https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

The Cryosphere Discussions
Open Access
EGU

Convolutional Neural Network and Long Short-Term Memory Models for Ice-Jam Prediction

Fatemehalsadat Madaeni[1], Karem Chokmani[1], Rachid Lhissou[1], Saied Homayouni[1], Yves Gauthier[1], and Simon Tolszczuk-Leclerc[2]

[1]INRS-ETE, Université du Québec, Québec City, G1K 9A9, Canada
[2]EMGeo Operations, Natural Resources Canada, Ottawa, K1S 5K2, Canada
Correspondence to: Fatemehalsadat Madaeni (Fatemehalsadat.Madaeni@ete.inrs.ca)

Abstract. In cold regions, ice-jam events result in severe flooding due to a rapid rise in water levels upstream of the jam. These floods threaten human safety and damage properties and infrastructures as the floods resulting from ice-jams are sudden. Hence, the ice-jam prediction tools can give an early warning to increase response time and minimize the possible corresponding damages. However, the ice-jam prediction has always been a challenging problem as there is no analytical method available for this purpose. Nonetheless, ice jams form when some hydro-meteorological conditions happen, a few hours to a few days before the event. The ice-jam prediction problem can be considered as a binary multivariate time-series classification. Deep learning techniques have been successfully applied for time-series classification in many fields such as finance, engineering, weather forecasting, and medicine. In this research, we successfully applied CNN, LSTM, and combined CN-LSTM networks for ice-jam prediction for all the rivers in Quebec. The results show that the CN-LSTM model yields the best results in the validation and generalization with F1 scores of 0.82 and 0.91, respectively. This demonstrates that CNN and LSTM models are complementary, and a combination of them further improves classification.

1 Introduction

Predicting ice-jam events gives an early warning of possible flooding, but there is no analytical solution to predict these events due to the complex interactions between involved hydro-meteorological variables. To date, a small number of empirical and statistical prediction methods that have been developed (threshold methods, multi-regression models, logistic regression models, and discriminant function analysis) for ice jams are site-specific with a high rate of false-positive errors (White, 2003). The numerical models developed for ice-jam prediction (e.g., ICEJAM (Flato and Gerard, 1986, cf.; Carson et al., 2011), RIVJAM (Beltaos, 1993), HEC-RAS (Brunner, 2002), ICESIM (Carson et al., 2001 and 2003), and RIVICE (Lindenschmidt, 2017)) show limitations in predicting ice-jam occurrence. This is because mathematical formulations in these models are complex which need many parameters that are often unavailable as they are challenging to measure in ice conditions. Hence, many simplifications corresponding to these parameters may degrade model accuracy (Shouyu & Honglan, 2005). A detailed overview of the previous models for ice-jam prediction based on hydro-meteorological data are presented in Madaeni et al. (2020).

Prediction of ice-jam occurrence can be considered a binary multivariate time-series classification (TSC) model when the time series of various hydro-meteorological variables (explained later) can be used to classify to jam or no jam. Time-series classification (particularly multivariate) has been widely used in various fields, including biomedical engineering, clinical prediction, human activity recognition, weather forecasting, and finance. Multivariate time-series provide more patterns and improve classification performance compared to univariate time-series (Zheng et al., 2016). Time-series classification is one of the most challenging problems in data mining and machine learning.

Most existing TSC methods are feature-based, distance-based, or ensemble methods (Cui et al., 2016). Feature extraction is challenging due to the difficulty of handcrafting useful features to capture intrinsic characteristics from time-series data (Karim et al., 2019; Zheng et al., 2014, June). Hence, distance-based methods work better in TSC (Zheng et al., 2014, June). Among the hundreds of developed methods for TSC, the leading classifier with the best performance was ensemble nearest neighbor with dynamic time warping (DTW) for many years (Fawaz et al., 2019, July; Karim et al., 2019).

In the k-nearest neighbors (kNN) classifier, the given test instance is classified by a majority vote of its k closest neighbors in the training data. The kNN needs all the data to make a prediction which requires high memory. Hence, it is computationally expensive and could be slow if the database is large, and sensitive to irrelevant features and the scale of the data. Furthermore, the number of neighbors to include in the algorithm should be wisely selected. The kNN is very challenging to be used for multivariate TSC. The dynamic time warping is a more robust alternative for Euclidean distance (the most widely used time-series distance measure) to measure the similarity between two given time series by searching for an optimal alignment (minimum distance) between them (Zheng et al., 2016). However, the combined kNN with DTW is time-consuming and inefficient for long multivariate time-series (Lin et al., 2012; Zheng et al., 2014, June). The traditional classification and classic data mining algorithms developed for TSC have high computational complexity or low prediction accuracy. This is due to the size and inherent complexity of time series, seasonality, noise, and feature correlation (Lin et al., 2012).

Deep learning is a type of neural network that uses multiple layers of nonlinear information to extract higher-level features from the input data. Although deep learning in recent years showed promising performance in various fields such as image and speech recognition, document classification, and natural language processing, only a few studies employed deep learning for TSC (Gu et al., 2018; Fawaz et al., 2019, July). Various studies show that deep neural networks significantly outperform the ensemble nearest neighbor with DTW (Fawaz et al., 2019, July). The main benefit of deep learning networks is automatic feature-extraction, which reduces the need for expert knowledge of the field and removes engineering bias in the classification task (Fawaz et al., 2019) as the probabilistic decision (e.g., classification) is taken by the network.

The most widely used deep neural networks for TSC are Multi-Layer Perceptron (MLP; i.e., fully connected deep neural networks), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) .

The application of CNNs for TSC has recently become more and more popular, and different types of CNN are being developed with superior accuracy performance for this purpose (e.g., Cui et al., 2016). Zheng et al. (2014, June) and Zheng et al. (2016) introduce a Multi-Channels Deep Convolutional Neural Network (MC-DCNN) for multivariate TSC, where each variable (i.e., univariate time series) is trained individually to extract features and finally concatenated using an MLP to perform classification (Fig. 1). Their results show that their model achieves a state-of-the-art performance both in efficiency and accuracy on a challenging dataset. The drawback of their model and similar architectures (e.g., Devineau et al., 2018, May) is that they do not capture the correlation between variables as the feature extraction is carried out separately for each variable.

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021
© Author(s) 2021. CC BY 4.0 License.

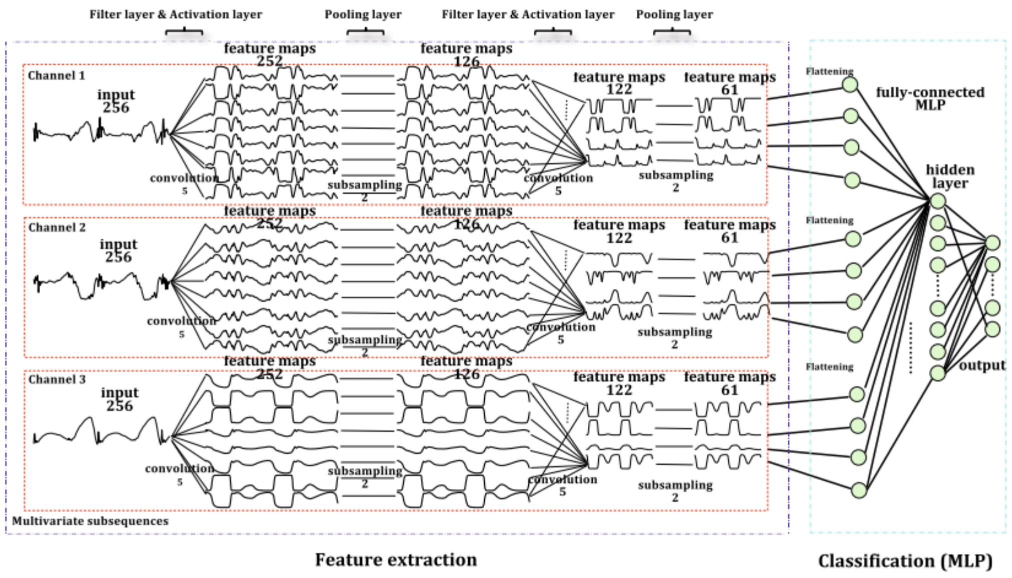

Figure 1. A 2-stages MC-DCNN architecture for activity classification. This architecture consists of three channels input, two filter layers, two pooling layers, and two fully-connected layers (after Zheng et al., 2014, June).

Brunel et al. (2019) present CNNs adapted for TSC in cosmology using 1D filters to extract features from each channel over time and a 1D convolution in depth to capture the correlation between the channels. They compared the results from LSTMs with CNNs, which shows that CNNs give better results than LSTMs. Nevertheless, both deep learning approaches are very promising.

The combination of CNNs and LSTM units has already yielded state-of-the-art results in problems requiring classification of temporal information such as human activity recognition (Li et al., 2017; Mutegeki and Han, 2020, February), text classification (Luan and Lin, 2019; March, She and Zhang, 2018, December; Umer et al., 2020), video classification (Lu et al., 2018 and Wu et al., 2015, October), sentiment analysis (Ombabi et al., 2020; Sosa, 2017; Wang et al., 2016, August; Wang et al., 2019), typhoon formation forecasting (Chen et al.,2019), and arrhythmia diagnosis (Oh et al., 2018). In this architecture, convolutional operations capture features and LSTMs capture time dependencies on extracted features. Ordóñez and Roggen (2016) propose a deep convolutional LSTM model (DeepConvLSTM) for activity recognition (Fig. 2). Their results are compared to the results from standard feedforward units showing that DeepConvLSTM reaches a higher F1 score and better decision boundaries for classification. Furthermore, they noticed that the LSTM model gives promising results with relatively small datasets. Furthermore, LSTMs present a better performance in capturing longer temporal dynamics, whereas the convolution filters can only capture the temporal dependencies dynamics within the length of the filter.

