Investigating Temporal Convolutional Neural Networks for Satellite Image Time Series Classification [*]

James Brock
Department of Engineering Mathematics
University of Bristol
Bristol
jb75426@gmail.com

Zahraa S. Abdallah
Department of Engineering Mathematics
University of Bristol
Bristol
zahraa.abdallah@bristol.ac.uk

Abstract

Satellite Image Time Series (SITS) of the Earth's surface provide detailed land cover maps, with their quality in the spatial and temporal dimensions consistently improving. These image time series are integral for developing systems that aim to produce accurate, up-to-date land cover maps of the Earth's surface. Applications are wide-ranging, with notable examples including ecosystem mapping, vegetation process monitoring and anthropogenic land-use change tracking. Recently proposed methods for SITS classification have demonstrated respectable merit, but these methods tend to lack native mechanisms that exploit the temporal dimension of the data; commonly resulting in extensive data pre-processing prohibitively long training times. To overcome these shortcomings, this paper seeks to study and enhance the new proposed method for SITS classification from literature; namely Temporal CNNs. Comprehensive experiments are carried out on two benchmark SITS datasets with the results demonstrating that Temporal CNNs display a superior or competitive performance to the benchmark algorithms for both datasets. Investigations into the Temporal CNNs architecture also highlighted the non-trivial task of optimising the model for a new dataset.

***K*eywords** satellite imagery; time series classification; Temporal Convolutional Neural Networks; land-cover mapping; deep learning

1 Introduction

In recent decades there has been a dramatic increase in the availability of satellite-based earth imagery, resulting from *Earth Observation (EO)* programs including the Sentinel and Landsat satellites [1]. Such satellites are able to capture images with spatial resolutions ranging from 10-100m with an increasing temporal density [2, 3]. These visual records of the Earth's ecosystems have existed since the 1970s but it has only been within the last decade that the data been available for public use [4]. These datasets often contain channels of electromagnetic information not present in traditional 3-channel (*e.g.* RGB) images, enabling a greater degree of data analysis. Whilst additional features are valuable for land-cover classification tasks, it consequently reduces the interpretability of the data and typically enlarges their volume [5]. An example land-cover map for a SITS classification task can be observed in Figure 1.

The advent of modern computing power and public data availability has opened the avenue for the wider scientific community to utilise land-cover maps within studies of limited funding [1]. Most notably, this data has proven an essential asset for wide-scale ecological mapping and monitoring; a key tool for shaping future climate change policies. Indeed, the extensive multi-spectral bands offered by *EO* imagery enables an accurate and comprehensive mapping of the planet for the evaluation of forest fire spread, land-use change and vegetation cover [7]. Increasingly more traditional disciplines such as ecology and biology are coming to rely on land-cover maps to evaluate ecosystems and help inform public policies [8]; current applications of SITS classification have included agricultural crop monitoring [9, 2], mapping dead forest cover [10], monitoring tropical forest succession patterns [11], mapping plant communities

[*] *Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

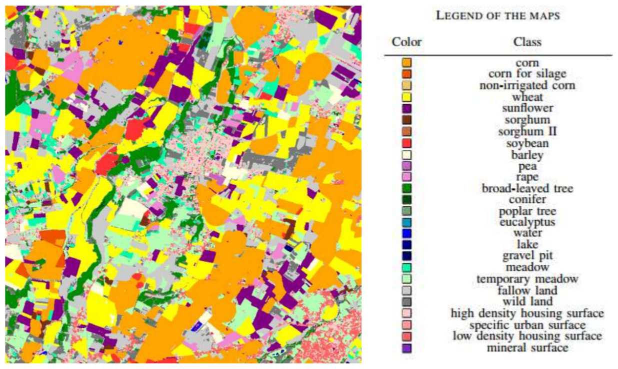


Figure 1: An example Land cover map from the SITS-TSI dataset used in this study. Each pixel in the image is assigned to a target class. This is snapshot of the dataset, with multiple images being stacked and geographically referenced to create the dataset. Image sourced from [6].

[12], and wider ecosystem monitoring as a whole [13]. With many sites being difficult to access and survey on the ground, satellite imagery offers a cheaper and more reliable alternative to previous mediums, with studies utilising SITS data often having reduced costs, fastest processing times and more accurate results.

This paper seeks to demonstrate the viability of Temporal CNNs for Satellite Image Time Series classification - focusing on the application of convolutions on the temporal channel. By performing experiments on two datasets with seasonal land-cover labels, the ability for the architecture to generalise will be thoroughly explored. The contributions of this paper can be summarised as:

- An exploration of a recently proposed Temporal CNNs architecture and how well it can generalise to different datasets
- Evaluating how sensitive the architecture's parameters are to the data it is applied to - focusing on the effect the temporal and spectral channels may have
- Identifying any significant advantages or shortcomings of the architecture for the purpose to generating deeper insights into potential caveats
- Providing a comprehensive analysis of the Temporal CNNs architecture against other recently proposed algorithms for STIS classification

The experiments conducted in this paper were designed to incorporate many of the recommendations found in the literature relevant to Temporal CNNs for SITS classification. However, this paper does still heavily investigate optimal parameter settings for aspects of the model that include the width and depth of the model, the degree of temporal dilation, and aspects of regularisation, These factors are considered across two datasets simultaneously. Following the discovery of the optimal parameters for the Temporal CNN, an extensive study is conducted that compares the model to other contemporary algorithms for SITS classification.

The rest of this paper is structured as follows. Section 2 provides a background on past and current methods for time series and SITS classification. Section 3 discuss the fundamentals of Temporal CNN. Next, Section 4 outlines the methods explored within this paper including the experimental Temporal CNNs architecture. Section 5 outlines the datasets used in this study and the experimental setup. Section 6 comprises the bulk of the paper, in which the results are presented. Finally, Section 7 concludes the paper with a critical discussion of the results and their subsequent conclusions.

