traditional DNN, which uses different parameters at each layer, an RNN model shares the same parameters (W_1, W_2), as depicted in Fig. 1(a), across all timesteps. This reflects that we are performing the same task at each step, using only different inputs, which greatly reduces the total number of learnable parameters.

Despite the advantage of the RNNs to remember information through time, training an RNN is not a simple task with respect to the backpropagation algorithm, which is used as in the traditional neural networks but with a little twist. Because the parameters are shared by all timesteps in the network, the gradient at each output depends not only on the calculations of the current timestep, but also on the previous timesteps. For example, in order to calculate the gradient at $t = j$ we need to backpropagate $j - 1$ steps and sum up the gradients. This is called Backpropagation Through Time (BPTT) and vanilla RNNs trained with BPTT have difficulties to learn long-term dependencies (i.e., dependencies between steps that are far away) [41]. This results in the so-called vanishing/exploding gradient problem. Thus, in order to address these issues, certain types of RNNs, such as LSTMs, which are described in the next subsection, were specifically designed to handle these drawbacks.

3.1.4 *Long Short-Term Memory Neural Networks*

Long Short-Term Memory is an artificial RNN model [42] used in the field of DL, which was developed to cope with

the exploding/vanishing gradient problems that can be encountered when training traditional RNNs. An LSTM has a similar control flow as a RNN. It processes data passing on information as it propagates forward. The differences with an RNN layer are the operations within the LSTM's units. LSTMs, sometimes also combined with other methods, and various extensions or variants of LSTMs do not only process single data points, such as images [43], but also entire sequences of data, such as speech or video and therefore are applicable to tasks, such as speech [44] and handwriting recognition [23], sign-language translation [26], etc.

A typical LSTM architecture will contain several LSTM units (as many as the number of timesteps). A common architecture of an LSTM unit at timestep t, as shown in Fig. 1(b), is composed of the cell state c_t , which is the memory part of the LSTM unit, and three "regulators", usually called gates, that control the flow of information inside the LSTM unit. Specifically, a forget gate F_t, which decides what is relevant to be kept from prior steps, an input gate I_t, which decides what information is relevant to be added from the current step, and an output gate O_t, which determines what the next hidden state h_t should be. Some variations [45], [46] of the LSTM units do not include one or more of these gates, or they have other gates.

The first step of an LSTM unit is to decide what information is thrown away from the cell state. This decision is made by a sigmoid layer called the forget gate layer. Taken into account the h_{t-1}, which is the hidden state vector, commonly referred to as output vector of the LSTM unit, and the current information x_t, and outputs a number between 0 and 1 for each number in the cell state c_{t-1}. 1 represents the completely keep this information, while a 0 represents completely get rid of this. The equation for the forget gate layer is as follows:

$$F_t = \sigma_g(W_F x_t + R_F h_{t-1} + b_F) \tag{4}$$

where σ_g denotes the gate activation function, W_F, R_F and b_F are the learnable weights of an LSTM layer, i.e., the input, the recurrent weights and the bias for the forget layer component, respectively.

The next step consisting of two parts is to decide what new information will be stored in the cell state. First, a sigmoid layer, called the input gate layer, decides which values will be updated and a tanh layer creates a vector of new candidate values, c'_t that could be added to the cell state. The equations describing these components at timestep t are the following:

$$I_t = \sigma_g(W_I x_t + R_I h_{t-1} + b_I) \tag{5}$$

$$c'_t = \sigma_c(W_{c'} x_t + R_{c'} h_{t-1} + b_{c'}) \tag{6}$$

where σ_c denotes the candidate cell state tanh activation function and W_I, R_I, b_I as well as $W_{c'}$, $R_{c'}$ and $b_{c'}$ are the learnable input and recurrent weights, and the bias for the input gate and cell candidate, respectively. These two steps are combined in order to update the old cell state c_{t-1} into a new cell state c_t based on the following equation:

$$c_t = F_t * c_{t-1} + I_t * c'_t \tag{7}$$

Finally, the LSTM unit decides its output. The output id based on the cell state, but might be a filtered version. Firstly, a sigmoid layer, called the output gate layer, decides what parts of the cell state to output. Then, the cell state passes via a tanh, so that the values are re-scaled between -1 and 1 and is multiplied with the output gate layer so that it outputs only the corresponding parts via the hidden state h_t. The equations describing these components are the following:

$$O_t = \sigma_g(W_O x_t + R_O h_{t-1} + b_O) \tag{8}$$

$$h_t = O_t * \sigma_c(c_t) \tag{9}$$

where W_O, R_O, b_O in eq. 8 are the learnable input and recurrent weights and the bias for the output gate. Note that depending on the task, LSTMs could also output a label at each timestep t, but for our application, which outputs a cell-type label given an input timeseries, only one output is calculated at the final timestep.

3.2 Regularization Methods

Although DL architectures are very powerful machine learning systems, they contain a large number of parameters, which makes them quite complex models. As a DNN learns, weights settle into their context within the network. Weights of nodes are tuned for specific features providing some specialization. Nodes of neighboring layers rely on this specialization, which if taken too deep could result in a fragile model too specialized to the training data. This reliant on context for a node during training is referred to complex co-adaptations and can lead to overfitting of the training data, meaning that the network produces over-optimistic predictions throughout the training process, but fails to generalize well on new data leading to a significant drop in its performance.

