Practical Machine Learning - Project 1

- Smartphone User Identification Kaggle Competition, 2022-2023 -

Ionescu Andrei, 407

andreei.ionescu@gmail.com

Abstract

The Smartphone User Identification Kaggle Competition (Radu, 2022) consists in a classification challenge to discriminate between 20 users, on a given dataset of mobile signal recordings. The works presented in this paper highlight the complete end-to-end pipeline of: data analysis and preprocessing, augmentation techniques, hyperparameter tuning and showcasing multiple approaches using both a classical model such as SVM, as well as deep neural networks variants: Temporal Convolutional Neural Network (TCNN), Attention-based Temporal Convolutional Neural Network (ATCNN) and Hybrid AutoEncoder SVM Classifer (HAESVM).

1 Dataset

1.1 Dataset Description

The task at hand is to classify the source (user) of mobile signal recordings that are taken at 100 Hz frequency, using an accelerometer, for an approximate duration of 1.5 *seconds*. For each user a fixed amount of 450 *recordings* are given, totaling up to 9.000 *samples* that have associated labels and can be used for training. On the other hand, the evaluation is done on a test set which has 5.000 *unlabeled samples*. Each time step in one recording has coordinates (x, y, z), meaning that the sizes of the subsets are as follows:

- $(N_{train}, T, C) \mapsto (9.000, 150, 3)$ *labeled* training samples.
- $(N_{test}, T, C) \mapsto (5.000, 150, 3)$ *unlabeled* testing samples.

Where N represents the number of samples, S represents the temporal/sequence dimension and C reflects the number of coordinates for one time step.

1.2 Dataset Analysis

As mentioned in the Dataset Description, there are 20 users (or classes) where each have 450 recordings, making the training **dataset balanced** *class-wise* from the beginning. As a first step in the analysis I inspected the global characteristics of the given data coordinates in Table 1, and observed that the computed *mean* and *stardard deviation* differ by a margin for each coordinate. Moreover, according to the *standard deviation* it seems that the values are a bit spread out which requires data scaling[1.3]. Also with regards to this, it can be an indication of residing outliers in the data which will be investigated further.

[2] \ [1]	x	y	z
count	1.348955e+06	1.348955e+06	1.348955e+06
mean	-2.151555e-01	4.731727e+00	8.017973e+00
std	8.352723e-01	1.205409e+00	1.060832e+00
min	-7.778163e+00	-6.853400e-01	1.212663e+00
25%	-7.314290e-01	3.876213e+00	7.340024e+00
50%	-2.184710e-01	4.532822e+00	8.043920e+00
75%	3.351880e-01	5.391742e+00	8.720282e+00
max	7.284957e+00	1.104026e+01	1.581849e+01

Table 1: Global Training Dataset Characteristics

1.2.1 Outliers Detection and Removal

Outliers present in the training data can be quite problematic as they can *destabilize* the training process by triggering big update steps in the wrong direction, leading the model parameters to a worse local minima. Considering this fact and the *std*, *min*, *max* values shown earlier, I displayed boxplots for each coordinate (x, y, z) for the **entire training dataset** and for **each class**, in order to view if this is really the case. These plots can be seen in the following figure:

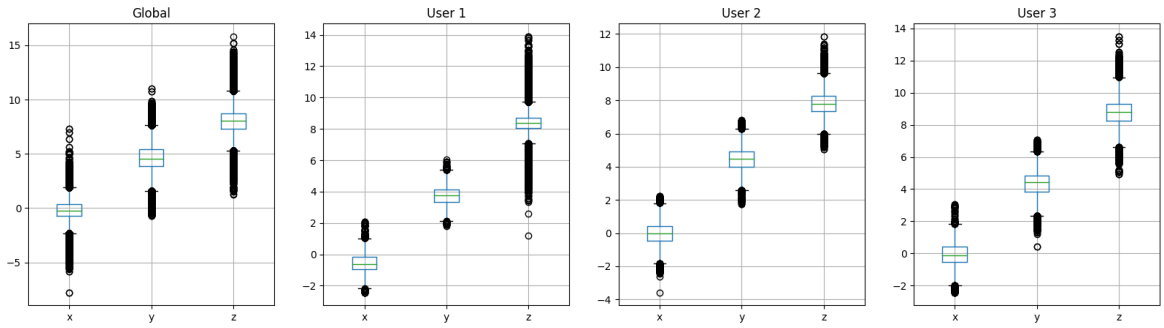

Figure 1: Training Data Outliers

Each boxplot has its whiskers set to the $Q1$ or $Q3$ shifted by $\pm 1.5 * \mid Q3 - Q1 \mid$. The first subplot illustrates the values for the **entire training dataset** and the rest of them are for the **first three users**. Similar values are present for other users as well but aren't included for brevity.

As it can be observed there are a large number of values residing outside the established $\pm 1.5 * IQR$ range. In order to eliminate them two approaches can be applied and the results can be seen in :

- Compute the $Q1$ and $Q3$ values for each (x, y, z) coordinate for the **entire dataset** and exclude those coordinates, which fall outside the whiskers as mentioned in Figure 1, from their respective recording while keeping the rest of the values.
- Similarly to the previous approach, compute the $Q1$ and $Q3$ values for each *class* and for each (x, y, z) coordinate inside that class and proceed the same way during elimination.