1.1 A note on terminology

For the context of this paper, it is useful to note the terminology for the various components related to the SITS classification domain. SITS is an abbreviation for *Satellite Image Time Series* which are series of images collected over a prolonged time period via optical remote sensing methods which depict Earth observation data [14]. These images are multi-spectral in nature, meaning that they are composed from a series of sensors operating in various areas of the electromagnetic. By extension, this also means that the data is multi-variate if more than one sensor value constitutes an observation. This data is therefore comprised of temporal components integrated with spectral and spatial dimensions, inherently making it 3D in nature. SITS datasets focus on documenting changes in the land cover of an area by monitoring and mapping the differences between the various temporal snapshots. Given this, SITS classification is a form of multi-variate time series classification; multi-variate in the sense of temporal snapshots comprised of spectral and spatial information. This data forms highly detailed remotely sensed land-cover maps that can be used for classification and change detection tasks through the employment of various time series analysis techniques.

2 Related Work

The availability and quality of SITS data has generated a renewed interest in discovering more efficient classification algorithms through which to process the data. Previous studies have demonstrated good performances for SITS classification when using traditional approaches [15], but these approaches commonly lack intrinsic methods of processing the temporal dimension of the data. Conversely, various deep learning methods have recently been studied that contain mechanisms for processing the temporal dimension of the data whilst discovering meaningful features within the data [3, 16]. Given the vast areas for which high quality SITS data now exists for, it is imperative that deep learning is thoroughly explored for the task so researchers can learn how best to apply it.

2.1 Supervised learning for SITS classification

Traditional approaches have commonly employed models that require features to be manually crafted , such as the Normalised Difference Vegetation Index (NDVI) [17]. Contemporary papers that introduce a new method for time series land-cover classification commonly compare performance with supervised classifiers such as Support Vector Machines (SVMs) [18], Random Forests (RFs) [19], and recurrent neural network (RNNs) variants [20]. RNNs and ensemble methods such as RFs have consistently proven to be strong baselines for remote-sensing classification tasks [9], resulting in their continued adoption. Of these methods, Random Forests and SVMs have no mechanisms for processing the temporal dimension of the data and so valuable information provided by seasonal changes is not utilised [3]. Previous studies incorporating these methods include mapping dead tree cover [10], mapping floodplain grassland communities [12], monitoring ecosystem states and dynamics [13] and agricultural crop analysis [20].

Comprehensive reviews of traditional classifiers for TSC report that machine learning and ensemble algorithms are more accurate and efficient than conventional classifiers such as SVMs when faced with high-dimensional, complex datasets [15, 21]. The review by Rustowicz [21] avidly emphasised the potential that CNNs hold in the TSC domain. Whilst these methods often convey respectable performance, their main drawback is the requirement for feature engineering which has a limited capacity for automation and is characteristically rigid [9].

Each of these non-parametric methods generally do not require large amounts of training data to achieve good performance; although SVMs struggle in small feature spaces [22]. Each method is computationally intense, with NNs and RFs acting as black boxes, making the interpretation of results difficult despite RFs having the capacity to determine variable importance [18]. Lastly, each method requires extensively tuned input parameters for optimal performance, which may exacerbate the already long training times [15]. If over-fitting is overcome, then NNs generally perform the best since feature representations are learnt automatically, reducing the required pre-processing needed for model development [20], although the discovery of a well-performing model is not trivial.

2.2 Deep learning approach for SITS

Recently, a wide-ranging variety of deep learning models have been applied to SITS tasks and time series classification (TSC) as a whole. Reviews such as those conducted by Fawaz et al. [16] and Hatami et al. [23] methodically compare the most notable methodologies, being tested on benchmarks such as the UCR archive [24]. The most common forms of deep neural networks (DNNs) for SITS classification as observed in papers like [25] are generally a choice between CNNs [10, 26] or RNN varieties such as Bi-LSTMs and GRUs [27]. More novel approaches utilise mechanisms such as self-attention [5] or modify coarsely detailed input images using approaches such as pixel set-encoders [28]. These approaches have been shown to offer state-of-the-art performance but only in limited circumstances, such as when the input data lacks a high spatial resolution [28] or has undergone little to none pre-processing [20].

RNNs and their variants are considered a prime candidate for time series representation due to their inherent nature to model sequential data [9]. They introduce a specialised component that represents the temporal dependency at various time-spans with gated recurrent connections [17]. Baseline LSTMs have been extended to make use of 2D convolutional layers to act as spatial feature extractors that provide inputs to the LSTM [29]; demonstrating the value of employing both the temporal and spatial channels [9].

Where RNNs seem appropriate for handling temporal relationships in time series data, CNNs are correspondingly remarkable for their ability to extract spatial features within 2D spaces such as images [30]. CNNs have translated well to the SITS domain, with 1D,2D and 3D ConvNets suggested for processing the spatial and spectral dimensions [31]. Models such as the revolutionary AlexNet have recently been adapted for use within the TSC domain [32], introducing an ensemble deep learning classifier based on the Inception-v4 classifier [33]. The model showed competitive performance with state-of-the-art methods, but with the ability to learn from an unprecedented quantity of training data within a short time frame. This demonstrates the strength of CNNs over RNN variants as the operations of a CNN can be parallelized, despite the high computational cost they incur in comparison to RNNs which has made them the predominant architecture for TSC tasks [32].

2.3 Investigation of Temporal CNNs

Building from the earlier discussion on CNNs for TSC tasks, CNNs of various dimensional capacities (1D, 2D, 3D) have been applied to time series datasets. 1D and 2D CNNs have demonstrated their effectiveness for multi-source, multi-temporal datasets, but are not able to take advantage of the temporal dimension [34]. The convolutions are either applied in the spectral or spatial domain, excluding the temporal domain. As a result, the order of the images has no influence on classification, removing an essential component for applications that use features such as vegetation growth patterns [3] to aid classification, *e.g.* crop classification [21]. In light of this limitation, a series of alternative methodologies have been proposed, including ensemble methods [35], methods derived from statistical analysis [1], pixel-set encoders [28], self-attention [20], and most notably for this paper, CNNs that incorporate the temporal channel [3, 31].