Dropout [47] is a regularization technique, which is essentially used to prevent overfitting while training ANNs and DNNs. The term dropout refers to dropping out units in a neural network, and thus these units are not considered during a particular forward and backward pass. More specifically, for the 1D-CNNs, at each training stage, individual nodes of a specific layer are either kept with probability *p* or dropped out of the net with probability *1-p*, so that a reduced network is left with the incoming and outgoing edges of the dropped-out node to be removed. Regarding the RNN and LSTM architectures, dropout can be also applied to the recurrent connections of these networks (i.e., the connections related to the hidden states), so that the recurrent weights could be regularized to improve performance. For any of the three architectures, each layer can be associated with a different probability *p*, meaning that dropout can be considered as a per-layer operation with some layers discarding more nodes compared to others dropping nodes with a lower rate or no rate at all.

3.3 Methodology

Our 1D-CNN architecture accepts the input vector of Ca^{2+} signal $x \in R^{1\times 4000}$, which is standard normalized (z-score).

The first convolutional layer of our system consists of 32 filters, while the rest of the layers consist of 62 filters, respectively. For the specific dataset studied, the most optimal depth (see next Section) is 3. Generally, the depth (i.e., number of convolutional layers) as well as the number of filters for the specific timeseries data application is kept small to avoid overfitting. Thus, regularization methods (i.e., dropout) are helpful for our system. Moreover, while in several applications, the size of the filter is 3 or 5, we used a trainable filter $f \in R^{1\times 10}$. Smaller filter sizes applied to our timeseries data are not effective enough, as Ca^{2+} signal has slow kinetics, and the signal is not notably distinguishable in less than 10 timesteps. In contrast, larger filter sizes lead to overfitting. Eventually, at the end of the network, a FC is attached, which receives the output of the previous layer and determines which features are mostly correlated to each one of the 4 classes.

Regarding RNN and LSTM models, they are usually more effective with timeseries data, whose length does not exceed the 100 timesteps in total, since they demand a considerable amount of time in order to be trained. Thus, in cases similar to ours, i.e. with long timeseries data, the following framework is proposed: each timeseries X of length N should be broken into T timesteps with each timestep $x_t \in R^d$, where d is the input dimensionality and $t = 1, ..., T$, such that $N = d \times T$ and $T \ll d$. Namely, each input timeseries X will consist of T timesteps, where each timestep is of dimensionality d. In our case, where each timeseries has a length of $N = 4000$ timesteps, we break each timeseries in $T = 2$, $T = 5$ and $T = 10$ timesteps, where each timestep x_t is of dimensionality $d = 2000$, $d = 800$ and $d = 400$, respectively.

4 Experimental Analysis and Discussion

The DL models that were used in our analysis were implemented using the Tensorflow [48] and Keras open-source libraries written in Python programming language. Both TensorFlow and Keras can perform the calculations on GPUs, and thus the training time is dramatically reduced. For our experiments we used Python version 3.6, the Tensorflow version 1.9 running on NVIDIA GeForce GTX 750 Ti GPU model under Windows 10 operating system.

4.1 Data Set

The data set used to train the classifiers was collected during a goal oriented task in awake, behaving mice [36]. Specifically, head-fixed mice ran on a 2-meter long treadmill belt equipped with a water delivery port (reward location) and the neural signals were recorded using the two-photon Ca^{2+} imaging technique. The mice learned the reward location after training on the belt for a few days. The recordings were obtained from the CA1 region of hippocampus, which is widely known to be involved in spatial memory formation. The data were then processed in order to translate the video recordings into fluorescence signals over time. Four different neuronal types were recorded during the aforementioned task. Namely, the excitatory PY cells, the PV, the SOM and the VIP inhibitory neurons making the problem a four-class classification task. Therefore, our design matrix consists of signals in time (timeseries) of four different neuronal types across all sessions/days and different animals.

4.2 Impact of Regularization and Network's Depth

In this subsection we study how the architecture of each model including the depth and dropout regularization hyper-parameters, affects the performance as well as the training time corresponding to 20 epochs. We present the results in Table 1. We used 3947 examples (1000 PY cells, 1000 SOM cells, 1000 VIP cells and 947 PV cells), where each example is a timeseries that corresponds to a specific neuronal cell-type consisting of 4000 timesteps (we have used the minimum length across all timeseries). We perform 10 random train-test splits, where in every split we use a fixed number of 3157 training and 790 testing examples, which have been z-score normalized based on the mean and standard deviation of the training set. Thus, we report the mean accuracy and training time of the 10 random train-test splits and their corresponding standard deviations. The hyper-parameter values for all the models have been selected after several pre-experiments in order to obtain the best set.

Regarding the architectures of the 1D-CNN model, which are presented in Table 1, each convolutional layer is followed by a ReLU activation function and the pipeline ends up with a FC layer. The optimizer that is used in order to train the network is the Adam gradient-descent optimizer with learning rate 0.001, and $beta_1$ as well as $beta_2$ parameters are 0.9 and 0.99 respectively. The first convolutional layer of each architecture is convolved with 32 filters with kernel size 10 and stride 1, while as we go deeper, from the second convolutional layer and onwards, the inputs to succeeding layers are convolved with 64 filters of kernel size 10 and stride 1. Table 1 shows that the optimal architecture is the fifth one (bolded mean accuracy), which is composed of 2 convolutional layers followed by a max pooling layer of size and stride equal to 2 followed by a last convolutional layer and a dropout layer.