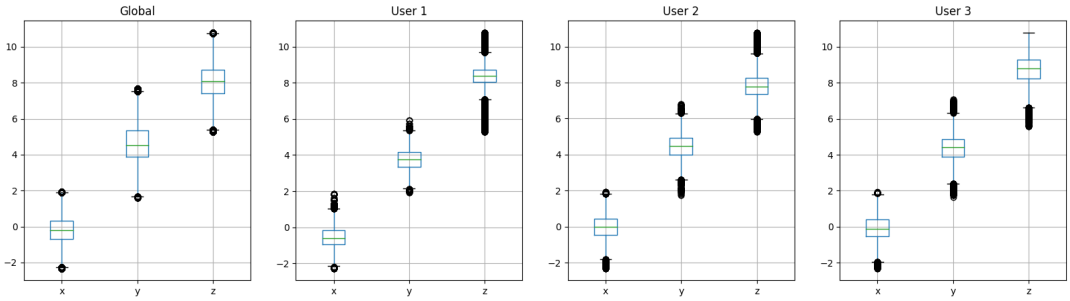

Figure 2: Global Removal of Outliers in Training Data

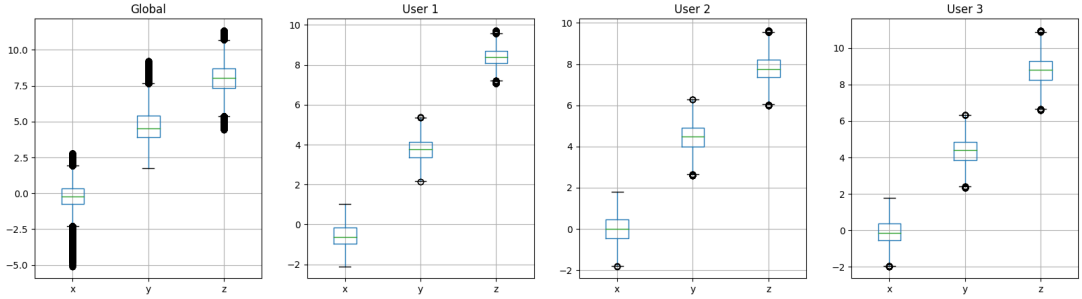

Figure 3: Intraclass Removal of Outliers in Training Data

The *intraclass* approach can only be applied on the training and validation sets and not on the test set as it does not have labeled samples. Even though the two approaches try to eliminate outliers it can be seen that when applying one method or the other there still remain some values either in the *global-scope* or the *class-scope*. Both methods could be applied simultaneously but this might result in recordings that have very few time points and the behavior of the user might not be preserved when trying to impute missing values.

During manual fine-tuning experiments, the removal methods tend to show small improvements during K-Fold Cross Validation when they are applied on both sets and slightly worse performance on the Kaggle public subset. A intuition for this would be that the task is easier to learn using the new representation but tends to overfit because the outliers might actually indicate individual spikes correlated to the user behavior.

1.2.2 Filling Missing Values

Both the training and the testing subsets contain user recordings that either **surpass** or **lack** values from the pre-established 150 sequence size. This can be an issue as many classical and deeplearning models expect a fixed input size in order to be trained and predict new values.

To overcome the first issue some entries need to be eliminated from the recordings in order to adjust an upper-bound of length 150. The values can be removed either from the start, end or inside the sequence. Because the sequence length also represents a temporal dimension which reflects the user's behavior, removing random values or intervals from inside the sequence might remove essential information. In the implementation I considered only the removal of values that are present at the **end of the temporal sequence**.

Overcoming the lack of entries is not as trivial as zero-padding or randomly filling in some statistics somewhere in the recording because this would disrupt the expected flow of information and would confuse the models between samples which needed filling and those which did not. A better approach is to consider this aspect and, in turn, interpolate new values randomly between existing ones:

1. Create an empty recording and set it's head and tail to match original recording's.
2. Insert the in-between values from the original recording into the new one, at random positions while maintaining their original ordering.
3. Fill in the gaps in the new recording by interpolating between existing non-gap values.

One thing that I considered during the proposed approach for *lack of entries* is to drop recordings that do not pass a lower-bound margin of sequence length because there would be too many interpolated values and the samples would mostly be generated and might not offer enough distinctive information to the model during the training phase. In the course of experiments I tried to use different lower-bounds such as: 50, 75, 100. Only the 50 lower-bound value showed marginally better performance, in some cases, in comparison to keeping those recordings, for certain models. Because of this I kept all recordings and did not consider dropping any of them. The results of the transformations described in this subsection can be visualized in Figure 4 and 5.

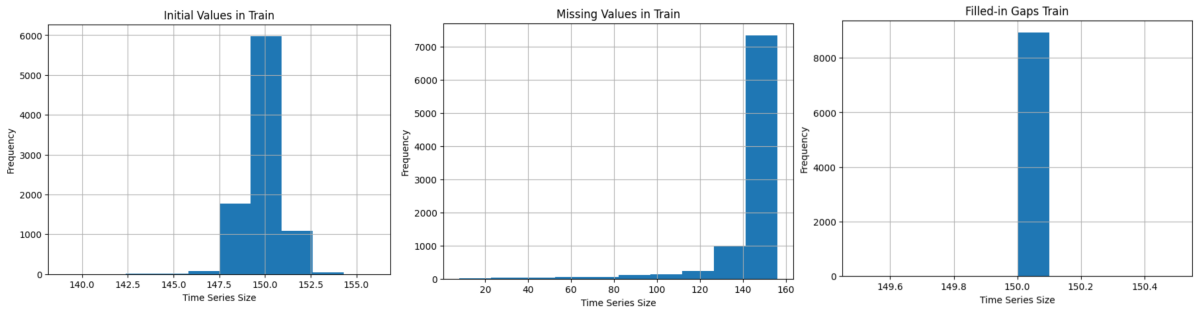

Figure 4: Before and After Histograms of Recording Sizes
The data is first capped at 150, global outliers are removed and new values are randomly in-between interpolated.

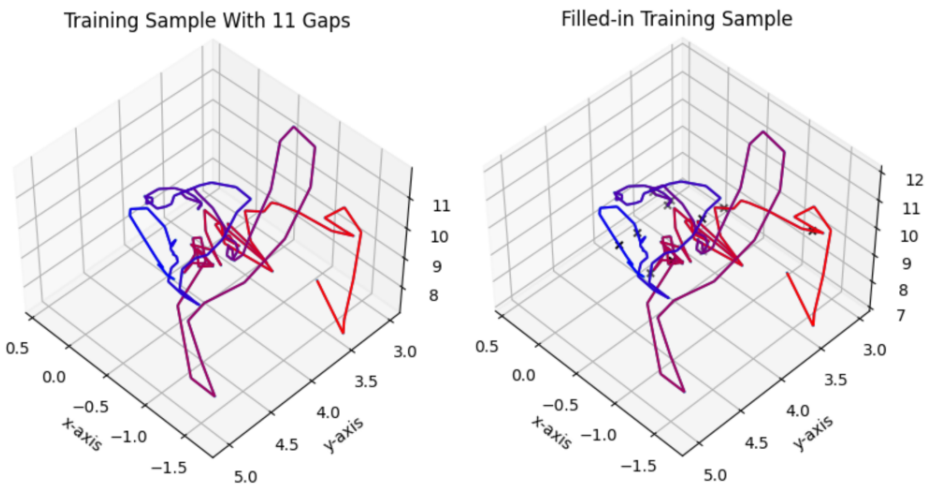

Figure 5: Gaps Filling Algorithm Applied on a Training Sample
The read-to-blue gradient indicates the start and end of the sequence. The filled-in gaps are marked by the 'x' symbol. These visualizations are generated prior to outlier elimination.

