Parkinsons-FOG Net

A Deep Network for Freezing of Gait Analysis

Hooman Ramezani

Abstract

Freezing of Gait affects the mobility of individuals with Parkinson's Disease. This paper investigates the deep learning models explored and implemented to address the problem of predicting the occurrence of FOG given a time series array of biometric data. The dataset used for training was the Multimodal Dataset of Freezing of Gait in Parkinson's Disease provided by Mendeley Data. From these various bio-signals (EEG, EMG, ECG, skin conductance), a InceptionTime model was selected from a variety of other options and implemented using the TSAI deep learning library to yield a test accuracy of 94 % with a timestamp batch of 25 frames. Optimizations were implemented and performance was compared with parameters such as dropout and learning rate. The methods for selecting the deep learning model as well challenges faced will be investigated in this report.

Introduction

Problem Space

People diagnosed with Parkinson's Disease (PD) are characterised by having unintended and uncontrollable movements, overall encompassing stiffness and shakiness and issues with mobility [1]. One of these specific symptoms that can occur with people living with PD, is Freezing of Gait, where patients will have a short period of time where they are unable to move their feet despite having the intention to walk, essentially becoming stuck in place [2]. This poses a significant risk to individuals who experience Freezing of Gait, as they are at an extremely high risk for falling and sustaining major injuries throughout their body, as the body movement shuts down [2]. The goal is to develop a method to quickly detect the occurrence of Freezing of Gait within an individual in a non-invasive manner, such that others can provide assistance immediately once detected. With the potential of classifying when Freezing of Gait occurs, it is possible to reduce the amount of injuries that a person with Parkinson's Disease, as well as return their independence to them.

Literature Review

Extensive review on previous works was conducted prior to the development of the model. Several previous studies have leveraged different types of deep learning approaches and networks attempting to classify instances of Freezing of Gait within patients. In many previous works, sensor data from ECG signals, EMG signals, 3D motion sensors, foot pressure sensors, and Inertial Measurement Units (IMUs) were all collected to form a dataset to detect FoG occurrences [3] [4] [5]. Two groups, one from the University of Tennessee, and a paper from the Journal of NeuroEngineering and Rehabilitation follow similar methodologies of tasking a patient to walk a specific route while measuring data from attached sensors [3] [5].

The paper from the University of Tennessee experimented with a group of classifiers to determine methods to improve the recognition of FoG in individuals [3]. Their work has shown that a Classifier Bagging model consisting of KNN, SVM, and MLP classifiers outperformed CNNs for detecting FoG [3]. Other approaches that were mentioned in separate papers proposed using CNNs, DNNs, and MLP classifiers, all of which were deemed suitable to detect Freezing of Gait [4].

Another paper proposed the use of Long Short Term Memory deep learning methods to predict the occurrence of FoG [5]. This approach primarily focuses on foot pressure measurements in the detection of FoG, and found that the LSTM model with a multi-layer network performed the best for FoG detection [5]. LSTM models have the property of being able to predict current events based on previous events, making it ideal for time based datasets and helpful for FoG classification [4].

Due to the dataset type provided for the generation of this model [6], research was performed to determine methods tried before with similar types of data. Additionally, due to the nature of the problem requiring time-series analysis, research was required into determining valid methods for deep learning with a time dimension.

Methodology

Dataset Description

A public dataset regarding the detection of Freezing of Gait using physiological data was released in late 2022 [7]. This dataset contains readings from various sensors reading electroencephalogram (EEG), electromyogram (EMG), gait acceleration (ACC), and skin conductance (SC) from 12 patients with Parkinson's Disease, as well as if Freezing of Gait is detected within a specific timeframe. The schema of the data contains 1 column for the timestamp in which the measurement occurs, 25 columns of EEG signals, 5 columns of EMG / ECG / Electrooculogram signals, 28 columns of acceleration data, and 1 column for the label whether Freezing of Gait is present.

12 patients with Parkinson's Disease participated in the study, performing two types of walking tasks performed twice for a total of 4 tasks. Tasks 1 and 2 involved patients to walk down a corridor while avoiding obstacles, performing a detour, and returning to their starting point. Tasks 3 and 4 involved patients walking and making a 180 degree turn within a small space, and returning to their starting position. From the 12 patients, Patient 8 performed the experiment twice, and has two separate readings. The filtered readings were provided in .txt files, separated by patient and then separated by task.

Data Cleaning

Data processing on the given dataset is required. The EMG signals given by the dataset had the ordering of the columns in a different order for each patient. Standardising the order of the columns of the data is essential to avoid misreadings of values when it is fed into a network, where this avoids needing to switch the readings for each patient within the entire dataset. As the goal of the detection model is to detect the presence of Freezing of Gait in a non-invasive way such that the patient can perform daily tasks, EEG signals were required to be removed as they require specialised tools to receive data, and that the patient is unlikely to continuously wear the EEG reader tools.

Upon further inspection of the dataset, it was discovered that Patient 4 and Patient 9 had additional tasks within the dataset. This does not correspond with the predefined tasks listed in the paper, thus these 3 sets of data should be removed to keep it consistent with the data files from the other patients. It was also noted that Patient 8 performed on two different days, thus having two sets of data for each of the 4 tasks. It was mentioned that the given data are considered valid results. As these results are all valid, the second trial of Patient 8 was dropped to avoid the model learning characteristically from Patient 8, and to ensure that data among all of the patients are considered relatively equal.

Dataset Preparation

When splitting the dataset we determined that a patient could not be both in the training set and testing set. If a patient was in both sets, the algorithm would detect factors from the data that are specifically related to those patients, and use those trends and patterns unique to an individual to apply to other patients. This results in the algorithm detecting false patterns and a lower accuracy, as well as potentially having an inflated test accuracy.

Implementation with Time Series Artificial Intelligence (TSAI)

The deep learning models were built utilising a Python package named TSAI, Time Series Artificial Intelligence. TSAI is a cutting edge deep learning package built on top of PyTorch and fastai, and is focused on training and deploying time-series models using state-of-the-art techniques []. The framework is capable of deploying classification, regression, and forecasting models with a wide variety of architectures. This framework was chosen in order to perform classification on the Multimodal FOG dataset due to its high level of documentation, abstraction of the intricacies of setting up time-series training loops, and wide variety of models that could be implemented. Additionally, the framework contains a set of helpers for visualisation and analysis of trained models.

With the library imported into a Python file, data loaded into the TimeSeriesDataset class of the framework with stratified, random train / validation splits. A special data augmentation was required in order to perform this step, which allowed for the use of the framework. TSAI requires data formatted in a 3D array of format samples x features x timesteps in order to be fed into models. Our dataset provided is formatted as a 2D array of features x timesteps with each timestep having its own label. In order for it to be compatible with the framework we batched our dataset into small sequences of n timesteps. Each batch was stacked together and fed as a sample to the framework. However in order for this to be done it needed to be refactored so each sample would receive its own label, where it was currently set up for each timestep to have a label. Thus, each sample was given a label based on the makeup of its labels per timestep. If a sample had a percentage of FOG labels over a threshold the entire sample was labelled as an FOG sample. It is worth noting due to the small sample size and large amount of timesteps within the data a vast majority of samples were entirely normal, or entirely FOG. The total number of samples that was passed into TSAI was the length of all sequences divided by n, each sample having 36 features, and n timesteps.

With data reformatted, TSA was capable of being utilised for training and inference on the Multimodel FOG data. A Learner was initialised, with a wide variety of models being able to be passed into it. LSTM, InceptionTime, TSTransformer, TCN models were all set up through the framework. A learning rate optimizer was additionally implemented. With all these initialisations finished, a wide variety of frameworks, hyperparameters, and model configurations was ready to be tested. Additionally, a wide range of functions were set up in order to plot confusion matrices, per_class probabilities, sample correctly classified and misclassified data, as well as accuracy, precision, recall metrics.

Architectures

An LSTM architecture was chosen as our baseline for the purpose of this report. LSTMs are well suited for taking in sequences of time-series data due to their short term memory capabilities, mitigation of vanishing gradient problems, and handling of state [5]. We felt it provided a strong baseline for analysing batches of sequences.

After the LSTM a set of alternate architectures were explored. Another type of RNN, the GRU was implemented. Features of GRUs that made them a strong alternative option were its higher efficiency, greater simplicity, and performance on smaller batches. Additionally, outside of RNN models alternate model types were chosen such as InceptionTime due to it being specifically designed for time-series problems similar to ours. The last model option was a temporal convolutional network, chosen for its abilities in viewing patterns over long series of data as well as computational efficiency[].

Consolidated together, the four architecture types are capable of trying to make predictions on our data in a wide variety of ways. We felt the variation necessary in order to be able to find an optimal model that is sufficiently generalizable with high enough accuracy.

Results and Discussion

Model Experimentation

Upon deciding the method for data preprocessing, and consulting with literature, an initial exploration was completed into 4 models, with some parameter optimization completed on promising models. The baseline architectures are compared below on 50 batchsize.

From best to worst performing, the architectures performed as follows: Inception (94%), LSTM (91%), TCN (90%), GRU (88%).

LSTM

This model reached an accuracy of 91% (Fig 1). After experimentation with hyperparameters this was our best result. Progress from hyperparameter adjustment can be

Figure 1: Losses for Final LSTM Model

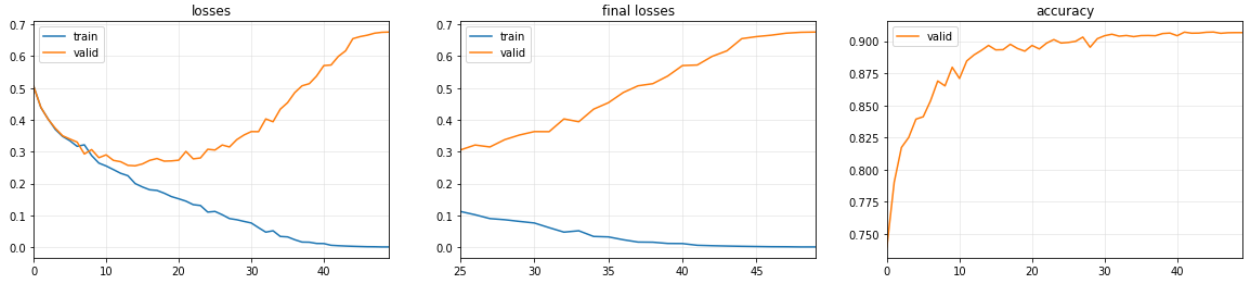
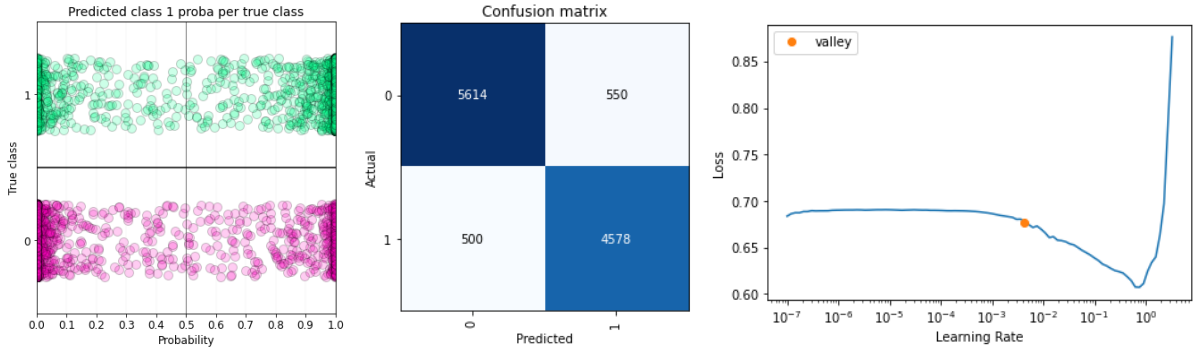


Figure 2: Additional LSTM Results



The LSTM provided a strong baseline architecture, however overfitting became a problem throughout training. Dropout layers between 20-80% were experimented, with 50% dropout experiencing the best overall improvement in LSTM accuracy while minimizing overfitting.