We observe that the architecture consisting of 2 convolutional layers gives a better classification performance compared to the architectures consisting of 1 and 3 convolutional layers, as by using only 1 layer, we create a very shallow network, which cannot be trained properly, while 3 layers lead to overfitting. By using the dropout regularization technique combined with a max pooling layer in order to control overfitting, we observe that the 5^{th} architecture, where the dropout layer is inserted just before the FC layer is the most effective one, as FC layers are more prone to overfitting due to their large number of connections. Training time, as expected, increases by adding more convolutional layers.

Regarding the RNN and LSTM models, each RNN and LSTM layer is followed by a ReLU activation function and the pipeline ends up with a FC layer. Moreover, each RNN and LSTM layer consists of 100 hidden units, which is essentially the dimensionality of the output space. The optimizer that we used in order to train the network is the Adam gradient-descent based algorithm with the same parameters as before.

More specifically, the optimal architecture for the RNN model, as shown in Table 1 is obtained in the case of 2

Models	Depth	Mean Acc.	St. Dev.	Mean Tr. Time (sec.)	St. Dev.
1D-CNN	1 Conv. Layer	0.8368	0.0116	50.2281	0.8775
	2 Conv. Layers	0.8674	0.0116	124.0125	2.7903
	2 Conv. Layers-Max P.-1 Conv. Layer	0.8739	0.0125	155.3015	1.9314
	2 Conv. Layers-Max P.-2 Conv. Layer	0.8746	0.0102	188.139	2.9593
	2 Conv. Layers-Max P.-1 Conv. Layer-Dropout	**0.8867**	0.0119	161.9093	2.3241
	2 Conv. Layers-Max P.-1 Conv. Layer-Dropout-1 Conv. Layer	0.8683	0.0114	197.5203	3.5911
	2 Conv. Layers-Dropout-2 Conv. Layers	0.8394	0.0219	345.664	2.3541
	3 Conv. Layers	0.8502	0.032	226.925	2.5193
RNN 2 timesteps	1 RNN Layer	**0.8205**	0.011	38.6984	0.6726
	2 RNN Layers	0.804	0.03	55.1843	1.1724
	2 RNN Layers-Dropout-1 RNN Layer	0.8067	0.0141	71.5421	0.9292
RNN 5 timesteps	1 RNN Layer	0.8167	0.01627	49.925	0.6941
	2 RNN Layers	0.8065	0.0151	80.2203	1.1572
	2 RNN Layers-Dropout-1 RNN Layer	0.7975	0.0227	109.6156	1.2772
RNN 10 timesteps	1 RNN Layer	0.8026	0.0194	71.8468	0.7897
	2 RNN Layers	0.8024	0.0273	124.6187	0.979
	2 RNN Layers-Dropout-1 RNN Layer	0.7969	0.0133	170.7687	1.6552
LSTM 2 timesteps	1 LSTM Layer	0.7734	0.0166	69.35	0.8836
	2 LSTM Layers	**0.7915**	0.015	102.8062	1.7754
	2 LSTM Layers-Dropout-1 LSTM Layer	0.7897	0.01	141.7281	2.1877
LSTM 5 timesteps	1 LSTM Layer	0.7869	0.0211	91.5953	1.3077
	2 LSTM Layers	0.7822	0.0158	156.3843	2.2983
	2 LSTM Layers-Dropout-1 LSTM Layer	0.7648	0.018	228.5359	3.7028
LSTM 10 timesteps	1 LSTM Layer	0.7911	0.0111	140.9765	1.3509
	2 LSTM Layers	0.7896	0.0171	256.5265	1.333
	2 LSTM Layers-Dropout-1 LSTM Layer	0.6464	0.1118	381.5828	4.6384

TABLE 1
Mean accuracy performance of the DL architectures on neuronal cell-type classification and their corresponding mean training time across 10 random train-test splits.

timesteps for a single RNN layer. Moreover, for all different values of timesteps, the single RNN layer architecture gives better classification results compared to the stacked RNNs (i.e., an RNN with more than one layer), which lead to network overfitting. In order to prevent overfitting, we added a dropout layer, but the performance was not improved (Table 1). We also experimented by adding recurrent dropout layers with and without the dropout layer but the accuracy performance was significantly dropped (i.e., $1\% - 3\%$ lower, data not shown). We observe again that increasing the complexity of the architecture by adding extra RNN layers, the training time is also increased. Regarding the LSTM model, the highest performing architecture is obtained in the case of 2 timesteps with 2 stacked LSTM layers (Table 1). In general, similarly to RNN model, various combinations of dropout and recurrent dropout layers among the LSTM layers did not improve the classification performance.

Fig. 2 demonstrates the normalized confusion matrices corresponding to the optimal architectures of the three models, as reported in Table 1 (given in bolded). We observe that for all models, classification errors concern primarily the VIP cell type, while the PY and PV cells are identified with higher accuracy.