Temporal Convolutional Neural Networks (Temporal CNNs) were first introduced for action segmentation [36] and object detection within videos [37], achieving competitive or superior performance to comparable methods whilst boasting a significant reduction in training time. Since then, they have been expanded into the domain of sequence modeling, most notably for SITS classification using crop data [3]. The deep learning approach applies convolutions in the temporal dimension in order to automatically learn temporal and spectral features of a dataset.

Given the structural similarities of video data to SITS and the promising results Temporal CNNs have displayed for SITS classification [9] and remote sensing [38], the choice to investigate this methodology is markedly justified. Unlike the more traditional 1D and 2D-CNNs, Temporal CNN architectures are able to fully exploit the temporal structure of SITS. 3D-CNNs have also shown promise for handling the spatial and temporal dimensions in video classification tasks, which could feasibly be adapted to SITS classification [39]. The aforementioned works highlight the potential of applying Temporal CNNs for SITS classification, with Temporal CNNs possessing higher accuracies and shorter training times when compared to more traditional approaches such as RFs and RNNs [3]. Temporal CNNs are a significant breakthrough for SITS tasks, with their prominence for handling temporal data well documented [40]. They are able to match and surpass more traditional methods such as RNNs and RFs by using an end-to-end deep learning architecture with significantly reduced training times [40]. Results from papers applying Temporal CNNs to SITS classification tasks such as [3] achieved overall accuracies of between 93-94% [3], outclassing RNNs and Random Forests by 1-3%. Another paper that also applied Temporal CNNs to SITS data in the form of a 2D CNN [41] achieved overall accuracies between 88% and 94%. From these results, a clear case can be made for exploring the use of Temporal CNNs within wider domains.

3 Temporal CNN Fundamentals

This section aims at presenting Temporal CNN models and their characteristic components. The theory for Temporal CNNs is introduced, followed by an overview of the Temporal CNN architecture that is experimented on in section 6 is introduced [2]. The two defining characteristics of a Temporal CNN include:

- The ability to take an input sequence of any length and map it to an output sequence of the same length, just like an RNN.
- The convolutions used by the architecture are *causal*, meaning that there is no 'leakage' of information from future to past.

To achieve these points, two components are required: causal convolutions and dilated convolutions.

3.0.1 Causal convolutions

To keep subsequent layers the same length as the previous ones, the Temporal CNN uses a 1D fully-convolutional network (FCN) architecture, where each hidden layer is the same length as the input layer, and a zero padding of length

[2]Appendix A contains information on general deep learning concepts that may be used as a useful prerequisite to this section.

(kernel size - 1) is added. To ensure convolutions are *causal*, convolutions within the architecture at time t are convolved only with elements from time t and earlier within the previous layer [37]. This rudimentary setup allows a Temporal CNN model to analyse the full history of the data available, although effectively capturing these long-term dependencies is challenging [37].

Convolutional layers extract features by correlating the input features with a set of D_h convolutional kernels $\theta = (\theta_0, \theta_1, ..., \theta_{D_h})$ with $\theta_d \in \mathbb{R}^{KxD}$. These kernels employ a convolutional operation followed by an elementwise non-linear activation function such as ReLU [20]. The size of the convolutional kernel K controls the size of the receptive field of each layer. The receptive field grows as the number of layers increases. The learned kernel is thus able to extract features in the temporal domain [37]. Aspects of the 1D FCN used can be customised to the specific domain task, with three key tuneable hyper-parameters to consider. These are the number of convolutional kernels used by the 1D convolutional layers which determine the dimensionality of hidden states, the respective kernel sizes $K \in \{3, 5, 7, 9\}$ and the dropout probability which is between 0 and 1. The number of convolutional kernels used is generally to the order of 2^n for a reasonably small value of n [20].

Despite tuning these hyper-parameters, the network is still incomplete. The disadvantage to using this design is that in order to achieve an effectively long history size, it is required to have either an extremely deep network and/or very large filters. Doing either of these however will result in a large model that may be infeasible to train [40].To overcome this issue, *Dilated Convolutions* [42, 43] are used to allow for both very deep networks with very long effective histories.

3.0.2 Dilated convolutions

Employing only simple causal convolutions on the temporal dimension of the time series data can only analyse the history of the series with a size linear to the depth of the network [40]. This aspect makes it challenging to apply causal convolutions on time series, especially those comprised of many timestamps. A suggested solution to this problem is the usage of dilated convolutions that trace their origins to Yu & Koltun [43], with a similar idea presented in [42]. Dilated convolutions are able to provide an exponentially large receptive field [44]. For a 1D input sequence $x \in R^n$ and a filter $f : \{0, ..., k-1\} \to R$, the dilated convolution operation F on element s of the sequence can be defined as [40]:

$$F(s) = (\mathbf{x} *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \mathbf{x}_{s-d \cdot i} \tag{1}$$

where d is the dilation factor, k is the filter size, and $s - d \cdot i$ accounts for the direction of the past. In this way, the effect of dilation is equivalent to including a fixed step between every two adjacent filter taps. If $d = 1$ then a dilated convolution reverts to a regular convolutional operation. Larger values of d enables an output at the top level to represent a wider range of inputs which expands the size of the receptive field [40]. An illustration of a dilated convolution is provided in Figure 2.

