Final Report PERS 2021: Application of Machine Learning Methods to Identify Piezo Ion Channel Kinetics

Renato Rios (student) and Jun Xu (faculty advisor)

SECTION I – INTRODUCTION

Patch-clamp electrophysiology data is vital for understanding ion channel behavior but is laborious to idealize for analysis in research. Celik et al. demonstrated that convolutional neural networks and long short-term memory architecture can automatically idealize complex single molecule data more quickly and accurately than traditional methods.[1] Although a basic understanding of machine learning, Python, matplot library, Keras, and Tensorflow is required, the simplicity of the deep learning model developed by Celik et al. requires simply updating the correct filename/path. In this preliminary study of the work presented by Celik et al. we attempt to recreate the (1) trained model development and (2) model prediction performance using a new unseen dataset using source code and recordings/data provided by the original authors. The recreation of semi-synthetic datasets described in the original paper was omitted due to equipment availability and time limitations. Utilizing datasets publicly available on GitHub, the ion channel data used included single ion channel recordings and five ion channel recordings. Even with limited background knowledge in machine learning and python programming, it is possible to train and validate models using the methods described by Celik et al. using the .csv files provided. This model requires no parameters set by researchers, as is required in traditional methods of ion channel classification, and requires less data processing steps as it only necessitates the data filename and path. This methodology may allow researchers to save time and resources by simplifying the data idealization, automatically specifying the number or channels present, and by possibly reducing the number of samples for analyzing patch-clamp electrophysiology data in ion channel research.

SECTION II – BACKGROUND

Patch-clamp electrophysiology techniques developed by E. Neher and B. Sakmann, produce functional electrical current data that was found to describe the role of ion channels in producing nerve action potentials.[2] Utilizing the ion channel data captured from these recordings requires idealization of the raw electrical recordings, which can be "noisy" and a time consuming to process. Multiple samples may even be required to obtain useable data for analysis as described by the original authors. Celik et al. proposes a hybrid recurrent convolutional neural network (RCNN) model to idealize ion channel records. Convolutional neural networks are used in deep learning for learning patterns given complex data and classification problems. Recurrent neural networks, as described in the original paper, can be used for time series analyses but performance degrades on longer time scales in a problem known as the vanishing gradient problem. Celik et al. utilize long short-term memory (LSTM) networks, a type of recurrent neural network, in the presented model to address the vanishing gradient problem. To train the deep learning model, the authors developed an analogue synthetic ion channel record generator system and used these semi-synthetic records to train and validate the neural model. This model is the first that can idealize ion channel records of up to five simultaneous ion channels events concurrently.

Here we attempt to train a deep learning model as outlined in the original paper, and then utilize it to analyze previously unseen data using ion channel records provided by the original authors. In essence, the main goal here is to recreate the model development stage to apply said model using ion channel recordings of new, less studied ion channels in future works. As previously mentioned, the source code for the model was acquired from GitHub and has been made fully available online. We chose to use the single ion channel data and the five ion channel data from this depository to attempt to replicate a similar

qualitative & performance comparison of a single ion channel and five ion channel data shown by Celik et al. in their original paper in addition to describing this process in detail.

SECTION III – SOFTWARE AND HARDWARE

Utilization of the deep-channel model requires correct configuration of a virtual environment on the Windows or Linux system. The model used here requires Keras framework with Tensorflow backend. Tensorflow is a popular end-to-end open-source platform for machine learning & Keras is a deep learning API written in Python, running on top of Tensorflow. Several computers were used in an attempt to create the Python environment as Tensorflow is not supported on an ARM based processors as attempted initially. After some failed attempts, using a Windows 10 computer gave a successful run. Additionally, this source code was also successfully run by downloading and utilizing Ubuntu, a popular Linux distribution system, on a Windows 10 computer from which a fully trained model was obtained. Later, however, Anaconda on Windows 10 was utilized for the rest of the work due its ease of use and interface when compared to using Ubuntu. Anaconda is a distribution system of the Python and R programming languages for scientific computing including data-science packages suitable for Windows, Linux, and macOS. The code was then edited and run on Spyder, a free and open-source scientific environment written in Python, for Python, and designed by and for scientists, engineers, and data analysts. There are a variety of ways to install these software, the order in which the programs were installed is discussed in detail.