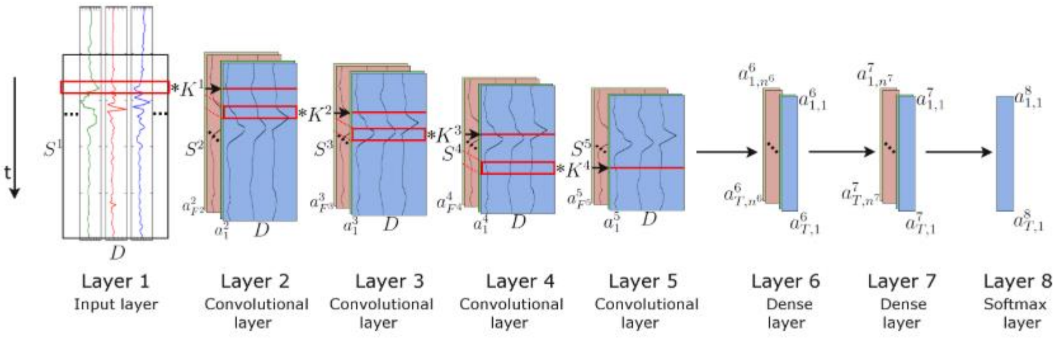

Figure 2. The architecture of the DeepConvLSTM framework for activity recognition (after Ordóñez and Roggen, 2016).

The objective of this research is to develop deep learning models to predict breakup ice-jam events to be used as an early warning system of possible flooding. Deep learning methods are promising to address the requirements of ice-jam predictions. Hence, we developed three deep learning models; a CNN, an LSTM, and a combined CN-LSTM (Convolutional-Long Short-Term Memory) for ice-jam predictions and compared the results. The previous studies show that these models show good capabilities in capturing features and the correlation between features (through convolution units) and time dependencies (through memory units) that will be later used for TSC. The combined CN-LSTM can reduce errors by compensating for the internal weaknesses of each model. In the CN-LSTM model, CNNs capture features, then the LSTMs give the time dependencies on the captured features.

2 Material and Methods

2.1 Input data and study area

It is known that specific hydro-meteorological conditions lead to the ice-jam occurrence (Turcotte and Morse, 2015, August and White, 2003). For instance, breakup ice jams occur when a period of intense cold is followed by a rapid peak discharge resulting from spring rainfall and snowmelt runoff (Massie et al., 2002). The period of intense cold can be represented by the changes in Accumulated Freezing Degree Days (AFDD). The sudden spring runoff increase is not often available at the jam location and can be represented by liquid precipitation and snow depth some days before the ice-jam occurrence (Zhao et al., 2012). Prowse and Bonsal (2004) and Prowse et al. (2007) evaluate various hydroclimatic explanations for river ice freeze-up and breakup, concluding that shortwave radiation is the most critical factor influencing the mechanical strength of ice and consequently the possibility of breakup ice jams to occur. Turcotte and Morse (2015, August) explain that Accumulated Thawing Degree Day (ATDD), an indicator of warming periods, partially covers the effect of shortwave radiation. In the previous studies of ice-jam and breakup predictions, discharge and changes in discharge, water level and changes in water level, AFDD, ATDD, precipitation, solar radiation, heat budget, and snowmelt or snowpack are the most readily used variables (Madaeni et al., 2020).

The inputs we used in this study are historical ice-jam or no ice-jam occurrence (Fig. 2) as well as hydro-meteorological variables including liquid precipitation (mm), min and max temperature (°C), AFDD (from August

1st; °C), ATDD (from January 1st; °C), snow depth (cm) and net radiation (W m^{-2}) in all rivers in Quebec. The net solar radiation, the total energy available to influence the climate, is calculated as the difference between incoming and outgoing energy. If the median temperature is greater than 1, the precipitation is considered liquid precipitation. The source, time period, and spatial resolution of the input variables are presented in Table 1. The “NaN” precipitation values get 0 values.

TThe ice-jam database is provided by the Quebec Ministry of Public Security (MSPQ; Données Québec, 2021) for 150 rivers in Quebec, mainly in the St. Lawrence basin. The database comes from the digital or paper event reports by local authorities under the jurisdiction of the MSPQ from 1985 to 2014. Moreover, some other data of this database are provided by the field observations from the Vigilance / Flood application from 2013 to 2019. It contains 995 recorded jam events that are not validated and contain many inaccuracies, mainly in the toponymy of the rivers, location, dating, and the redundancy of jam events.

The names of the watercourse of several ice jams are not given or completely wrong or affected by a typo or an abbreviation. The toponymy of the rivers was corrected using the National Hydrographic Network (NHN; National Hydrographic Network - Natural Resources Canada (NRCan)), the Geobase of the Quebec hydrographic network (National Hydro Network - NHN - GeoBase Series - Natural Resources Canada), and the Toporama Web map service (The Atlas of Canada - Toporama - Natural Resources Canada) of the Sector of Earth Sciences.

Several ice jams are placed on the banks at a small distance (less than 20 m) from the polygon of the river. In this case, the location of the ice jam is moved inside the river polygon. In other cases, the ice-jam point is posed further on the flooded shore at a distance between 20 and 200 m. This has been corrected based on images with very high spatial resolution, the sinuosity and the narrowing of the river, the history of ice jams at the site in question, and the press archives. In addition, some ice jams were placed too far from the mentioned river due to a typo in entering their coordinates. A single-digit correction in longitude or latitude returned the jam to its exact location. There are certain cases where the date of jam formation is verified by searching the press archives, notably when the date of formation is missing or several jams with the same dates and close locations in a section of a river are present.

The ice jam database contains many duplicates. This redundancy can be due to merging two data sources, the double entry during ice-jam monitoring, or recording an ice jam for several days. The duplicates are removed from the database. The corrected ice-jam database contains 850 jams for 150 rivers, mainly in southern Quebec (Fig. 3). The ice jams formed in November and December (freeze-up jams) are removed to only include breakup jams (from January 15th) in the modelling as these two types of jams are formed due to different processes. The final breakup ice-jam database that used in this study includes 504 jam events.

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

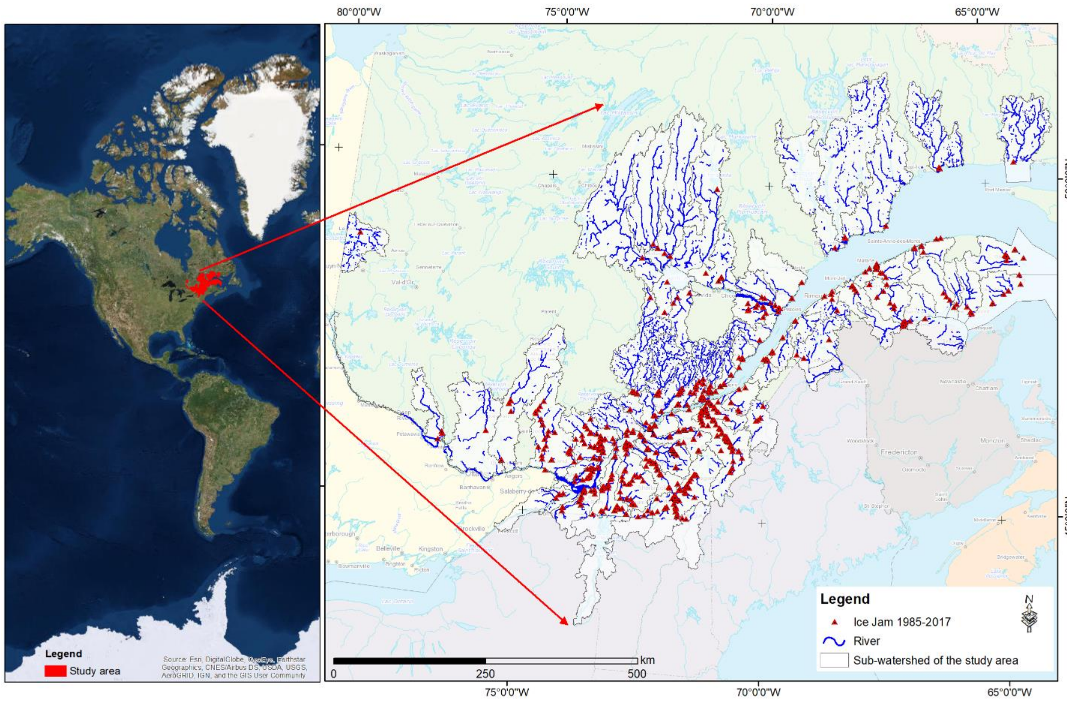
Legend
Study area
Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
80°0'0"W
75°0'0"W
70°0'0"W
65°0'0"W
50°0'0"N
45°0'0"N
Legend
Ice Jam 1985-2017
River
Sub-watershed of the study area
0
250
500
km

Figure 3. Study area and historic ice-jam locations recorded in Quebec from 1985-2017.

Table 1. Hydro-meteorological data used as the input to the model.

Data	**Source**	**Duration**	**Spatial resolution**
Min and Max temperature*	Daily Surface Weather Data (Daymet; Thornton et al., 2020)	1979-2019	1 km
Liquid precipitation	Canadian Precipitation Analysis (CaPA; Mahfouf et al., 2007)	2002-2019	10-15km
Liquid precipitation	North American Regional Reanalysis (NARR; Mesinger et al., 2006)	1979-2001	30 km
Infrared radiation emitted by the atmosphere	North American Regional Reanalysis (NARR)	1979-2019	30 km
Infrared radiation emitted from the surface	North American Regional Reanalysis (NARR)	1979-2019	30 km
Snow depth	North American Regional Reanalysis (NARR)	1979-2019	30 km

* The average was used to derive the AFDD and the ATDD.

