LSTM Fully Convolutional Networks for Time Series Classification

Sion Chun[1] (sc3791), Michelle Twan[1] (mt3565), George Weale[2] (gmw2143)
[1]Computer Science, [2]Biomedical Engineering
Columbia University
New York, USA

***Abstract*—Time series classification is critical in many fields, including healthcare and finance, necessitating models that effectively capture both short-term and long-term temporal patterns. In this project, we replicate, evaluate, and enhance the Long Short-Term Memory Fully Convolutional Network (LSTM-FCN) architecture originally proposed by Karim, Majumdar, and Darabi in their influential work "LSTM Fully Convolutional Networks for Time Series Classification" [1]. Utilizing TensorFlow, we developed implementations of both the standard LSTM-FCN and its attention-enhanced variant, ALSTM-FCN, and evaluated their performance using the UCR Time Series Classification Archive [2]. Although we successfully reconstructed the model architectures, the classification accuracies achieved did not reach the levels reported in the original study. To bridge this performance gap, we created our own model with changes in the architecture, where we are able to match the performance of this paper on many data sets. This report outlines our implementation process, highlights the technical challenges encountered, and presents our model performance results. We look at the reasons for the observed discrepancies and propose strategies for future work aimed at enhancing model performance to more closely align with the original findings.**

I. Introduction

Time series classification has been extensively studied, with many different approaches proposed to capture temporal dependencies and feature representations. Traditional methods like Shapelets [5] and Time Series Forests [6] rely on hand-crafted features, which can be limiting in capturing complex patterns. Deep learning models, particularly Fully Convolutional Networks (FCNs) [8] and Long Short-Term Memory (LSTM) networks [3], have shown significant improvements by learning feature representations directly from raw data.

Recent advancements incorporate attention mechanisms to enhance model interpretability and performance [11]. For instance, the Attention LSTM-FCN (ALSTM-FCN) model integrates multi-head attention with LSTM layers, allowing the network to focus on the most relevant temporal segments [1]. Additionally, Squeeze-and-Excitation (SE) blocks [3] have been employed to recalibrate channel-wise feature responses, further improving model performance and robustness.

This project builds upon these developments by replicating and enhancing the LSTM-FCN and ALSTM-FCN architectures, incorporating SE blocks to investigate their impact on classification accuracy and model interpretability.

Time series classification has emerged as a critical task in machine learning, with applications in healthcare [5] and finance [6]. Traditional methods often rely on hand-crafted feature extraction and pre-processing, which can be computationally expensive and require domain expertise [7]. Recently, deep learning approaches such as Fully Convolutional Networks (FCNs) [8] and Long Short-Term Memory (LSTM) networks [3] have demonstrated state-of-the-art performance in end-to-end time series classification [9].

In this work, we replicate and analyze the LSTM-FCN architecture proposed in the original paper, which augments FCN models with LSTM or Attention-LSTM blocks for improved time series classification performance [1]. The proposed model leverages temporal convolutions for efficient feature extraction and LSTMs to capture long-term temporal dependencies [10]. Additionally, the attention mechanism enables more interpretability by identifying the most relevant parts of the time series data.

We implemented the LSTM-FCN and ALSTM-FCN models using TensorFlow [12] and evaluated them on UCR datasets [2]. Despite successfully replicating the model architecture, our results did not achieve the accuracy levels reported in the original paper.

II. Summary of the Original Paper

A. Methodology

The original paper combines FCNs [8] with LSTM networks to improve time series classification. FCNs extract local features using temporal convolutions, but they lack mechanisms to capture long-term dependencies. To address this, LSTM blocks are integrated to model temporal relationships effectively [14].

They used an attention mechanism [11] to aid the LSTMs by focusing on key regions of the input sequence, improving interpretability [15]. The resulting models–LSTM-FCN and Attention LSTM-FCN (ALSTM-FCN)–achieve state-of-the-art performance with minimal pre-processing and a nominal increase in model size.

1) Temporal Convolutional Networks: The paper uses a temporal convolutional network (TCN) which processes time series data $\mathbf{X}_t \in \mathbb{R}^{F_0}$ at each time step t, where F_0 is the feature dimension and $0 < t \leq T$. Layer l includes T_l time steps, and the class labels are $y_t \in \{1, ..., C\}$ for C classes [16].

The l-th layer applies 1D filters $\mathbf{W}^{(l)} \in \mathbb{R}^{F_l \times d \times F_{l-1}}$ with biases $\mathbf{b}^{(l)} \in \mathbb{R}^{F_l}$. The activation $\hat{E}_t^{(l)}$ is computed as:

$$\hat{E}_{i,t}^{(l)} = f\left(b_i^{(l)} + \sum_{t'=1}^{d} \left\langle W_{i,t'}^{(l)}, E_{\cdot,t+d-t'}^{(l-1)} \right\rangle\right), \tag{1}$$

where $f(\cdot)$ is the Rectified Linear Unit (ReLU) [12]. Each TCN block consists of convolution, batch normalization, and an activation function (ReLU or Parametric ReLU).

2) Recurrent Neural Networks: Recurrent Neural Networks (RNNs) model temporal relationships by recurrently connecting units across time steps [16]. The hidden state $\mathbf{h}_t$ at step t is updated as:

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{I}\mathbf{x}_t), \tag{2}$$

where $\tanh$ is the activation function, $\mathbf{W}$ is the recurrent weight matrix, and $\mathbf{I}$ is the input projection matrix.

The hidden state $\mathbf{h}_t$ produces predictions using:

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{W}\mathbf{h}_{t-1}), \tag{3}$$

where softmax normalizes outputs over C classes [9]. For stacked RNNs, the hidden states at layer l are updated as:

$$\mathbf{h}_t^l = \sigma(\mathbf{W}\mathbf{h}_{t-1}^l + \mathbf{I}\mathbf{h}_t^{l-1}), \tag{4}$$

where σ is the logistic sigmoid function [15].

3) Long Short-Term Memory: LSTMs address the vanishing gradient issue in RNNs using gating mechanisms [3]. At time step t, an LSTM maintains a hidden state $\mathbf{h}_t$ and memory vector $\mathbf{m}_t$, computed as:

$$\mathbf{g}^u = \sigma(\mathbf{W}^u\mathbf{h}_{t-1} + \mathbf{I}^u\mathbf{x}_t), \quad \mathbf{g}^f = \sigma(\mathbf{W}^f\mathbf{h}_{t-1} + \mathbf{I}^f\mathbf{x}_t),$$
$$\mathbf{g}^o = \sigma(\mathbf{W}^o\mathbf{h}_{t-1} + \mathbf{I}^o\mathbf{x}_t), \quad \mathbf{g}^c = \tanh(\mathbf{W}^c\mathbf{h}_{t-1} + \mathbf{I}^c\mathbf{x}_t),$$
$$\mathbf{m}_t = \mathbf{g}^f \odot \mathbf{m}_{t-1} + \mathbf{g}^u \odot \mathbf{g}^c, \quad \mathbf{h}_t = \tanh(\mathbf{g}^o \odot \mathbf{m}_t),$$

where σ is the sigmoid function, $\odot$ denotes element-wise multiplication, and $\mathbf{W}$, $\mathbf{I}$ represent the recurrent and input weight matrices [3].

4) Attention Mechanism: The attention mechanism generates a context vector $\mathbf{c}_i$ for a target sequence by selectively focusing on key parts of the input sequence [11]. Given encoder annotations $\mathbf{h}_j$, the context vector $\mathbf{c}_i$ is:

$$\mathbf{c}_i = \sum_{j=1}^{T_x} \alpha_{ij}\mathbf{h}_j, \tag{6}$$

where the attention weight α_{ij} is:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{7}$$

and $e_{ij} = a(s_{i-1}, \mathbf{h}_j)$ is the alignment score [11]. The alignment model a, implemented as a feedforward network, computes a soft alignment to ensure gradients propagate effectively [9].

5) LSTM Fully Convolutional Network: Their architecture consists of two main components: a Temporal Convolutional Block (TCB) for feature extraction and an LSTM block for capturing long-term temporal dependencies [1].