1.3 Data Preprocessing

As mentioned in the beginning, the recordings are given as a set of coordinates, $x \in \mathbb{R}^3$, that change along the temporal axis. According to the values observed in Table 1, the scaling differs from one coordinate to another as well as other characteristics. This can have negative impacts during model training in various ways such as:

- the training process is destabilized and convergence can take more time because of odd updates;
- models can learn biases towards features that have a greater scale, but which do not contribute much at inference time;

To alleviate these issues there are multiple ways of preprocessing the data such that the features are transformed to similar scales and have equal importance when given to the model. During experimentation I tried to apply *standardization*, *min-max scaling* and *L2 normalization* across different dimensions:

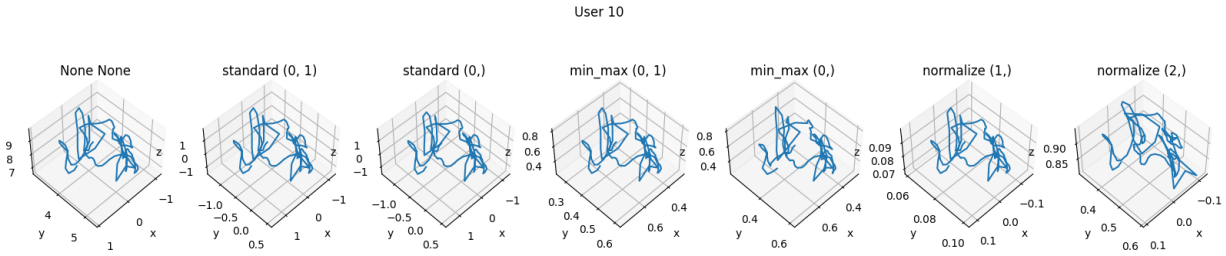

Figure 6: Data Scaling Approaches Visualized for a Training Sample
Reads from left to right. The first approach does not apply any scaling, following are the operations mentioned above. $(0, 1)$ indicates that the dataset and temporal dimensions are collapsed, while $(0,)$ reflects that only the dataset axis is collapsed. The normalization is applied per-instance across the temporal or spatial axis.

In order to investigate whether any of these known techniques improve either the training process or the accuracy, I ran some models for a small amount of epochs on the same 5-Fold Cross Validation, and recorded the mean accuracy values for each type of scaling that was applied. The data was only filled to size 150 and no recordings/outliers were removed. The results can be seen in Table 7 and showcase that *min-max scaling* and *L2 normalization* both disrupt the training process and lead to bad validation results. On the other hand, the *standardization* computed by either collapsing or not the temporal axis, shows **slighly better results** than training the models directly on the raw data. With these empirical observations in mind, the *standard* $(0,)$ approach is the one kept for further experiments.

2 Models and Tried Approaches

2.1 Support Vector Machines (SVM)

The first model that I have considered is Support Vector Machines (SVM), with the LIBSVM implementation (Chang & Lin, 2011), as the task at hand is a classification problem, with numerical data given as input features. The model is well known for its robustness against overfitting on tasks, because of its objective of maximizing the boundary in binary classification. In order to adapt the algorithm to the current problem, which implies multiclass prediction, two approaches can be taken:

- One versus all - train $N_{models} = 20$ by keeping one class as a positive label $(+1)$ and the other ones as negative predictions (-1);
- One versus one - use $N_{models} = \frac{N_{class} * (N_{class} - 1)}{2} = 190$, where each is trained on two different classes;

Futhermore, the prediction is given by a **majority vote** between the learnt models. Across the mentioned procedures, I considered using the *one versus all* approach because of its lower computational cost and similar results to the other method. Using a LinearSVM on the dataset has not shown great results compared to using the *Radial Basis Function Kernel* (Shashua, 2009) as a similarity measure, with the features being mapped in a higher dimensional space. In turn, this gives a strong indication that the data may not be linearly separable. The results of hyperparameter tuning can be seen in Table 2.

Overall, the fine-tuned classical model offers a good starting point with *RBF* kernel and the tweaked C regularization, which allows a softer margin to generalize better on the validation data. However, one weakness of applying the SVM model on this dataset is the fact that it **ignores the temporal dimension**. In other words, shuffling alongside the temporal axis gives the same results as each space-time point has an associated weight that receives no indication of any positional value. In order to overcome this issue, one approach would be to add *positional encoding* (Vaswani et al., 2017) in the form of:

$$\hat{X} = \{\, x_{ij} + p_j \mid p_j = \frac{j}{T},\ j \in [1, T] \cap \mathbb{Z},\ i \in [1, N] \cap \mathbb{Z},\ x_{ij} \in X \subset \mathbb{R}^{N \times T \times C} \}$$

Where X is the dataset, N is the dataset axis, T is the temporal sequence and C represents the (x, y, z) coordinates.

Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		5-Fold Mean		C	Kernel	Outliers	P.E.
Train	Valid	Train	Valid	Train	Valid	Train	Valid	Train	Valid	Train	Valid				
0.997	0.927	0.995	0.937	0.996	0.933	0.995	0.937	0.996	0.932	0.996	**0.933**	7	RBF	Keep	Yes
0.996	0.938	0.996	0.922	0.996	0.934	0.996	0.937	0.996	0.926	0.996	0.931	7	RBF	Keep	No
0.997	0.913	0.997	0.932	0.996	0.922	0.997	0.926	0.996	0.928	0.997	0.924	7	RBF	Global	Yes
0.997	0.922	0.995	0.922	0.997	0.929	0.996	0.936	0.997	0.926	0.997	0.927	7	RBF	Global	No
0.991	0.805	0.99	0.786	0.99	0.785	0.99	0.786	0.994	0.788	0.99	0.79	2	Linear	Keep	Yes
0.997	0.758	0.996	0.758	0.997	0.764	0.997	0.761	0.996	0.747	0.997	0.758	10	Linear	Global	No

Table 2: SVM Hyperparameter Tuning
P.E. indicates if *positional embeddings* are added.

2.2 Hybrid AutoEncoder SVM (HAESVM)

An alternative approach to solve the temporal issue mentioned above is to reduce the features of the input data by mapping them into a **lower dimensional space**. Generally, when data is recorded it contains a quantity of noise that is seen by models and this can present itself as a barrier during optimization. Moreover, if a dataset has many dimensions this can be another issue called ***Curse of Dimensionality***, which imposes the requirement of large amounts of data in order to capture relationships in the high dimensional space which has sparse data.