2.2 Deep learning models for time-series classification (TSC)

The most popular deep neural networks for TSC are MLP, CNNs, and LSTM. Despite their power, however, MLP has limitations that each input (i.e., time-series element) and output are treated independently, which means that the temporal or space information is lost (Lipton et al., 2015). Hence, an MLP needs some temporal information in the input data to model sequential data such as time series (Ordóñez and Roggen, 2016). In this regard, Recurrent Neural Networks (RNNs) are specifically adapted to sequence data through the direct connections between individual layers (Jozefowicz et al., 2015). Recurrent Neural Networks perform the same repeating function with a straightforward structure, e.g., a single tanh (hyperbolic tangent) layer, for every input of data (xt), while all the inputs are related to each other with their hidden internal state, which allows it to learn the temporal dynamics of sequential data (Fig. 4).

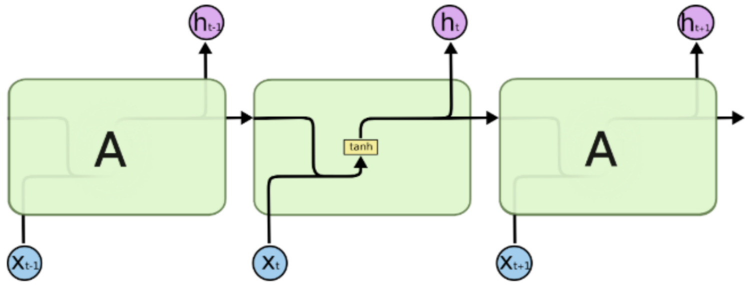

Figure 4. An RNN with a single tanh layer, where A is a chunk of the neural network, xt is input data, and ht is output data (after Olah, 2015).

Recurrent Neural Networks were rarely used in TSC due to their significant problems. Recurrent Neural Networks mainly predict output for each time-series element, they are sensitive to the first examples seen, and it is also challenging to capture long-term dependencies due to vanishing gradients, exploding gradients, and their complex dynamics (Devineau et al., 2018, June; Fawaz et al., 2019).

Long short-term memory RNNs are developed to improve the performance of RNNs by integrating a memory to model long-term dependencies in time-series problems (Brunel et al., 2019; Karim et al., 2019). Long short-term memory networks do not have the problem of exploding gradients. LSTMs have four interacting neural network layers in a very special way (Fig. 5). An LSTM has three gates (sigmoid layers; σ) to control how much of each component should be let through by outputting numbers between zero and one. The input to an LSTM goes through three gates (“forget”, “input”, and “output gates”) that control the operation performed on each LSTM block (Ordóñez and Roggen, 2016). The first step is the “forget gate” layer that gets the output of the previous block (ht−1), the input for the current block (Xt), and the memory of the previous block (Ct-1) and gives a number between 0 and 1 for each number in the cell state (Ct−1; Olah, 2015). The second step is called the “input gate” with two parts, a sigmoid layer that decides which values to be updated and a tanh layer that creates new candidate values for the cell state. These two new and old memories will then be combined and control how much the new memory should influence the old memory. The last step (output gate; step 3 in Fig. 5) gives the output by applying a sigmoid layer deciding how much new cell memory goes to output, and multiply it by tanh (giving values between −1 and 1).

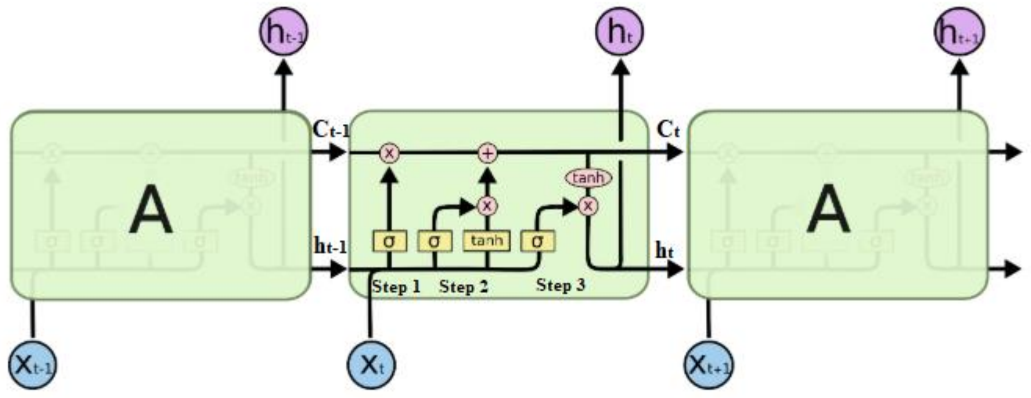

Figure 5. Structure of LSTM block with four interacting layers (adopted from Olah, 2015).

Recently, convolutional neural networks challenged the assumption that RNNs (e.g., LSTMs) have the best performance when working with sequences. Convolutional neural networks show state-of-the-art performance in sequential data such as speech recognition and sentence classification, similar to TSC (Fawaz et al., 2019). Convolutional neural networks are the most widely used deep learning methods in TSC problems (Fawaz et al., 2019). They learn spatial features from raw input time series using filters (Fawaz et al., 2019). Convolutional neural networks are robust and need a relatively small amount of training time comparing with RNNs or MLPs. They work best for extracting local information and reducing the complexity of the model.

A CNN is a kind of neural network with at least one convolutional layer (or filter). A CNN usually involves several convolutional layers, activation functions, and pooling layers for feature extraction following by dense layers (or MLP) as a classifier (Devineau et al., 2018, June). The reason to use a sequence of filters is to learn various features from time series for TSC. A convolutional layer consists of a set of learnable filters that compute dot products between local regions in the input and corresponding weights. With high-dimensional inputs, it is impractical to connect neurons to all neurons in the previous layer. Therefore, each neuron in CNNs is connected to only a local region of the input, namely the receptive field, which equals the filter size (Fig. 5). This feature reduces the number of parameters by limiting the number of connections between neurons in different layers. The input is first convolved with a learned filter, and then an element-wise nonlinear activation function is applied to the convolved results (Gu et al., 2018). The pooling layer performs a downsampling operation such as maximum or average, reducing the spatial dimension (Fig. 6). One of the most powerful features of CNNs is called weight or parameter sharing, where all neurons share filters (weights) in a particular feature map (Fawaz et al., 2019) to reduce the number of parameters.

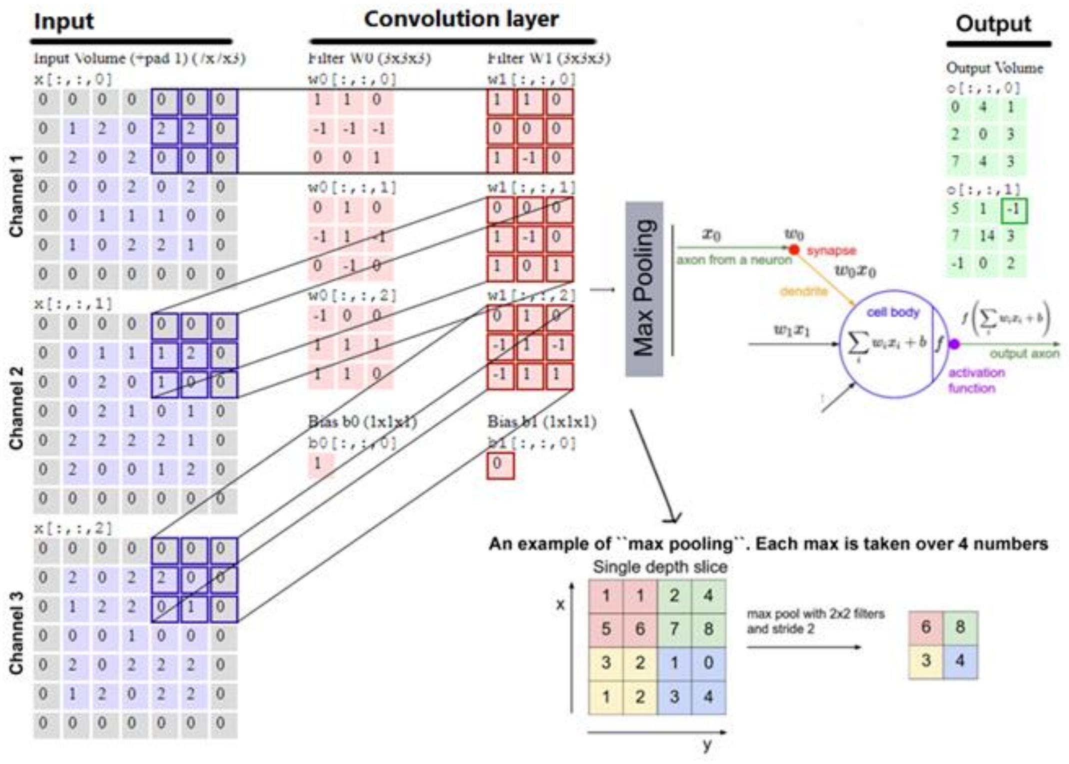

Figure 6. . A CNN Architecture for image classification (modified from Karpathy, 2017).

2.3 Model libraries

In an anaconda (Analytics, C., 2016) environment, Python is implemented CNN, LSTM, and CN-LSTM networks for TSC. To build and train networks, the networks are implemented in Theano (Bergstra et al., 2010, June) using the Lasagne (Dieleman et al., 2015) library. The other core libraries used for importing, preprocessing, training data, and visualization of results are Pandas (Reback et al., 2020), NumPy (Harris et al., 2020), Scikit-Learn (Pedregosa et al., 2011), and Matplotlib.PyLab (Hunter, J. D., 2007). Spyder (Raybaut, 2009) package of Anaconda is utilized as an interface, or the command window can be used without any interface.

2.4 Preprocessing

The data is comprised of variables with varying scales, and the machine learning algorithms can benefit from rescaling the variables to all have the same scale. Scikit-learn (Pedregosa et al., 2011) is a free library for machine learning in Python that can be used to preprocess data. We examined Scikit-learn MinMaxScaler (scaling each variable between 0 and 1), Normalizer (scaling individual samples to the unit norm), and StandardScaler (transforming to zero mean and unit variance separately for each feature). The results show that MinMaxScaler (Eq. (1)) works the best in our models. The scaling of validation data is done with min and max from train data.