We observe that 1D-CNN is the optimal model for this task (Table 1 and Fig. 2), as it generates the most accurate predictions. This finding suggests that long-term dependencies are unlikely to be significant in the particular scenario. This is because both RNN and LSTM models, which are capable of identifying such dependencies, have a poorer performance than the CNN architectures. The efficacy of 1D-CNN can be attributed to the fact that they take bi-directional temporal information into account, compared to the single-temporal-direction features extracted by RNN and LSTM models.

Notably, increasing the number of timesteps T, and thus automatically reducing the input dimensionality d, the performance accuracy drops for both RNN and LSTM models, which strengthens the aforementioned finding (Table 1). We also tested the extreme case of $T = 4000$ timesteps and $d = 1$, which practically means that we feed the network with a scalar value at each timestep. Performance was very poor, as the maximum classification accuracy that was achieved was around 45% for both models. Thus, we conclude that for these two models a higher input dimensionality is preferable than an unfolding of too many timesteps, which possibly indicates that no significant long-

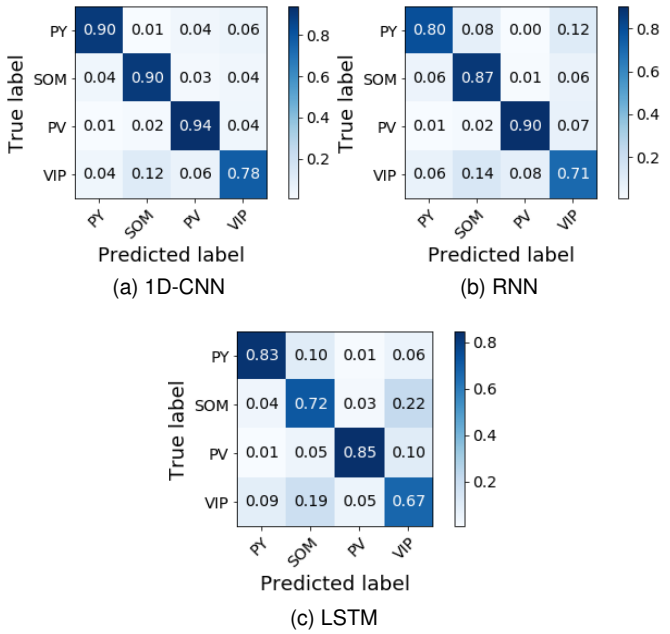

Fig. 2. Normalized confusion matrices for the best-performing 1D-CNN, RNN and LSTM models.

term dependencies exist in the input data. Overall, we observe that the LSTM model has lowest performance than the RNN model, with respect to both classification accuracy and training time. The low performance of LSTM model can be justified from the highest complexity of LSTMs in comparison with RNNs. In addition, as we did not use a large amount of training data, the intrinsic complexity of the LSTM model is probably unnecessary for the modeling of this specific data-set. This is also evidenced by the training accuracy, during the last epochs, while 1D-CNN and RNN models reached a training accuracy of $95\% - 100\%$, the LSTM model achieved a lower training accuracy of $88\% - 97\%$.

We also investigated the impact of the number of training epochs in the accuracy of the tested architectures. Specifically, we trained our best-performing models for 60, and 100 training epochs and compared their performance with the one obtained when 20 epochs are used. For the optimal 1D-CNN architecture, the resulting accuracy was 0.8787 and 0.881, for 60 and 100 epochs respectively, compared to 0.8867 for 20 epochs. This demonstrates that the number of 20 epochs chosen to train the 1D-CNN model is adequate. Similarly, for the best-performing RNN model, the accuracies achieved were 0.8188 and 0.8169 for 60 and 100 epochs, respectively compared to 0.8205 for 20 epochs. Finally, for the best-performing LSTM model, the accuracies achieved were 0.7911 and 0.7894 for 60 and 100 epochs, respectively, compared to 0.7915 for 20 epochs. Overall, this analysis confirmed that using 20 epochs is sufficient for ensuring the optimal performance while avoiding overfitting and larger amount of training time.

Table 2 shows the precision, recall and specificity metrics (for each neuronal cell-type separately) derived from our best-performing model (i.e., the 1D-CNN architecture, whose mean accuracy in Table 1 is in bold).

Metrics	precision	recall	specificity
PY	0.908 (0.05)	0.885 (0.06)	0.967 (0.02)
SOM	0.874 (0.05)	0.913 (0.03)	0.954 (0.02)
PV	0.932 (0.01)	0.916 (0.02)	0.977 (0.006)
VIP	0.819 (0.06)	0.779 (0.05)	0.938 (0.03)

TABLE 2
Mean performance and standard deviation (parentheses) of the best-performing 1D-CNN model for each cell-type

4.3 Impact of the training set size

Unlike other types of datasets used in machine learning like Imagenet, whose size is directly related to the amount of human effort involved in annotating images, the case of biological data offers a significantly more challenging setting. This is due to the fact that annotations, cell types in this case, are obtained though the postmortem biophysical analysis of cells. As such, the availability of training/validation data is substantially more limited compared to other cases. One of the objectives in this paper is also to understand the performance of different DL methods given a relativity limited set of training examples.