GRU

This model reached an accuracy of 88% (Figure 3)

Figure 3: Losses for Final GRU Model

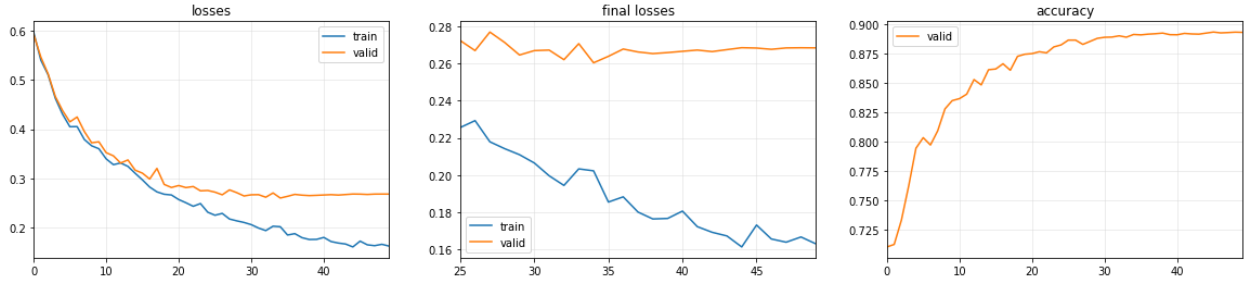
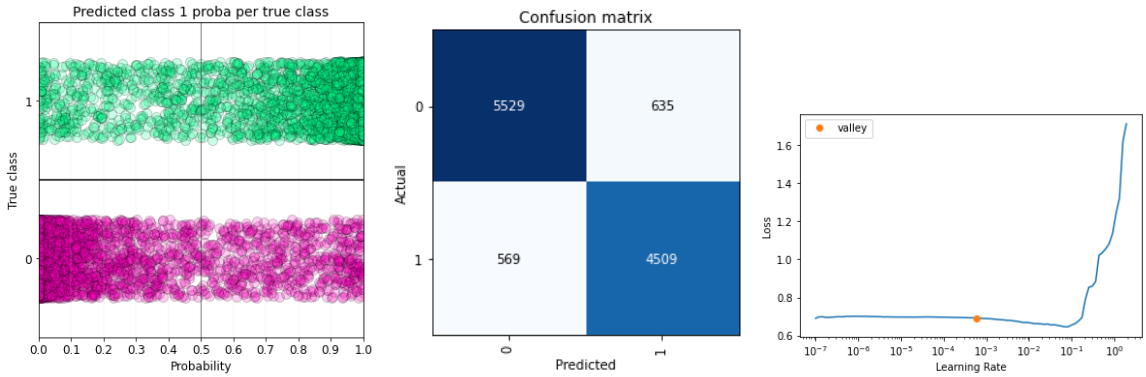


Figure 4: Additional GRU Results



The GRU model performed very well out of the box. GRU models perform more efficiently on smaller data batches compared to LSTMs, since the number of timestamps were relatively low this accuracy made sense with the literature. Overfitting remained high and more regularization would improve model generalisability.

TCN Model Results

This model reached an accuracy of 90% (Figure 5)

Figure 5: Losses for Final TCN Model

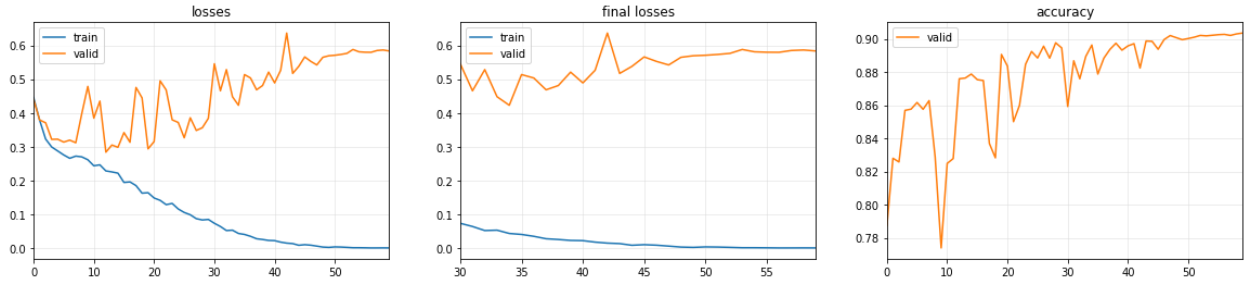
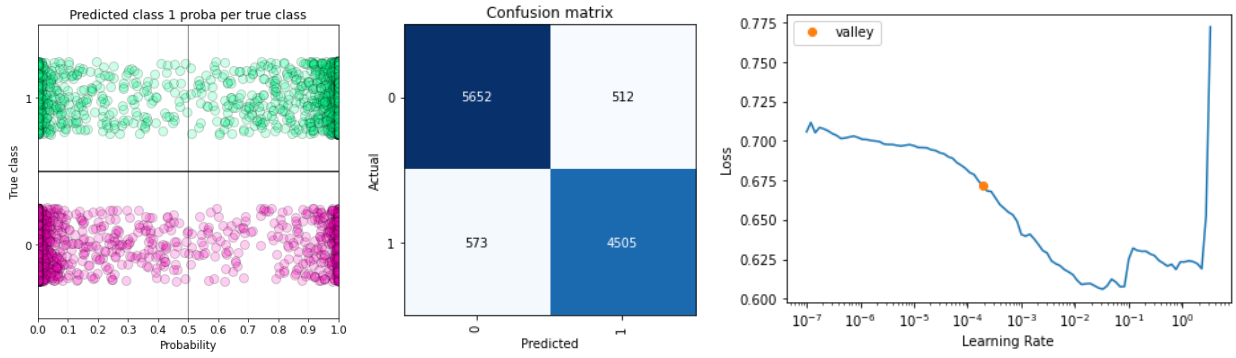


Figure 6: Additional TCN Results



The TCN model contained similar results to the GRU. It experienced high levels of overfitting while maintaining high validation accuracy. Regularization is recommended for this model in order to reduce overfit.

InceptionTime Model Results

This model reached an accuracy of 94% (Figure 7)

Figure 7: Losses for Final InceptionTime Model

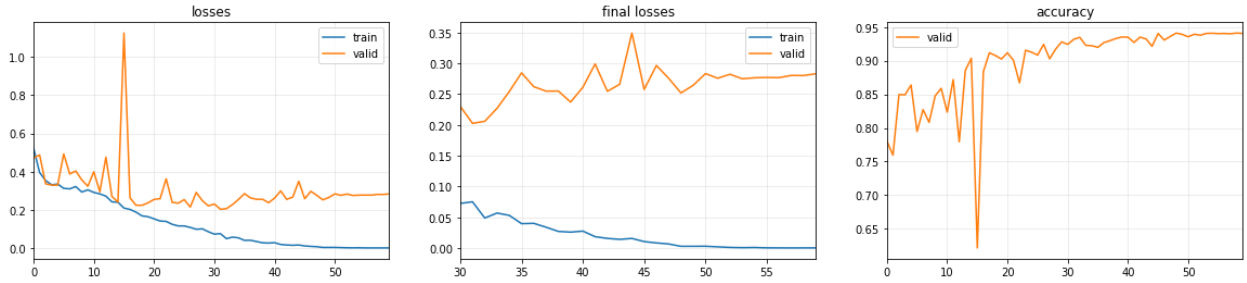
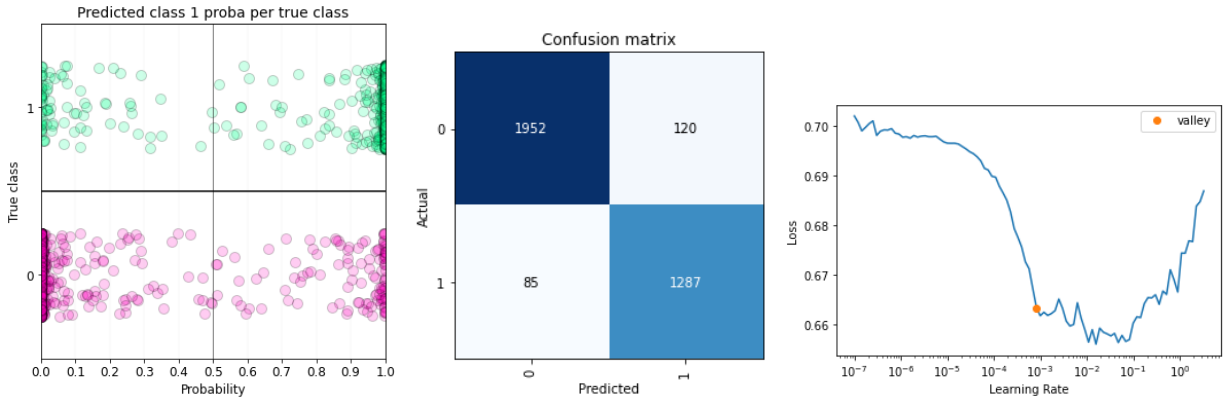


Figure 8: Additional InceptionTime Results



The InceptionTime model was the best all around performer. It provided a very strong 94% accuracy on the test set. Additionally confusion matrices and class distributions indicated a well balanced model. The

InceptionTime architecture is highly specialized in order to be able to handle data that directly suits our use case, and this experimentation verified its efficiency in classifying time-series data.

Parameter Experimentation

The main parameters can be broken down into data preprocessing parameters and architecture hyperparameters.

Data Preprocessing

Firstly, for data preprocessing, task specific data was first used to train on a baseline model (LSTM). This gave 4 models, one for each task from task 1 to 4, where data from patients performing the same task were concatenated. This model was compared to a single non-task specific data model that concatenated all task (1 to 4) and patient (1 to 12) data. This dataset incorporated an additional feature, 'task', specifying what task the given sample came from. The best performing tasks specific model, task 1, reached an accuracy of 86% (Figure 1) whereas the all tasks data reached an accuracy of 68% (Section 2.2 in appendix). Since the single task dataset is smaller, it is more vulnerable to overfitting. Although task 1 performed better, to design for generalizability across all tasks, the all tasks dataset was selected for further experimentation.