Anaconda was first installed directly from the official website following the provided instructions. The system used was a 64-bit operating system with x64-based processor PC running Windows 10. The PC ran on an 11th Gen Intel® Core™ i7-1165G7 with only 16.0 GB of RAM and no GPU. Once Anaconda

was successfully installed, the software was updated to the newest available version at the time, Anaconda Navigator version 2.0.4 specifically.

To begin creating the virtual environment, it is possible to use the command line and command prompts. There are several issues that can arise from utilizing this method, so the built-in environment menu available in Anaconda was used to create the environment by choosing the Python 3.6 package. It is important to note that additional packages, such as Tensorflow and Keras, are not downloaded at this point in the process. These are to be downloaded after the environment is created and all the associated Python 3.6 packages have downloaded. At this point it's recommended to download all research files directly from GitHub. One can also install the dependencies from this repository using the *'environment.yml'* file provided, but there were issues with versions of certain packages that were not compatible with newer packages of other dependencies. To circumvent this issue, we opted to install packages as necessary when working with the different files using pip, a package manager for Python. This tool allowed for installation of additional libraries and dependencies that we not included when initially creating the python environment using the built-in environment menu in Anaconda. To determine which packages were required, we ran the '*deepchannel_train.py'* file downloaded from GitHub by launching Spyder under the new virtual Python 3.6 environment and downloaded the required or missing packages by using pip command "pip install" followed by the missing package/dependency in the Spyder console. The version of Spyder used here was version 5.0.5, the newest available at the time of running this project. As this project depends on utilization of Tensorflow and Keras for the machine learning development, pip was also used to properly install both. The following installed here are *keras-nightly* 2.5.0.dev2021032900, *tensorflow* 2.5.0, and *tensorflow-addons* 0.13.0. Once the environment was completely built, it is possible proceed to the model development stage.