Their TCB is made up of three stacked 1D convolutional layers with filter sizes of 128, 256, and 128. Each convolutional layer is followed by Batch Normalization [13] (momentum = 0.99, epsilon = 0.001) and a ReLU activation function. Residual connections are added to improve gradient flow [8]. Global average pooling [10] is applied at the end of the convolutional block to reduce the output dimensionality and aggregate the extracted features.

Following the convolutional block, the model applies a dimension shuffle to transpose the temporal and feature dimensions of the input [9]. This allows the LSTM block to process the output of the TCB as a multivariate time series. This LSTM block can consist of either a standard LSTM layer or an Attention LSTM layer [11].

A rather high dropout layer [14] (rate = 0.8) is applied to the LSTM output to mitigate overfitting as the model was trained over 2000 epochs. The output of the LSTM block is then flattened and concatenated with the global average pooling output of the TCB. The concatenated output is passed through a dense layer with a softmax activation function, producing the final class probabilities [10]. This combination of temporal convolutional features and LSTM-based temporal modeling allows the LSTM-FCN architecture to efficiently capture local and global dependencies within the time series data.

The overall architecture is illustrated in Fig. 1.

B. Key Results of the Original Paper

The LSTM-FCN model [1] achieved the highest classification accuracy across 85 UCR datasets, outperforming baseline models such as FCNs [8], pure LSTMs [3], Shapelets [5], and Time Series Forests [6] shown in Fig. 2. The FCN branch, with multiple convolutional layers and varying kernel sizes, captured temporal patterns for class discrimination [7], while the LSTM branch captured long-term dependencies critical for classification [3]. The attention layer in the LSTM branch improved focus on relevant time steps, increasing interpretability and performance [11]. Global average pooling in the FCN branch reduced parameters, keeping efficient training and inference [10]. Ablation studies confirmed significant contributions from both the FCN and LSTM components [1]. The average arithmetic rank in Fig. 2 indicates the superiority of the original paper's proposed models over the existing state-of-the-art models [4]. Fine-tuning improved accuracy but required additional training time [13].

III. Implementation - Replication of Paper Methods

A. Objectives and Technical Challenges

The primary objective of this project is to replicate the architecture and results presented in the original paper for the LSTM Fully Convolutional Network (LSTM-FCN) and its Attention LSTM (ALSTM-FCN) variant [1]. We aim to

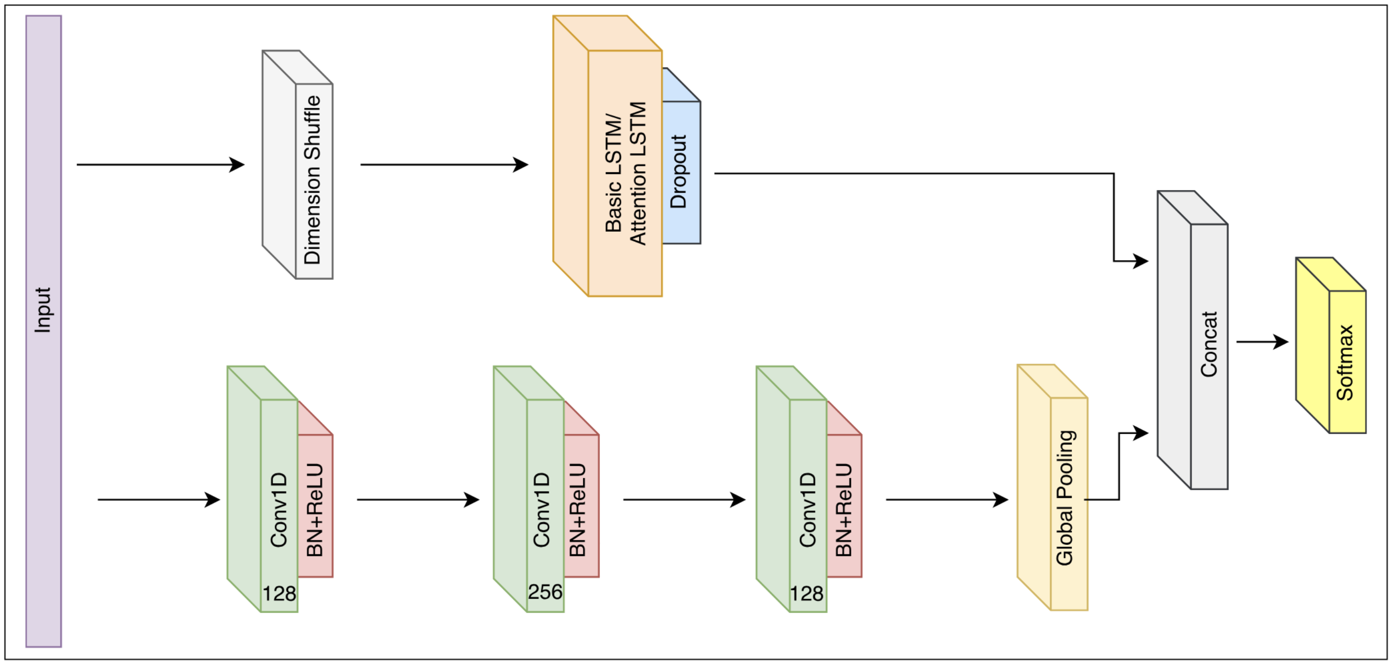


Fig. 1. The original paper LSTM-FCN architecture. LSTM cells can be replaced by Attention LSTM cells to construct the ALSTM-FCN architecture [1].

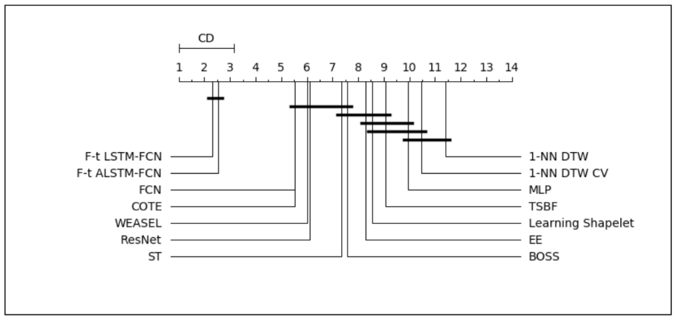


Fig. 2. Original paper's critical difference diagram of the arithmetic means of the ranks [1].

implement the LSTM-FCN model architecture as described in the paper, reproduce the reported classification accuracies on benchmark time series datasets [2], and validate the model's ability to use temporal convolution and attention mechanisms for improved performance [8].

The paper provided high-level details of the model architecture but lacked explicit specifications for data pre-processing steps and training configurations. Any assumptions that we had to make for these steps may have contributed to performance discrepancies [7]. Although we normalized the data sets following standard practices [10], subtle differences in scaling, encoding, or handling missing values could have affected the model's convergence and accuracy [13].

B. Data

For our experiments, we used the same dataset as the original paper: the 85 time series datasets from the University of California, Riverside (UCR) Time Series Classification Archive [2]. The UCR archive is a comprehensive and widely recognized repository that has a large range of datasets, including healthcare [5], finance [6], environmental monitoring [7], and more. This diversity ensures that our model is evaluated across different types of time series patterns and complexities, showing its generalizability and robustness [4]. By using the UCR archive, we ensure that our results are comparable to the original paper and also relevant within the broader research community [2].