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

$$X_{\text{scaled}} = \left(\frac{X - X.\text{min}}{X.\text{max} - X.\text{min}}\right), \tag{1}$$

For each jam or no jam event, we used 15 days of information before the event to predict the event on the 16th day. We generate a balanced dataset with the same number of jam and no-jam events (1008 small sequences totally), preventing the model from becoming biased to jam or no-jam events. The hydro-meteorological data related to no-jam events are constructed by extracting data from the reaches of no-jam records. We used ShuffleSplit subroutine from the Scikit-learn library, where the database was randomly sampled during each re-shuffling and splitting iteration to generate training and validation sets. We applied 100 re-shuffling and splitting iterations with 80 % of data for training and 20 % for validation. There are 806 and 202 small sequences with the size of (16, 7), 16 days of data for the seven variables; for training and validation, respectively. To examine models' generalization, we hold out 30 small sequences for testing and 80 % and 20 % of remaining data for training and validation, respectively.

2.5 Training

Training a deep neural network with an excellent generalization to new unseen inputs is challenging. As a benchmark, a CNN model with the parameters and layers similar to previous studies is developed. The model shows underfitting or overfitting with various architectures and parameters. To overcome underfitting, deeper models and more nodes in each layer are beneficial; however, overfitting is more challenging to overcome. The ice-jam dataset is small, which easily causes the network to memorize training examples and consequently results in overfitting, as a small dataset may not appropriately describe the relationship between input and output spaces.

2.5.1 Overcome overfitting

2.5.1.1 Noise layer

The first approach to overcome overfitting is acquiring more data that is not possible with ice-jam records. Another popular approach to increase the number of samples is data augmentation, including cropping, rotating, blurring, color modification, and noise injection in image classification. Data augmentation can act as a regularizer, prevent overfitting, and improve performance in imbalanced class problems (Wong et al., 2016). However, the application of data augmentation in deep learning for time series classification still has not been studied thoroughly (Fawaz et al., 2019). To expand the size of the dataset, noise layers, as a simple form of random data augmentation, can be used. Over the training process, each time an input sample is exposed to the model, the noise layer creates new samples in the vicinity of the training samples resulting in various input data every time, increases randomness, making the model less prone to memorize training samples and learns more general features (resulting in better generalization).
We added the Gaussian noise layer (from the Lasagne library), where the noise values are Gaussian-distributed with zero-mean and a standard deviation of 0.1 to the input. The noise layer is usually added to the input data but can also be added to other layers.

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

2.5.1.2 Dropout

The other approach to tackle overfitting is dropout (Srivastava et al., 2014). The dropout, the most successful method for neural network regularization, randomly sets inputs to zero (Fig. 7). To overcome overfitting and examine the effectiveness of dropout in our models, the dropout with the recommended rates of 0.1 for the input layer and between 0.5 and 0.8 for hidden layers (Garbin et al., 2020) are applied in different layers of the models.

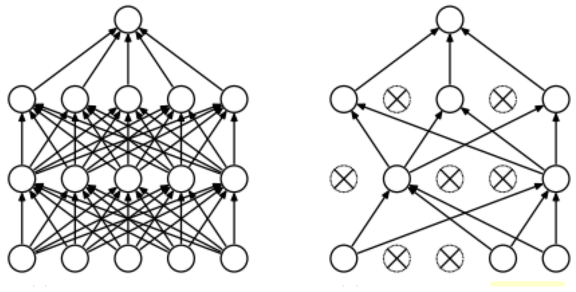

Figure 7. A neural network with two hidden layers (left) and a neural network with dropout (right; after Srivastava et al., 2014).

2.5.1.3 Early stopping

Early stopping is another efficient method to tackle overfitting via halting the training procedure where further training would decrease training loss, while validation loss starts to increase.

2.5.1.4 Batch normalization

As explained earlier, the input data is scaled separately for each feature to be between 0 and 1. However, in deep learning, the distribution of the input of each layer will be changed by updates to all the preceding layers, so-called internal covariate shift. Hence, hidden layers try to learn to adapt to the new distribution slowing down the training process. Batch normalization (Ioffe and Szegedy, 2015, June) is a recent method that provides any layer with inputs of zero mean and unit variance and consequently prevents internal covariate, solves exploding or vanishing gradient problems, allows the use of higher learning rates, improves the training efficiency, and speeds up the training. Batch normalization adjusts the value for each batch, results in more noise acting as a regularizer, similar to dropout, and thus reduces the need for dropout (Garbin et al., 2020). We performed batch normalization over each channel in different layers in our models to find its best locations through trial and error.

2.5.1.5 Regularization

There are two general ways to keep a deep neural network simple and consequently prevent overfitting; through the number of weights and values of weights. The number of weights can be controlled by the number of layers and nodes optimized via the grid or random search. A network with large weights can be more complex and unstable as large weights increase loss gradients exponentially, resulting in exploding gradients that cause massive output changes with minor changes in the inputs. In turn, the exploding gradients can force the model loss and weights to “NaN” values (Brownlee, 2017).

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

The simplest and most common approach to keep the weights small is regularization methods that involve checking model weights and adding an extra penalty term to the loss function in proportion to the size of weights' size in the model. The two main methods used to calculate the size of the weights are L1 (the sum of the absolute values of the weights; Eq. (2)) and L2 or weight decay (the sum of the squared values of the weights; Eq. 3). In Eq. (2) and (3), λ is a parameter that controls the importance of the regularization, and W is the network parameters. The L1 regularization encourages weights to be 0.0 (causing underfitting) and very few features with non-zero weights, while L2 regularization forces the weights to be small rather than zero. Hence, L2 can predict more complex patterns when output is a function of all input features. We used an L2 regularization cost by applying a penalty to the parameters of all layers in the networks in CNN, LSTM, and CN-LSTM models.

$$Cost\ function + \lambda \sum_{i=1}^{n} |w_i| \tag{2}$$

$$Cost\ function + \lambda \sum_{i=1}^{n} w_i^2 \tag{3}$$

2.5.2 Architecture Tuning

Finding hyperparameter values in deep learning has been challenging due to the complex architecture of deep learning and a large number of parameters (Garbin et al., 2020). To find the best model architecture, we study the performance of models with different layers and parameters such as number of noise, batch normalization, convolutional, pooling, LSTM, dropout, and dense layers, as well as different pooling sizes and strides, different batch sizes, various scaling of data (standardization and normalization), various filter sizes, number of units in LSTM and dense layers, the type of the activation functions, regularization and learning rates, weight decay and number of filters in convolutional layers. We also applied various combinations of these layers and parameters.

2.5.2.1 Activation function

The activation function adds non-linearity to the network allowing the model to learn more complex relationships between inputs and outputs (Zheng et al., 2014, June). Each activation function that is used in deep learning has its advantages and disadvantages, and typical activation functions in deep learning are Rectified Linear Unit (ReLU; Eq. (4)), sigmoid (Eq. (5)), and hyperbolic tangent (tanh; Eq. (6); Fig. 8; Gu et al., 2018). In deep neural networks, adding more layers with certain activation functions results in the vanishing gradient problem where the gradients of the loss function become almost zero, causing difficulties in training. For instance, the sigmoid function maps a large input space into a small one between 0 and 1. Hence, when the input is very positive or very negative, the sigmoid function saturates (becomes very flat) and becomes insensitive to small changes in its input, causing the derivatives to disappear (Goodfellow et al., 2016). Therefore, in backpropagation, small derivatives are multiplied together, causing the gradient to decrease exponentially, propagating back to the first layer. This causes ineffective updates of weights and biases of the initial layers and consequently inaccuracy. Some solutions to overcome this problem include using specific activation functions like ReLU and tanh and using batch normalization layers to prevent the activation functions from becoming saturated. The ReLU recently drown lots of attention and has been widely used in recent deep learning models (Gamboa, 2017). The advantage of ReLU over sigmoid and tanh is a better generalization, making the training faster and simpler. Hence, we investigated the performance of the model with ReLU, sigmoid, or tanh activation functions in convolutional layers.

$$ReLU(x) = max\,(0, x) \tag{4}$$

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{5}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6}$$

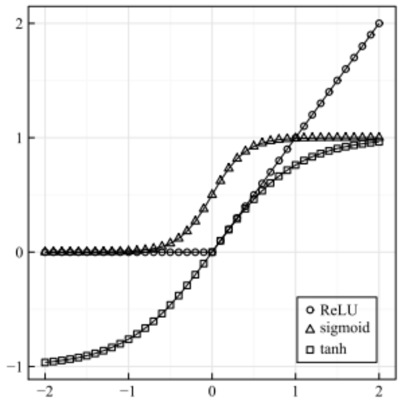

Figure 8. Illustration of sigmoid, tanh, and ReLU activation functions (after Zheng et al., 2016).

2.5.2.2 Learning rate

To find the minimum cost function, a move in the negative direction of the gradient is required. This movement is called the "learning rate," which is the most significant hyperparameter in training a deep neural network. The model error is calculated, and the errors corresponding to weights updated by the learning rate are backpropagated in the network. A too-small learning rate needs many updates and epochs, reaching the minimum. On the other hand, a too-large learning rate causes dramatic updates and leads to oscillations in loss over epochs. A good learning rate quickly reaches the minimum point between 0.1 to 1e-6 on a log scale and can be found through a grid or random search (Fig. 9).