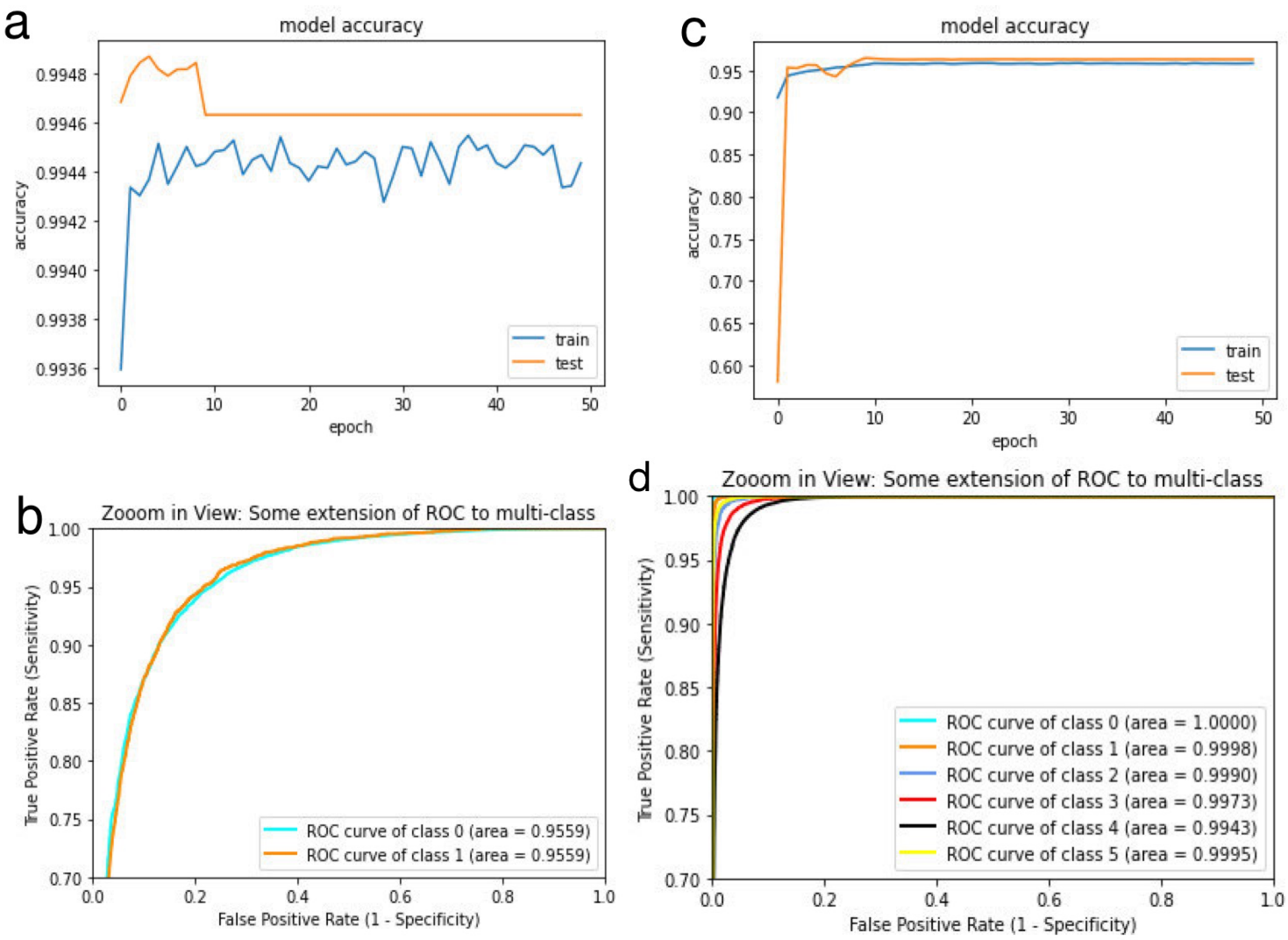

Fig 1. Model Training Metrics. a Single Channel Model Accuracy. **b** ROC and AUC for Single Channel Model Training for 0 and 1 channels. **c** Multi-Channel Model Accuracy. **d** ROC and AUC for Multi-Channel Model Training for 0 -5 channels.

SECTION IV – MODEL BACKGROUND & DEVELOPMENT

To develop any deep learning model, the data used is instrumental in influencing the results and performance of the model. To generate quality data to train the model in a manner that would work on genuine recordings, Celik et al. developed an approach to simulate idealized records using stochastic,

Markovian methods described by Gillespie.[3] To also simulate the authenticity of traditional "noisy" recordings, the signals were passed through a patch-clamp amplifier and recording it using CED's Signal software via Axon electronic "modal cell" as described in more detail by the original authors. This gave the signal authentic "electrophysiological" noise. This data contained the raw current data, ground truth labels from the stochastic model (idealization), and fiducial record/ground truth that labeled the number (0-5) of open channels open at a given time. The process in which the semi-synthetic data was created and validated is explained in more detail in the original work and it was these sets of data that were used in here for model training.