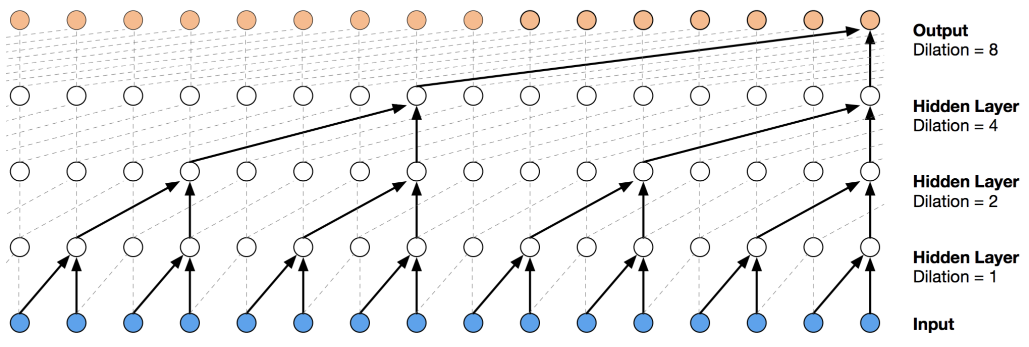


Figure 2: A dilated causal convolution with dilation factors d = 1, 2, 4 and filter size k = 3. The receptive field is able to cover all values from the input sequence. Image sourced from [40].

Through the aforementioned mechanisms, two methods of altering the receptive field of the Temporal CNN are created. Either the filter size k can be increased or the dilation factor d can be. The observable history of any layer within the Temporal CNN can thus be defined as: $(k-1)d$. A common procedure to employ when using dilated convolutions is to increase d exponentially with respect to the depth of the network such that $d = O(2^i)$ at layer i of the network. Doing so guarantees that there is always some filter that hits each input within the effective history whilst simultaneously enabling for an extremely large history [40],[43].

4 State-of-the-art methods in SITS

This section reviews a number of benchmark algorithms to offer a comprehensive overview of the methodologies available for SITS classification in contrast to the Temporal CNN. These algorithms are now briefly introduced.

4.1 Temporal CNN network

The architecture being investigated is that proposed by Pelletier et al. [3], which has seen recent usage in [45, 20]. Figure 3 displays the Temporal CNN architecture that this paper experiments on, the only difference being that the 3 output classes are replaced by the appropriate number of output classes for each dataset. The architecture is composed of three convolutional layers, one dense layer and one Softmax layer. The filter size is set to five [45]. The outputs of all layers except the Softmax layer employ batch normalisation and ReLU operations. The number of units in the convolutional layers is set to 128 units and 256 units in the dense layer.

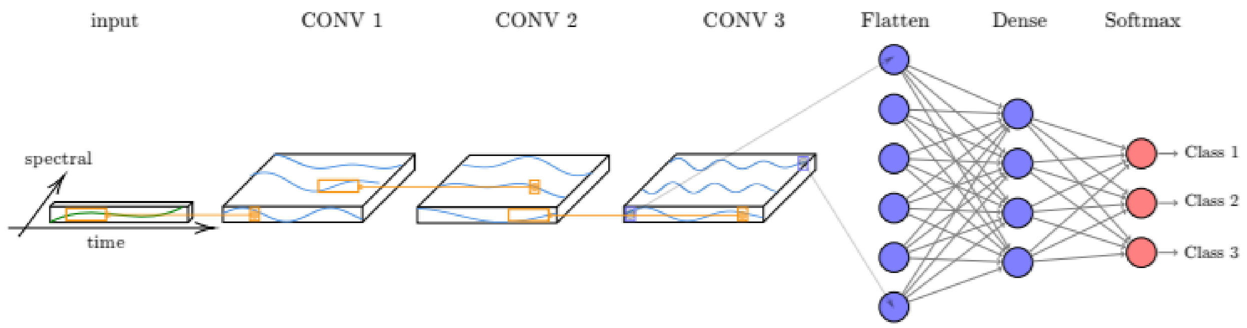


Figure 3: The input to the architecture is a multivariate time series; SITS in this case. Three consecutive convolutional layers are applied that each use dilated convolutions. This is then followed by one dense layer and a final Softmax layer that produces the output class predictions. Image sourced from [3].

The regularisation techniques employed by [3] are also repeated here to help control overfitting. The dropout is between 0.182 and 0.25 depending on the dataset being used; this is also reflected in their follow up paper: [45]. The validation set size selected to be 10% rather than 5% to help improve the clarification of the cross validation results. Weight decay is also retained as it was shown to have a positive effect on model performance. The parameters of the network are trained using the Adam optimiser [46] with default settings, using a range of batch size of 128 or 256 depending on the dataset. The models are trained using 20 and 30 epochs respective of the dataset used.

4.2 Multi-Channel Deep CNN (MCDCNN)

The methodology proposed by Zheng et al. [47, 48] introduced a multi-channel deep CNN for time series classification. The aim of the architecture is to exploit a presumed independence between the different features of the multi-modal time series data by applying the convolutions independently (*i.e.* parallel) on each dimension of the input. Given a multivariate time series, the architecture first learns features from the individual univariate time series in each channel, which is then concatenated together to create an overall feature representation that is fed to the final layer. Classification is performed through a Multilayer Perceptron (MLP) to calculate class probabilities. Predictions are generated using the categorical crossentropy loss function paired with a Softmax output layer and an Adam optimiser.

The process can be described as follows: each dimension for an input multivariate TS goes through two convolutional stages using 8 filters of length 5 with a ReLU activation function. Each convolution is followed by a max pooling operation using a kernel of size 2. The outputs of the second convolution for each dimension is then concatenated together which is then fed to a fully-connected layer with 732 neurons with ReLU as the activation function. A Softmax classifier is then used to make predictions with the layer containing a number of neurons equal to the number of classes [48].