Furthermore, the timestamp batch size (i.e. the number of steps) and number of batches (i.e. the number of timesteps) were experimented with. The fully concatenated dataset has a shape of (430512, 34), with a target shape of (430512,). The preprocessing method reshapes this into three dimensions. First, a timestamp batch size of 500 was selected, yielding a shape of (861, 34, 500). On the baseline LSTM, the accuracy achieved was 68% (Section 2.2 in appendix). Next a timestamp batch size of 250 yielded a shape of (1722, 34, 250), achieving an accuracy of 79% on the baseline. Finally a timestamp batch size of 50 yielded a shape of (8610, 34, 50), achieving an accuracy of 91% (Figure 1). This negative correlation between batchsize and accuracy indicates that the baseline architecture does not rely on longer temporal frames to accurately predict the FOG state. Furthermore, the positive correlation between the number of samples and accuracy indicates that the number of training samples is much more impactful in maximising accuracy than the batch size. Therefore, a batch size of 50 will be used to proceed.

Literature states that ECG and SC conductance signals are the most important in predicting FOG. Finally, we compare a dataset using only ECG and SC features to a dataset with all 33 features, on the baseline model. The ECG + SC dataset yields an accuracy of 74% (Section 2.5 in appendix) whereas all features yield 91% (Figure 1). This indicates that all features do not add noise, rather they all provide useful information for the network to learn and improve on.

Architecture Hyper Parameters

Due to the rather small sample size of patients, we found that overfitting could be a rather large concern, and hence experimented with adding dropout. This proved to be extremely influential in yielding high results in our LSTM model, with a 20% improvement in accuracy. This can be seen in the table in Appendix 2.2's first graph compared to Appendix 2.3's first graph.

Due to the same small sample size, another parameter which we found could have reduced our accuracy was batch size, so this was experimented after the model was proven to work, and in all cases where it was applied the results were helped by this. Again, this is visible in the appendix.

Conclusions

The Freezing of Gait detection presented an opportunity to leverage time series deep learning models to learn patterns in the data over the period time. Several models were experimented with to learn over the time series data starting with a simple Pytorch RNN, and eventually settling on the TSAI deep learning library which provided access to a breadth of more complex models able to learn complex feature relations and infer when FOG occured in patients. TSAI provided access to LSTM, GRU, TCN and InceptionTime models which could easily be interchanged if the data was formatted appropriately. This allowed for fast iterations on model architecture and hyperparameter tuning to optimise the models' performance. The results of the various training experiments revealed InceptionTime performed the best over all tasks with a test accuracy of 94%.

Next Steps

The next steps would be to investigate methods to combat overfitting in the models tested. For instance, LSTM performed well on a single task however, the accuracy dropped substantially when trained on all tasks, indicating it was overfitting to a task and poorly generalising the detection of FOG. There are a variety of methods to address this, including sourcing more data, modifying batch size, dropout, regularisation, etc. Additionally, exploring optimising some of these models further with hyperparameter optimization would likely lead to some interesting results. Expanding this research to wearable devices would also prove to be valuable as it would allow this to be applied to a large audience, reducing the late detection of parkinsons that many people suffer from.

References

[1] National Institute on Aging, “Parkinson’s Disease: Causes, Symptoms, and Treatments,” *National Institute on Aging*, 14-Apr-2022. [Online]. Available: https://www.nia.nih.gov/health/parkinsons-disease. [Accessed: 06-Dec-2022].

[2] American Parkinson Disease Association, “Freezing of Gait & Parkinson's Disease: APDA,” *American Parkinson Disease Association*. [Online]. Available: https://www.apdaparkinson.org/article/freezing-gait-and-parkinsons-disease/. [Accessed: 06-Dec-2022].

[3] N. Naghavi, A. Miller, and E. Wade, “Towards Real-Time Prediction of Freezing of Gait in Patients With Parkinson’s Disease: Addressing the Class Imbalance Problem,” *Sensors*, vol. 19, no. 18, p. 3898, 2019.

[4] M. Sun, A. Watson, and G. Zhou, “Wearable computing of Freezing of Gait in Parkinson's disease: A survey,” *Smart Health*, vol. 18, p. 100143, 2020.

[5] G. Shalin, S. Pardoel, E. D. Lemaire, J. Nantel, and J. Kofman, “Prediction and detection of freezing of gait in Parkinson’s disease from plantar pressure data using long short-term memory neural-networks,” *Journal of NeuroEngineering and Rehabilitation*, vol. 18, no. 1, 2021.

[6] H. Li, “Multimodal Dataset of Freezing of Gait in Parkinson's Disease,” *Mendeley Data*, 26-Jan-2021. [Online]. Available: https://data.mendeley.com/datasets/r8gmbtv7w2/3. [Accessed: 06-Dec-2022].

[7] W. Zhang, Z. Yang, H. Li, D. Huang, L. Wang, Y. Wei, L. Zhang, L. Ma, H. Feng, J. Pan, Y. Guo, and P. Chan, “Multimodal Data for the Detection of Freezing of Gait in Parkinson’s Disease,” *Scientific Data*, vol. 9, no. 1, 2022.

Individual Contributions

Deep Learning Individual Report - Hooman Ramezani

My contribution to this deep learning project was building the training loop, architecture, and model variations for our FOG predictor. To begin I received the process dataset from other group members, and began working towards training an initial model. My technical contributions began by researching the problem, as well as relevant papers on time series medical classification. From the literature, from papers such as [x] and [x], it became apparent that for time-series data similar to our problem RNN architectures were the most logical architectural option. There were other possibilities related to course material such as Temporal Convolutional Network (TCN), and Time Series Transformer alternatives, however the RNN seemed like the safest option for the initial architecture. Specifically, I began with an LSTM architecture due to its ability to process large amounts of sequential data.

Significant PyTorch documentation as well as online tutorials were initially read. A dataloader and LSTM architecture was configured for this problem, however a set of errors ultimately led me to abandon this approach after attempting to solve it for a couple days. A lack of personal knowledge as well as minimal course resources related to setting up RNN models could be attributed to this model not being finished.

After deciding to move on, I researched to find another possible route to train a model for this task. This led me to find a package named TSAI for Python, which stands for Time Series Artificial Intelligence. The library contains many helper functions to build deep learning models on time-series data. It had the potential to fit the needs of this project and I began configuring it to be used. After reading significant documentation, I realised that this framework required our data to be reformatted. TSAI requires data formatted in a 3D array of samples x features x timesteps. The Multimodal FOG data provided to us is formatted as a 2D array of features x timesteps with each timestep having its own label. I needed to refactor the data so each sample would need its own label rather than each timestep. To solve this problem, with Jakob Zerbs help, we reconfigured the dataset into small batches with a fixed number of timesteps. Each batch would turn into a sample with a label fed into TSAI. The label was assigned based on the makeup of its labels per timestep. With data reformatted, TSAI could now be utilised for training and inference on the Multimodel FOG data.

TSAI was effective when applied, to produce an LSTM classification model with great initial accuracy on the data. My work proceeded with a wide variety of deep learning experimentation. I trained models with LSTM, InceptionTime Transformer, TST, RCN, and GRU architectures. There was significant overfitting in initial models, which was initially mitigated with more data, adding all tasks and patients to the dataset. Dropout layers were added to my architecture ranging from 20% to 80%. Another parameter I changed was the number of timesteps batched, and a key insight I found is that models performed better with smaller batches of data fed in as samples. It seemed like huge sequences were not needed to analyse FOG data and smaller sequences, with more samples, gave better overall accuracy. The last step of my technical contribution was producing confusion matrices, per class probabilities, and result analysis through various graphs for all the models we utilised.

My contributions' impact on the project was the selection of the deep learning package TSAI which allowed for training a variety of models. As well as parameter tuning that ultimately resulted in strong validation accuracy on the MultiModal FOG data. For the group report, I wrote the *Methodology* and *Results* section.

Jakob

My contribution to the deep learning project was first building a PyTorch RNN and Data Loader with Joshua. The Data Loader was successfully built however I was having issues implementing the RNN architecture. Specifically, the training loop was experiencing errors due to incorrect formatting of the input data. Therefore, I pivoted my focus to running architecture experiments using a tool called 'tsai'. This is an open-source deep learning package built on top of Pytorch and fastai, focused on state-of-the-art techniques for time series tasks like classification, regression, and forecasting. Thus my contribution was to preprocess the time series data into a format that tsai could accept. While Kha performed some initial data engineering on the raw dataset, my data engineering took the 2D *timesteps x features* to the 3D *samples x features x timesteps* format tsai required. This involved concatenating all of the tasks and patients into a single dataset first. I also transformed the 'label' feature from the dataset into the target vector. Furthermore, once I had obtained preprocessed data, I worked with Hooman to build, train, and tune all of the LSTM, GRU, TCN, and Inception architectures. Following all the training runs, I performed inference with Hooman. Finally, I was tasked with generating the loss, accuracy, confusion, predicted probabilities, and optimal learning rate plots for all trained models.

Hooman performed initial research into the problem space's literature and identified that RNNs performed well for this problem. This is what led me to initially experimenting with the PyTorch RNN. When I pivoted to tsai, I performed further research and found that a LSTM, GRU, TCN, or InceptionTime Transformer could also be promising, leading me to experiment with all of these architectures.

For the group report, I generated all of the observed plots. Additionally, I wrote the *Results* section which includes the subsections of *Model Experimentation Variables* and *Parameter Experimentation Results and Discussion*.

Deep Learning Individual Report - Joshua Wilkinson

My contribution to the deep learning project was developing the dataloader, and implementing a simple RNN architecture with Jakob. The dataloader successfully prepared the datasets in a suitable format for training, including dropping timestamp, and separating the labels column as the target values for training, test, and validation sets. To build the dataloaders the data was first read from a text file as csv and then converted into Pytorch Datasets, dropping the unnecessary columns as mentioned earlier. The data was then converted to Pytorch tensors to be used in the pytorch RNN model. The initial RNN approach used a batch of 64 frames, with 2 hidden layers, and 1 output to predict whether or not FOG occurred in that sequence. Cross Entropy Loss was selected since we were targeting a classification approach and SGD was selected as the optimizer. While I worked to optimise the provided Pytorch RNN, Hooman and Jakob worked to investigate more advanced models. The initial attempt with the Pytorch RNN architecture and struggled to learn on the given features and hyper parameters. In order to address this I modified the learning rate, and visualised the data signals to identify features that I could drop according to what correlation they had with FOG occurrence. This led to slight improvements but hardly better than a random 50% guess. There was the option to perform further feature engineering to optimise the learning of the RNN but I opted to begin research into other more complicated architectures for the sake of time. In particular, I researched the use of LSTM and GRU which enable learning complex relations in sequential/time series data. From existing implementations online, I worked to adapt our existing Dataloader and perform data engineering to transform the data into suitable dimensions to be run on the GRU architecture. Before I could successfully implement the GRU architecture, Hooman and Jakob were able to achieve notable performance with Tsai and focus shifted towards their efforts. For the group report, I presented the main findings in the abstract, as well as wrote the conclusion and next steps.

Deep Learning Individual Report - Kha Nguyen

My contribution to the deep learning project was reading up on several papers that involved the detection of FoG using machine learning and deep learning techniques. Many papers I have read up on mentioned the use of classifiers, forest algorithms, convolutional neural networks, recurrent neural networks, and long short term memory models. The significant papers I have read primarily focused on using motion sensors and foot plantar pressure sensors to determine FoG, the latter of which we did not have data for. These findings were relayed to the team regarding the existing options of potential architectures for the deep learning model.