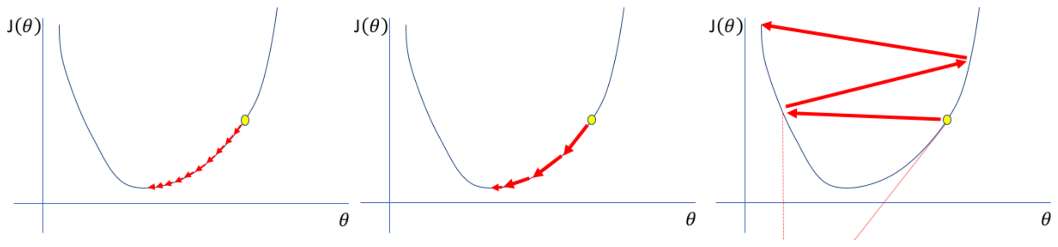

Figure 9. Too small, good, and too large learning rates from left to right (after Jordan, 2018).

2.5.2.3 Update expression

There are various algorithms to update the trainable parameters at each mini-batch. The parameter updating procedure includes feedforwarding, backpropagation, and applying gradients. We tried the Stochastic Gradient Descent (SGD) with Nesterov momentum, RMSProp, Adadelta, and Adam updates to update the parameters in Lasagne. The SGD with momentum updates the model weights by adding a momentum so that the overall gradient depends on the current and previous gradients, causing the weights to move in the previous direction without oscillation.

2.5.3 Network optimization

Training CNN involves global optimization by defining a loss expression to be minimized overtraining. For the classification task, the loss function of the models is calculated using categorical cross-entropy between network outputs and targets (Eq. (7)), where L is the loss, p is the prediction (probability), t is the target, and c is the number of classes. Then, the mean of the loss is computed over each mini-batch.

$$L = -\sum_{i=1}^{c=2} t_i \log(p_i) \tag{7}$$

2.5.4 Model evaluation

The network on the validation set is evaluated after each epoch during training to monitor the training progress. During validation, all non-deterministic layers are switched to deterministic. For instance, noise layers are disabled, and the update step of the parameters is not performed.
The classification accuracy cannot appropriately represent the model performance for unbalanced datasets, as the model can show a high accuracy by biasing towards the majority class in the dataset (Ordóñez and Roggen, 2016). While we built a balanced dataset (with the same number of jam and no jam events), randomly selecting test data and shuffling the inputs, and splitting data into train and validation sets can result in a slightly unbalanced dataset. In our case, the number of jams and no jams for train and validation and test sets is presented in Table 2. Therefore, the F1 score (Eq. (8)), which considers each class equally important, is used to measure the binary classification accuracy. The F1 score, as a weighted average of the precision (Eq. (9)) and recall (Eq. (10)), has the best and worst scores of 1 and 0, respectively. In Eqs. 9 and 10, TP, FP, and FN are true positive, false positive, and false negative, respectively.

Table 2. The number of jam and no jam events in train and validation and test datasets.

	Train and validation	Test
Jam	504	48
No jam	403	53

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{8}$$

$$Precision = \frac{TP}{TP+FP} \tag{9}$$

$$Recall = \frac{TP}{TP+FN} \tag{10}$$

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

Although the model accuracy is usually used to examine the performance of deep learning models, the model size (i.e., number of parameters) provides a second metric, which represents required memory and calculations, to be compared among models with the same accuracy (Garbin et al., 2020).
After training the model, the well-trained network parameters are saved to a file and are later used for testing the network generalization using a test dataset, which is not seen during training and validation.

3 Results and Discussion

3.1 Hyperparameters optimization

3.1.1 Batch size

The inputs and corresponding targets are iterated in mini-batches for training and validation. Batch size significantly influences the training time (Fawaz et al., 2019, July), and the batch size of 32 is usually used in previous studies. However, we investigated batch sizes of 16, 32, and 64, and the mini-batches of 16 demonstrate to improve the results slightly.

3.1.2 Noise layers

The performance of CNN and LSTM models developed for the ice-jam prediction problem is improved by adding a noise layer to the input, while the CN-LSTM model showed underfitting. Adding a noise layer to other layers does not improve any of the developed models for ice-jam prediction.

3.1.3 Dropout layer

Adding dropout layers could not improve any developed models. This agrees with previous studies revealing that dropout does not work well with LSTMs (Zaremba et al., 2014) and CNNs, and dropout layers do not work when batch size is small (less than 256; Garbin et al., 2020). Furthermore, it is in agreement with Garbin et al. (2020) stating that utilizing batch normalization layers in a model reduces the need for dropout layers.

3.1.4 Number of layers

The depth is related to the sequence length (Devineau et al., 2018, May), as deeper networks need more data to provide better generalization (Fawaz et al., 2019, July). In the previous studies of CNNs, there are usually one, two, or three convolution stages (Zheng et al., 2014, June). We tried different numbers of CNN, LSTM, and dense layers and selected three, two, and two such layers, respectively, as the sequence length in this study is small (16), and we could not improve the model performance by merely adding more depth.

3.1.5 Number and size of CN filters

Fawaz et al. (2019, July) explain the number and length of filters used in CNNs. Data with more classes need more filters to classify the inputs accurately. Longer time series need longer filters to capture longer patterns and consequently to produce accurate results. However, longer kernels significantly increase the number of parameters and increase the potential for overfitting small datasets, while a small kernel size risks poor performance. In our models, the optimum number of filters is attained to be 128 by searching among the typical number of filters (i.e., 32, 64, and 128). The kernel sizes of 3, 5, and 7 are often applied in deep CNNs. We tried these filter sizes, and the best performance was achieved through using two convolutional layers with 1-D filters of (5, 1) with the stride of (1, 1) to capture temporal variation for each variable separately.

Furthermore, one convolutional layer with 2-D filters of size (5, 3) with the stride of (1, 1) is then used to achieve the correlation between variables via depth-wise convolution of input time-series. A big stride might cause the model to miss valuable data used in predicting and smoothing out the noise in the time series. The layers in CNNs have a bias for each channel, sharing across all positions in each channel.

3.1.6 Padding

The convolution is applied where the input and the filter overlap. Hence, we pad the input by zeros with half the filter size on both sides. Using stride of 1 with "Pads = same" (in Lasagne) in the convolutional 2-D layers results in an output size equal to the input size for each layer.

3.1.7 Activation functions in CN layers

The experiments demonstrate that errors are very high using tanh, whereas ReLU and sigmoid show almost the same performance. As ReLU performs slightly better than sigmoid, we used ReLU in our models.

3.1.8 Weight initialization

Among the various types of methods available in Lasagne for weight initialization, the GLOROT uniform (i.e., Xavier; Glorot and Bengio, 2010, March) and He initializations (He et al., 2015), the most popular initialization techniques, are used to set the initial random weights in convolutional layers. The results reveal that these methods yield almost the same F1 scores. However, the histograms of F1 scores reveal that GLOROT uniform yields slightly better results (Fig. 10).

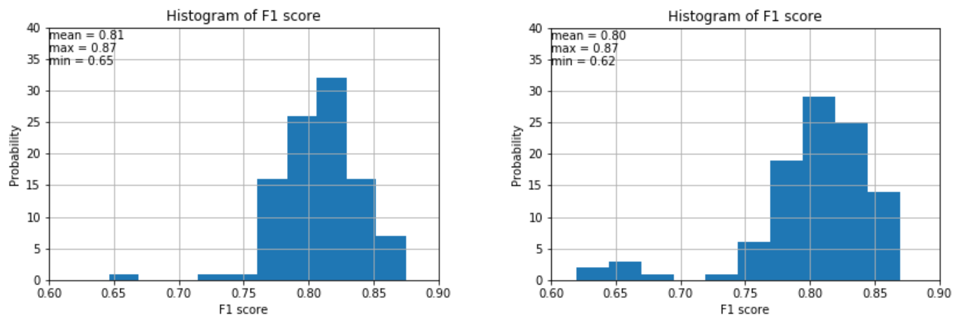

Figure 10. Histograms of F1 score for CNN using He (left) and GLOROT uniform (right) weight initialization with 100 random train-validation splits.

3.1.9 Number of LSTM units and their activation functions

The optimal number of units in LSTM layers was found through a search over typical numbers of 32, 64, and 128. We found that 128 units yield the best results in our models. We used the default activation function of tanh in LSTM layers.

3.1.10 Dense layer

The dense layers with RecLU functions following by one dense layer with softmax function are applied after the feature learning and LSTM layers to perform classification. The common number of units in dense layers are 16, 32, 128, and 256. We found that 32 gives the best results in our models. To output the binary classes from the network, softmax or sigmoid functions can be used. We applied softmax as it gives a probability for each class where their total sum is one.

3.1.11 Adaptive learning rates

The adaptive learning rate decreases the learning rate and consequently weights over each epoch. We tried different base learning and decay rates for each model and found that the learning rate significantly impacts the model performance. Finally, we chose a base learning rate of 0.1, 0.01, and 0.001 for LSTM, CNN, and CN-LSTM and, respectively. A decay rate of 0.8 was used for CNN and CN-LSTM, while for the LSTM model, this rate was 0.95. Table 3 shows the adaptive learning rates for CNN, LSTM, and CN-LSTM calculated using Eq. (11) for each epoch.

$$\text{adaptive learning rate} = \textit{base learning rate} \times \textit{decay}^{epoch} \qquad \text{Eq. (11)}$$

The experiments show that the learning rate is the most critical parameter influencing the model performance. A small learning rate can cause the cost function to get stuck in local minima, and a large learning rate can result in oscillations around global minima without reaching it.
Our CN-LSTM model is deeper than the other two models, and deeper models are more prone to a vanishing gradient problem. To overcome the vanishing gradients, it is recommended that lower learning rates, e.g., lower than 1e-4, be used. Interestingly, we found that our CN-LSTM model works better with lower learning rates than the other two models.

Table 3. The adaptive learning rate for 50 epochs.