C. LSTM.py

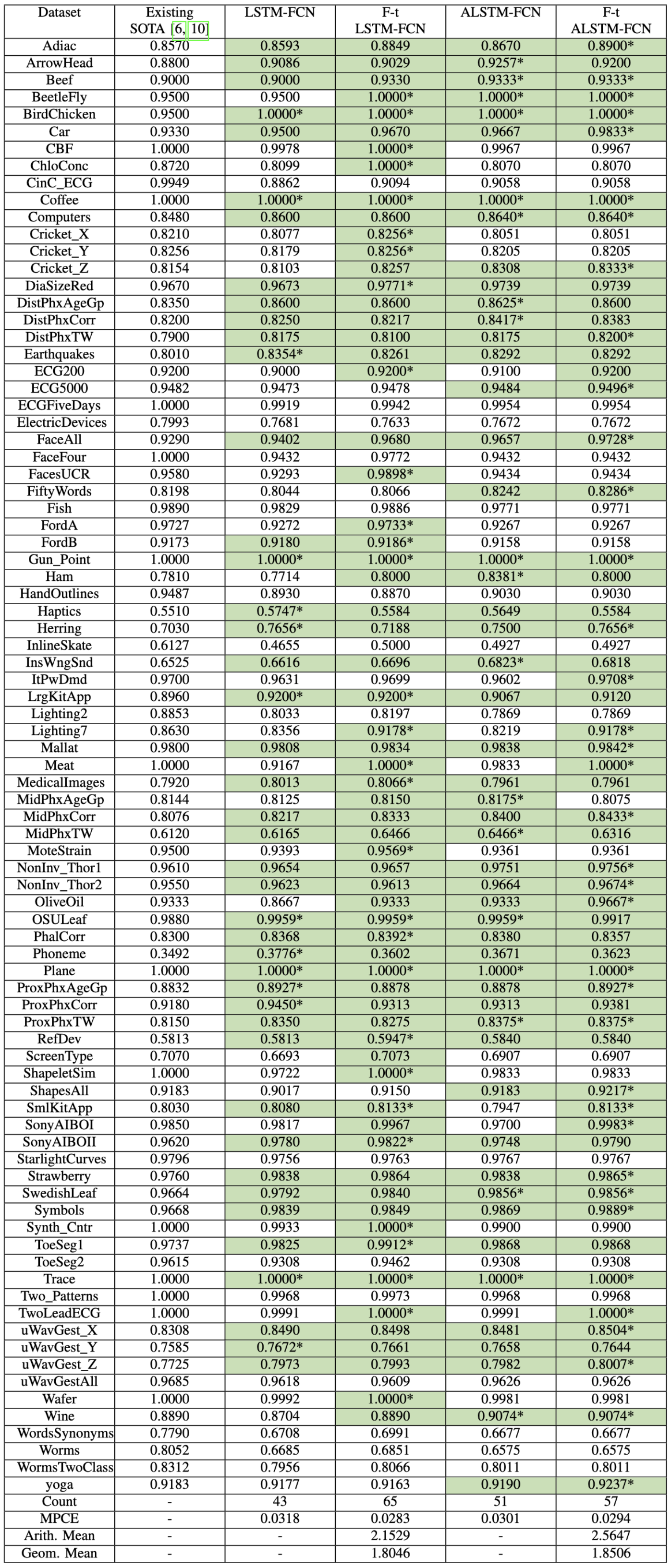
Our implementation of the LSTM Cell is designed to give flexibility and control over the LSTM's internal mechanisms compared to standard library implementations [3].

a) Custom LSTM Cell: Our 'LSTMCell' class defines a custom LSTM cell by extending the 'tf.keras.Model' class [12]. It manages the hidden state $\mathbf{h}_t$ and the cell state $\mathbf{c}_t$ at each timestep [3]. The cell explicitly calculates and updates the input, forget, cell, and output gates, which regulate the flow of information and enable our LSTM cell to capture long-term dependencies in sequential data [14].

b) Initialization and Building: We set up the necessary parameters for the number of units and initializers for the kernel, recurrent kernel, and bias [12]. Our 'build' method dynamically constructs the weight matrices based on the input shape:

$$\mathbf{W} \in \mathbb{R}^{F \times 4U}, \quad \mathbf{U} \in \mathbb{R}^{U \times 4U}, \quad \mathbf{b} \in \mathbb{R}^{4U},$$

where F is the input feature dimension and U is the number of LSTM units [3].



Dataset	Existing SOTA [6, 10]	LSTM-FCN	F-t LSTM-FCN	ALSTM-FCN	F-t ALSTM-FCN
Adiac	0.8570	0.8593	0.8849	0.8670	0.8900*
ArrowHead	0.8800	0.9086	0.9029	0.9257*	0.9200
Beef	0.9000	0.9000	0.9330	0.9333*	0.9333*
BeetleFly	0.9500	0.9500	1.0000*	1.0000*	1.0000*
BirdChicken	0.9500	1.0000*	1.0000*	1.0000*	1.0000*
Car	0.9330	0.9500	0.9670	0.9667	0.9833*
CBF	1.0000	0.9978	1.0000*	0.9967	0.9967
ChloConc	0.8720	0.8099	1.0000*	0.8070	0.8070
CinC_ECG	0.9949	0.8862	0.9094	0.9058	0.9058
Coffee	1.0000	1.0000*	1.0000*	1.0000*	1.0000*
Computers	0.8480	0.8600	0.8600	0.8640*	0.8640*
Cricket_X	0.8210	0.8077	0.8256*	0.8051	0.8051
Cricket_Y	0.8256	0.8179	0.8256*	0.8205	0.8205
Cricket_Z	0.8154	0.8103	0.8257	0.8308	0.8333*
DiaSizeRed	0.9670	0.9673	0.9771*	0.9739	0.9739
DistPhxAgeGp	0.8350	0.8600	0.8600	0.8625*	0.8600
DistPhxCorr	0.8200	0.8250	0.8217	0.8417*	0.8383
DistPhxTW	0.7900	0.8175	0.8100	0.8175	0.8200*
Earthquakes	0.8010	0.8354*	0.8261	0.8292	0.8292
ECG200	0.9200	0.9000	0.9200*	0.9100	0.9200
ECG5000	0.9482	0.9473	0.9478	0.9484	0.9496*
ECGFiveDays	1.0000	0.9919	0.9942	0.9954	0.9954
ElectricDevices	0.7993	0.7681	0.7633	0.7672	0.7672
FaceAll	0.9290	0.9402	0.9680	0.9657	0.9728*
FaceFour	1.0000	0.9432	0.9772	0.9432	0.9432
FacesUCR	0.9580	0.9293	0.9898*	0.9434	0.9434
FiftyWords	0.8198	0.8044	0.8066	0.8242	0.8286*
Fish	0.9890	0.9829	0.9886	0.9771	0.9771
FordA	0.9727	0.9272	0.9733*	0.9267	0.9267
FordB	0.9173	0.9180	0.9186*	0.9158	0.9158
Gun_Point	1.0000	1.0000*	1.0000*	1.0000*	1.0000*
Ham	0.7810	0.7714	0.8000	0.8381*	0.8000
HandOutlines	0.9487	0.8930	0.8870	0.9030	0.9030
Haptics	0.5510	0.5747*	0.5584	0.5649	0.5584
Herring	0.7030	0.7656*	0.7188	0.7500	0.7656*
InlineSkate	0.6127	0.4655	0.5000	0.4927	0.4927
InsWngSnd	0.6525	0.6616	0.6696	0.6823*	0.6818
ItPwDmd	0.9700	0.9631	0.9699	0.9602	0.9708*
LrgKitApp	0.8960	0.9200*	0.9200*	0.9067	0.9120
Lighting2	0.8853	0.8033	0.8197	0.7869	0.7869
Lighting7	0.8630	0.8356	0.9178*	0.8219	0.9178*
Mallat	0.9800	0.9808	0.9834	0.9838	0.9842*
Meat	1.0000	0.9167	1.0000*	0.9833	1.0000*
MedicalImages	0.7920	0.8013	0.8066*	0.7961	0.7961
MidPhxAgeGp	0.8144	0.8125	0.8150	0.8175*	0.8075
MidPhxCorr	0.8076	0.8217	0.8333	0.8400	0.8433*
MidPhxTW	0.6120	0.6165	0.6466	0.6466*	0.6316
MoteStrain	0.9500	0.9393	0.9569*	0.9361	0.9361
NonInv_Thor1	0.9610	0.9654	0.9657	0.9751	0.9756*
NonInv_Thor2	0.9550	0.9623	0.9613	0.9664	0.9674*
OliveOil	0.9333	0.8667	0.9333	0.9333	0.9667*
OSULeaf	0.9880	0.9959*	0.9959*	0.9959*	0.9917
PhalCorr	0.8300	0.8368	0.8392*	0.8380	0.8357
Phoneme	0.3492	0.3776*	0.3602	0.3671	0.3623
Plane	1.0000	1.0000*	1.0000*	1.0000*	1.0000*
ProxPhxAgeGp	0.8832	0.8927*	0.8878	0.8878	0.8927*
ProxPhxCorr	0.9180	0.9450*	0.9313	0.9313	0.9381
ProxPhxTW	0.8150	0.8350	0.8275	0.8375*	0.8375*
RefDev	0.5813	0.5813	0.5947*	0.5840	0.5840
ScreenType	0.7070	0.6693	0.7073	0.6907	0.6907
ShapeletSim	1.0000	0.9722	1.0000*	0.9833	0.9833
ShapesAll	0.9183	0.9017	0.9150	0.9183	0.9217*
SmlKitApp	0.8030	0.8080	0.8133*	0.7947	0.8133*
SonyAIBOI	0.9850	0.9817	0.9967	0.9700	0.9983*
SonyAIBOII	0.9620	0.9780	0.9822*	0.9748	0.9790
StarlightCurves	0.9796	0.9756	0.9763	0.9767	0.9767
Strawberry	0.9760	0.9838	0.9864	0.9838	0.9865*
SwedishLeaf	0.9664	0.9792	0.9840	0.9856*	0.9856*
Symbols	0.9668	0.9839	0.9849	0.9869	0.9889*
Synth_Cntr	1.0000	0.9933	1.0000*	0.9900	0.9900
ToeSeg1	0.9737	0.9825	0.9912*	0.9868	0.9868
ToeSeg2	0.9615	0.9308	0.9462	0.9308	0.9308
Trace	1.0000	1.0000*	1.0000*	1.0000*	1.0000*
Two_Patterns	1.0000	0.9968	0.9973	0.9968	0.9968
TwoLeadECG	1.0000	0.9991	1.0000*	0.9991	1.0000*
uWavGest_X	0.8308	0.8490	0.8498	0.8481	0.8504*
uWavGest_Y	0.7585	0.7672*	0.7661	0.7658	0.7644
uWavGest_Z	0.7725	0.7973	0.7993	0.7982	0.8007*
uWavGestAll	0.9685	0.9618	0.9609	0.9626	0.9626
Wafer	1.0000	0.9992	1.0000*	0.9981	0.9981
Wine	0.8890	0.8704	0.8890	0.9074*	0.9074*
WordsSynonyms	0.7790	0.6708	0.6991	0.6677	0.6677
Worms	0.8052	0.6685	0.6851	0.6575	0.6575
WormsTwoClass	0.8312	0.7956	0.8066	0.8011	0.8011
yoga	0.9183	0.9177	0.9163	0.9190	0.9237*
Count	-	43	65	51	57
MPCE	-	0.0318	0.0283	0.0301	0.0294
Arith. Mean	-	-	2.1529	-	2.5647
Geom. Mean	-	-	1.8046	-	1.8506