I had an early start on working with the data and understanding what was given within the dataset. After downloading the dataset and using pandas to describe it, I performed the required data engineering to clean the dataset. The filtered data was verified to contain the supposed amount of columns. Headers were added to each of the columns to determine which columns needed to be removed. The EMG columns were individually labelled within each patient and any swapping to standardize the column order was performed to avoid any discrepancy in the data and to ensure each neuron in the input layer was associated with the same data type. A separate dataset was created with the data combined for each patient with the task number associated with each line. An initial training, testing, and validation dataset was created as well to split the group up. The steps outlined above were placed into python scripts and sent out to the other group members to avoid recreating the cleaned data for each training session that is run, which would presumably save time in running the models.

After an architecture was chosen, I read the implementation of RNNs from PyTorch and attempted to follow a few tutorials in setting up an RNN model on Google Colab, however I ran into issues with actually running the model. I was not able to run it due to multiple unknown errors with Colab. To troubleshoot this, Joshua sent a working version of his early RNN model and I was not able to run that either, despite the same code working for the other members.

I wrote the following sections in the report: Introduction, Literature Review, Dataset Description, Data Cleaning

Deep Learning Individual Report - Karanbir Singh

My contribution to the project began with performing research into time-series deep networks and determining how best to apply them. Upon developing that base of information I started to explore the datasets and processing methods. Alongside this I read papers that other group members had recommended to get a better understanding of the problem, and synthesise that knowledge with the time series deep learning research I had completed. I was occupied with other machine learning projects at the time, and hence other team members took the data preprocessing tasks, and I focused on the completion stages of the project.

I provided insight on what models to experiment with as well as research and explanations of how to implement them, when the team was implementing the models, and worked to generate the strategies for testing implementation, such as working with individual tasks rather than the whole model for proof of concept. With the architecture solidified on this and other group members help, I ran the code on my local machine to help with testing, and began researching optuna optimization as I presented the idea of using it to the group from the previous deep learning assignment.

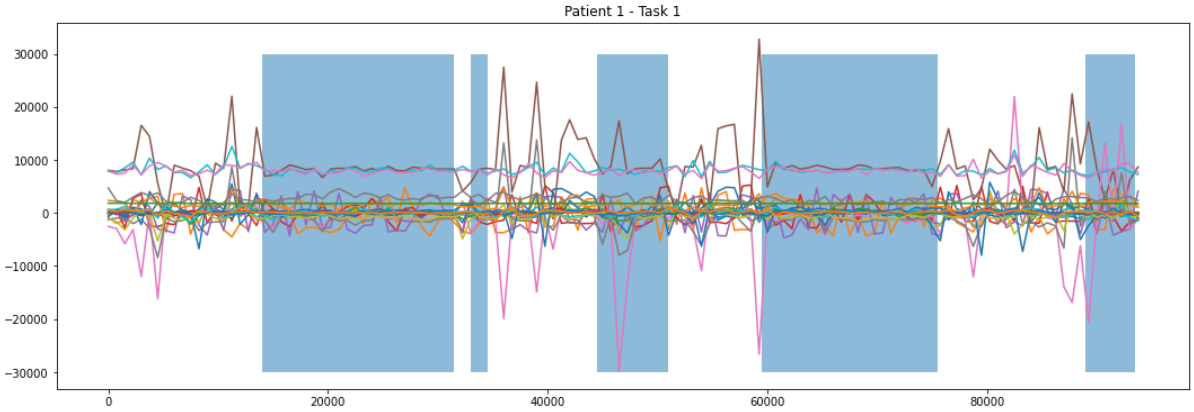
Using optuna tutorials I determined how hyperparameter optimization would work and begun to implement it, however facing a large amount of errors with trying to run optuna with TSAI, and our

results being very good with existing parameters; the group decided to abandon optuna optimization for now.

Appendices

1: FoG Visualization

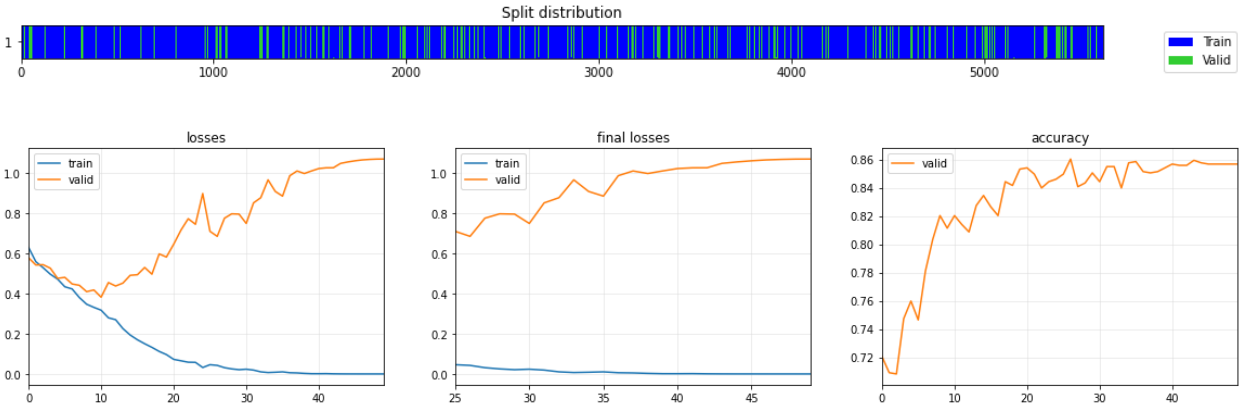
FOG Locations in Blue

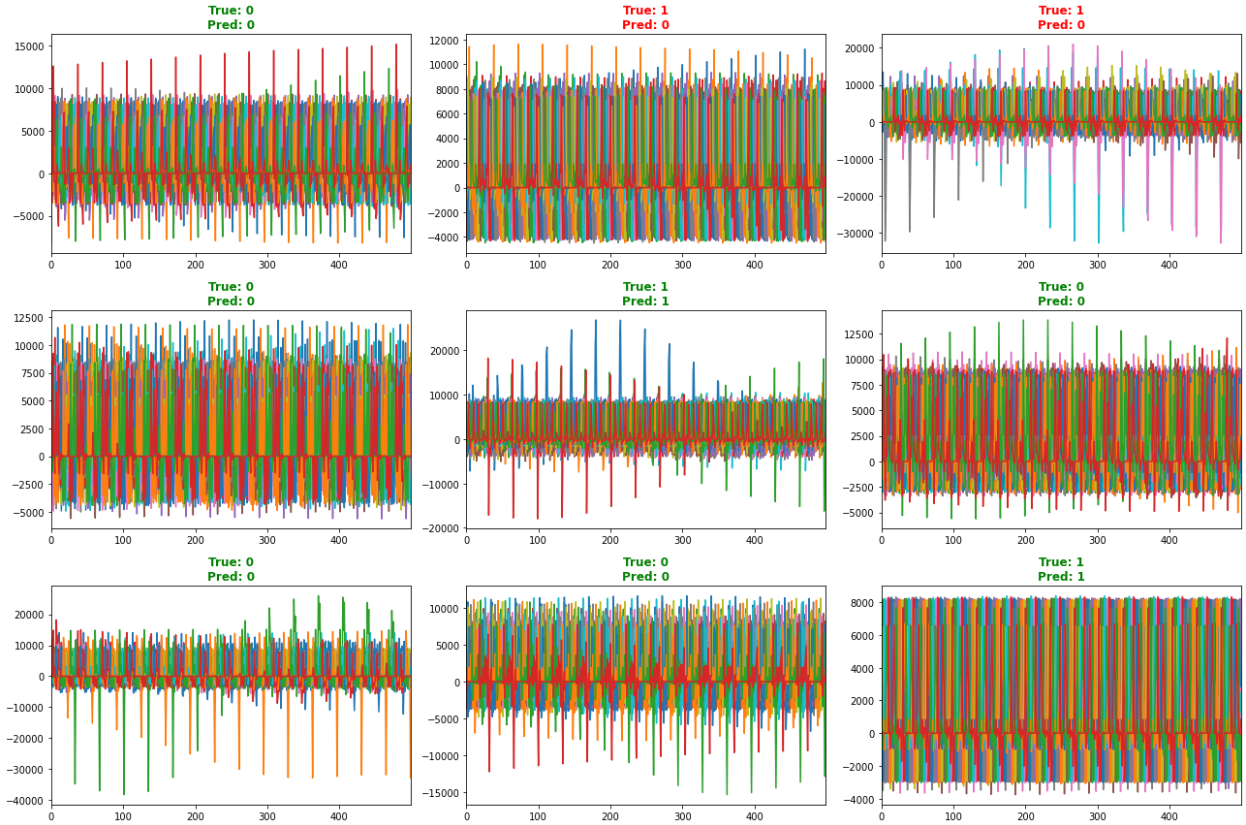


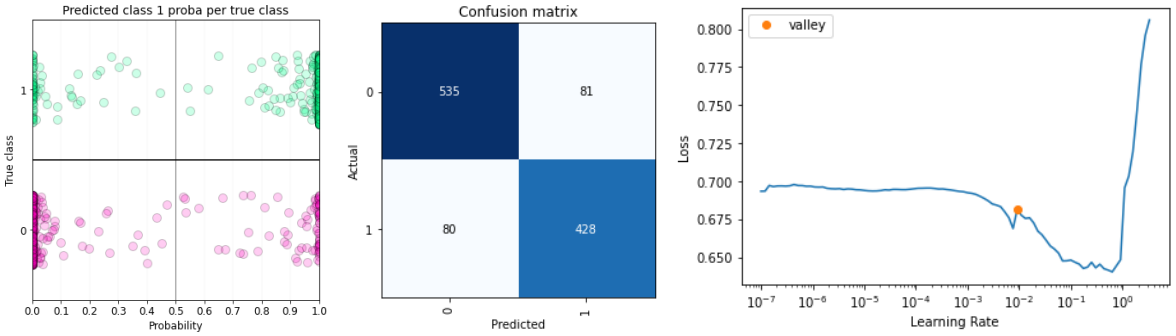
2: Full Model Development and Experimentation Results