	Learning rate		
Epochs	**CNN**	**CN-LSTM**	**LSTM**
1	0.008	8.00E-04	0.095
2	0.006	6.40E-04	0.09
3	0.005	5.12E-04	0.086
4	0.004	4.10E-04	0.081
.	.		.
.	.		.
40	1.30E-06	1.33E-07	0.013

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

.	.		-
50	1.40E-07	1.43E-08	-

3.1.12 Update expression

We found that SGD with momentum works better than other methods in our cases. The typical values for momentum are 0.99, 0.9, and 0.5. We applied different values and found that 0.9 gives the best results in our models; this high momentum results in larger update steps. It is recommended to scale the learning rate by "1 – momentum" for using the high momentums, which gives 0.1. Interestingly, we already have applied the base learning rate of 0.1 for the LSTM model chosen through trial and error (as explained earlier); however, smaller values are chosen for CNN and CN-LSTM networks.

3.2 Architecture of models

The architectures of CNN, LSTM, and CN-LSTM models that are finally selected are presented in Figs. 11, 12, and 13, respectively. The layers, their output shapes, and their number of parameters are presented in Tables 4, 5, and 6 for CNN, LSTM, and CN-LSTM models, respectively.

The ice-jam dataset for Quebec contains 1008 balanced sequence instances (with a length of 16), which is small for deep learning. The deep learning models often tend to overfit small datasets by memorizing inputs rather than training. The noise layers applied to the CNN and LSTM models significantly overcome the overfitting problem through data augmentation. However, the performance of the CN-LSTM model dramatically deteriorates, including a noise layer (Fig. 14; showing underfitting).

The CNN models often include pooling layers to reduce data complexity and dimensionality. However, it is not always necessary that every convolutional layer is followed by a pooling layer in the time-series domain (Ordóñez and Roggen, 2016). For instance, Fawaz et al. (2019, July) do not apply any pooling layers in their models for TSC. We tried max-pooling layers after different convolutional layers in CNN and CN-LSTM networks and found that a pooling layer following only the last convolutional layer improves the performance of both models. This can be due to subsampling the time series and using time series with a length of 16 that reduces the need for reducing dimensionality.

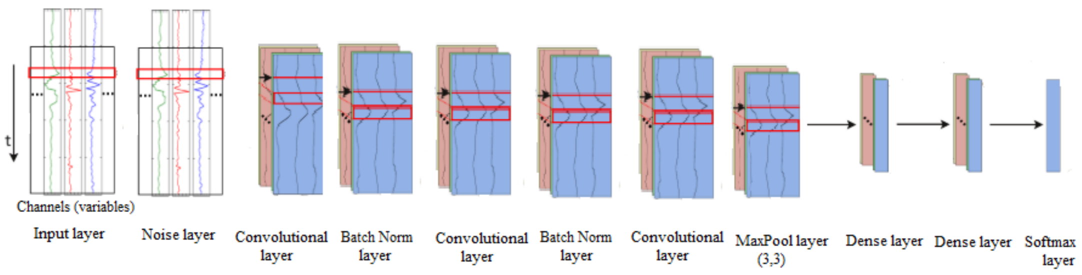

Figure 11. The architecture of the CNN model for ice-jam prediction (adapted after Ordóñez and Roggen, 2016).

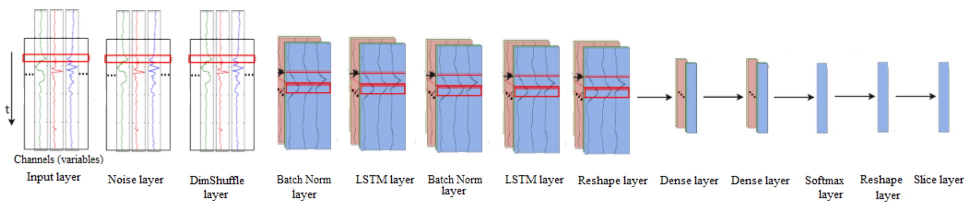

Figure 12. The architecture of the LSTM model for ice-jam prediction (adapted after Ordóñez and Roggen, 2016).

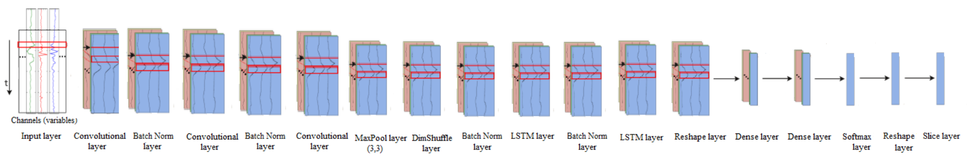

Figure 13. The architecture of the CN-LSTM model for ice-jam prediction (adapted after Ordóñez and Roggen, 2016).

Table 4. The layers, their output shapes, and their number of parameters for the CNN model.

Layers	Output shape	Number of parameters
Input	(16, 1, 16, 7)	0
GaussianNoise	(16, 1, 16, 7)	0
Conv2D	(16, 128, 16, 7)	640
BatchNorm	(16, 128, 16, 7)	512
Nonlinearity	(16, 128, 16, 7)	0
Conv2D	(16, 128, 16, 7)	81920
BatchNorm	(16, 128, 16, 7)	512
Nonlinearity	(16, 128, 16, 7)	0
Conv2D	(16, 128, 16, 7)	245888
MaxPool2D	(16, 128, 5, 2)	0
Dense	(16, 32)	40992
Dense	(16, 32)	1056
Softmax	(16, 2)	66

Table 5. The layers, their output shapes, and their number of parameters for the LSTM model.

Layers	Output shape	Number of parameters
Input	(16, 1, 16, 7)	0
GaussianNoise	(16, 1, 16, 7)	0
Dimshuffle	(16, 16, 1, 7)	0
BatchNorm	(16, 16, 1, 7)	64
LSTM	(16, 16, 128)	70272
BatchNorm	(16, 16, 128)	64
Nonlinearity	(16, 16, 128)	0
LSTM	(16, 16, 128)	132224
Reshape	(256, 128)	0

Dense	(256, 32)	4128
Dense	(256, 32)	1056
Softmax	(256, 2)	66
Reshape	(16, 16, 2)	0
Slice	(16, 2)	0

Table 6. The layers, their output shapes, and their number of parameters for the CN-LSTM model.

Layers	Output shape	Number of parameters
Input	(16, 1, 16, 7)	0
Conv2D	(16, 128, 16, 7)	640
BatchNorm	(16, 128, 16, 7)	512
Nonlinearity	(16, 128, 16, 7)	0
Conv2D	(16, 128, 16, 7)	81920
BatchNorm	(16, 128, 16, 7)	512
Nonlinearity	(16, 128, 16, 7)	0
Conv2D	(16, 128, 16, 7)	245888
MaxPool2D	(16, 128, 5, 2)	0
Dimshuffle	(16, 5, 128, 2)	0
BatchNorm	(16, 5, 128, 2)	20
LSTM	(16, 5, 128)	197760
BatchNorm	(16, 5, 128)	20
Nonlinearity	(16, 5, 128)	0
LSTM	(16, 5, 128)	132224
Reshape	(80, 128)	0
Dense	(80, 32)	4128
Dense	(80, 32)	1056
Softmax	(80, 2)	66
Reshape	(16, 5, 2)	0
Slice	(16, 2)	0

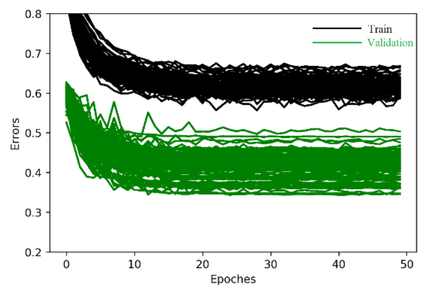

Figure 14. Train and validation errors over epochs for CN-LSTM model with a noise layer.

3.3 Model evaluation

LSTM needs only early stopping at 40 epoch among the developed models, as its validation error starts to increase, while its training error continues to decrease (Fig. 15). Hence, we set the number of epochs to 40 for the LSTM model.

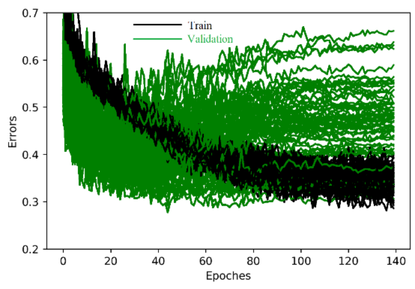

Figure 15. Train and validation errors over epochs for an LSTM model showing overfitting after 40 epochs.

3.3.1 Learning curves and F1 scores

Line plots of the loss (i.e., learning curves), which are loss over each epoch, are widely used to examine the performance of models in machine learning. Furthermore, line plots clearly indicate common learning problems, such as underfitting or overfitting. The learning curves for CNN, LSTM, and CN-LSTM models are presented in Fig. 16. The LSTM model starts to overfit at epoch 40, so an early stopping is conducted. CN-LSTM performs better than the other two models, as its training loss is the lowest and is lower than its validation loss. Histograms of F1 scores (Fig. 16 and Table 7) show that CN-LSTM outperforms the other two models since it results in the highest average and the lowest F1-scores for validation (0.82 and 0.75, respectively). Figure 16 shows that training error of CNN is lower than that of LSTM, which means that CNN trained better than LSTM model. However, it is not true for the validation error. The LSTM network is valdated better than the CNN model since its average and minimum F1 scores for validation are better than the CNN model (by 1 % and 32 %, respectively), and also LSTM yielded no F1 scores below 0.74 (Fig. 17 and Table 7). This reveals that LSTM is showing underfitting.

As shown in Fig. 16, training loss is higher than validation loss in some of the results. Some reasons are explaining that. Regularization reduces the validation and testing (i.e., evaluation) loss at the expense of increasing training loss. The regularization techniques such as noise layers are only applied during training, but not during evaluation resulting in more smooth and usually better functions in evaluation. There is no noise layer in CN-LSTM model that may caused a lower training error than validation error. However, other regularization methods such as L2 regularization are used in all the models, including the CN-LSTM model.