Table 3 demonstrates how the size of the training set affects the mean accuracy of the best-performing 1D-CNN, RNN and LSTM models (i.e., models whose mean accuracy in Table 1 is in bold). We used 4 different training set sizes, but in each case we retained a fixed-size testing set of 640 examples for a fair comparison.

For a training set of 2560, we used 640 cells from each category, while for a training set of 3157 we used 800 examples from each category of PY, SOM and VIP, and 757 PV cells. For the dataset with 5137 training examples we used 1600 examples from each of the PY and VIP classes, 1180 SOM cells and 757 PV cells. Finally, in the case of 6737 training examples we used 2400 examples from each of the PY and VIP classes, 1180 SOM cells and 757 PV cells.

We found that differences in the performance across different training set sizes were small for all types of models. Specifically, the largest difference was observed between the 2 extreme cases of 2560 and 6737 training examples and was smaller than $6 - 7\%$. However, this relatively small improvement came with a very high cost of required time for training the models. Indicatively, the mean training times required by the 1D-CNN, given the extreme cases of 2560 and 6737 examples were 132.46 and 361.155 seconds, respectively. The same observation is derived from the other two models, where RNN needs 20.078 and 51.643 seconds and the LSTM 52.257 and 129.973 seconds.

4.4 Comparison with other classifiers

To assess whether our approach is better than other types of popular classifiers, we compared the best-performing 1D-CNN, RNN and LSTM models (models of Table 1 in bold), against the traditional Machine Learning approaches, such as the Adaboost and the Support Vector Machine (SVM) classifiers. In particular, we used SVMs with linear, Gaussian and polynomial kernels. Table 4 corroborates the claim that 1D-CNN is the most efficient algorithm for the

Training Set Size	1D-CNN	RNN	LSTM
2560	0.854 (0.02)	0.792 (0.01)	0.794 (0.01)
3157	0.88 (0.01)	0.829 (0.01)	0.803 (0.01)
5137	0.907 (0.008)	0.833 (0.01)	0.813 (0.01)
6737	0.918 (0.01)	0.868 (0.01)	0.848 (0.009)

TABLE 3
Mean accuracy and standard deviation (parentheses) of the best-performing models for training sets of different size.

Model	Mean Accuracy	St. Dev.
1D-CNN	**0.8867**	0.0119
RNN	0.8205	0.011
LSTM	0.7915	0.015
Linear SVM	0.6168	0.0125
Gaussian SVM	0.8184	0.0108
Polynomial SVM	0.6477	0.0152
Adaboost	0.568	0.01

TABLE 4
Comparison of the best-performing 1D-CNN, RNN and LSTM models against the Adaboost and SVM classifiers.

task at hand. Its main competitor is the Gaussian SVM, which outperforms all other models except the RNN, while Adaboost [49] has the worst performance.

In order to infer the most efficient combination of parameter values for each of the classifiers, we used the GridSearchCV implementation [49], which performs an exhaustive search over the hyperparameter space. The specific hyperparameter related to the Adaboost classifier is the number of estimators at which boosting is terminated. Correspondingly, the hyperparameter related to the Gaussian and polynomial SVMs are the gamma and the penalty parameters. For the polynomial SVM, the degree hyperparameter is also defined, while the hyperparameter related to the linear SVM classifier is the penalty parameter. Gamma is a hyperparameter for the non-linear hyperplanes. The higher its value the harder the classifier tries to fit the training data set. The penalty parameter of the error term controls the trade off between a smooth decision boundary and classifying the training points correctly. Degree is a parameter used with a polynomial kernel and is essentially the degree of the polynomial kernel used to find the hyperplane to split the data.

Using GridSearchCV, we found that the Adaboost classifier works best with 100 estimators (i.e., classifiers). Regarding the Gaussian SVM, gamma equal to 0.0001 and the penalty parameter equal to 100 are the best-performing combinations. For the polynomial SVM the optimal combination of hyperparameter is a polynomial degree equal to 3, and the gamma as well as the penalty parameters are equal to 0.001 and 100, respectively. Eventually, for the linear classifier, the most optimal penalty parameter equals to 10. In general, pre-experiments showed that increasing the values of gamma and the penalty parameters leads to overfitting, as the classifier tries to perfectly fit the training data, while for smaller values, the classifier is not trained properly making more errors during the testing phase.

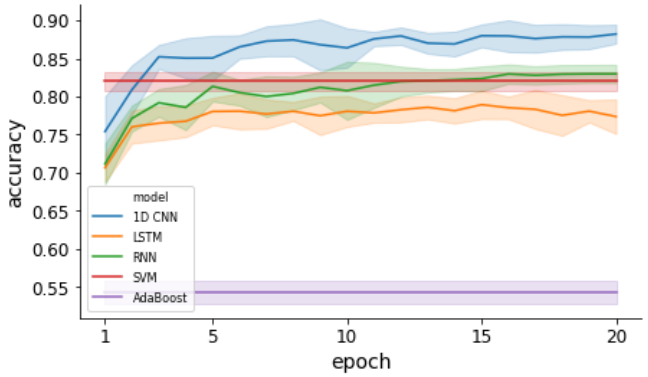

Fig. 3. Comparison of 1D-CNN, RNN and LSTM with traditional machine learning models (Gaussian SVM, AdaBoost). Solid lines denote means, while shaded areas standard deviation across 10 random train-test splits respectively.