2.1: LSTM Results From Task 1

LSTM - Task 1 - Timestamp Batch Size 500

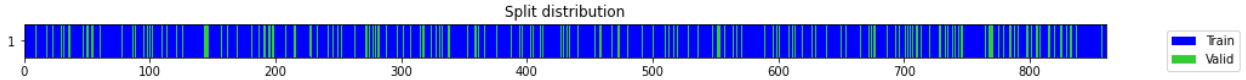


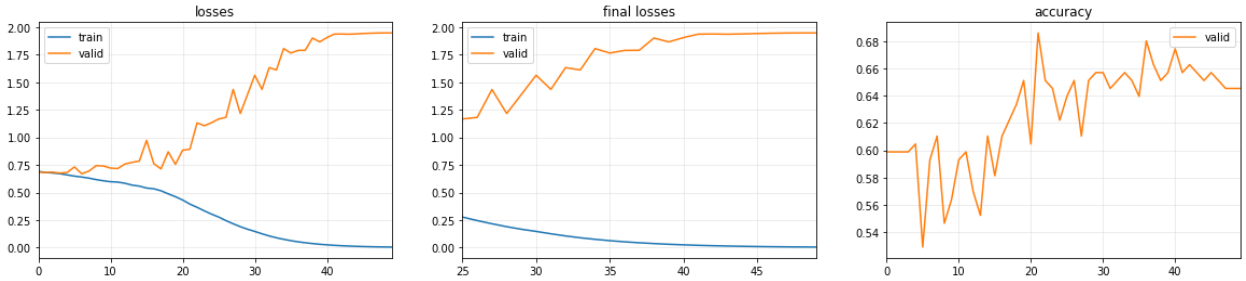


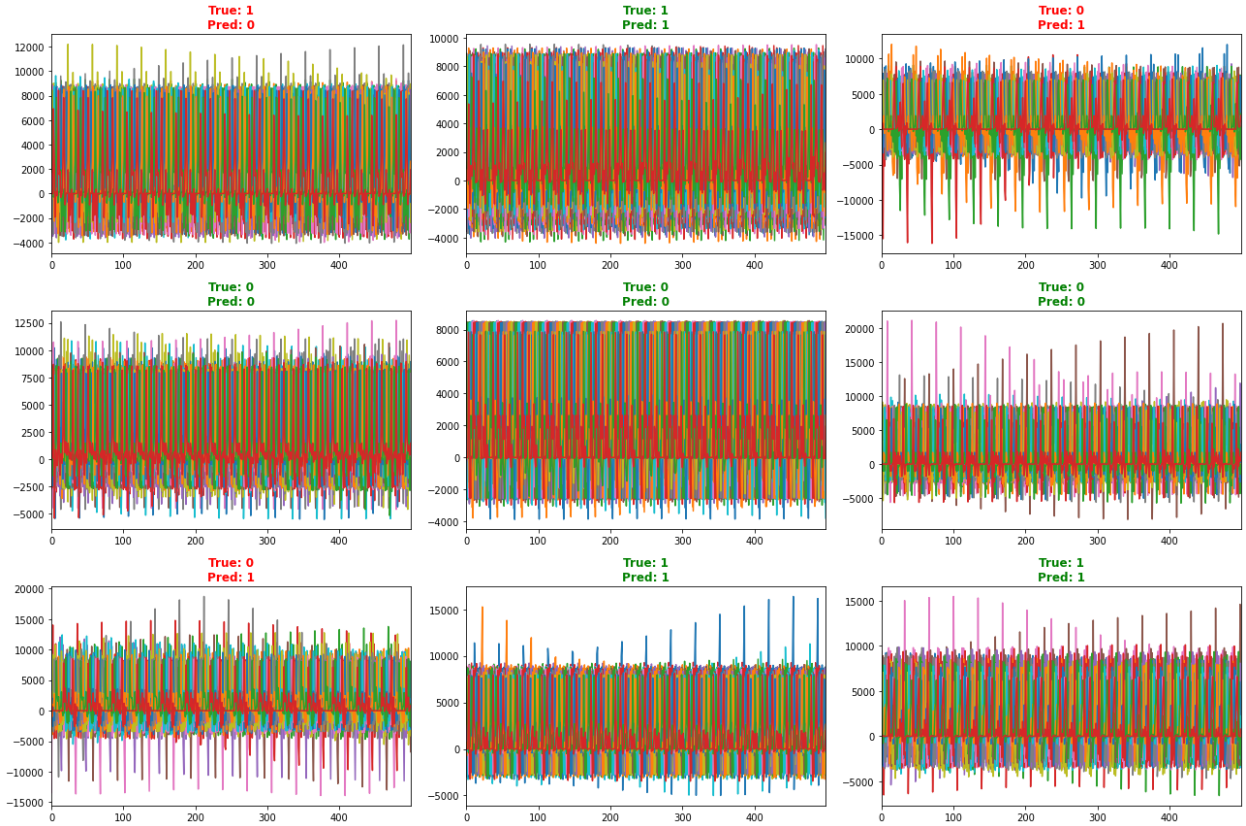


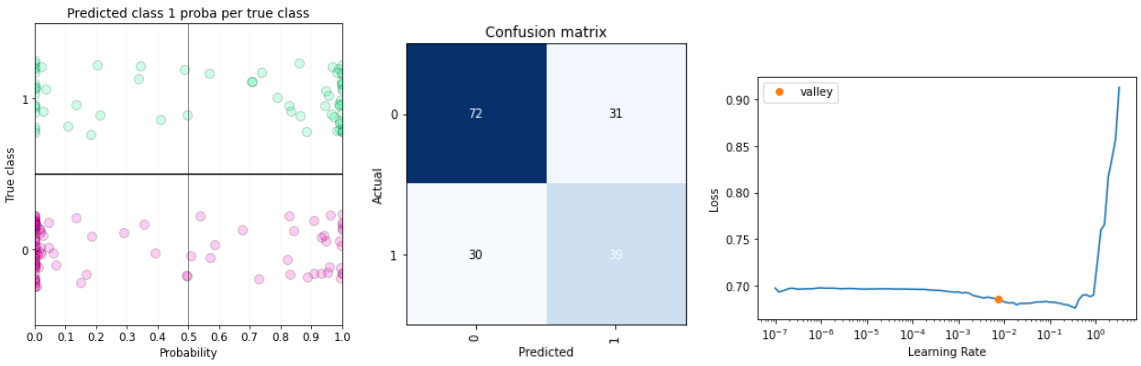
2.2: LSTM Results From All Tasks

LSTM - All Tasks - Timestamp Batch Size 500



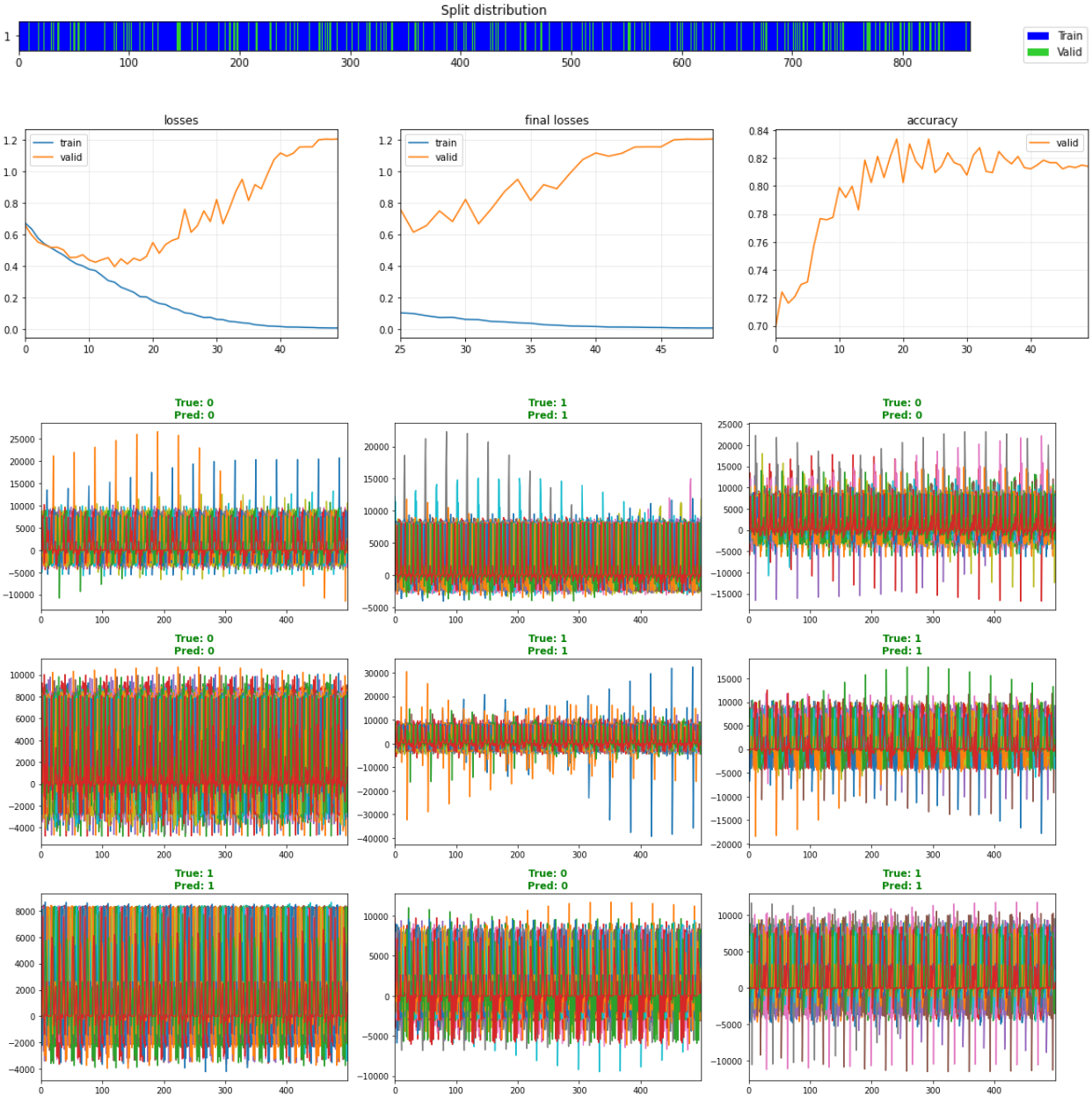

losses
train
valid
final losses
train
valid
accuracy
valid

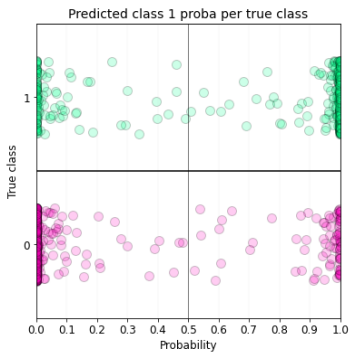

True: 1
Pred: 0
True: 1
Pred: 1
True: 0
Pred: 1
True: 0
Pred: 0
True: 0
Pred: 0
True: 0
Pred: 0
True: 0
Pred: 1
True: 1
Pred: 1
True: 1
Pred: 1

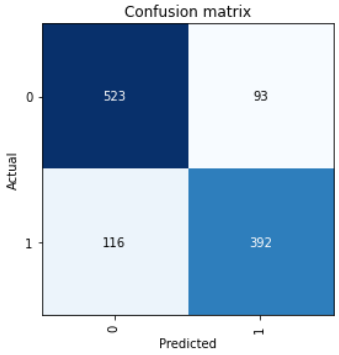

Predicted class 1 proba per true class
True class
Probability
Confusion matrix
Actual
Predicted
72
31
30
39
valley
Loss
Learning Rate

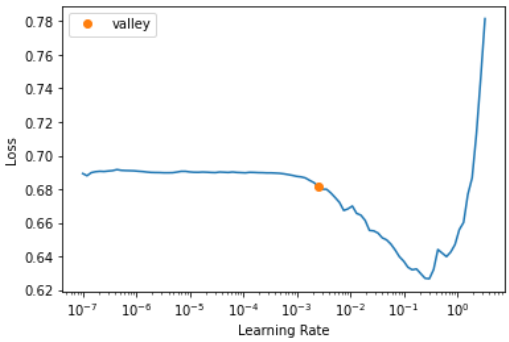
2.3: LSTM Results From All Tasks with Dropout Modification