According to Siddique, Sakib, and Rahman (2019), training a classifier on top of a lower representation of the given data can improve the performance for classical models such as KNN and SVM. In addition, the recordings present in this dataset may contain floating-point errors, noise because of user movement and, the space-time axes $(150, 3)$ of a single sample might be a cause for *Curse of Dimensionality*. As an idea

of improvement over the previous classifier, I modeled and trained an AutoEncoder, using convolutional layers, to map the input data to a reduced feature space of length 128. After this reconstruction stage, a SVM is trained over the bottleneck to solve the task at hand. This architecture can be seen in the following diagram:

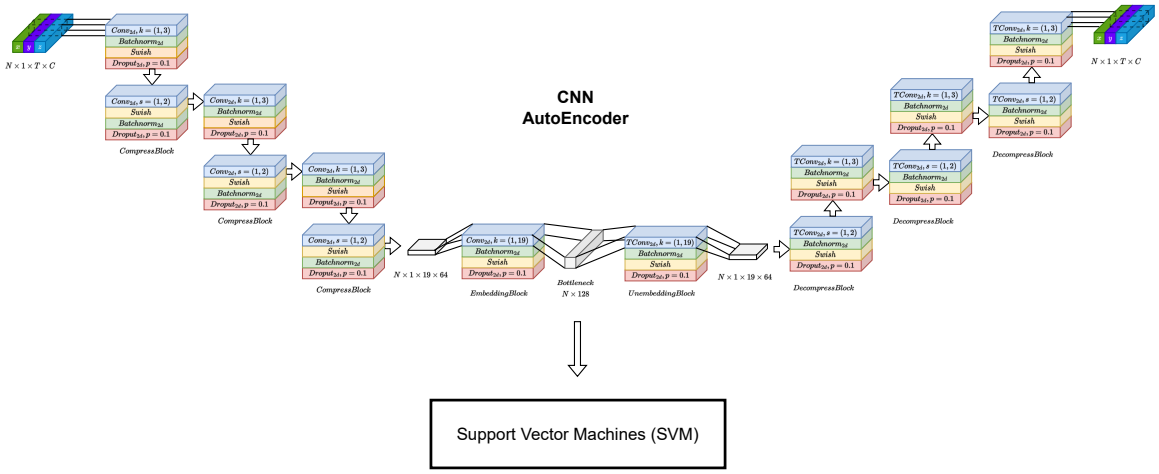

Figure 7: Hybrid AutoEncoder SVM Architecture Diagram

The AutoEncoder architecture is made up of three compression blocks, that reduce the temporal dimension using strided convolutions and double the amount of channels, followed by one embedding layer that flattens the feature maps and lastly, the inverse operations which restore the temporal axis. All of the inner block operations are read in top-to-bottom order.

The model presented above includes some architectural choices which facilitate a smoother and shorter training process leading to a well represented bottleneck which can be used for further prediction using the SVM:

- The *Swish* activation function is a better alternative to *ReLU* according to the empirical performance shown by Ramachandran, Zoph, and Le (2017).
- The temporal axis is the one reduced and reconstructed, while the coordinates are considered as the channels. This should lead the model to learn the behavioral changes of the users across multiple time points.
- The pooling and unpooling operations are given by strided and transposed convolutions in order to learn richer operations as the operations are not applied on an image but on data sequences.
- 2D spatial dropout (Tompson, Goroshin, Jain, LeCun, & Bregler, 2015) with a small probability, $p = 0.1$, is used to improve generalization performance, because the channels become more correlated as the temporal axis is reduced.

The objective of the **first stage** is to minimize the *Mean Squared Error (MSE)* or the *Mean Absolute Error (MAE)* loss functions. In order to speed up the training process and keep it stable, I used various techniques:

- Mini-batches of size 32 seems to be the *optimum* value between training time and loss convergence. Lower values led to very slow training times and higher values showed convergence issues which required many more epochs.
- The *Adam* optimizer (Kingma & Ba, 2017) is used with the *learning rate* $= 10^{-3}$, and hits a pleateau around 50 epochs with no improvement given by reducing the learning rate, neither by lowering it initially or reducing it on the go.
- To have a better starting point during model training I used *Xavier Normal Initialization* (Glorot & Bengio, 2010) and zeroed out the bias terms.

The first stage of the training process can be observed in the following graphs, on a single fold from 5-Fold Cross Validation, and the results of hyperparameter tuning can be observed in Table 3.

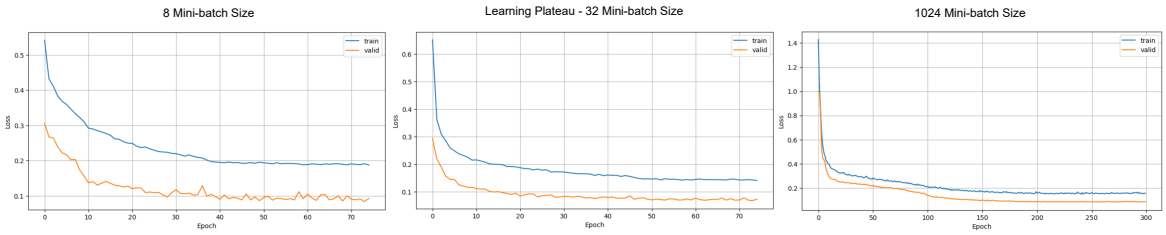

Figure 8: 1^{st} Stage Training Process Across a Single Fold from 5-Fold Cross Validation
The 8 mini-batch run indicates a slow learning process with validation spikes along the epochs. The 32 mini-batch run shows a plateau that's hit around epoch 50 even though the learning rate was decreased gradually to 10^{-7}. The final 1024 mini-batch experiment showed a smooth but long learning process requiring many more epochs to hit a good enough value.

Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		5-Fold Mean			C	K	Outliers	P.E.	Loss	CH	BK	D
T	V	T	V	T	V	T	V	T	V	R	T	V								
0.95	0.88	0.95	0.87	0.95	0.89	0.95	0.88	0.95	0.89	**0.03**	**0.953**	0.881	7	RBF	Keep	No	MSE	64	128	0
0.95	0.87	0.96	0.87	0.96	0.89	0.95	0.88	0.95	0.88	0.03	0.953	0.878	7	RBF	Keep	No	MSE	16	128	0
0.94	0.89	0.94	0.87	0.94	0.88	0.96	0.88	0.94	0.87	0.08	0.946	0.879	10	RBF	Keep	Yes	MSE	16	128	0.1
0.94	0.88	0.94	0.87	0.94	0.86	0.94	0.88	0.94	0.87	0.06	0.943	0.873	100	RBF	Global	No	MSE	32	150	0.1
0.98	0.92	0.98	0.92	0.98	0.91	0.97	0.91	0.97	0.92	0.15	0.977	**0.919**	7	RBF	Keep	Yes	MAE	32	1024	0.1

Table 3: HAESVM Hyperparameter Tuning
Some abbreviations were needed in order to fit the table. D - indicates dropout. CH - highlights the starting amount of convolutional filters applied. K - represents the used kernel in SVM. BK - shows the size of the final embeddings. T, V - are symbols used to indicate the training and validation stages. R - is the reconstruction loss obtained on the validation set.

After extensive trials no improvement was obtained even though the CNN AutoEncoder was able to reconstruct the data quite well even with the low embedding size of 128. This might indicate that the SVM model might not be suited well enough to train on the newer representation.