Fig. 3 compares the performance of the proposed DL models with traditional machine learning methods, such as the Gaussian SVM and Adaboost. Solid lines denote means, while the shaded areas denote standard deviation for the 10 random train-test splits. We observe that the mean performance of the DL models improves across epochs (having also small standard deviations), until they reach a plateau at 20 epochs or earlier. Thus, while 1D-CNN starts in the first epoch with almost 75% accuracy, it eventually outperforms all other models. Adaboost on the other hand, has the worst performance.

4.5 Impact of the number of timesteps

In this subsection, we assess how the classification accuracy and the training time of the best-performing architectures (Table 1) are affected by the number of timesteps in the input. As expected, the results in Table 5 confirm that decreasing the number of timesteps in the input, results in lower performance accuracy and smaller training time. However, when 2000 timesteps are used, while the classification accuracy (especially for the 1D-CNN model) has a small drop compared to the 4000 timesteps, the training time has a sharp decrease, suggesting that the use of 2000 timesteps is an acceptable option for fast and accurate cell-type classification.

	Mean Acc. & Time (sec.)		
Timesteps	1D-CNN	RNN	LSTM
1000	0.8097 (61.314)	0.7184 (35.793)	0.7246 (90.784)
2000	0.8505 (96.140)	0.7715 (37.95)	0.7651 (94.021)
4000	0.8867 (161.90)	0.8205 (38.698)	0.7915 (102.806)

TABLE 5
Mean accuracy and training time of the best-performing methods for various numbers of timesteps

4.6 Testing the models on a new dataset

To assess the generalization performance of our models, we used a completely new dataset, derived from different animals performing a different behavioral task (removal of

the reward location, random foraging, unpublished results from the Losonczy lab). The new dataset includes 119 time-series of length 2606 timesteps, where 91 of the timeseries corresponded to the activity of SOM interneurons, while the rest of them corresponded to the activity of the PV interneurons.

To ensure equal-length examples, we reduced the length of examples in the first dataset from 4000 to 2606 timesteps. We then re-trained the best-performing models using only the first dataset and tested them on both datasets (i.e., the 210 new examples were added to the 790 testing examples considered in the previous sections). The mean accuracy that was achieved by the 1D-CNN model was 81.65%, while the RNN and LSTM models achieved a mean accuracy of 78.3% and 73.28%, respectively. These results are quite promising given that the datasets are completely different and the task performed by the animals is not identical. They further suggest that discriminatory features of the different cell types may be behavior-independent, thus making our approach an extremely powerful tool for the online classification of cell types in behaving animals.

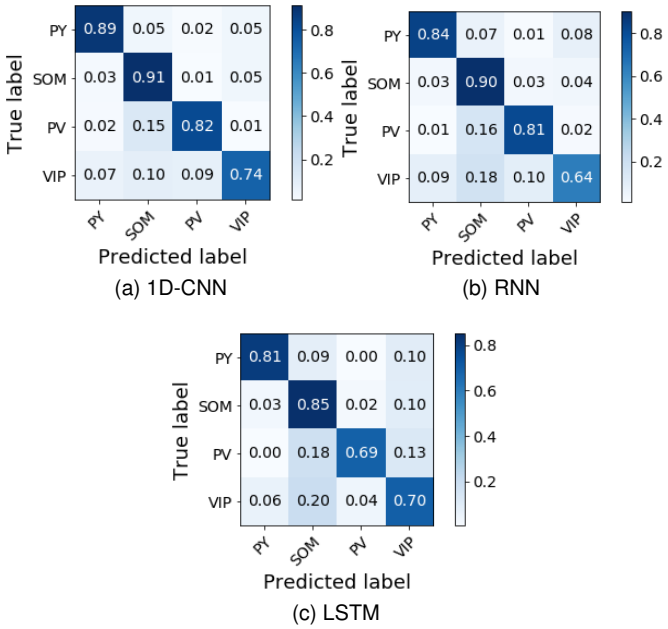

Fig. 4. Normalized confusion matrices of the best-performing architectures of 1D-CNN, RNN and LSTM models tested on a new dataset

As shown in Fig. 4, all models misclassify more often the VIP and PV neurons, which are mainly predicted to be SOM cells. More specifically, the VIP neurons are again misclassified as SOM cells, as in the first dataset (Fig. 2), while now, PV neurons are also misclassified as SOM cells. This could be explained by the small number of the newly added PV cells (just 28), which probably hinders their correct recognition by the classifier, compared to the 191 newly added SOM cells, where there is more information to be exploited.

5 Conclusion

In this work we propose a DL-based formalization for the task of automatic neuronal cell-type classification based on the Ca^{2+} activity signal of neurons in behaving animals. We considered 1D-CNN, RNN and LSTM architectures and showed that these DL network architectures are capable of capturing hidden dynamics/features in the activity signal and therefore to make accurate predictions. We found that 1D-CNN is the optimal classifier for this task, suggesting the absence of significant long-term dependencies in the particular dataset. Our results reveal a great potential for replacing current approaches for cell-type classification (e.g., with molecular markers and/or feature extraction algorithms) with 1D-CNN Ca^{2+} imaging-based classifiers, making neuronal cell-type classification automate.