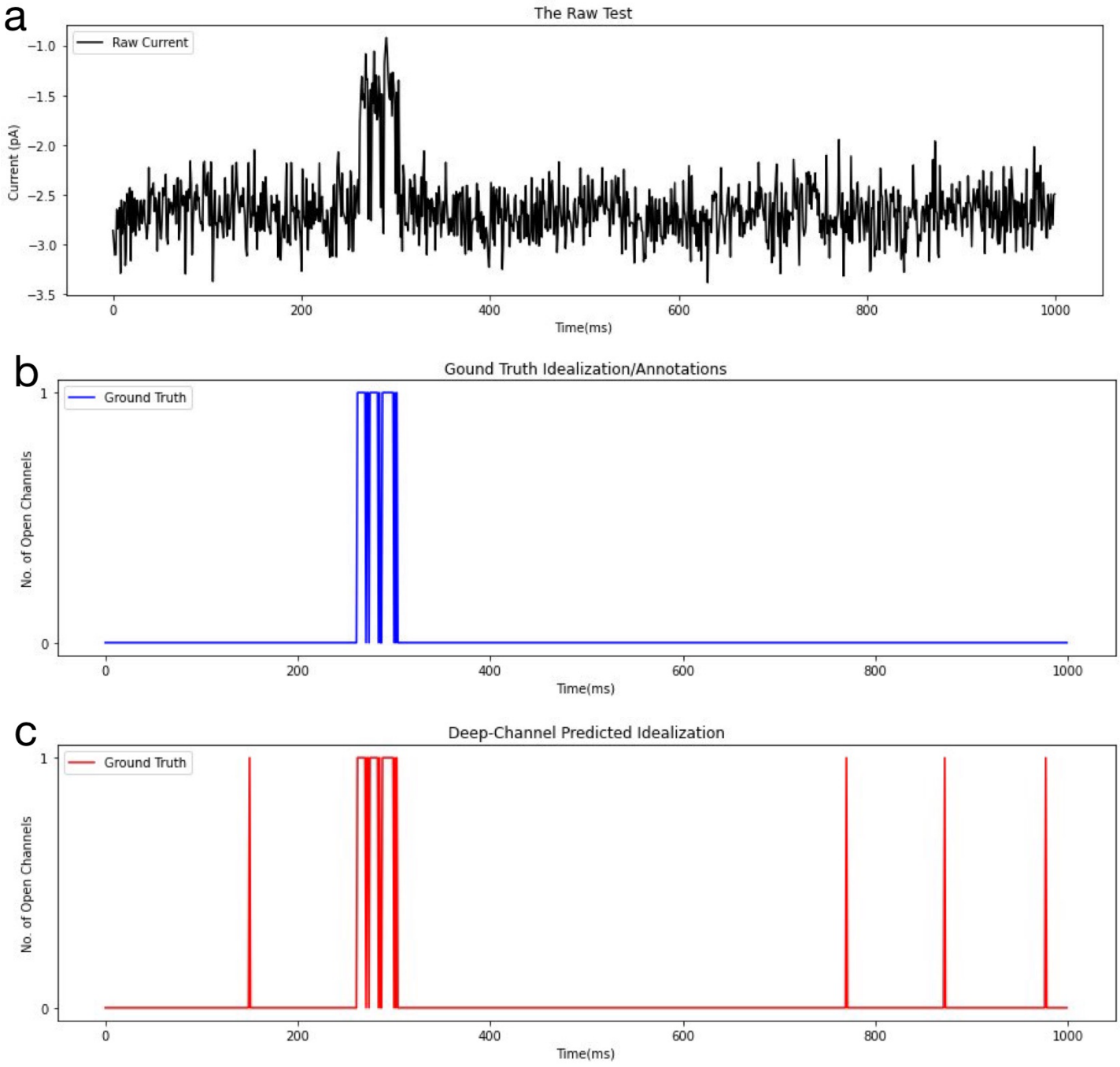

Fig 2. Qualitative performance Deep-Channel with previously unseen data for single ion channel model. a-c Representative example of Deep-Channel performance using single ion channel data training **a** The raw semi-simulated ion channel event data (black). **b** The ground truth/idealization annotation labels (blue) from the raw data above in (**a**). **c** The Deep-Channel predictions (red) for the raw data above (**a**).