Fig. 3. Original paper's performance comparison between proposed models and the rest [1].

c) Forward Pass: Given the input $\mathbf{x}_t$ and the previous states $\mathbf{h}_{t-1}$ and $\mathbf{c}_{t-1}$, it computes the following:

$$\mathbf{z} = \mathbf{x}_t\mathbf{W} + \mathbf{h}_{t-1}\mathbf{U} + \mathbf{b}, \quad (19)$$

$$\mathbf{i}, \mathbf{f}, \mathbf{g}, \mathbf{o} = \mathrm{split}(\mathbf{z}), \quad \mathbf{i} = \sigma(\mathbf{i}),\ \mathbf{f} = \sigma(\mathbf{f}),\ \mathbf{g} = \tanh(\mathbf{g}),\ \mathbf{o} = \sigma(\mathbf{o}), \quad (20)$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}, \quad (21)$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t), \quad (22)$$

where $\sigma(\cdot)$ denotes the sigmoid activation function and $\odot$ represents element-wise multiplication [3]. These equations govern the gating mechanisms that control information flow within the LSTM cell.

d) LSTM Model: We added a custom 'LSTMCell' within a functional model for time-series prediction. It uses the 'tf.keras.layers.RNN' layer to process input sequences and generates outputs through a Dense layer with a specified activation function [10].

$$\mathbf{y} = \mathrm{Dense}(\mathbf{h}_T), \quad (23)$$

where $\mathbf{h}_T$ is the final hidden state from the LSTM, and Dense is a fully connected layer that projects the LSTM outputs into the desired number of output classes using our specified activation function [10].

D. *attention_lstm.py*

In our 'attention_lstm.py', we combine multi-head attention with an LSTM layer [11]. This allows the model to identify and focus on important features within the input sequence, helping both performance and interpretability [9].

a) Attention LSTM Architecture: The 'AttentionLSTM' class extends the 'tf.keras.layers.Layer' [12] and uses multi-head attention mechanisms to change the traditional LSTM's [11]. The key components are the query, key, and value projections, scaled dot-product attention, and the combination of attention heads [9].

b) Initialization: The 'AttentionLSTM' is initialized with a specified number of LSTM units and attention heads [11]. The dimensionality of each head is determined by:

$$\text{head\_dim} = \frac{\text{lstm\_units}}{\text{num\_heads}}.$$

Trainable dense layers project the input into query, key, and value vectors:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V,$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{F\times U}$ are the weight matrices for queries, keys, and values, respectively [11].

c) Forward Pass: Our forward pass method performs these operations:

1) **Linear Transformations:** The input tensor $\mathbf{X} \in \mathbb{R}^{B\times T\times F}$ is projected into queries, keys, and values:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V. \quad (24)$$

2) **Splitting Heads:** The projected tensors are reshaped and transposed to be able use multiple attention heads:

$$\mathbf{Q}_{\text{heads}} = \text{reshape}(\mathbf{Q}, (B, T, H, d_k)) \\ \rightarrow \text{transpose}(\mathbf{Q}_{\text{heads}}, (B, H, T, d_k)), \quad (25)$$

where H is the number of heads and d_k is the dimensionality per head [9].

3) **Scaled Dot-Product Attention:** For each head, attention scores are computed and scaled:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}, \quad (26)$$

ensuring stability through the scaling factor $\sqrt{d_k}$ [9].

4) **Concatenation and Transformation:** Outputs from all attention heads are concatenated and passed through a final dense layer to merge the heads:

$$\mathbf{A} = \text{concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_H)\mathbf{W}_O, \quad (27)$$

where $\mathbf{W}_O \in \mathbb{R}^{H \cdot d_k \times U}$ is the output projection matrix [9].

5) **Context Vector:** The multi-head attention output is averaged across the time dimension to produce a context vector:

$$\mathbf{c}_{\text{attn}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{A}_t, \quad (28)$$

which gives us the most important features of the input sequence [11].

d) Integration with LSTM: The context vector $\mathbf{c}_{\text{attn}}$ is integrated with the LSTM's hidden states to help our model focus on important regions of the input sequence [11]. This allows the model to prioritize informative features, improving classification accuracy and interpretability [9].