4.3 Time-CNN

The Time-CNN as introduced by Zhao et al. [31] seeks to build upon the work of the MCDCNN, addressing the limitations they see with model. They argue that the model cannot learn the relationships between the various univariate time series that compose the multi-modal data. To overcome this issue they modify the MCDCNN algorithm in the following ways:

- Replace the categorical crossentropy loss and Softmax output layer with an MSE loss and sigmoid output layer
- Employ Average Pooling instead of the standard Max Pooling
- Omit the use of a pooling layer after the last convolutional layer

The authors hoped to that instead of learning the features from individual univariate time series, the multivariate time series is jointly trained during the feature extraction process prior to the calculation of the class prediction. The network as described in [49] is composed of two consecutive convolutional layers with 6 and 12 filters respectively which is then followed by a local average pooling operation of length 3. The sigmoid activation function is adopted as the activation function of the convolutional layers. The output layer consists of a fully-connected layer with neurons equal to the number of output classes [16]. The model is compiled with an Adam optimser.

4.4 Recurrent Neural Network (RNN)

Recurrent Neural Networks are regularly used as a comparative baseline within deep learning research for time series classification [5, 40]. These networks share the learned features across different positions within a sequence; intuitively modeling the temporal dependencies of the data. However, as the error is back-propagated at each time step, the computational cost can become significant and cause potential learning issues such as the vanishing gradient problem, in which inputs earlier in the sequence are forgotten. More recent RNN architectures use LSTM and GRU units that help to capture the long distance connections and solve the vanishing gradient problem [3]. The specific RNN architecture used in this comparison is the one proposed by Pelletier et al. [3] which uses three stacked bidirectional GRUs, one dense layer and a Softmax output layer with Adam optimisation. Multiple models were developed for both datasets with only the best results on the test data being reported. Early stopping was used to reduce overfitting with the validation loss being the monitored metric.

4.5 InceptionTime

InceptionTime [32] is an ensemble of five deep CNN models specialised for image recognition tasks, being most heavily inspired by the Inception-v4 CNN network from which the model derives its name. It is reported in the introductory paper that the model outperforms the previous state-of-the-art models such as HIVE-COTE [50] and does so whilst being much more scalable. This model has demonstrated strong performance for SITS classification tasks; having previously been applied to the SITS-TSI dataset. The core of the InceptionTime model is the Inception module which works by applying several filters of various resolutions to the input multi-modal time series data.

The Inception module of the network is composed of the following layers:

- A bottleneck layer to reduce the dimensionality (*i.e.* depth) of the inputs, consequently cutting the computational cost by reducing the number of parameters, speeding up training and improving generalisation.
- The output of the bottleneck is fed to three 1D convolutional layers of kernel size 10, 20 and 40.
- The input of the Inception model is passed through a max pooling layer of size 3 and then through a bottleneck layer.
- The layer is a depth concatenation layer where the outputs of the four convolutional layers at the second step are concatenated along the depth dimension.

5 Overview of study sites and experimental setup

This section provides an overview of the two datasets used in training and evaluation of the studied methods. Summaries of the two datasets and their reference data are first given. This is then followed by an overview of the data preprocessing steps. Finally the performance metrics are briefly discussed.

5.1 TiSeLaC dataset details

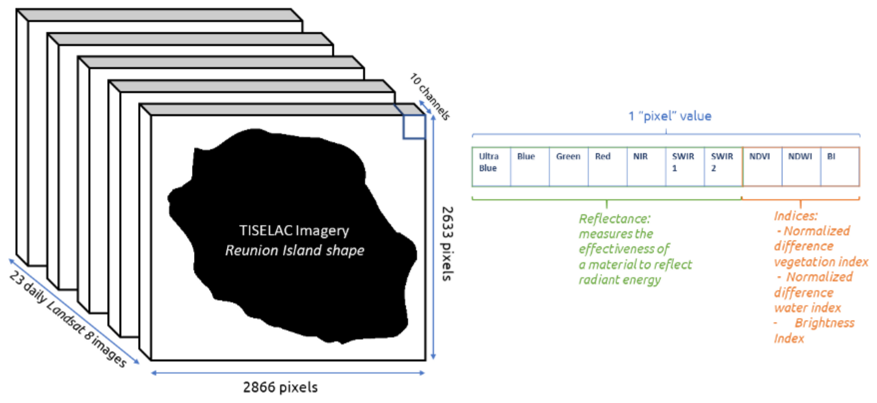
The first dataset comes from the Time Series Land Cover Classification Challenge (TiSeLaC) [51] that covers Reunion Island and is roughly 80MB in size. Between the training and test datasets, there are a total of 99,687 time series represented by pixels from 23 satellite images that are taken over the annual period of 2014 at a spatial resolution of 30m. Each image contains 2866x2633 pixels but many pixels are left blank. These images use the L2A processing level, with the source data having been further processed to fill cloudy observations via a pixel-wise multi-temporal linear interpolation on each of the multi-spectral bands (OLI) independently. This enabled the computation of 3 complementary radiometric indices (NDVI, NDWI and BI). This processing provides a total of 10 features per each pixel at each timestamp.

Each sample in the dataset is temporally ordered, meaning that features 1 to 10 represent the first timestamp and features 220 to 230 represent the last timestamp. The order of the features is also consistent, being composed of 7 surface reflectance values (Ultra Blue, Blue, Green, Red, NIR, SWIR1 and SWIR2) and 3 indices calculated from these surface reflectance values (NDVI, NDWI and BI).

Each sample is classified as one of 9 possible classes. The reference land-cover labels were gathered from two publicly available datasets, namely the 2012 Corine Land Cover (CLC) map and the 2014 farmers' graphical land parcel registration (RPG). Of the potential classes, the most significant ones were retained and spatial processing was applied which was aided by photo-interpretation. This was performed to ensure consistency with the image geometry. Pixel-based random sampling was applied to the dataset to produce the most balanced and representative ground truth possible. The distribution of classes in the TiSeLaC dataset are provided in Table 1. As can be seen, there are some noticeable class imbalances in the *Other crops* and *Other built-up surfaces* classes that the trained models will need to address. Figure 4 provides an overview of the TiSeLaC dataset composition.