Moreover, our results set the stage for a deeper investigation of whether automated classification of neuronal subtypes (i.e., basket and axoaxonic cells that are subtypes of of the PV cells) is possible. Our current datasets consist of large families of these subtypes, whose automated discrimination, if possible, will have major contribution to the design of experiments, leading to important resource savings (cost, time, effort and number of animals). Another avenue of application involves the development of a continuous learning ANN. This network should be able to understand when new cell-types are introduced during testing, and adjust its parameters so as to recognize them in a next appearance without requiring to re-train its parameters. These two research perspectives will be investigated in future work.

Acknowledgments

This research is co-funded by Greece and the European Union (European Social Fund-ESF) through the Operational Programme 'Human Resources Development, Education and Lifelong Learning' in the context of the project 'Strengthening Human Resources Research Potential via Doctorate Research' (MIS-5000432), implemented by the State Scholarships Foundation (IKY).

References

[1] A. J. Trevelyan, D. Sussillo, B. O. Watson, and R. Yuste, "Modular propagation of epileptiform activity: evidence for an inhibitory veto in neocortex," *Journal of Neuroscience*, vol. 26, no. 48, pp. 12 447–12 455, 2006.

[2] T. F. Freund and I. Katona, "Perisomatic inhibition," *Neuron*, vol. 56, no. 1, pp. 33–42, 2007.

[3] E. M. Powell, D. B. Campbell, G. D. Stanwood, C. Davis, J. L. Noebels, and P. Levitt, "Genetic disruption of cortical interneuron development causes region-and gaba cell type-specific deficits, epilepsy, and behavioral dysfunction," *Journal of Neuroscience*, vol. 23, no. 2, pp. 622–631, 2003.

[4] P. S. Kalanithi, W. Zheng, Y. Kataoka, M. DiFiglia, H. Grantz, C. B. Saper, M. L. Schwartz, J. F. Leckman, and F. M. Vaccarino, "Altered parvalbumin-positive neuron distribution in basal ganglia of individuals with tourette syndrome," *Proceedings of the National Academy of Sciences*, vol. 102, no. 37, pp. 13 307–13 312, 2005.

[5] K. Tabuchi, J. Blundell, M. R. Etherton, R. E. Hammer, X. Liu, C. M. Powell, and T. C. Südhof, "A neuroligin-3 mutation implicated in autism increases inhibitory synaptic transmission in mice," *science*, vol. 318, no. 5847, pp. 71–76, 2007.

[6] V. S. Dani, Q. Chang, A. Maffei, G. G. Turrigiano, R. Jaenisch, and S. B. Nelson, "Reduced cortical activity due to a shift in the balance between excitation and inhibition in a mouse model of rett syndrome," *Proceedings of the National Academy of Sciences*, vol. 102, no. 35, pp. 12 560–12 565, 2005.

[7] G. Gonzalez-Burgos and D. A. Lewis, "Gaba neurons and the mechanisms of network oscillations: implications for understanding cortical dysfunction in schizophrenia," *Schizophrenia bulletin*, vol. 34, no. 5, pp. 944–961, 2008.

[8] D. A. Lewis, T. Hashimoto, and D. W. Volk, "Cortical inhibitory neurons and schizophrenia," *Nature Reviews Neuroscience*, vol. 6, no. 4, p. 312, 2005.

[9] C. R. Muratore, C. Zhou, M. Liao, M. A. Fernandez, W. M. Taylor, V. N. Lagomarsino, R. V. Pearse II, H. C. Rice, J. M. Negri, A. He *et al.*, "Cell-type dependent alzheimer's disease phenotypes: probing the biology of selective neuronal vulnerability," *Stem cell reports*, vol. 9, no. 6, pp. 1868–1884, 2017.

[10] N. G. Skene and S. G. Grant, "Identification of vulnerable cell types in major brain disorders using single cell transcriptomes and expression weighted cell type enrichment," *Frontiers in neuroscience*, vol. 10, p. 16, 2016.

[11] H. Zeng and J. R. Sanes, "Neuronal cell-type classification: challenges, opportunities and the path forward," *Nature Reviews Neuroscience*, vol. 18, no. 9, p. 530, 2017.

[12] G. Maccaferri and J.-C. Lacaille, "Interneuron diversity series: hippocampal interneuron classifications–making things as simple as possible, not simpler," *Trends in neurosciences*, vol. 26, no. 10, pp. 564–571, 2003.

[13] S. A. Booker and I. Vida, "Morphological diversity and connectivity of hippocampal interneurons," *Cell and tissue research*, vol. 373, no. 3, pp. 619–641, 2018.

[14] L. Guerra, L. M. McGarry, V. Robles, C. Bielza, P. Larrañaga, and R. Yuste, "Comparison between supervised and unsupervised classifications of neuronal cell types: a case study," *Developmental neurobiology*, vol. 71, no. 1, pp. 71–82, 2011.

[15] X. Vasques, L. Vanel, G. Villette, and L. Cif, "Morphological neuron classification using machine learning," *Frontiers in neuroanatomy*, vol. 10, p. 102, 2016.

[16] L. M. McGarry, A. M. Packer, E. Fino, V. Nikolenko, T. Sippy, and R. Yuste, "Quantitative classification of somatostatin-positive neocortical interneurons identifies three interneuron subtypes," *Frontiers in neural circuits*, vol. 4, p. 12, 2010.