E. temporal_conv.py

The Temporal Convolutional Block (TCB) extracts features from input time series data using 1D convolutional layers [8]. Each layer applies a series of filters across the temporal dimension, capturing local dependencies [7]. Batch Normalization [13] and ReLU activation [12] follow each convolution to stabilize training and introduce non-linearity [14]. The TCB processes an input tensor $\mathbf{X} \in \mathbb{R}^{B \times T \times F}$, where B is the batch size, T is the number of time steps, and F is the number of features [2]. The block consists of three Conv1D layers with filters $[128, 256, 128]$, a kernel size of 8, and a stride of 1. Residual connections [8] are optionally applied to mitigate the vanishing gradient problem and accelerate training [1]. The output of each Conv1D layer is computed as:

$$\mathbf{X}^{(l)} = \text{ReLU}\left(\text{BN}\left(\text{Conv1D}\left(\mathbf{X}^{(l-1)}, \mathbf{W}^{(l)}\right)\right)\right), \quad (8)$$

where $\mathbf{W}^{(l)}$ are the convolutional filters for the l-th layer, BN denotes Batch Normalization with momentum 0.99 and epsilon 0.001, and ReLU is the activation function [12]. When the residual connections are enabled, the input $\mathbf{X}^{(l-1)}$ is added to the output $\mathbf{X}^{(l)}$, ensuring consistent dimensions. A 1x1 convolution adjusts mismatched dimensions:

$$\mathbf{X}_{\text{res}} = \text{Conv1D}\left(\mathbf{X}^{(l-1)}, \mathbf{W}_{\text{res}}\right), \quad (9)$$

where $\mathbf{W}_{\text{res}}$ projects $\mathbf{X}^{(l-1)}$ to the desired output shape [8]. The final TCB output aggregates features while preserving the temporal structure, making it a suitable input for subsequent LSTM or Attention LSTM layers [1]. This block improves local feature extraction efficiency while maintaining a small parameter footprint [7].

F. model.py

Our LSTM Fully Convolutional Network (LSTM-FCN) model combines a Temporal Convolutional Block (TCB) with an LSTM or Attention LSTM module to perform time series classification [1]. The architecture uses temporal convolutions for local feature extraction and LSTMs for capturing long-term temporal dependencies. We also use global average pooling [10] and dimension shuffling [9] to make sure there is compatibility between each of the convolutional and LSTM components.

The model takes as input a time series tensor $\mathbf{X} \in \mathbb{R}^{B \times T \times F}$, where B is the batch size, T is the number of time steps, and F is the feature dimension [2].

a) Temporal Convolutional Block (TCB): As described earlier, the Temporal Convolutional Block extracts local features using three stacked Conv1D layers with filter sizes of $[128, 256, 128]$ and a kernel size of 8. Each convolutional layer is followed by Batch Normalization [13] and a Rectified Linear Unit (ReLU) activation [12].

The output of the TCB is then passed through a Global Average Pooling layer to aggregate the features across the temporal dimension:

$$\mathbf{X}_{\text{global}} = \text{GAP}\left(\mathbf{X}_{\text{TCB}}\right), \quad (10)$$

where GAP denotes Global Average Pooling [10].

b) Dimension Shuffle: To allow the LSTM block to process the convolutional outputs as multivariate input, a dimension shuffle operation transposes the tensor [9]. Given input $\mathbf{X} \in \mathbb{R}^{B \times T \times F}$, the transposed tensor $\mathbf{X}_{\text{shuffled}}$ is:

$$\mathbf{X}_{\text{shuffled}} = \text{Permute}(2, 1)\left(\mathbf{X}\right), \quad (11)$$

where Permute rearranges the tensor's dimensions [9].

c) LSTM and Attention LSTM: The LSTM block as stated previously process the transposed tensor $\mathbf{X}_{\text{shuffled}}$ to capture long-term dependencies [3]. Two versions of the LSTM block are implemented:

- **Standard LSTM:** A custom LSTM cell processes the input sequence and outputs hidden states across all time steps [3].
- **Attention LSTM:** The Attention LSTM adds multi-head attention into the LSTM module to focus on key regions of the input sequence [11]. Each attention head projects the input into query, key, and value vectors, enhancing

the model's ability to identify important features as mentioned previously [9].

The LSTM output is passed through a dropout layer [14] with a rate of 0.8 to mitigate overfitting:

$$\mathbf{H} = \mathrm{Dropout}(0.8)\left(\mathbf{H}_{\mathrm{LSTM}}\right). \tag{12}$$

The LSTM output is flattened to prepare it for concatenation with the global average pooling output [10].

d) Final Concatenation and Classification.: The global average pooling output $\mathbf{X}_{\mathrm{global}}$ and the flattened LSTM output $\mathbf{H}$ are concatenated to form the final feature representation:

$$\mathbf{F} = \mathrm{Concatenate}\left(\mathbf{X}_{\mathrm{global}}, \mathbf{H}\right). \tag{13}$$

A dense layer with softmax activation produces the final class probabilities:

$$\mathbf{y} = \mathrm{softmax}\left(\mathbf{W}\mathbf{F} + \mathbf{b}\right), \tag{14}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the dense layer weights and biases, respectively [10].

IV. Implementation - New Model Methods

A. Objectives and Technical Challenges

While our primary goal of the project was to replicate the paper's findings, we were curious to add new more modern methods to improve prediction accuracy.

B. Data

We used the same dataset: the 85 time series datasets from the University of California, Riverside (UCR) Time Series Classification Archive [2].

C. SE_block.py

The Squeeze-and-Excitation (SE) block is a channel-wise attention mechanism that adaptively re-calibrates channel-wise feature responses by explicitly modeling inter-dependencies between channels. Introduced by Hu, Shen, and Sun [3], the SE block helps the representational power of a network by enabling it to perform dynamic channel-wise feature selection. This improves the network's ability to capture salient features.

The SE block can be mathematically represented as:

$$\mathbf{X}' = \mathbf{X} \odot \sigma\left(\mathbf{W}_2 \cdot \mathrm{ReLU}\left(\mathbf{W}_1 \cdot \mathrm{GAP}(\mathbf{X})\right)\right), \tag{15}$$

a) Architecture of the SE Block: The SE block comprises two main operations: Squeeze and Excitation. These operations are implemented as follows:

1) **Squeeze:** The squeeze operation aggregates feature maps across the temporal dimension to produce a channel descriptor. This is achieved using Global Average Pooling (GAP), which computes the average of each feature channel:

$$\mathbf{z} = \mathrm{GAP}(\mathbf{X}) = \frac{1}{T}\sum_{t=1}^{T} \mathbf{X}_t, \tag{16}$$

where $\mathbf{X} \in \mathbb{R}^{B\times T\times F}$ is the input tensor, B is the batch size, T is the number of time steps, and F is the number of feature channels.

2) **Excitation:** The excitation operation models the inter-dependencies between channels and generates channel-wise weights. This is implemented using two fully connected (Dense) layers with a bottleneck (reduction ratio) to reduce computational complexity:

$$\mathbf{u} = \mathrm{ReLU}(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1), \tag{17}$$

$$\mathbf{s} = \mathrm{Sigmoid}(\mathbf{W}_2\mathbf{u} + \mathbf{b}_2), \tag{18}$$

where $\mathbf{W}_1 \in \mathbb{R}^{\frac{F}{r}\times F}$ and $\mathbf{W}_2 \in \mathbb{R}^{F\times \frac{F}{r}}$ are the weight matrices for the first and second Dense layers, respectively, and r is the reduction ratio (set to 16). The output $\mathbf{s} \in \mathbb{R}^F$ represents the channel-wise scaling factors, showing the importance of each feature channel.

3) **Scale:** Our last step scales the original input tensor with the generated channel-wise weights:

$$\mathbf{X}' = \mathbf{X} \odot \mathbf{s}, \tag{19}$$

where $\odot$ is element-wise multiplication. This scaling operation emphasizes informative channels while suppressing less useful ones, thereby enhancing the network's discriminative ability.

b) Integration within the Model.: The SE block is integrated after a convolutional layer within the network architecture to help the feature representations before passing them to subsequent layers. By embedding the SE block, our network can emphasize more informative features and limit the less relevant ones.

D. model.py

The SE blocks were added into the TCB to recalibrate channel-wise feature responses. Residual connections were also added within the TCB to help better gradient flow and mitigate the vanishing gradient problem, expressed as:

$$\mathrm{Output} = \mathrm{Conv}(U) + U,$$

where $\mathrm{Conv}(U)$ represents the convolution operations applied to the input tensor U. This addition helps in training deeper networks by providing shortcut paths for gradients.

We added Batch Normalization layers immediately after each convolutional layer to stabilize and accelerate the training process. The Batch Normalization operation is defined as:

$$\mathrm{BN}(x) = \gamma\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta,$$

where μ and σ^2 are the mean and variance of the batch, γ and β are learnable scaling and shifting parameters, and ϵ is a small constant for numerical stability.