LSTM - All Tasks - Dropout = 0.6 - Timestamp Batch Size 500



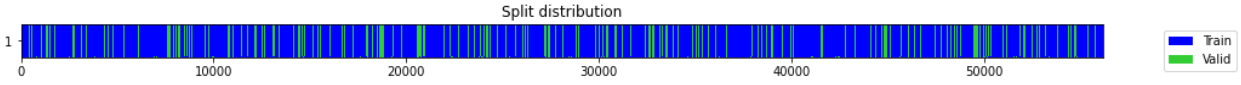


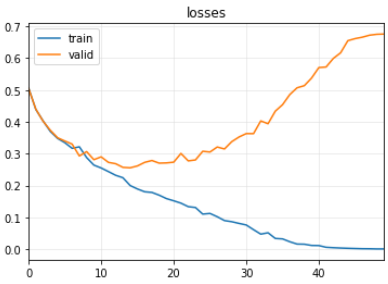


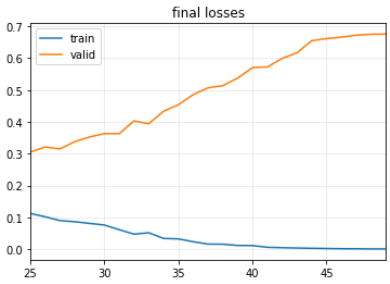


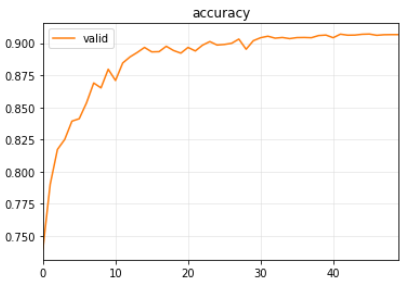
2.4: LSTM Results From All Tasks with Batch Size Reduction

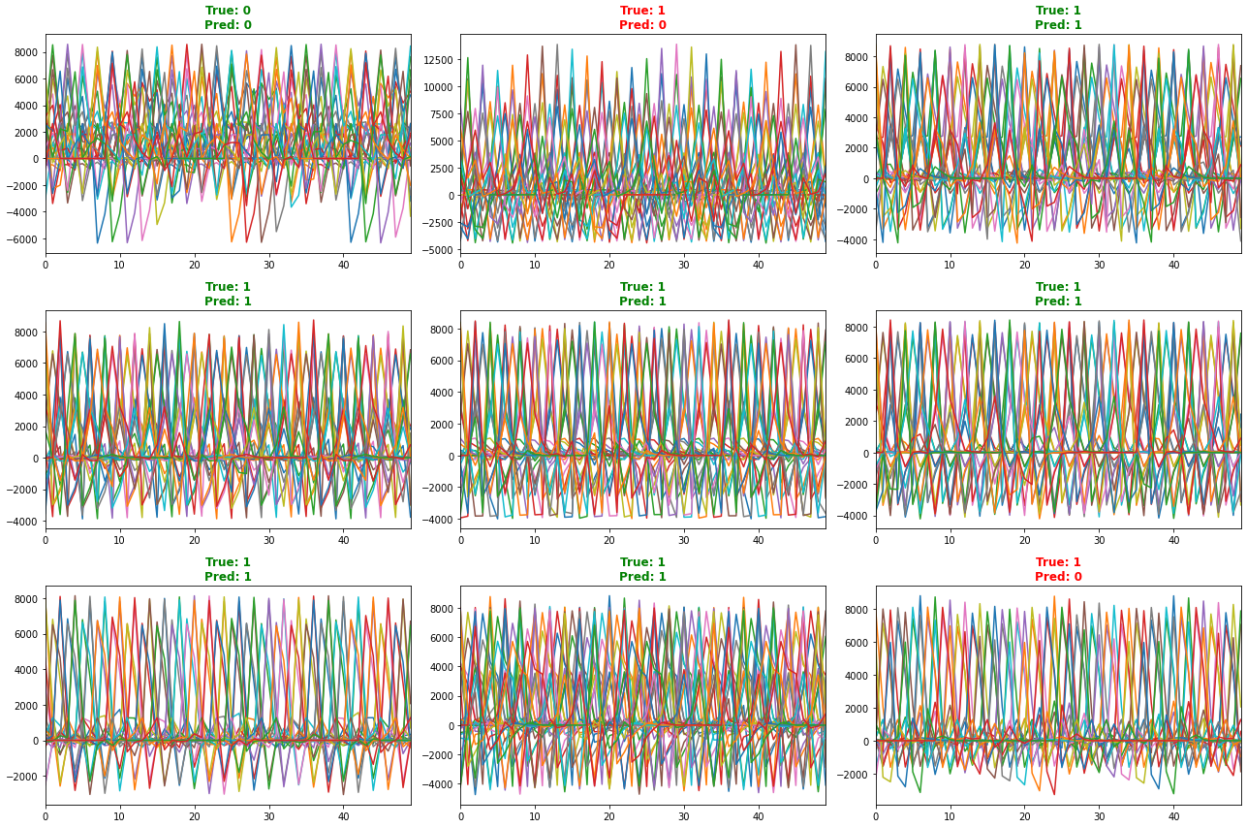
LSTM - All Tasks - Timestamp Batch Size 50

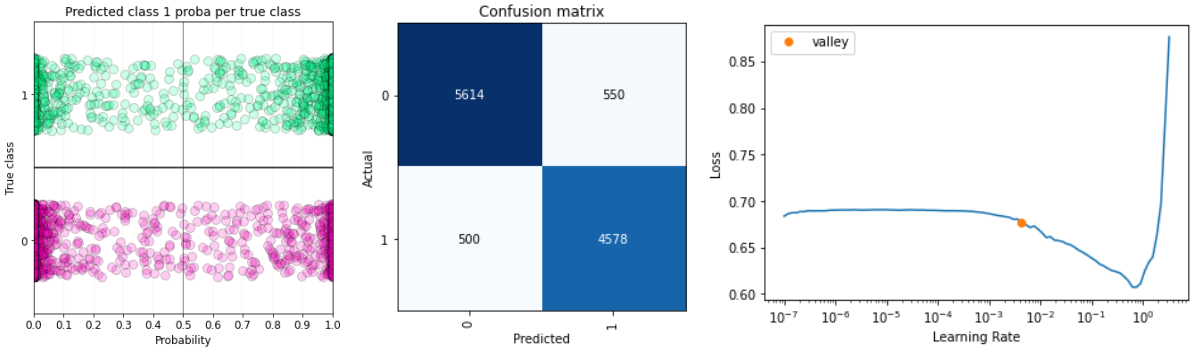






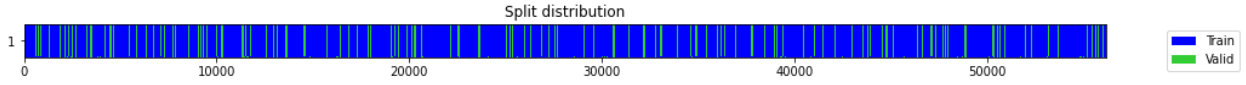


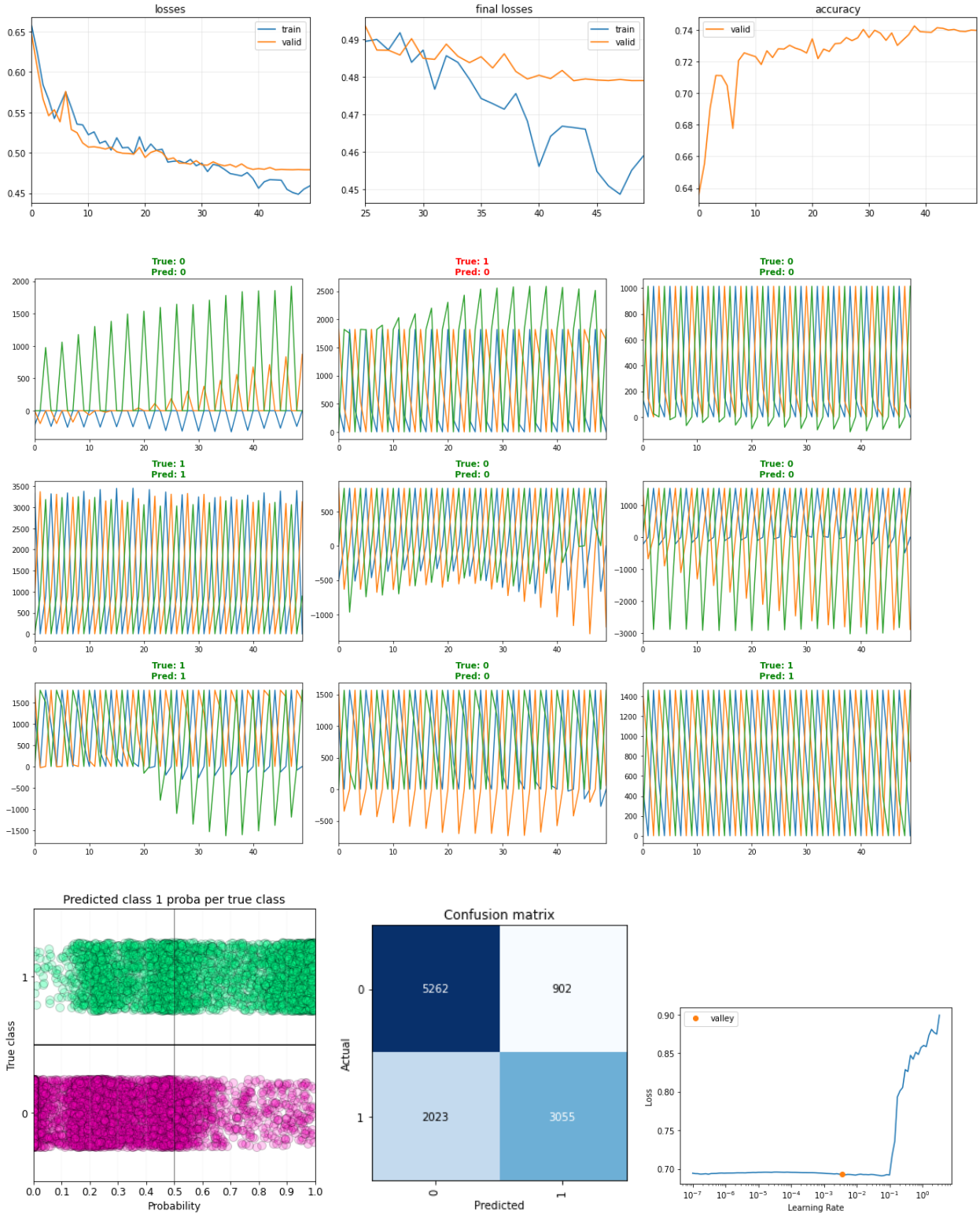




2.5: LSTM Results with ECG and SC Only

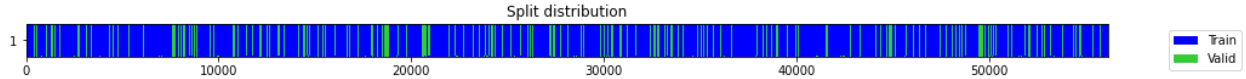
LSTM - All Tasks - Timestamp Batch Size 50