[17] G. A. Ascoli, L. Alonso-Nanclares, S. A. Anderson, G. Barrionuevo, R. Benavides-Piccione, A. Burkhalter, G. Buzsáki, B. Cauli, J. DeFelipe, A. Fairén *et al.*, "Petilla terminology: nomenclature of features of gabaergic interneurons of the cerebral cortex," *Nature Reviews Neuroscience*, vol. 9, no. 7, p. 557, 2008.

[18] C. Stosiek, O. Garaschuk, K. Holthoff, and A. Konnerth, "In vivo two-photon calcium imaging of neuronal networks," *Proceedings of the National Academy of Sciences*, vol. 100, no. 12, pp. 7319–7324, 2003.

[19] A. Birkner, C. H. Tischbirek, and A. Konnerth, "Improved deep two-photon calcium imaging in vivo," *Cell calcium*, vol. 64, pp. 29–35, 2017.

[20] S. L. Resendez and G. D. Stuber, "In vivo calcium imaging to illuminate neurocircuit activity dynamics underlying naturalistic behavior," *Neuropsychopharmacology*, vol. 40, no. 1, p. 238, 2015.

[21] W. Gobel and F. Helmchen, "In vivo calcium imaging of neural network function," *Physiology*, vol. 22, no. 6, pp. 358–365, 2007.

[22] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light gated recurrent units for speech recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 92–102, 2018.

[23] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Advances in neural information processing systems*, 2008, pp. 577–584.

[24] J.-C. Hou, S.-S. Wang, Y.-H. Lai, Y. Tsao, H.-W. Chang, and H.-M. Wang, "Audio-visual speech enhancement using multimodal deep convolutional neural networks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 117–128, 2018.

[25] K. Zheng, W. Q. Yan, and P. Nand, "Video dynamics detection using deep neural networks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 3, pp. 224–234, 2017.

[26] L. Gao, Z. Guo, H. Zhang, X. Xu, and H. T. Shen, "Video captioning with attention-based lstm and semantic consistency," *IEEE Transactions on Multimedia*, vol. 19, no. 9, pp. 2045–2055, 2017.

[27] Q. Chen, W. Wang, F. Wu, S. De, R. Wang, B. Zhang, and X. Huang, "A survey on an emerging area: Deep learning for smart city data," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2019.

[28] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[29] M.-C. Su, C.-Y. Cheng, and P.-C. Wang, "A neural-network-based approach to white blood cell classification," *The scientific world journal*, vol. 2014, 2014.

[30] R. Tomari, W. N. W. Zakaria, M. M. A. Jamil, F. M. Nor, and N. F. N. Fuad, "Computer aided system for red blood cell classification in blood smear image," *Procedia Computer Science*, vol. 42, pp. 206–213, 2014.

[31] R. Geetha, S. Sivasubramanian, M. Kaliappan, S. Vimal, and S. Annamalai, "Cervical cancer identification with synthetic minority oversampling technique and pca analysis using random forest classifier," *Journal of medical systems*, vol. 43, no. 9, p. 286, 2019.

[32] C. Shah and A. G. Jivani, "Comparison of data mining classification algorithms for breast cancer prediction," in *2013 Fourth international conference on computing, communications and networking technologies (ICCCNT)*. IEEE, 2013, pp. 1–4.

[33] A. Kepecs and G. Fishell, "Interneuron cell types are fit to function," *Nature*, vol. 505, no. 7483, pp. 318–326, 2014.

[34] K. A. Pelkey, R. Chittajallu, M. T. Craig, L. Tricoire, J. C. Wester, and C. J. McBain, "Hippocampal gabaergic inhibitory interneurons," *Physiological reviews*, vol. 97, no. 4, pp. 1619–1747, 2017.

[35] J. DeFelipe, P. L. López-Cruz, R. Benavides-Piccione, C. Bielza, P. Larrañaga, S. Anderson, A. Burkhalter, B. Cauli, A. Fairén, D. Feldmeyer *et al.*, "New insights into the classification and nomenclature of cortical gabaergic interneurons," *Nature Reviews Neuroscience*, vol. 14, no. 3, pp. 202–216, 2013.

[36] G. F. Turi, W.-K. Li, S. Chavlis, I. Pandi, J. OHare, J. B. Priestley, A. D. Grosmark, Z. Liao, M. Ladow, J. F. Zhang *et al.*, "Vasoactive intestinal polypeptide-expressing interneurons in the hippocampus support goal-oriented spatial learning," *Neuron*, vol. 101, no. 6, pp. 1150–1165, 2019.

[37] R. Scorcioni, M. T. Lazarewicz, and G. A. Ascoli, "Quantitative morphometry of hippocampal pyramidal cells: differences between anatomical classes and reconstructing laboratories," *Journal of Comparative Neurology*, vol. 473, no. 2, pp. 177–193, 2004.

[38] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[39] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[40] J. Ebrahimi and D. Dou, "Chain based rnn for relation classification," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 1244–1249.

[41] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, 2013, pp. 1310–1318.

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with lstm recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3547–3555.

[44] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[45] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[46] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015, pp. 802–810.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.