An additional intermediate Dense layer with 512 units and ReLU activation was incorporated before the final output layer to help the model's learning capacity:

$$\mathrm{Dense}_{\text{intermediate}} = \mathrm{ReLU}(W \cdot x + b),$$

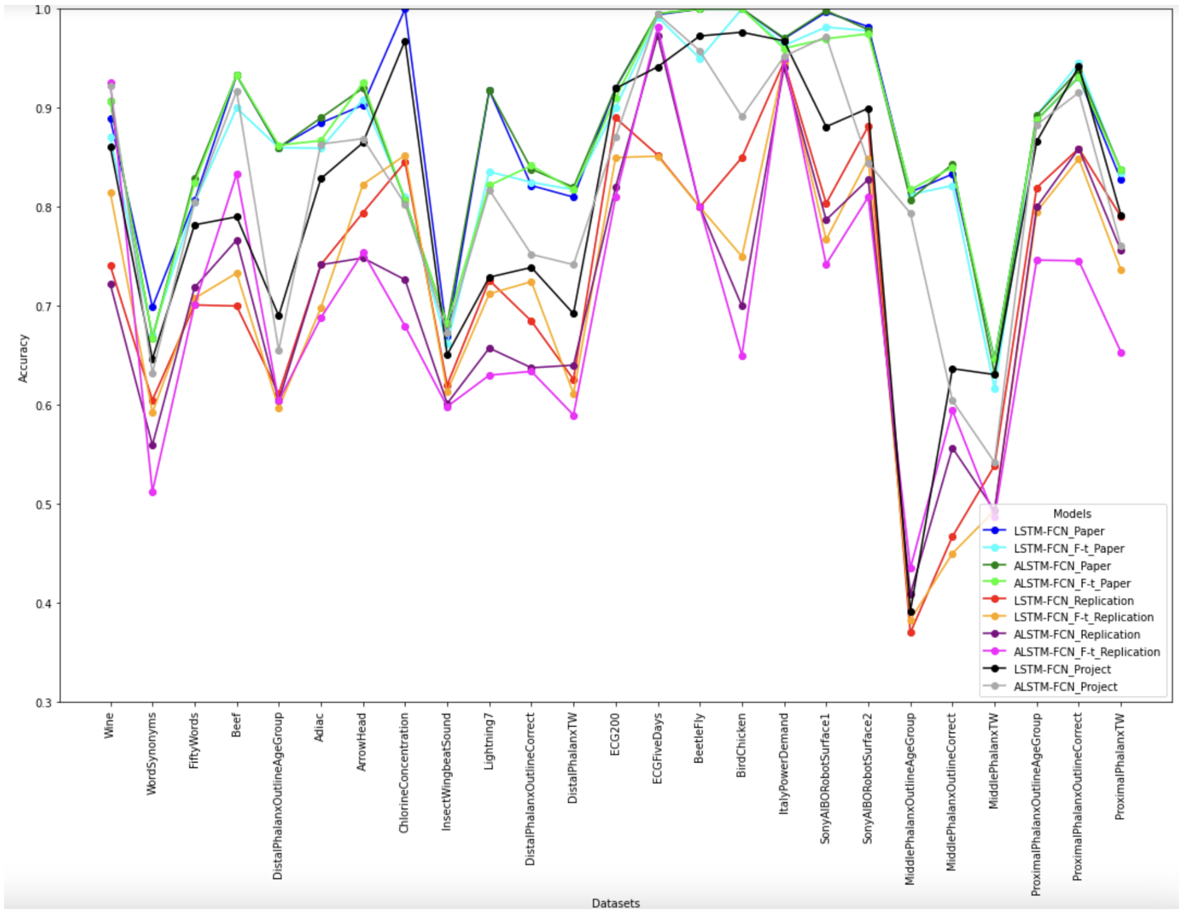


Fig. 4. Model performance across datasets[1].

where W and b are the weight matrix and bias vector, respectively. This layer increases the model's ability to learn complex representations prior to classification.

The model's output construction was changed to include intermediate processing steps before the final classification layer. The sequence now follows concatenation of global pooling and LSTM outputs, an intermediate Dense layer, Batch Normalization, Dropout, and the Dense output layer.

V. Training Algorithm Details

The training procedure for the LSTM-FCN and ALSTM-FCN models follows the methodology outlined in the original paper [1]. We train and evaluate the models on the 85 UCR Time Series Classification datasets [2]. Each dataset is loaded, preprocessed, and used for both model replication and fine-tuning experiments [4].

a) Data Pre-processing: The training and testing datasets are loaded from the UCR archive [2]. Class labels are balanced using a weighting scheme, where class weights are computed as:

$$w_c = \frac{n}{k \cdot n_c}, \tag{20}$$

where n is the total number of samples, k is the number of classes, and n_c is the number of samples for class c [7]. These

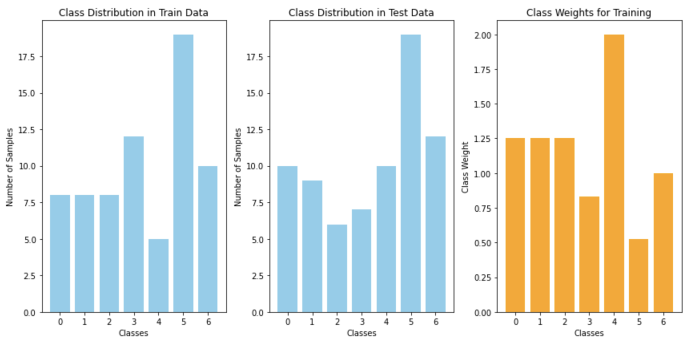


Fig. 5. Class Imbalance - Dataset Lightning7

weights helped mitigate class imbalance during training as shown in Fig. 5[13].

b) Model Training: We use the Adam optimizer [11] with a sparse categorical cross-entropy loss function [10]. The models are trained for 2000 epochs, as specified in the original paper [1], with an initial batch size of 128 [10]. A validation split of 20% is applied to monitor model performance during training [13]. The training process consists of two phases:

1) **Replication Phase:** Both LSTM-FCN and ALSTM-FCN models are trained without fine-tuning [1]. Results

are recorded for comparison [4].

2) **Fine-Tuning Phase:** The models are fine-tuned over $K = 5$ iterations, following the learning rate and batch size adjustments described in the paper [1]. The learning rate is updated as:

$$\eta_{i+1} = \eta_i \cdot \frac{1}{2^{1/3}}, \tag{21}$$

and the batch size is halved at each iteration [11].

c) Implementation and Callbacks.: To ensure convergence and improve generalization, we use callbacks during training [14]:

- **ReduceLROnPlateau:** Reduces the learning rate by a factor of 0.5 if the validation loss plateaus for 10 epochs, with a minimum learning rate of 10^{-4} [11].
- **EarlyStopping:** Stops training if the validation loss does not improve for 15 consecutive epochs, restoring the best model weights [14].

d) Model Evaluation.: After training, the models are evaluated on the test set. Predictions are made using:

$$\hat{y} = \text{argmax}(\mathbf{p}), \tag{22}$$

where $\mathbf{p}$ is the softmax probability output [10]. The accuracy is computed as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}}. \tag{18}$$

e) Training Pipeline: The complete training pipeline consists of:

1) Loading each dataset and pre-processing inputs [2].
2) Training LSTM-FCN and ALSTM-FCN models with and without fine-tuning [1].
3) Saving results, including accuracy scores and model configurations, for further analysis [4].

The models are trained iteratively across all datasets, ensuring consistency with the original paper's methodology [1].

VI. Results

A. Project Results

Model performance for both the LSTM-FCN model and the ALSTM-FCN model is graphed in Fig. 4 for each dataset. The performance for both our replication model and modified model (labeled as ”project” in the figure) follow similar trends. Out of the replication models and the modified models, the replication models performed worse on average.

In Fig. 6, we can view the performance in detail by observing the accuracies of our replication of the original paper's architecture and our modified model are showcased.