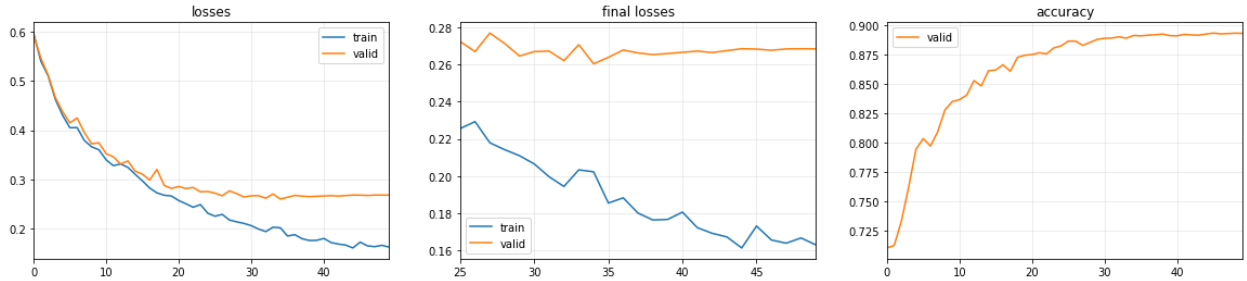


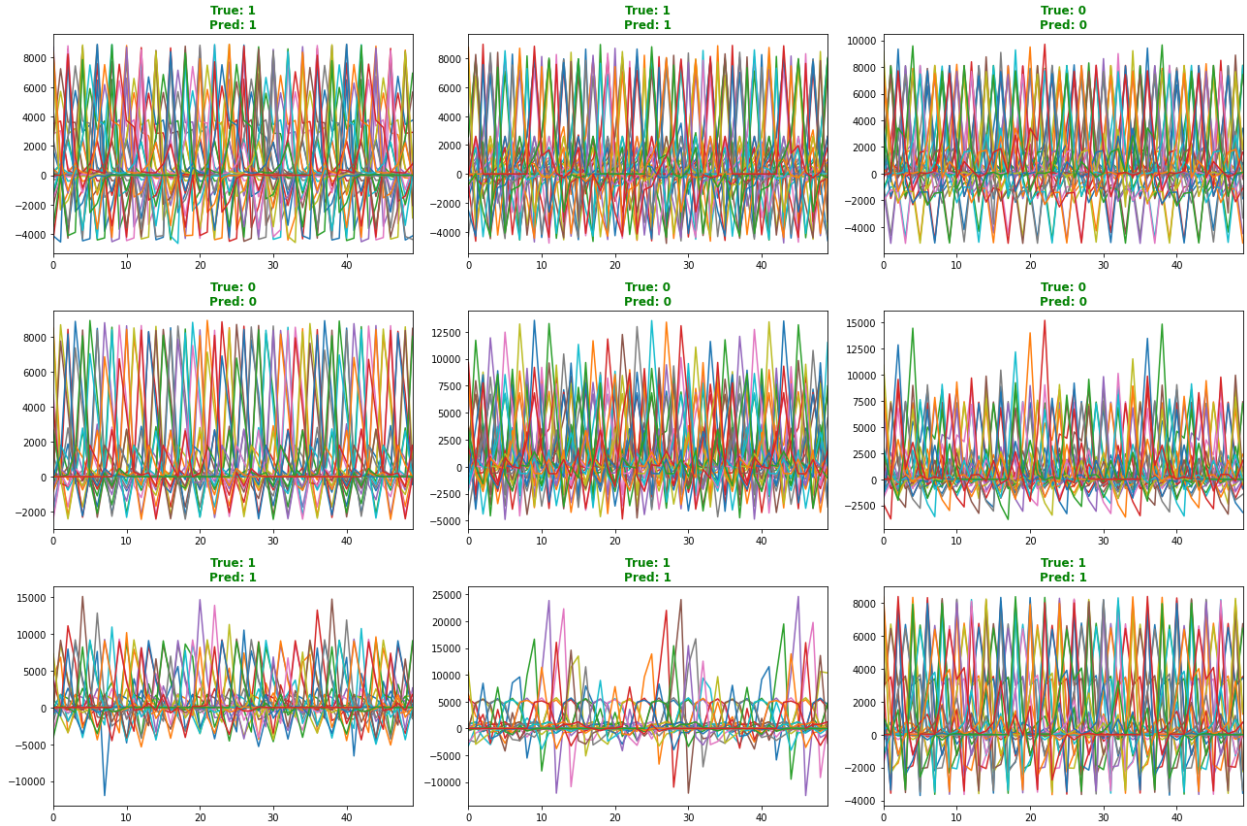


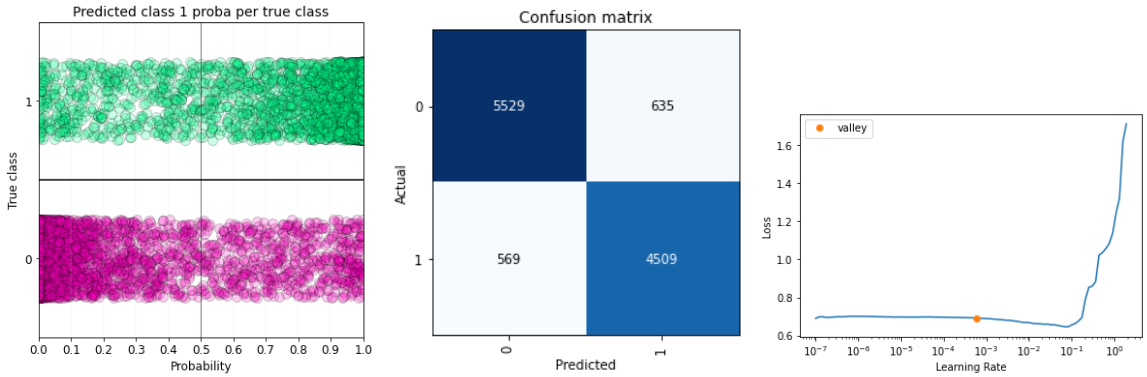
2.6: GRU Results with All Tasks

GRU - All Tasks - Timestamp Batch Size 50


Split distribution
Train
Valid

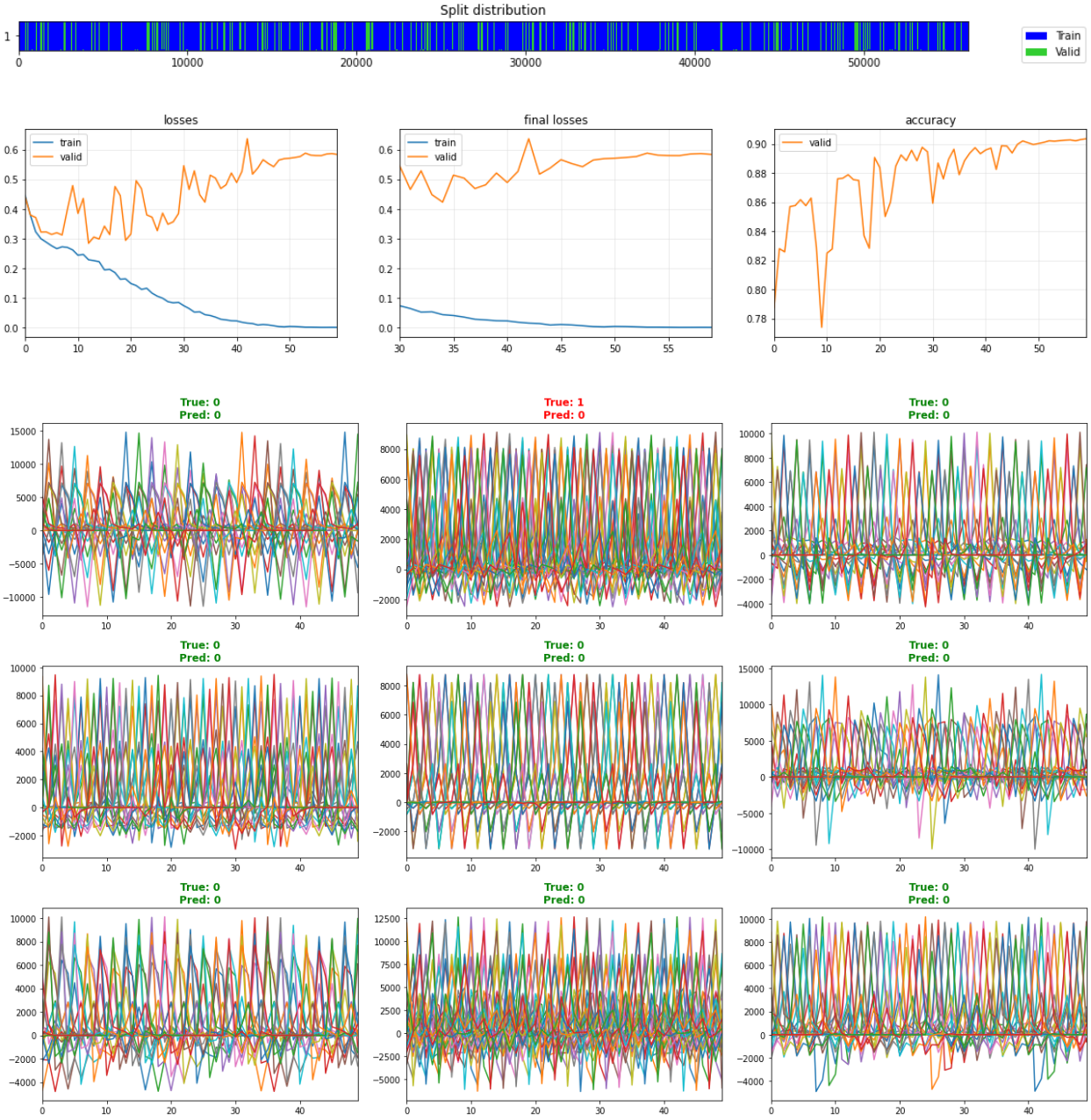

losses
train
valid
final losses
train
valid
accuracy
valid

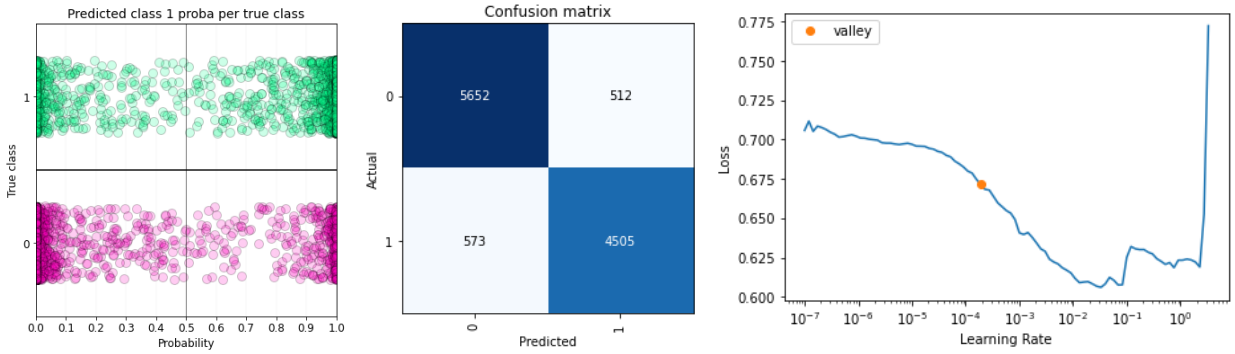

True: 1
Pred: 1
True: 1
Pred: 1
True: 0
Pred: 0
True: 0
Pred: 0
True: 0
Pred: 0
True: 0
Pred: 0
True: 1
Pred: 1
True: 1
Pred: 1
True: 1
Pred: 1