Furthermore, the other issue is that batch normalization uses the mean and variance of each batch in training, whereas, in evaluation, it uses the mean and variance of the whole training dataset. Plus, training loss is averaged over each epoch, while evaluation losses are calculated after each epoch once the current training epoch is completed. Hence, the training loss includes error calculations with fewer updates.

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

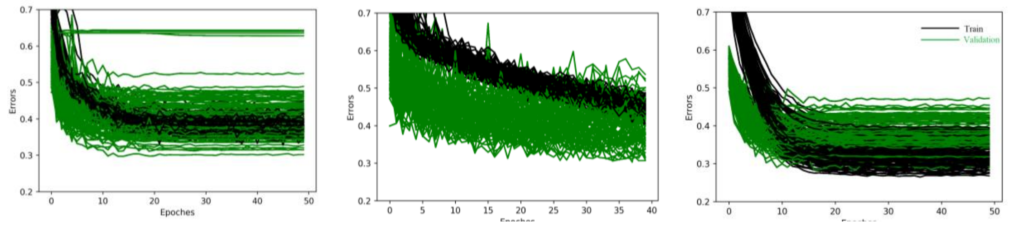

Figure 16. Train and validation errors over epochs for CNN (left), LSTM (middle), and CN-LSTM (right) models with 100 random train-validation splits.

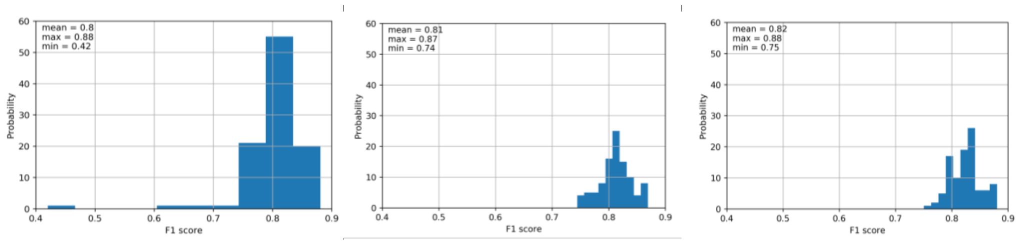

Figure 17. Histograms of F1 scores of validation for CNN (left), LSTM (middle), and CN-LSTM (right) models with 100 random train-validation splits.

Table 7. F1 scores of validation for CNN, LSTM, and CN-LSTM models with 100 random train-validation splits.

Models	F1 score		
	mean	max	min
CNN	0.80	0.88	0.42
LSTM	0.81	0.87	0.74
CN-LSTM	0.82	0.88	0.75

3.3.2 Number of parameters and run time

The total number of parameters in CNN, LSTM, and CN-LSTM networks are 371586, 207874, and 664746, respectively. The best performance has resulted from CN-LSTM with the highest number of parameters. Even though the number of parameters for the LSTM model is less than CNN, the LSTM model shows better validation performance. Furthermore, the number of parameters in the CN-LSTM model is much higher than the two other models, but the computation time is not much higher. All three models take less than 24 hours to train with 100 shuffle splits for training and validation. The models are run on a CPU with four cores, 3.4 GHz clock speed, and 12 GB RAM.

3.4 Order of input variables

Although the order of input variables in the input file is important through using 2-D filters and 2-D max-pooling layers, there is no guideline for this order for multivariate TSC. In the benchmark, we randomly used this order from left to right: precipitation, minimum temperature, maximum temperature, net radiation, ATDD, AFDD, and snow depth. We randomly changed this order and applied the new order: snow depth, maximum temperature, precipitation, AFDD, net radiation, minimum temperature, and ATDD. Both models yielded the same average and minimum F1 scores, whereas the maximum F1 score from the order in the benchmark model (0.88) is higher than that of the second-order (0.86). Therefore, it can be concluded that the order does not significantly impact the results.

3.5 Generalization

To examine the ability of the models to generalize to new unseen data, we randomly set aside 10 % of data from training and validation. We trained a CNN, an LSTM, and a CN-LSTM model, then the trained parameters are saved, and finally, the well-trained parameters are utilized for testing. The test dataset is almost a balanced dataset with 101 samples with the size of (16, 7), including 48 jams and 53 no jams.

The results of the test models show that CN-LSTM models represent the best F1 score of 0.91 (Table 8). Tables 7 and 8 show that although LSTM has slightly better validation performance, CNN works a little better in generalization by only 1 %. The better generalization of CNN can be because LSTM is a little underfitted as LSTM models are often harder to regularize, agreeing with previous studies (e.g., Devineau et al., 2018, June).

Table 8. Test F1 scores for LSTM, CNN, and CNSTM models.

Models	F1 score
CNN	0.80
LSTM	0.79
CN-LSTM	0.91

3.6 Model comparison

Multiple combined classifiers can be considered for pattern recognition problems to reduce errors as different classifiers can cover internal weaknesses of each other (Parvin et al., 2011). The ensemble classifier may be less accurate than the most accurate classifier. However, the accuracy of the combined model is always higher than the average accuracy of individual models. Combining two models improved our results compared to convolution-only or LSTM-only networks in both training and generalization. It can be because the CN-LSTM model incorporates both the temporal dependency of each variable by using LSTM networks and the correlation between variables through CNN models.

The better generalization results from CNN compared to LSTM can be because of the ability of CNN to partially include both temporal dependency and the correlation between variables by using 1D and 2D filters, respectively, while LSTM is unable to incorporate the correlations between variables.

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

4 Conclusion

This project is a part of a project called DAVE, which aims to develop a tool to provide regional ice jam watches and warnings, based on the integration of three aspects: the current conditions of the ice cover; hydrometeorological patterns associated with breakup ice jams; and channel predisposition to ice-jam formation. The outputs of the previous tasks will be used to develop an ice-jam monitoring and warning module and transfer the knowledge gained to end-users to manage the risk of ice jams better.

While most TSC research in deep learning is performed on 1D channels (Hatami et al., 2018, April), we propose deep learning frameworks for multivariate TSC for ice-jam prediction. The main finding from the comparison of results is that the CN-LSTM model is superior to the CNN-only and LSTM-only networks in both training and generalization accuracy, supporting the previous studies (e.g., Sainath et al., 2015). Though the LSTM network demonstrates quite good performance, the CNN model performed slightly better generalization, which agrees with previous studies (e.g., Brunel et al., 2019).

To our best knowledge, this study is the first study introducing these deep learning models to the problem of ice-jam prediction. Even though our training data in supervised ice-jam prediction is small, the results reveal that deep learning techniques can give accurate results, which agrees with a previous study conducted by Ordóñez and Roggen (2016) in activity recognition. The excellent performance of CNN and CN-LSTM models may be partially due to the characteristic of CNN that decreases the total number of parameters which does training with limited training data easier (Gao et al., 2016, May) and including the correlation between involved variables. However, our models will be improved in the future by a larger dataset.

The developed models do not apply to freeze-up jams that occur in early winter and are based on different processes than breakup jams. We studied only break-up ice jams as usually they result in flooding and are more dangerous than freeze-up jams.

The hydro-meteorological variables are not the only drivers of ice-jam formation. The geomorphological indicators that control the formation of ice jams include the river slope, sinuosity, a barrier such as an island or a bridge, narrowing of the channel, and confluence of rivers. In the future, a geospatial model using deep learning will be developed to examine the impacts of these geospatial parameters on the ice-jam formation.

Author contribution

Fatemehalsadat Madaeni designed and carried out the experiments under Karem Chokmani and Saeid Homayouni supervision. Fatemehalsadat Madaeni developed the model code and performed the simulations using hydro-meteorological and ice-jam data provided and validated by Rachid Lhissou. Fatemehalsadat Madaeni wrote the bulk of the paper with conceptual edits from Karem Chokmani and Saeid Homayouni. Yves Gauthier and Simon Tolszczuk-Leclerc helped in the refinement of the objectives and the revision of the methodological developments.

Acknowledgment

This study is part of the DAVE project, funded by the Defence Research and Development Canada (DRDC), Canadian Safety and Security Program (CSSP), with partners from Natural Resources Canada (NRCan), and Environment and Climate Change Canada.

References

Analytics, C. (2016). Anaconda Software Distribution: Version 2-2.4. 0.

Beltaos, S. (1993). Numerical computation of river ice jams. Canadian Journal of Civil Engineering, 20(1), 88-99.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., ... & Bengio, Y. (2010, June). Theano: A CPU and GPU math compiler in Python. In Proc. 9th Python in Science Conf (Vol. 1, pp. 3-10).

Brownlee, J. (2017). A gentle introduction to exploding gradients in neural networks. Retrieved from https://machinelearningmastery.com/exploding-gradients-in-neural-networks/.

Brunel, A., Pasquet, J., PASQUET, J., Rodriguez, N., Comby, F., Fouchez, D., & Chaumont, M. (2019). A CNN adapted to time series for the classification of Supernovae. Electronic Imaging, 2019(14), 90-1.

Brunner, G. W. (2002). Hec-ras (river analysis system). In North American Water and Environment Congress & Destructive Water (pp. 3782-3787). ASCE.

Carson, R. W., Beltaos, S., Healy, D., & Groeneveld, J. (2003, June). Tests of river ice jam models–phase 2. In Proceedings of the 12th Workshop on the Hydraulics of Ice Covered Rivers, Edmonton, Alta (pp. 19-20).

Carson, R., Beltaos, S., Groeneveld, J., Healy, D., She, Y., Malenchak, J., ... & Shen, H. T. (2011). Comparative testing of numerical models of river ice jams. Canadian Journal of Civil Engineering, 38(6), 669-678.

Chen, R., Wang, X., Zhang, W., Zhu, X., Li, A., & Yang, C. (2019). A hybrid CNN-LSTM model for typhoon formation forecasting. GeoInformatica, 23(3), 375-396.

Cui, Z., Chen, W., & Chen, Y. (2016). Multi-scale convolutional neural networks for time series classification. arXiv preprint arXiv:1603.06995.