Both the replication models (for LSTM-FCN and ALSTM-FCN) had unpredictable accuracies across the datasets, with the LSTM-FCN alone scoring as low as 0.37 on the MidPhxAgeGp dataset and as high as 0.98 on the CBF dataset. Fine-tuning interestingly rarely improved these replication models, and the fine-tuned replication models often performed worse than the replication models without fine-tuning.

The use of attention in the ALSTM-FCN model seemed to be unpredictable in its ability to improve the LSTM-FCN model. For example, it improved upon the LSTM-FCN model's accuracy on the ECGFiveDays dataset by 0.12 points, but worsened the accuracy of the LSTM-FCN model on the CinC ECG dataset by 0.39 points.

Our modified models (labeled as ”New Model” in the figure) showed large improvements across all datasets. The results of fine-tuning were still unpredictable, though we do see some large jumps in improvement with the addition of fine-tuning with the new model, one of them being a 0.40 point increase on the MidPhxAgeGp dataset.

The modified models may be performing better because of a combination of different settings we implemented. The SE attention mechanism improved the model's ability to focus on important feature channels. Lowering the dropout rate balanced regularization and information retention. The intermediate dense layer increased the model's capacity to learn complex features. Batch normalization stabilized training and accelerated convergence. Residual connections enhanced gradient flow and allowed for deeper network training. Dropout in LSTM cells reduced overfitting by preventing neuron co-adaptation. With all these changes that were implemented, the modified LSTM-FCN and ALSTM-FCN models were able to learn better across the datasets.

B. Comparison of the Results Between the Original Paper and the Our Project

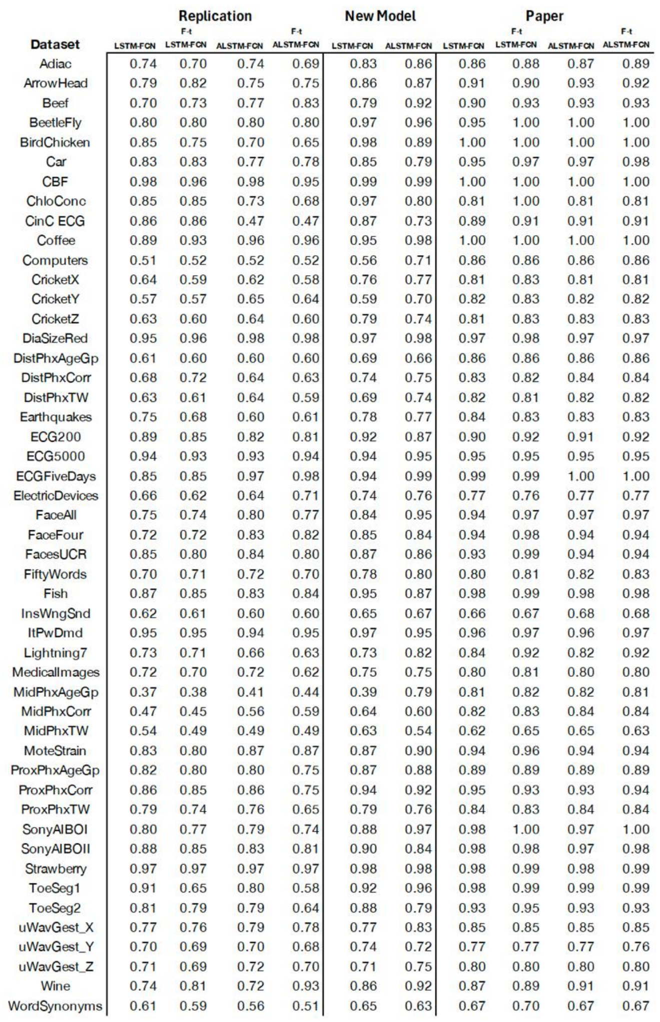
The replication models aimed to reproduce the performance of the original LSTM-FCN and ALSTM-FCN architectures as described in the paper. However, the results showed noticeable differences, with the replication models generally under-performing compared to the paper's reported scores.

In Fig. 4, we can compare our replication models and modified models with the original paper's models. The original paper's models for LSTM and ALSTM performed the best on average. Although our replicant models performed the worst on average, our modified models performed on par with the original paper's models on some datasets.

Accuracies between the results of our replication of the original paper's architecture, our improved model, and the original paper are compared in Fig. 6.

As noted in Fig. 4, our replication models performed the worst on average compared to the modified models and the original paper's models. The interesting part of the comparisons between these three main categories is the impact of fine-tuning. Fine-tuning generally improved the accuracies of the original paper's models. However, as mentioned in the previous subsection, fine-tuning generally worsened the accuracies of the replication models, while it generally increased the accuracies of our modified models. The original paper's models may be performing better both with and without fine-tuning because of pre-processing and different ways of addressing class imbalance.

Our modified model showed significant improvements across all datasets. Although it rarely achieved the same level



	Replication				New Model		Paper			
Dataset	LSTM-FCN	F-t LSTM-FCN	ALSTM-FCN	F-t ALSTM-FCN	LSTM-FCN	ALSTM-FCN	LSTM-FCN	F-t LSTM-FCN	ALSTM-FCN	F-t ALSTM-FCN
Adiac	0.74	0.70	0.74	0.69	0.83	0.86	0.86	0.88	0.87	0.89
ArrowHead	0.79	0.82	0.75	0.75	0.86	0.87	0.91	0.90	0.93	0.92
Beef	0.70	0.73	0.77	0.83	0.79	0.92	0.90	0.93	0.93	0.93
BeetleFly	0.80	0.80	0.80	0.80	0.97	0.96	0.95	1.00	1.00	1.00
BirdChicken	0.85	0.75	0.70	0.65	0.98	0.89	1.00	1.00	1.00	1.00
Car	0.83	0.83	0.77	0.78	0.85	0.79	0.95	0.97	0.97	0.98
CBF	0.98	0.96	0.98	0.95	0.99	0.99	1.00	1.00	1.00	1.00
ChloConc	0.85	0.85	0.73	0.68	0.97	0.80	0.81	1.00	0.81	0.81
CinC ECG	0.86	0.86	0.47	0.47	0.87	0.73	0.89	0.91	0.91	0.91
Coffee	0.89	0.93	0.96	0.96	0.95	0.98	1.00	1.00	1.00	1.00
Computers	0.51	0.52	0.52	0.52	0.56	0.71	0.86	0.86	0.86	0.86
CricketX	0.64	0.59	0.62	0.58	0.76	0.77	0.81	0.83	0.81	0.81
CricketY	0.57	0.57	0.65	0.64	0.59	0.70	0.82	0.83	0.82	0.82
CricketZ	0.63	0.60	0.64	0.60	0.79	0.74	0.81	0.83	0.83	0.83
DiaSizeRed	0.95	0.96	0.98	0.98	0.97	0.98	0.97	0.98	0.97	0.97
DistPhxAgeGp	0.61	0.60	0.60	0.60	0.69	0.66	0.86	0.86	0.86	0.86
DistPhxCorr	0.68	0.72	0.64	0.63	0.74	0.75	0.83	0.82	0.84	0.84
DistPhxTW	0.63	0.61	0.64	0.59	0.69	0.74	0.82	0.81	0.82	0.82
Earthquakes	0.75	0.68	0.60	0.61	0.78	0.77	0.84	0.83	0.83	0.83
ECG200	0.89	0.85	0.82	0.81	0.92	0.87	0.90	0.92	0.91	0.92
ECG5000	0.94	0.93	0.93	0.94	0.94	0.95	0.95	0.95	0.95	0.95
ECGFiveDays	0.85	0.85	0.97	0.98	0.94	0.99	0.99	0.99	1.00	1.00
ElectricDevices	0.66	0.62	0.64	0.71	0.74	0.76	0.77	0.76	0.77	0.77
FaceAll	0.75	0.74	0.80	0.77	0.84	0.95	0.94	0.97	0.97	0.97
FaceFour	0.72	0.72	0.83	0.82	0.85	0.84	0.94	0.98	0.94	0.94
FacesUCR	0.85	0.80	0.84	0.80	0.87	0.86	0.93	0.99	0.94	0.94
FiftyWords	0.70	0.71	0.72	0.70	0.78	0.80	0.80	0.81	0.82	0.83
Fish	0.87	0.85	0.83	0.84	0.95	0.87	0.98	0.99	0.98	0.98
InsWngSnd	0.62	0.61	0.60	0.60	0.65	0.67	0.66	0.67	0.68	0.68
ItPwDmd	0.95	0.95	0.94	0.95	0.97	0.95	0.96	0.97	0.96	0.97
Lightning7	0.73	0.71	0.66	0.63	0.73	0.82	0.84	0.92	0.82	0.92
MedicalImages	0.72	0.70	0.72	0.62	0.75	0.75	0.80	0.81	0.80	0.80
MidPhxAgeGp	0.37	0.38	0.41	0.44	0.39	0.79	0.81	0.82	0.82	0.81
MidPhxCorr	0.47	0.45	0.56	0.59	0.64	0.60	0.82	0.83	0.84	0.84
MidPhxTW	0.54	0.49	0.49	0.49	0.63	0.54	0.62	0.65	0.65	0.63
MoteStrain	0.83	0.80	0.87	0.87	0.87	0.90	0.94	0.96	0.94	0.94
ProxPhxAgeGp	0.82	0.80	0.80	0.75	0.87	0.88	0.89	0.89	0.89	0.89
ProxPhxCorr	0.86	0.85	0.86	0.75	0.94	0.92	0.95	0.93	0.93	0.94
ProxPhxTW	0.79	0.74	0.76	0.65	0.79	0.76	0.84	0.83	0.84	0.84
SonyAIBOI	0.80	0.77	0.79	0.74	0.88	0.97	0.98	1.00	0.97	1.00
SonyAIBOII	0.88	0.85	0.83	0.81	0.90	0.84	0.98	0.98	0.97	0.98
Strawberry	0.97	0.97	0.97	0.97	0.98	0.98	0.98	0.99	0.98	0.99
ToeSeg1	0.91	0.65	0.80	0.58	0.92	0.96	0.98	0.99	0.99	0.99
ToeSeg2	0.81	0.79	0.79	0.64	0.88	0.79	0.93	0.95	0.93	0.93
uWavGest_X	0.77	0.76	0.79	0.78	0.77	0.83	0.85	0.85	0.85	0.85
uWavGest_Y	0.70	0.69	0.70	0.68	0.74	0.72	0.77	0.77	0.77	0.76
uWavGest_Z	0.71	0.69	0.72	0.70	0.71	0.75	0.80	0.80	0.80	0.80
Wine	0.74	0.81	0.72	0.93	0.86	0.92	0.87	0.89	0.91	0.91
WordSynonyms	0.61	0.59	0.56	0.51	0.65	0.63	0.67	0.70	0.67	0.67