Class ID	**Class Name**	**# Instances Train**	**# Instances Test**
1	Urban Areas	16000	4000
2	Other built-up surfaces	3236	647
3	Forests	16000	4000
4	Sparse Vegetation	16000	3398
5	Rocks and bare soil	12942	2599
6	Grassland	5681	1136
7	Sugarcane crops	7656	1531
8	Other crops	1600	154
9	Water	2599	519

Table 1: Overview of class distributions in the training and testing datasets for the TiSeLaC dataset.



10 channels
TISELAC Imagery
Reunion Island shape
2633 pixels
2866 pixels
23 daily Landsat 8 images
1 "pixel" value
Ultra Blue
Blue
Green
Red
NIR
SWIR 1
SWIR 2
NDVI
NDWI
BI
Reflectance: measures the effectiveness of a material to reflect radiant energy
Indices:
- Normalized difference vegetation index
- Normalized difference water index
- Brightness Index

Figure 4: The Reunion Island study site and the corresponding composition of the dataset. 7 channels are raw sensor values whereas the final 3 are calculated manually. Image sourced from [49].

5.2 SITS-TSI dataset

The second dataset named the SITS-TSI dataset originates from a paper by Tan et al. [6]. The dataset consists of 46 chronological FORMOSAT-2 satellite images of an unspecified study area at an 8m spatial resolution with each image containing 1 million pixels, hence each image covers $64km^2$. Each pixel in the image forms a time series of a length of 46, resulting in a dataset of 1 million time series. Only a single value is provided at each timestamp for each pixel with there being no mention of how this value is calculated [6].

Each image has been manually corrected by geo-science experts using photo-interpretation, urban databases and ground-sampling campaigns. This ensures a reliable and accurate ground-truth labeling for each sample. Each sample is classified as one of 24 possible classes, with class imbalances being more severe than in the TiSeLaC dataset.

From this dataset, 10 cross-validation folds were created as there are no defined training/testing datasets. Each fold contains 1 million samples, but training on all ten folds yields 10 million data samples. One fold is 270MB in size with all ten folds being roughly 2.64GB. Details of how this cross-validation set was created are not included within [6] and so the results on a single fold of data may be biased towards that particular fold. To ensure a consistency of results, each model is trained on all ten folds of the dataset and then compared to each other. Figure 1 gives an example Land Cover map for the SITS-TSI dataset.

5.3 Dataset preprocessing

To comprehensively evaluate the effectiveness of Temporal CNNs, the two datasets introduced earlier are used for training and testing purposes. Since the characteristics of these datasets vary greatly, they were be used to evaluate how the following factors affect performance: the number and type of input feature channels, class imbalances, the number of output classes, temporal sequence length and number of training samples available. Once conclusions were made in regards to these factors, further experiments investigated hyper-parameter tuning and the altering of the architectural configuration.

Prior to training, each dataset was split into a training, validation and test set. The test set size is dependent on the dataset being used, but the training data is always split to a ratio of 80:20 to provide training and validation data. This was done to ensure that over-fitting could be monitored and a fair analysis could be made. The feature data in each dataset was also normalised using a min-max normalisation per feature type. Traditional min-max normalisation subtracts the minimum and then divides by the range (*i.e.* the maximum value minus the minimum value). In [3], they note that this method is sensitive to extreme values and so instead use the 98$\%$ percentile rather than the minimum or maximum value. For each feature type per timestamp, the percentile values are extracted and the normalisation is applied, assisting the back-propagation of data through the neural network. Normalisation ensures each feature value in the dataset is scaled between 0 and 1, ensuring each feature uses the same scale and distribution, ensuring no single input channel influences the model performance disproportionately. The target values were also converted into a one-hot encoding for use in the training step for calculating the loss.

5.4 Performance metrics

When evaluating the various classification algorithms, quantitative assessments are enabled via the use of the Overall Accuracy (OA) and F1 score. A per-class analysis is graphically provided by the use of confusion matrices [52]. The training time was also investigated to help distinguish similarly performing models. hese evaluation metrics are typically popular and offer a diverse interpretation of the results [53]. To monitor the progress of learning during the model's during training, Categorical Cross Entropy was employed [54]. Finally, to monitor overfitting, training graphs and cross-validation techniques were employed to assure confidence in the training and testing of each model across all classes.

6 Experimental Results

This section reports the results of the experiments carried out on the methods discussed in Section 4. These experiments are designed to study:

- Which dataset dependent features affect Temporal CNN performance,
- Which architecture configuration alterations affect performance the most,
- The degree to which the Temporal CNN architecture can generalise,
- Which hyper-parameter values for regularisation methods provide optimal performance and whether the dataset employed affects these values.

- How Temporal CNN compares to other SITS classification methodologies,

Due to processing limitations, only the best experimental result for each model's performance and time is reported in place of an average of multiple runs. To enable a fair comparison of training times for each methodology, when comparing times, each implemented neural network was trained using a batch size of 128. Each model was trained using early stopping which monitored the validation loss with epochs adjusted to prevent overfitting.

To clearly answer the questions the experiments pose, the rest of this section is laid out as follows: Section 6.1 explores the optimal architectural configuration for the Temporal CNN on both studied datasets using recommendations from Pelletier et al. [3]. Next, an overview of the investigation into regularisation for the Temporal CNN and the final optimised hyper-parameter values are given in Section 6.2. Sections 6.3 and6.4 report the performance of each benchmark model against the optimal Temporal CNN for the TiSeLaC and SITS-TSI datasets respectively. Any results not discussed in this section are provided in Appendix B.

6.1 Experiments on the Temporal CNN architecture

Testing of the Temporal CNN architecture included experiments that analysed factors such as how big and deep to construct models, the effect of changing the kernel size and how to control over-fitting by adjusting the dropout and batch size. Due to the smaller dataset size, experiments were carried out on the TiSeLaC dataset utilising the reduced training time. Promising architectures were then applied to a singular fold of the SITS-TSI dataset to confirm the ability of the model to generalise.