2.3 Temporal Convolutional Neural Network (TCNN)

In contrast to the previous approach, which used a learned representation and trained a classical algorithm over it, one step forward is to train a model **end-to-end** over the dataset. This would imply a restriction on the network layers to both learn a temporal representation and predict what user gave that behavior. Starting from the previous idea, I kept various elements but changed some aspects, which have proved to lead to great success in the end:

- Similarly to the AutoEncoder, the representation of the data is built using 2D convolutional layers that are being applied across the **temporal axis** while having the **coordinates in the channel dimension**. This seems to be the more intuitive approach as the convolutional operations have strong local biases and implicitly encode local changes pertaining to the user behavior. By using kernels of at least size 3, the network can cover a great part of the sequence in deeper layers because of its gradually increasing *receptive field*.
- Max pooling operations with $kernel\_size = (1, 3)$ are used instead of strided convolutions, which reduce some of the overhead of learning more complex operations, while exposing only the more important changes in user behavior to further layers.
- Instead of learning a separate classifier over the flattened feature maps, given by the convolutional backend, four fully-connected layers are added on top, and a final layer that predicts logits which are then transformed to class probabilities using the *softmax* activation. This approach makes use of the *shared representations* concept by having a single *deeper* network.
- 2D spatial dropout, which drops entire channels, is replaced by the usual dropout operation which zeroes out certain activations with a given probability.
- More activation functions are considered: $ReLU$, $LeakyReLU$ and *Swish*.

- Stronger regularization is applied by using some of the following:
  - Dropout with $p = 0.3$ or higher;
  - Weight decay between $[2 * 10^{-4},\ 2 * 10^{-5}]$;
  - Addition of small Gaussian noise values: $\hat{X} = X + \phi_{noise},\ \phi_{noise} \sim \mathcal{N}(0,\ 10^{-3^2})$;
- Similar observations regarding the optimization process are kept:
  - The Adam optimizer is used with lr such as $2 * 10^{-4}$ and periodic reduction of its value, starting from an $epoch = 30$ and decreasing exponentially every $step\_size = 10$ by a factor $\gamma = 8 * 10^{-1}$;
  - Mini-bacthes of size 32 offer good time / convergence tradeoff;
  - *Xavier Normal Initialization* is used;

The architecture of the **Temporal Convolutional Neural Network (TCNN)** can be inspected in the following diagram:

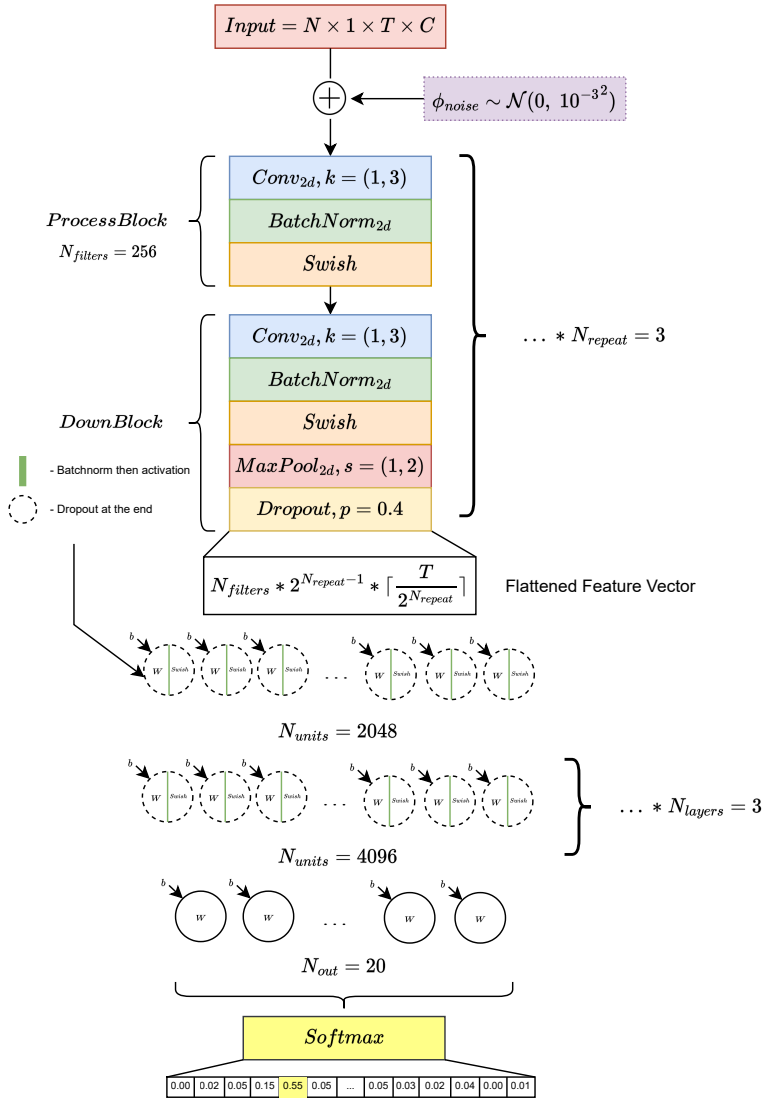

Figure 9: Temporal Convolutional Neural Network (TCNN) Architecture Diagram
The architecture uses small $(1, 3)$ kernels to extract user related-information from the recordings and starts from $N_{filters} = 256$ channels and doubles them for $N_{repeat} - 1 = 2$ times, while reducing the dimensionality of the input. This offers a rich set of feature maps, that are then flattened to a 9128 feature vector, which is classified by the multiple fully-connected layers. The network is **strongly regularized** by using dropout values up to $p = 0.4$ at every layer as depicted. Moreover, placing the *batch normalization* before the activation functions led to much better results.