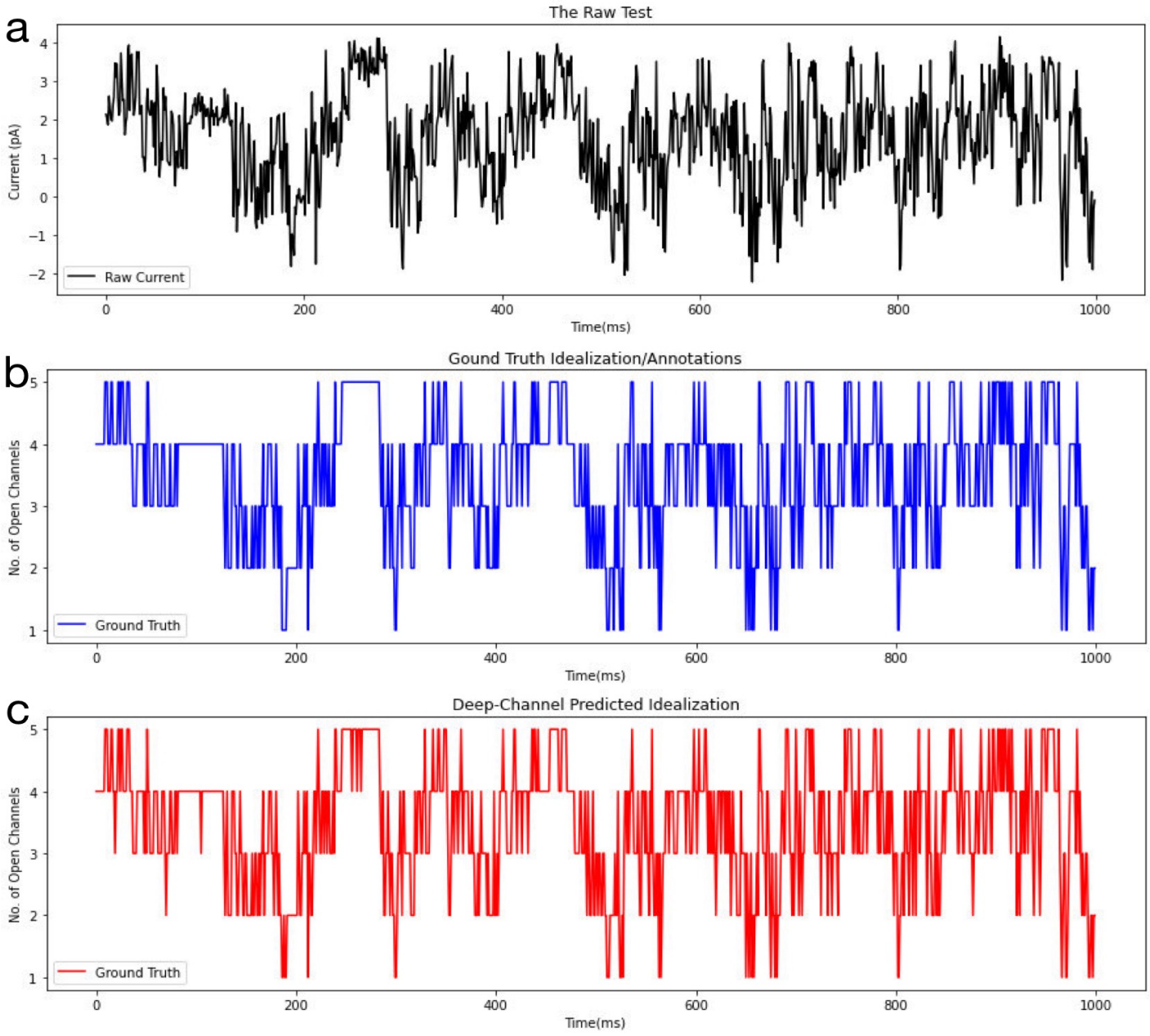

Fig 3. Qualitative performance Deep-Channel with previously unseen data for a five ion channel model. a-c Representative example of Deep-Channel performance using five ion channels training data **a** The raw semi-simulated ion channel event data (black). **b** The ground truth/idealization annotation labels (blue) from the raw data above in (**a**). **c** The Deep-Channel predictions (red) for the raw data above (**a**).