At the centre of this investigation is the prevalence of the *bias-variance-trade-off*. Since dataset size is a key component of the bias-variance-trade-off, the optimal model settings found for each dataset may vary. Within machine learning, the more complex a model is (*i.e.* more parameters), the higher its bias. Lower bias translates to a lower number of errors when making predictions on the training data, which may reduce the generalisation of the model. On the other hand, an increase in variance is observed as the model complexity increases for a fixed number of training samples. If the complexity of a model outstrips the provided data for training, it cannot effectively learn to accommodate new data outside of the observed training distribution [3].

Controlling this trade off between bias and variance within neural networks requires a careful consideration of model complexity and supplementary regularisation techniques. Within neural networks, the bias is solely influenced by the complexity of the model, whereas the variance is dependent on the amount of training data provided. Similar experimentation on model complexity conducted within [3] use the number of parameters within each generated model as a proxy for model complexity as their study context had a static data size and number of classes. This paper investigates model performance on two datasets, each being of vastly different sizes and containing a different number of classes. Therefore, the number of parameters cannot be as so assuredly linked to an increase in complexity. Model complexity within this context is hence observed as a combination of the depth (*i.e.* number of convolutional layers) and width (*i.e.* number of units in the convolutional and dense layers). Investigations into the optimal depth range from using 2 convolutional layers up to 5, with values for the number of units within the convolutional and dense layers ranging from 64 to 2048.

6.1.1 Investigating Optimal Width

To investigate the optimal width of each model, 50 architectures were generated by varying the configuration of various parameters. Of these 50 architectures, only the results with consistent values for the dropout, batch size and filter size are studied. More extreme configurations were tested on the TiSeLaC dataset due to the smaller size of the dataset. Promising configurations from these experiments were then applied to a singular fold of the SITS-TSI dataset and a few settings where then applied on all ten folds of the SITS-TSI dataset.

To find the optimal width of the network, the number of units in the convolutional layers was varied alongside the number of units in the fully connected layer. This method of experimentation generated insights into the number of required units for effective feature extraction in the convolutional layers, as well as what ratio of units to use in the subsequent fully connected dense layer. In accordance with the findings of [3] and [45] the number of convolutional units used is never below 64 due to adverse effects on performance. The results for these experiments are reported, with each model comprised of three convolutional layers, a dense layer and a Softmax layer. Dropout is set to 0.2, the filter size is set to 5 and a batch size of 128 is employed. The results in Table 2 and Table 3 displays the accuracies of the various models chosen with the bold accuracy of each table indicating the best performing architecture.

Although the tested configurations are not exhaustive, the results confer a number of conclusions. For the SITS-TSI dataset, since it only uses a single feature per timestamp, adjusting the width of the network seemed to have almost no effect on the observed performance as the extra neurons could not be effectively utilised. For both datasets, performance

# Convolutional Units	# Fully Connected Units	OA
64	64	90.9
64	128	92.64
64	256	92.81
64	512	91.33
128	128	93.57
128	256	**95.02**
256	256	94.58
512	256	93.48
512	512	93.36
512	1024	93.02
512	2048	93.54
1024	256	93.83

Table 2: Results for adjusting the width of the model on the TiSeLaC dataset.

# Convolutional Units	# Fully Connected Units	OA
64	256	86.95
128	256	**87.16**
256	128	86.87
256	256	87.13

Table 3: Results for adjusting the width of the model on one fold of the SITS-TSI dataset.

decreases when the number of convolutional units is greater than the number of fully connected units. For the TiSeLaC dataset, optimal results are displayed when there are either 128 or 256 units in the convolutional layers, with the number of fully connected units being either equal to or double the number of convolutional units. Smaller and larger numbers of units for the convolutional layers display adverse performance. The model either lacks the computational complexity to extract relevant features, or has too great a complexity that it looses the ability to sufficiently generalise. An experiment on all ten folds of the SITS-TSI dataset that used 256 convolutional and fully connected units produced the greatest accuracy of 96.58%.

Given these observations, the recommendation would be to employ either 128 or 256 units in the convolutional layers and 256 in the fully connected layer. Using the greater number for the choice of convolutional units will increase model complexity, resulting in longer training times and an increased risk of over-fitting. Adjusting the batch size and dropout therefore must be included during model development.

6.1.2 Investigating Optimal Depth

To investigate the depth of the neural network, 8 architectures were created: 4 per studied dataset. The number of convolutional layers varies between 2 and 5. Results from [3] demonstrated markedly reduced performance for values outside of the previously stated range. Their experiments kept the model complexity the same by reducing the width of the models as the number of layers increased. For the experiments within this paper, since the model complexity is partly dataset dependent, the complexity cannot be controlled in the same manner. Hence, the width of the architectures are kept constant in order to make fair comparisons. The width of each model is kept at 128 convolutional units and 256 full connected units in accordance to the findings of the previous section. These experiments contained various widths and dropout values. It was found that as the complexity of the model increased, updating the dropout correspondingly improved performance as expected.

Figure 5 shows that for the TiSeLaC dataset, 3 convolutional layers is optimal, whereas there is no discernible difference for the SITS-TSI dataset. It can be concluded the usage of 3 convolutional layers is optimal for both datasets. Values lower than this may reduce the ability of the model to learn effective feature representations, and more may increases complexity to the degree that it reduces generalisation and training speed. If a greater number of layers is desired, then an increase in batch size and dropout is recommended to combat over-fitting.