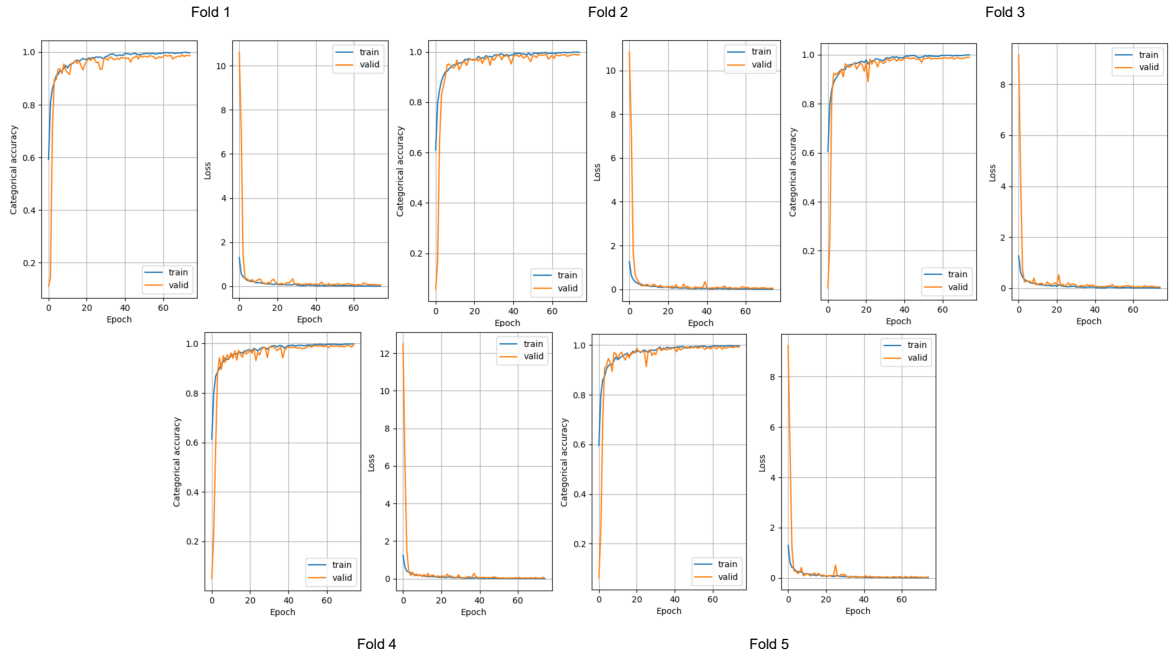

Figure 10: 5-Fold Cross Validation Using the Hyperparams with the **Highest Kaggle Private Score**
The model starts with a low training loss and the validation loss shortly follows in the next few epochs, which can attributed to the *Xavier Normal Initialization* and the accomodation to the high dropout value. The two metrics closely follow each other with minimal variance. Even though there are some spikes in the validation these minimize gradually as the **model converges** and the *learning rate is reduced significantly*.

Kaggle		5-Fold CV Mean		f_a	$N_{filters}$	K_{size}	N_{units}	N_{epochs}	σ_{Noise}	L_{rate}	W_{decay}	Outliers	D	MB
Public	Private	Train	Valid											
0.971	**0.9705**	0.997	0.989	Swish	256	3	2048	75	None	$2 * 10^{-4}$	None	Keep	0.4	48
0.975	0.9697	**0.999**	0.993	LeakyReLU	128	3	1024	150	None	$2 * 10^{-4}$	$2 * 10^{-4}$	Keep	0.3	32
0.964	0.9625	0.998	0.991	LeakyReLU	128	3	1024	120	None	10^{-3}	$2 * 10^{-4}$	Global	0.3	32
0.959	0.9517	0.998	0.989	LeakyReLU	128	3	1024	150	10^{-1}	$2 * 10^{-4}$	$2 * 10^{-4}$	Keep	0.3	32
0.968	0.9672	0.997	**0.994**	ReLU	72	3	512	150	None	$2 * 10^{-4}$	$1 * 10^{-4}$	Keep	0.3	16

Table 4: TCNN Hyperparameter Tuning
For brevity reasons, MB indicates the mini-batch size, f_a represents the activation function and K_{size} is the filter size for convolutional operations.

From the results obtained it seems that a higher number of filters and units with **strong regularization** leads to better results on the **private kaggle dataset** while having a marginally lower local validation value than the other approaches. Similarly to this approach, the last entry in the table shows that reducing the number of filters and units has a close but slightly weaker effect. Moreover, **stopping earlier** the training process at epoch 75 offered better generalization. In contrast to these approaches, the second row offers the best result on the **public kaggle dataset**, yet this actually led to *overfitting* because of the lack of stronger regularization and a high amount of epochs that did not increase the validation accuracy. Finally, the addition of *Gaussian Noise* did not prove useful and the improved accuracy by removing the global outliers might actually be caused due to the lower number of epochs used instead.

2.4 Attention-based TCNN (ATCNN)

Recent advances in sequence models (Vaswani et al., 2017) have shown great potential in focusing certain features in the data and avoiding others as they might represent noise or irelevant elements for the task at hand. Moreover, this focus-based mechanism translated well into visual tasks such as Georgescu et al. (2022) or Esser, Rombach, and Ommer (2020), with varying implementations for generating the attention maps. Wang, Girshick, Gupta, and He (2017) shows the general formulation of this process which can be used as a general recipe on various tasks. One intuition for using such a formulation on the current task

is that the attention-based mechanism might recognize **global behaviors** that might be missed by the convolutional layers which have stronger implicit local-bias. Considering this, I applied a *self-attention single-headed mechanism* on the data $(N, 1, T, C)$ along the temporal dimension in multiple layers.

Another improvement over the previous architecture is the addition of *skip-connections* (He, Zhang, Ren, & Sun, 2015) in order to facilitate better learning curves and prevent the issue of *vanishing gradients* due to the deeper nature of the architecture. Furthermore, these added connections can aid the network with the **reusal** of previously seen feature maps, in order to propagate the information forward, as suggested by Huang, Liu, and Weinberger (2016). Despite these improvements, the computational cost increases and the network is slower to train on consumer-grade GPUs. To overcome this issue, I used 1×1 2D convolutional operations, as indicated by Szegedy et al. (2014) and Lin, Chen, and Yan (2013), to reduce the number of channels by powers of 2. This led to shorter experimentation time and even increased results.

Many of the previous observations and optimization ideas are kept with the following contrasting points of view:

- Strided max pooling operations are **removed** in order to preserve the sequential dimension of the data to be exploited multiple times by the *self-attention mechanism*.
- Dropout operations are replaced with the 2D spatial variant, which drops entire channels similar to the way it was used in Section 2.2. This is done because the channels become highly correlated due to the added attentions.
- The flattening of the feature maps is replaced with **GlobalMaxPooling** that collapses the temporal dimension by extracting only the more important characteristics and exposing them to further layers.
- The optimizers used are either Adam or its variant AdamW (Loshchilov & Hutter, 2017), which is said to give better generalization due to its decoupled weight decay.
- The *learning_rate* is decayed every $step\_size = 40$ by a factor of $\gamma = 2 * 10^{-1}$.

The arhictecture for the Attention-based Temporal Convolutional Neural Network is illustrated in Figure 11 with varying structural aspects as hyperparameters which can be adjusted.

Kaggle		5-Fold CV Mean		f_a	$N_{filters}$	K_{size}	N_{units}	N_{epochs}	Optim	L_{rate}	W_{decay}	D	BK
Public	Private	Train	Valid										
0.946	**0.9465**	0.875	0.942	Swish	256	3	512	100	Adam	$2 * 10^{-4}$	$3 * 10^{-4}$	0.3	4
0.941	0.9457	**0.995**	**0.988**	Swish	512	3	1024	100	AdamW	$2 * 10^{-4}$	$2 * 10^{-4}$	0.3	16
0.935	0.9342	0.993	**0.988**	Swish	256	3	1024	85	AdamW	$2 * 10^{-4}$	$2 * 10^{-4}$	0.3	16

Table 5: ATCNN Hyperparameter Tuning
BK represents the bottleneck factor which reduces the number of channels used in the *inner modules*: $N_{inner_i} = \frac{N_{filters_i}}{BK}$, where i indicates the respective layer, as the number of channels is doubled periodically as shown in the diagram.