Devineau, G., Moutarde, F., Xi, W., & Yang, J. (2018, May). Deep learning for hand gesture recognition on skeletal data. In 2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018) (pp. 106-113). IEEE.

Devineau, G., Xi, W., Moutarde, F., & Yang, J. (2018, June). Convolutional neural networks for multivariate time series classification using both inter-and intra-channel parallel convolutions. In Reconnaissance des Formes, Image, Apprentissage et Perception (RFIAP'2018).

Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S.K., Nouri, D., … & Degrave,J. (2015). Lasagne: First release. (Version v0.1). Zenodo. Retrieved from http://doi.org/10.5281/zenodo.27878.

Données Québec: Historique (publique) d'embâcles répertoriés au MSP - Données Québec,. Retrieved from https://www.donneesquebec.ca/recherche/dataset/historique-publique-d-embacles-repertories-au-msp. (last access: 15 June 2021).

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019, July). Deep neural network ensembles for time series classification. In 2019 International Joint Conference on Neural Networks (IJCNN) (pp. 1-6). IEEE.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019). Deep learning for time series classification: a review. Data Mining and Knowledge Discovery, 33(4), 917-963.

Gamboa, J. C. B. (2017). Deep learning for time-series analysis. arXiv preprint arXiv:1701.01887.

Gao, Y., Hendricks, L. A., Kuchenbecker, K. J., & Darrell, T. (2016, May). Deep learning for tactile understanding from visual and haptic data. In 2016 IEEE International Conference on Robotics and Automation (ICRA) (pp. 536-543). IEEE.

Garbin, C., Zhu, X., & Marques, O. (2020). Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimedia Tools and Applications, 1-39.

Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT press.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. Pattern Recognition, 77, 354-377.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362.

Hatami, N., Gavet, Y., & Debayle, J. (2018, April). Classification of time-series images using deep convolutional neural networks. In Tenth International Conference on Machine Vision (ICMV 2017) (Vol. 10696, p. 106960Y). International Society for Optics and Photonics.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. IEEE Annals of the History of Computing, 9(03), 90-95.

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.

Jordan, J. (2018). Setting the learning rate of your neural network. Retrieved from https://www. jeremyjordan. me/nn-learning-rate.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015, June). An empirical exploration of recurrent network architectures. In International conference on machine learning (pp. 2342-2350). PMLR.

Karim, F., Majumdar, S., Darabi, H., & Harford, S. (2019). Multivariate lstm-fcns for time series classification. Neural Networks, 116, 237-245.

Karpathy, A. (2017). Convolutional neural networks for visual recognition. Retrieved from http://cs231n.github.io/convolutional-networks/.

Li, X., Zhang, Y., Zhang, J., Chen, S., Marsic, I., Farneth, R. A., & Burd, R. S. (2017). Concurrent activity recognition with multimodal CNN-LSTM structure. arXiv preprint arXiv:1702.01638.

Lin, J., Williamson, S., Borne, K., & DeBarr, D. (2012). Pattern recognition in time series. Advances in Machine Learning and Data Mining for Astronomy, 1, 617-645.

Lindenschmidt, K. E. (2017). RIVICE—a non-proprietary, open-source, one-dimensional river-ice model. Water, 9(5), 314.

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019.

Lu, N., Wu, Y., Feng, L., & Song, J. (2018). Deep learning for fall detection: Three-dimensional CNN combined with LSTM on video kinematic data. IEEE journal of biomedical and health informatics, 23(1), 314-323.

Luan, Y., & Lin, S. (2019, March). Research on text classification based on CNN and LSTM. In 2019 IEEE international conference on artificial intelligence and computer applications (ICAICA) (pp. 352-355). IEEE.

Madaeni, F., Lhissou, R., Chokmani, K., Raymond, S., & Gauthier, Y. (2020). Ice jam formation, breakup and prediction methods based on hydroclimatic data using artificial intelligence: A review. Cold Regions Science and Technology, 103032.

Mahfouf, J. F., Brasnett, B., & Gagnon, S. (2007). A Canadian precipitation analysis (CaPA) project: Description and preliminary results. Atmosphere-ocean, 45(1), 1-17.

Massie, D.D., White, K.D., Daly, S.F., 2002. Application of neural networks to predict ice jam occurrence. Cold Reg. Sci. Technol. 35 (2), 115–122.

Mesinger, F., DiMego, G., Kalnay, E., Mitchell, K., Shafran, P. C., Ebisuzaki, W., ... & Shi, W. (2006). North American regional reanalysis. Bulletin of the American Meteorological Society, 87(3), 343-360.

Mutegeki, R., & Han, D. S. (2020, February). A CNN-LSTM approach to human activity recognition. In 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC) (pp. 362-366). IEEE.

National Hydro Network - NHN - GeoBase Series - Natural Resources Canada. Retrieved from https://open.canada.ca/data/en/dataset/a4b190fe-e090-4e6d-881e-b87956c07977.

National Hydrographic Network - Natural Resources Canada. Retrieved from https://www.nrcan.gc.ca/science-and-data/science-and-research/earth-sciences/geography/topographic-information/geobase-surface-water-program-geeau/national-hydrographic-network/21361.

Oh, S. L., Ng, E. Y., San Tan, R., & Acharya, U. R. (2018). Automated diagnosis of arrhythmia using combination of CNN and LSTM techniques with variable length heart beats. Computers in biology and medicine, 102, 278-287.

Olah, C. (2015). Understanding LSTM Networks. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Ombabi, A. H., Ouarda, W., & Alimi, A. M. (2020). Deep learning CNN–LSTM framework for Arabic sentiment analysis using textual information shared in social networks. Social Network Analysis and Mining, 10(1), 1-13.

Ordóñez, F. J., & Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. Sensors, 16(1), 115.

Parvin, H., Minaei, B., Beigi, A., & Helmi, H. (2011, April). Classification ensemble by genetic algorithms. In International Conference on Adaptive and Natural Computing Algorithms (pp. 391-399). Springer, Berlin, Heidelberg.

https://doi.org/10.5194/tc-2021-194
Preprint. Discussion started: 23 July 2021

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12.

Prowse, T. D., & Bonsal, B. R. (2004). Historical trends in river-ice break-up: a review. Hydrology Research, 35 (4-5), 281-293.

Prowse, T. D., Bonsal, B. R., Duguay, C. R., & Lacroix, M. P. (2007). River-ice break-up/freeze-up: a review of climatic drivers, historical trends and future predictions. Annals of Glaciology, 46, 443-451.

Raybaut, P. (2009). Spyder-documentation. Retrieved from pythonhosted. org.

Reback, J., McKinney, W., Den Van Bossche, J., Augspurger, T., Cloud, P., Klein, A., ... & Seabold, S. (2020). pandas-dev/pandas: Pandas 1.0. 3. Zenodo.

Sainath, T. N., Vinyals, O., Senior, A., & Sak, H. (2015, April). Convolutional, long short-term memory, fully connected deep neural networks. In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 4580-4584). IEEE.

She, X., & Zhang, D. (2018, December). Text classification based on hybrid CNN-LSTM hybrid model. In 2018 11th International Symposium on Computational Intelligence and Design (ISCID) (Vol. 2, pp. 185-189). IEEE.

Shouyu, C., & Honglan, J. (2005). Fuzzy Optimization Neural Network Approach for Ice Forecast in the Inner Mongolia Reach of the Yellow River/Approche d'Optimisation Floue de Réseau de Neurones pour la Prévision de la Glace Dans le Tronçon de Mongolie Intérieure du Fleuve Jaune. Hydrological sciences journal, 50(2).

Sosa, P. M. (2017). Twitter sentiment analysis using combined LSTM-CNN models. Eprint Arxiv, 1-9.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

The Atlas of Canada - Toporama - Natural Resources Canada. Retrieved from https://atlas.gc.ca/toporama/en/index.html.

Thornton, M.M., Shrestha, R., Wei, Y., Thornton, P.E., Kao, S. & Wilson, B.E. (2020). Daymet: Daily Surface Weather Data on a 1-km Grid for North America, Version 4. ORNL DAAC, Oak Ridge, Tennessee, USA.

Turcotte, B., & Morse, B. (2015, August). River ice breakup forecast and annual risk distribution in a climate change perspective. In 18th Workshop on the Hydraulics of Ice Covered Rivers, CGU HS Committee on River Ice Processes and the Environment, Quebec (Vol. 35).

Umer, M., Imtiaz, Z., Ullah, S., Mehmood, A., Choi, G. S., & On, B. W. (2020). Fake news stance detection using deep learning architecture (cnn-lstm). IEEE Access, 8, 156695-156706.

Wang, J., Yu, L. C., Lai, K. R., & Zhang, X. (2016, August). Dimensional sentiment analysis using a regional CNN-LSTM model. In Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers) (pp. 225-230).

Wang, J., Yu, L. C., Lai, K. R., & Zhang, X. (2019). Tree-structured regional CNN-LSTM model for dimensional sentiment analysis. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 28, 581-591.

White, K. D. (2003). Review of prediction methods for breakup ice jams. Canadian Journal of Civil Engineering, 30(1), 89-100.

Wong, S. C., Gatt, A., Stamatescu, V., & McDonnell, M. D. (2016, November). Understanding data augmentation for classification: when to warp?. In 2016 international conference on digital image computing: techniques and applications (DICTA) (pp. 1-6). IEEE

Wu, Z., Wang, X., Jiang, Y. G., Ye, H., & Xue, X. (2015, October). Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In Proceedings of the 23rd ACM international conference on Multimedia (pp. 461-470).

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.

Zhao, L., Hicks, F. E., & Fayek, A. R. (2012). Applicability of multilayer feed-forward neural networks to model the onset of river breakup. Cold Regions Science and Technology, 70, 32-42.

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014, June). Time series classification using multi-channels deep convolutional neural networks. In International Conference on Web-Age Information Management (pp. 298-310). Springer, Cham.

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2016). Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. Frontiers of Computer Science, 10(1), 96-112.