Predicted class 1 proba per true class
True class
Probability
Confusion matrix
Actual
Predicted
5529
635
569
4509
valley
Loss
Learning Rate

2.7: TCN Results with All Tasks

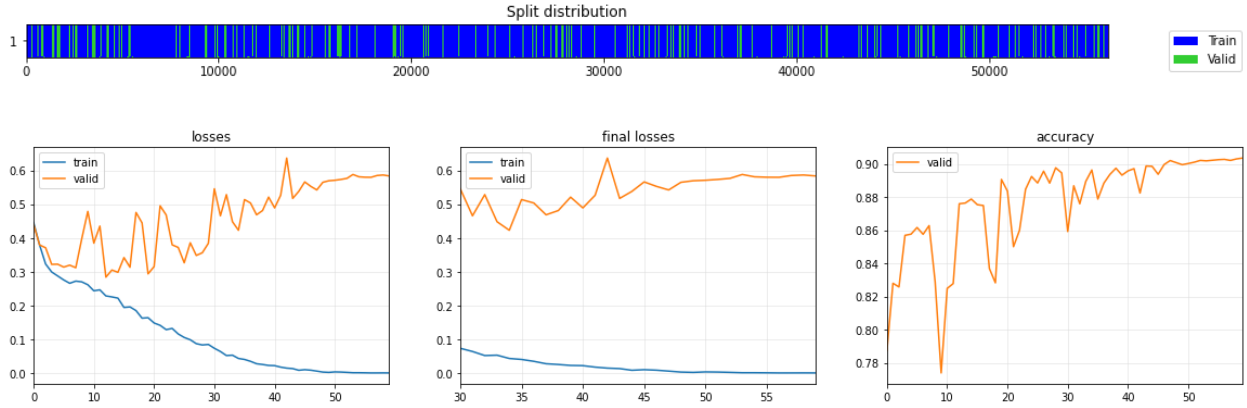
TCN - All Tasks - Timestamp Batch Size 50

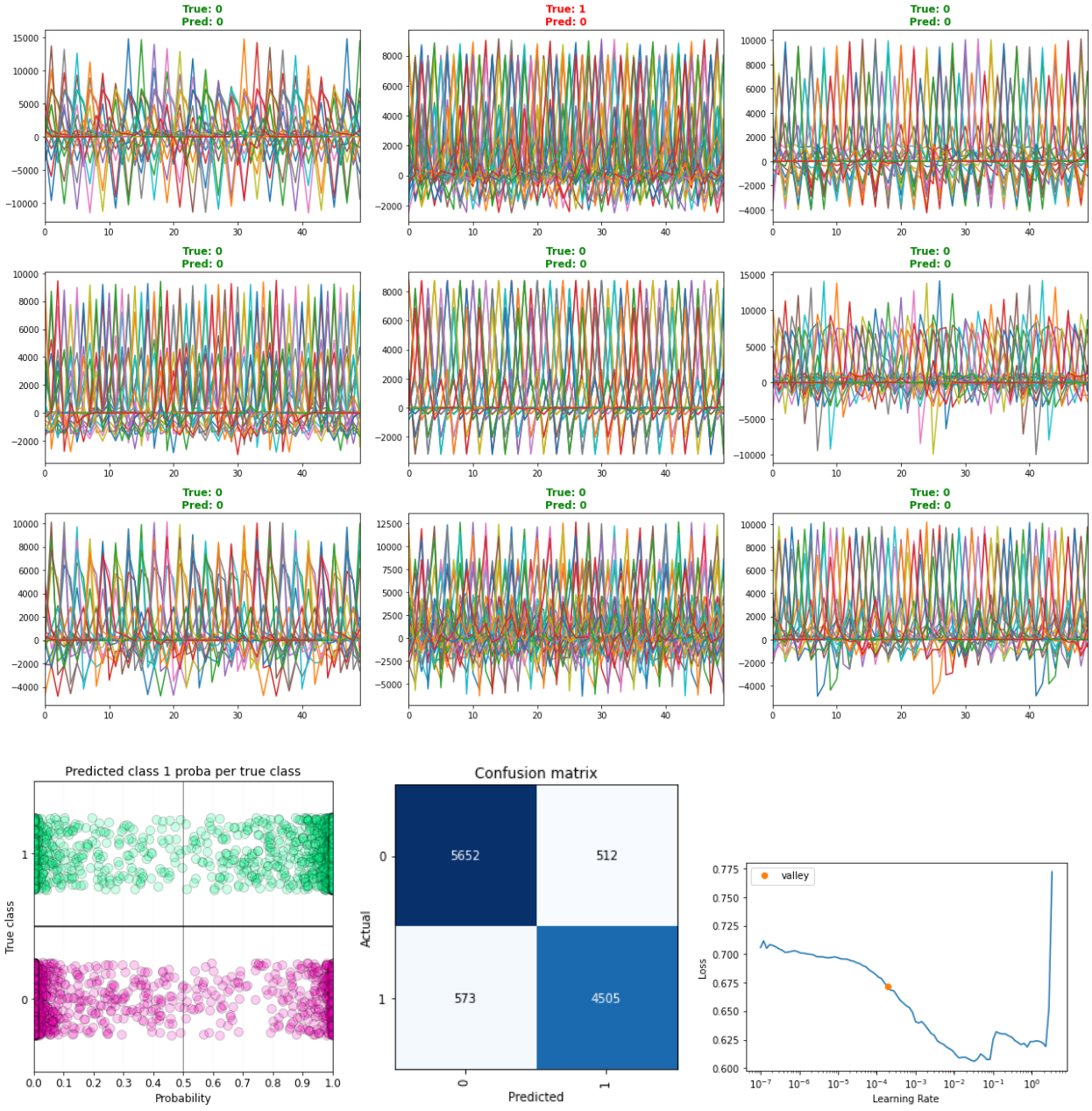




2.8: InceptionTime Results with All Tasks

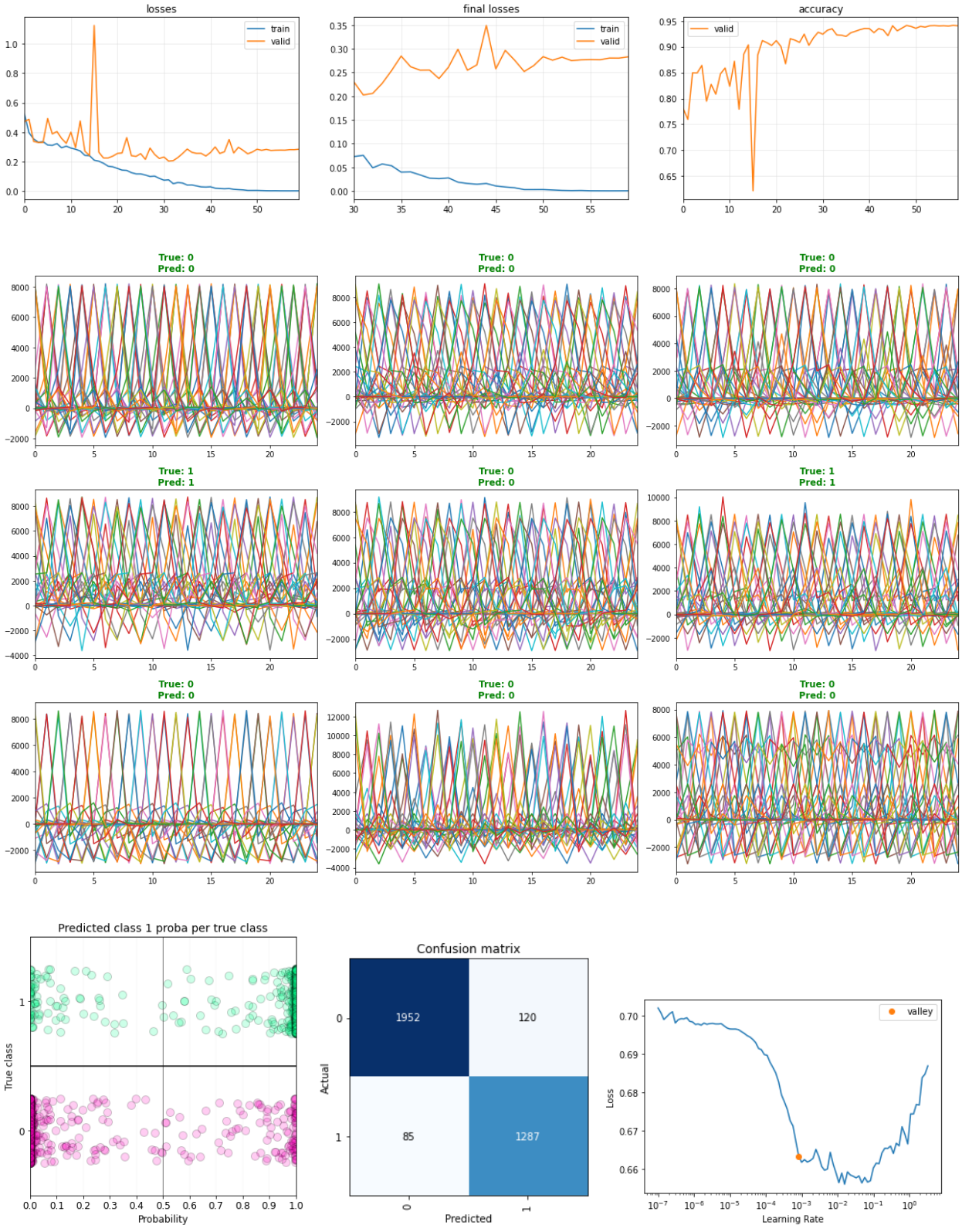
InceptionTime - All Tasks - Timestamp Batch Size 50





2.9: TCN Results with Reduced Batch Size

InceptionTime - All Tasks - Timestamp Batch Size 25


losses
train
valid
final losses
train
valid
accuracy
valid
True: 0
Pred: 0
True: 0
Pred: 0
True: 0
Pred: 0
True: 1
Pred: 1
True: 0
Pred: 0
True: 1
Pred: 1
True: 0
Pred: 0
True: 0
Pred: 0
True: 0
Pred: 0
Predicted class 1 proba per true class
True class
Probability
Confusion matrix
Actual
Predicted
1952
120
85
1287
valley
Loss
Learning Rate