Despite the modified network architecture with the added *self-attention mechanism* and the *skip-connections*, no improvements were obtained in terms of classification accuracy over the previous model, and instead the network tends to overfit the given training set. Due to time constraints, further architectural adjustments and optimizations are left as future work but, the network seems to be a strong starting point for more experiments due to its ability to adjust to complex sequences.

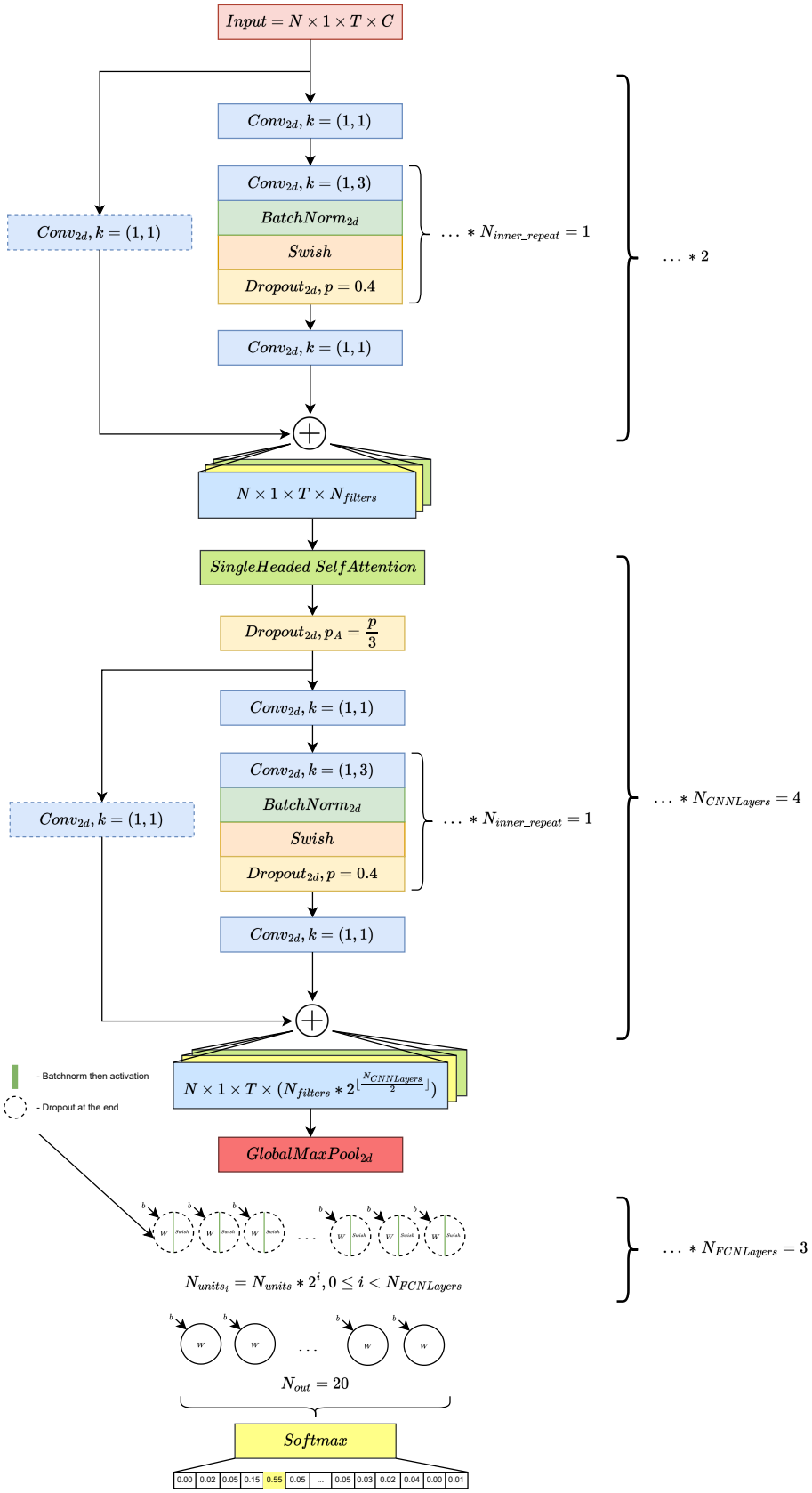

Figure 11: Attension-based Temporal Convolutional Neural Network (ATCNN) Architecture Diagram
In the CNN part of the network, the number of *channels* is doubled for every odd layer. The *skip-connection* may contain a 1×1 $Conv_{2d}$ operation in case the number of channels is doubled at the end of that layer.

3 Results

In the course of this paper four architectures with gradually increasing complexity and transferable ideas were presented: SVM, HAESVM, TCNN and ATCNN. Across the results seen so far, the end-to-end deep neural networks gave the best performance both locally and on the public & private kaggle datasets, with two architectures hitting a **top-10** place and one of them **winning the competition** with various hyperparams. To get a better overview of relative performance between the models, Table 6 presents the aggregated results obtained on a 5-Fold Cross Validation run, with the same *seed*, along with the Kaggle scores where present. Moreover, in Figure 12 the algorithms are also fit on the first 7000 training entries and confusion matrices are reported for the last 2000 entries, which is considered as a validation subset. The model hyperparameters are the ones used in previous tables and are chosen according to the established metrics given in following order:

1. Highest accuracy score on the private Kaggle subset;
2. Highest accuracy score on the public Kaggle subset;
3. Highest mean validation accuracy score on the previous 5-Fold CVs;

Model	Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		5-Fold CV Mean		Kaggle	
	Train	Valid	Train	Valid	Train	Valid	Train	Valid	Train	Valid	Train	Valid	Public	Private
SVM	0.996	0.925	0.995	0.93	0.996	0.924	0.995	0.931	0.996	0.928	0.996	0.928		
HAESVM	0.977	0.921	0.976	0.918	0.981	0.91	0.974	0.917	0.976	0.927	0.978	0.918		
TCNN	0.996	0.989	0.998	0.986	0.998	0.99	0.999	0.992	0.998	0.991	0.998	**0.99**	**0.971**	**0.9705**
ATCNN	0.853	0.923	0.837	0.936	0.907	0.962	0.879	0.949	0.935	0.978	0.882	0.949	0.941	0.945

Table 6: 5-Fold Cross Validation with the Best Model Hyperparams
The TCNN model showed the best performance followed by its *attention*-based variant.