Fig. 6. Accuracy comparison between our replication, our improved model, and the original paper, F-t indicating fine-tuning[1].

of accuracy of the results in the original paper, it bridged much of the gap between the results of our replication model and the original paper's model.

C. *Discussion of Insights Gained*

Throughout the replication and enhancement of the LSTM-FCN and ALSTM-FCN architectures, our team learned several insights that helped our understanding of time series classification.

We learned that results can be significantly influenced data pre-processing and hyperparameter tuning. We still believe that simpler models demonstrated competitive performance and have more real world application; but we do see our more complex models have better performance.

We also learned that integrating Squeeze-and-Excitation (SE) blocks proved helpful in capturing inter-feature dependencies within multivariate time series data. The SE mechanism dynamically re-calibrates channel-wise feature responses, enhancing the model's focus on the most informative aspects of the input. By adjusting feature weights dynamically, SE blocks contributed to more robust and discriminative feature extraction, particularly in complex and high-dimensional datasets. This adaptability helped mitigate overfitting and enhanced generalization across diverse time series domains.

VII. Future Work

We think that integrating transformer-based layers [17] may improve the model's ability to capture long-range dependencies without the sequential limitations of RNNs. Applying transfer learning [19] and self-supervised learning [18] techniques could enhance feature representations and reduce training time, especially in domains with limited labeled data. Looking at advanced attention mechanisms [20] might refine the model's focus on relevant temporal segments, potentially increasing classification accuracy.

Addressing scalability and computational efficiency is also important. Implementing model compression techniques [21] such as pruning and quantization can make the models suitable for deployment in resource-constrained environments. Combining Graph Neural Networks [22] with LSTM-FCN could allow the model to better capture inter-feature relationships in multivariate time series data. Additionally, employing data augmentation and synthetic data generation methods [23] can improve the model's generalization by increasing the diversity of training samples.

Improving model interpretability through explainability frameworks [23] will make the models more transparent and trustworthy, particularly in sensitive applications like healthcare. Finally, evaluating the models on a wider range of real-world datasets will demonstrate their robustness and applicability across different domains. These directions can help enhance the performance, efficiency, and usability of LSTM-FCN models for time series classification.

VIII. Conclusion

We replicated the LSTM-FCN and ALSTM-FCN architectures for univariate time series classification based on Karim et al. [1]. We noticed that we were not able to obtain similar results with the same methods that were explicitly stated in the report. We created a new similar model where we introduced a Squeeze-and-Excitation (SE) attention mechanism, reduced the dropout rate, added an intermediate dense layer, applied batch normalization throughout, implemented residual connections, and added dropout within LSTM cells. These changes addressed overfitting, improved feature emphasis, and stabilized training.

Our modified models achieved higher classification accuracies compared to both the replication models across the UCR datasets, and they also were able to achieve the same performance as the original paper's models on some datasets. The SE attention mechanism and residual connections in the modified models contributed to performance improvements. Future work includes hyperparameter optimization, integrating transformer-based layers, exploring advanced attention mechanisms, applying transfer learning, improving scalability, enhancing interpretability, evaluating on more datasets, and combining LSTM-FCN with Transformers or Graph Neural Networks.

References

[1] Karim F, Majumdar S, Darabi H, Chen S. LSTM fully convolutional networks for time series classification. IEEE Access. 2017 Dec 4;6:1662-9.

[2] Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen AM, Batista G. The UCR Time Series Classification Archive. 2015.

[3] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–1780.

[4] Bagnall A, Lines J, Keogh E. A comprehensive evaluation of representations for time series classification. Data Min Knowl Discov. 2017;31(3):606–660.

[5] Read N, Schmidt M, Keogh E. Learning Shapelets for Time Series Classification. ICDM. 2011;4:417-424.

[6] Breiman L. Random forests. Mach Learn. 2001;45(1):5–32.

[7] Zhao R, Zhang S, Wang Y, Zhao G. Feature extraction techniques for time series classification: A survey. Inf Fusion. 2020;57:1–13.

[8] Wang Y, Zheng Y, Malinowski M, Yuan Y, Wang Y. Time series classification using fully convolutional networks. ICML. 2017;70:849–858.

[9] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. *Attention is All You Need*. NIPS. 2017;30:5998–6008.

[10] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15(1):1929-1958.

[11] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. ICLR. 2015.

[12] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.

[13] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML. 2015;37:448–456.

[14] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15:1929–1958.

[15] Elman JL. Finding structure in time. Cognitive Science. 1990;14(2):179–211.

[16] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–444.

[17] He K, Zhang X, Ren S, Sun J. *Momentum Contrast for Unsupervised Visual Representation Learning*. CVPR. 2020.

[18] Pan S J, Yang Q. *A Survey on Transfer Learning*. IEEE Transactions on Knowledge and Data Engineering. 2010;22(10):1345-1359.

[19] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu P J. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. JMLR. 2020;21(140):1-67.

[20] Frankle J, Carbin M. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. ICLR. 2019.

[21] Wu Z, Pan S, Chen F, Long G, Zhang C, Philip S Y. *A Comprehensive Survey on Graph Neural Networks*. IEEE Transactions on Neural Networks and Learning Systems. 2020.

[22] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. *Generative Adversarial Nets*. NIPS. 2014.

[23] Lundberg SM, Lee S I. *A Unified Approach to Interpreting Model Predictions*. NeurIPS. 2017.