To begin training a model, we first made sure to already have downloaded the code and files from GitHub. The file *'deepchannel_train.py'* was opened and ran on Spyder by first opening Anaconda as described previously. At this point, it is important to select the appropriately built environment that should have been set up as previously described. This file trains the Deep-channel model based on LSTM-convolution layers using the training set and evaluates the model on a different test set. In order to train and validate the model, a dataset for training and a dataset for validation is required. The training data is split 80% training and 20% testing, and then validated (tested) using a different file. The file paths for the appropriate training and validation files were updated accordingly in the code of *'deepchannel_train.py'*. Additionally, the specifications of the training stage were changed to match that described in the original paper. The training specifications used here are as follows; batch sizes were set at 256, initial learning rate was set at 0.001, momentum set at 0.9, trained for 50 epochs. In the original work, 17 experiments were carried out with previously unseen data. Due to the limited number of data sets available, we limited the model training and exposure to a previously unseen dataset to one ion channel data and five ion channel data generated models. For the single ion channel testing *'outfinaltest10.csv'* was used, and for validation *'outfinaltest78.csv'* was used. For the multi-channel testing *'outfinaltest328.csv'* was used, and for validation *'outfinaltest534.csv'* was used. Before running we updated the correct file path/names for the files used. Additionally, we updated the name the created model would have prior to running the testing stage. It is important to do this so as to not save over a previously created model with the same name. The model generated is saved as a hierarchical data format file HDF5.

In Fig. 1 we show the resultant graphs obtained from a *single* ion channel training run and from a *five* ion channel training run. In both cases, similar results were obtained in comparison to those published be Celik et al., as one can note the high receiving operating characteristics of (ROC) of both the single and five channel training results. The high ROC curves plot two parameters, true positive rate, and false

positive rate. The true positive (TP) rate represents the proportion of positive samples that are correctly predicted, and the false positive (FP) rate represents the proportion of actual negatives the model predicted incorrectly. Here the changes in model accuracy are tracked across the entire 50 epochs. The model accuracy is a metric used in evaluating classifications models and is a fraction of predictions that the model got correct. The metrics used in determining the efficacy and performance of deep learning models include accuracy, precision, recall, and F-score. All of which rely on true positives and false positives. A perfect classifier would have a precision value and recall value of 1. In Fig. 1 the area under the receiver operating curve (AUC) is also included and is shown to be very close to a value of 1. The area under the ROC is typically measured from 0.5 (useless model) to 1.0 (perfect classifier). The equations that describe precision, recall, and F-score are shown below.

$$Accuracy = \frac{True\ Positives+True\ Negatives}{True\ Positives+True\ Negatives+False\ Positives+False\ Negatives} \quad (1)$$

$$Precision = \frac{True\ Positives}{True\ Positives-False\ Positives} \quad (2)$$

$$Recall = \frac{True\ Positives}{True\ Positives-False\ Negatives} \quad (3)$$

$$FScore = \frac{2 \times Precision \times Recall}{Precision+Recall} \quad (4)$$

After a successful training stage for both single and multi-channels, the respective models were then tested on previously unseen data. For this, '*predictor.py'* was opened in the same virtual environment as previously used. Here it is important to correctly update the file path for the previously generated model so the correct model can be tested. The file path/name for the previously unseen data must also be updated. In this file, however, the file path/name must be updated to accommodate testing for the Deep Channel (DC) model, QuB SKM, and QuB half-amp (50% threshold-crossing). The files used for this were

Table 1. Classification Reports for Single Channel Models on Previously Unseen Data

classification report of DC:

	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	0.94	0.99	0.97
accuracy			1.00
macro avg	0.97	0.99	0.98
weighted avg	1.00	1.00	1.00

classification report of QuB SKM:

	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	0.94	0.98	0.96
accuracy			1.00
macro avg	0.97	0.99	0.98
weighted avg	1.00	1.00	1.00

classification report QuB half-amp:

	Precision	Recall	F1-score
0	1.00	0.99	1.00
1	0.90	0.99	0.95
accuracy			0.99
macro avg	0.95	0.99	0.97
weighted avg	0.99	0.99	0.99

Table 2. Classification Reports for Multi-Channel Models on Previously Unseen Data

classification report of DC:

	Precision	Recall	F1-score
0	0.96	0.95	0.95
1	0.97	0.96	0.96
2	0.95	0.98	0.96
3	0.96	0.97	0.96
4	0.93	0.97	0.95
5	1.00	0.87	0.93
accuracy			0.95
macro avg	0.96	0.95	0.95
weighted avg	0.95	0.95	0.95

classification report of QuB:

	Precision	Recall	F1-score
0	1.00	0.57	0.72
1	0.95	0.81	0.88
2	0.95	0.92	0.93
3	0.96	0.96	0.96
4	0.96	0.98	0.97
5	0.98	0.98	0.98
accuracy			0.96
macro avg	0.97	0.87	0.91
weighted avg	0.96	0.96	0.96

classification report of QuB half-amp:

	Precision	Recall	F1-score
0	1.00	0.88	0.93
1	0.98	0.90	0.94
2	0.97	0.93	0.95
3	0.96	0.96	0.96
4	0.96	0.97	0.97
5	0.95	0.99	0.97
accuracy			0.96
macro avg	0.97	0.94	0.95
weighted avg	0.96	0.96	0.96

'outfinaltest44.csv', *'outfinaltest44_SKM.csv'*, and *'outfinaltest44_halfamp.csv'* for the single channel. The files used for the multi-channel run were *outfinaltest747.csv'*, *'outfinaltest747_SKM.csv'*, and *'outfinaltest747_halfamp.csv'*. Quantify Unknown Biophysics QuB is used to explore the dynamics of hidden states in a memoryless systems and simulates outputs from a models.[4] QuB has been used for to solve problems with proteins, molecular motors, and ion channel research. In Fig. 2 and Fig. 3 we demonstrate the obtained results which show a qualitative performance of Deep-Channel (DC) with previously unseen data. Additionally, in Table 1 and Table 2 we show the obtained classification reports for the DC model, QuB SKM, and QuB half-amp for single channel and multi-channel data respectively. Here we can see that the DC model performs very closely to QuB SKM and QuB half-amp. Although the results from the multi-channel data performance of DC is slightly lower, the single channel performance is slightly better than that of QuB half-amp. Although we were not able to compare the performance of the model to the extent the original paper did due to a lower number of available recordings, these results demonstrate the efficacy of the model for single and multi-channel ion classification.

SECTION V – CONCLUSION & FUTURE DIRECTION

Here the results from the model training stage and testing on a previously unseen dataset result in similar results as those presented by the original paper. Looking at the qualitative performance of the DC model with previously unseen data shown in Fig. 2 and Fig. 3 highlight the similarities between the classification of DC in comparison to the ground truth idealization demonstrates the efficacy of the DC model. Additionally, the quantifiable comparison between DC, SKM, and half-amp shown in Table 1 and Table 2 demonstrates that the performance of DC is on par with that of previous models. More data availability is required to truly compare the models as was done in the original paper, but these preliminary results are encouraging, nonetheless.

The model development stage and testing on previously unseen data requires some background in Python, machine learning, and basic debugging skills. Future vision stemming from this current work involves utilizing additional recordings and recreating larger sections of the methodology outlined in the work by Celik et al. and extending use of this the Deep-Channel model to other ion channel types while testing efficacy and performance. Demonstrating the efficacy and application of this model to other studies may allow researchers to save time by utilizing this easier to use model that requires less parameters to set and manage. This investigational study demonstrates similar performance to that outlined by Celik et al. in their original work and results are encouraging with respect to the future application of this model.

REFERENCES

1. Celik, N., O'Brien, F., Brennan, S. et al. Deep-Channel uses deep neural networks to detect single-molecule events from patch-clamp data. *Nature* 3, 3 (2020). https://doi.org/10.1038/s42003-019-0729-3
2. Neher, E. & Sakmann, B. Single-channel currents recorded from membrane of denervated frog muscle fibres. *Nature* 260, 799–802 (1976).
3. Gillespie, D. T. Exact stochastic simulation of coupled chemical-reactions. *J. Phys. Chem*. 81, 2340–2361 (1977).
4. Nicolai, C. & Sachs, F. Solving ion channel kinetics with the QuB software. *Biophys*. *Rev. Lett*. 8, 191–211 (2013).