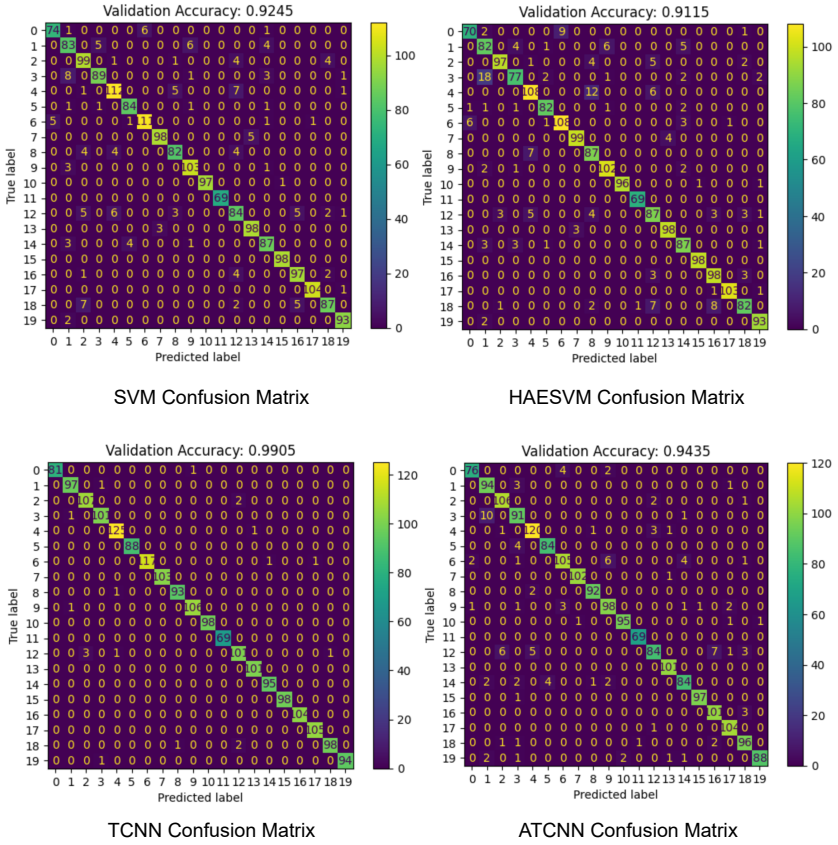

Figure 12: Confusion Matrices Obtained on the Last 2000 Samples
The models and their hyperparams are the same as in Table 6, and are trained on the first 7000 entries and the reported confusion matrices are computed on a validation set formed with the last 2000 samples.

Model	5-Fold CV		Scaling	Axes Collapsed	Out Shape
	Train	Valid			
SVM	1	0.909	None		
SVM	1	0.906	Standard Scaling	N	$T \times C$
SVM	1	0.906	Standard Scaling	N, T	C
SVM	0.836	0.622	MinMax Scaling	N	$T \times C$
SVM	0.781	0.216	MinMax Scaling	N, T	C
SVM	0.173	0.146	L2 Norm	T	$N \times C$
SVM	0.519	0.564	L2 Norm	C	$N \times T$
HAESVM	0.993	0.927	None		
HAESVM	0.994	0.922	Standard Scaling	N	$T \times C$
HAESVM	0.995	0.928	Standard Scaling	N, T	C
HAESVM	0.984	0.617	MinMax Scaling	N	$T \times C$
HAESVM	0.969	0.25	MinMax Scaling	N, T	C
HAESVM	0.746	0.391	L2 Norm	T	$N \times C$
HAESVM	0.86	0.714	L2 Norm	C	$N \times T$
TCNN	0.99	0.985	None		
TCNN	0.989	0.987	Standard Scaling	N	$T \times C$
TCNN	0.989	0.987	Standard Scaling	N, T	C
TCNN	0.979	0.753	MinMax Scaling	N	$T \times C$
TCNN	0.99	0.33	MinMax Scaling	N, T	C
TCNN	0.987	0.957	L2 Norm	T	$N \times C$
TCNN	0.984	0.942	L2 Norm	C	$N \times T$

Table 7: Data Scaling Methods for Various Models
Each model is trained with the same hyperparameters across identical 5-Folds, the only element that differs is the scaling method applied.

References

Chang, C.-C., & Lin, C.-J. (2011). Libsvm. *ACM Transactions on Intelligent Systems and Technology*, *2*(3), 1–27. DOI: 10.1145/1961189.1961199

Esser, P., Rombach, R., & Ommer, B. (2020). Taming transformers for high-resolution image synthesis. *CoRR*, *abs/2012.09841*. Retrieved from https://arxiv.org/abs/2012.09841

Georgescu, M.-I., Ionescu, R. T., Miron, A.-I., Savencu, O., Ristea, N.-C., Verga, N., & Khan, F. S. (2022). *Multimodal multi-head convolutional attention with various kernel sizes for medical image super-resolution.* arXiv. Retrieved from https://arxiv.org/abs/2204.04218 DOI: 10.48550/ARXIV.2204.04218

Glorot, X., & Bengio, Y. (2010). *Understanding the difficulty of training deep feedforward neural networks.* Retrieved from https://www.researchgate.net/publication/215616968_Understanding_the_difficulty_of_training_deep_feedforward_neural_networks

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*. Retrieved from http://arxiv.org/abs/1512.03385

Huang, G., Liu, Z., & Weinberger, K. Q. (2016). Densely connected convolutional networks. *CoRR*, *abs/1608.06993*. Retrieved from http://arxiv.org/abs/1608.06993

Kingma, D. P., & Ba, J. (2017, Jan). *Adam: A method for stochastic optimization.* Retrieved from https://arxiv.org/abs/1412.6980

Lin, M., Chen, Q., & Yan, S. (2013, Dec). *Network in network.* Retrieved from https://arxiv.org/abs/1312.4400v1

Loshchilov, I., & Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, *abs/1711.05101*. Retrieved from http://arxiv.org/abs/1711.05101

Radu. (2022). *Smartphone user identification.* Kaggle. Retrieved from https://kaggle.com/competitions/pml-2022-smart

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. *CoRR*, *abs/1710.05941*. Retrieved from http://arxiv.org/abs/1710.05941

Shashua, A. (2009, Apr). *Introduction to machine learning.* Retrieved from https://arxiv.org/abs/0904.3664v1

Siddique, M. A. B., Sakib, S., & Rahman, M. A. (2019). Performance analysis of deep autoencoder and NCA dimensionality reduction techniques with knn, ENN and SVM classifiers. *CoRR*, *abs/1912.05912*. Retrieved from http://arxiv.org/abs/1912.05912

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., ... Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, *abs/1409.4842*. Retrieved from http://arxiv.org/abs/1409.4842

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., & Bregler, C. (2015). Efficient object localization using convolutional networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr.2015.7298664

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. Retrieved from http://arxiv.org/abs/1706.03762

Wang, X., Girshick, R. B., Gupta, A., & He, K. (2017). Non-local neural networks. *CoRR*, *abs/1711.07971*. Retrieved from http://arxiv.org/abs/1711.07971