Due to the choice of dataset, the number of convolutional layers, and width of the network, the number of parameters in each constructed model ranges from 140,000 to $\sim$18 million. The optimal model for the TiSeLaC dataset contained $\sim$2.1 million parameters, and $\sim$3.8 million for the SITS-TSI dataset. The increase in parameters for the SITS-TSI dataset results from the number of output classes being predicted for; 24 instead of 9. In general, the number of parameters does not significantly affect the performance of the model, with results ranging from $\sim$91-95% for the

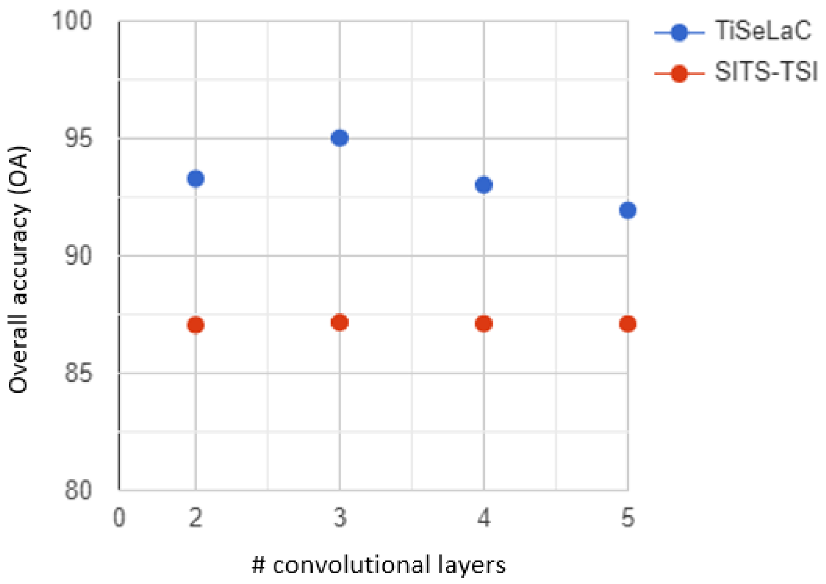


Figure 5: Overall accuracy as a function of the number of convolutional layers (depth influence) on the TiSeLaC and SITS-TSI datasets.

TiSeLaC dataset, and 87 $\pm$ 0.25% for SITS-TSI. Using a model with a lower number of parameters is preferable since the complexity and training time can be dramatically reduced.

If models need to be developed for a fixed complexity (*i.e.* number of parameters) given specific hardware or software limitations, the use of an inappropriate number of convolutional layers or number of units in the convolutional and fully-connected layers may lead to an underestimated Temporal CNN. In situations where additional resources are available, more exhaustive and computationally expensive cross-validation procedures could be used to further optimise the Temporal CNN architecture and avoid developing under-optimised models. Overall, through these experiments, the optimal architecture for the two datasets was found to be very similar, with the only major difference being the optimal kernel size, which is justified in the following section.

6.1.3 Influence of filter size

CNNs that employ temporal guidance perform convolutions across the temporal domain of a sample time series. Investigating the size of the filter for these convolutions is of special interest for finding which filter size works best for the temporal sampling used by each studied dataset. The gap in temporal sampling for the two datasets is not explicitly reported so it is unknown how a filter size of f will abstract the temporal information. The filter f (f being odd) abstracts the temporal information over $\pm(f-1)$ timestamps, before and after each point in the time series. This is commonly coined the *reach* of the convolution, corresponding to half of the width of the temporal neighbourhood used for the temporal convolutions [3]. Experiments conducted on the temporal reach in [3] found large filter sizes reduced the quality of the temporal resolution within the SITS data. Their studied dataset had considerably more timestamps, and also a denser temporal frequency that allowed them to experiment with larger filter sizes. This investigation hence focused on a smaller number of filter sizes, investigating three filters $f = 3, 5, 7$ for the TiSeLaC dataset as the temporal length is only 23 timestamps. Four filters are investigated for one fold of the SITS-TSI dataset where $f = 3, 5, 7, 9$. Since the temporal length is 46, an increase in the upper filter bound used could be investigated. Models with the following configuration were used during the selection stage of the filter size: 128 units per convolutional layer, 256 units in the fully-connected layer, a batch size of 128 and a dropout of 0.2.

Reach	2	4	6	8
TiSeLaC OA	93.94	**95.02**	93.93	NA
SITS-TSI OA (1 fold)	86.70	86.80	**86.98**	86.91

Table 4: Overall accuracy of each model as the filter size (reach) is fine-tuned. The bold value indicates the highest result.

Table 4 highlights that for the TiSeLaC dataset the maximum OA is achieved with a reach of 4 timestamps whereas for the SITS-TSI dataset, it is achieved with a reach of 6 timestamps. One factor that may have caused this result is the greater number of temporal observations in the SITS-TSI dataset. Results seem to be relatively consistent for the SITS-TSI dataset however, so these findings are not alarming. The effect of temporal reach is slightly more pronounced in the TiSeLaC data. Generally however, these results demonstrate the importance of high temporal resolution within SITS [3]. Due to the Sentinel-2 and FORMASAT-2 satellites having high acquisition frequencies, the Temporal CNN is able to abstract enough temporal information from the convolution operations. Crucially, the optimal value for the reach used heavily depends upon the focus of the SITS data. Tasks such as crop monitoring and urban expansion require much larger temporal reach values when capturing useful information for classification.

6.2 Controlling over-fitting

Similar studies have investigated regularisation techniques for Temporal CNNs [3], focusing on the use of four regularisation techniques: 1) regularisation of the weights, 2) dropout, 3) usage of a validation set and 4) using batch normalisation layers. The conclusions of [3] were that dropout had the most significant effect, whilst weight decay and the use of a validation test were nominal. Although insignificant for the goal of improving performance, the validation set is helpful for monitoring over-fitting and as such is retained here. The experiments conducted for controlling over-fitting therefore investigate the optimal value to use for dropout for each studied dataset.

6.2.1 Selecting the batch size

The batch size was found to have no notable adverse effects on performance, as was reported by Pelletier et al. [3]. Increasing the batch size does however cause a significant reduction in training time, whereby a doubling in the batch size corresponds to roughly halving the training time. The recommendation is to therefore use larger values for the batch size if memory is available as training can be vastly sped up, allowing for a greater degree of experimentation. A batch size of 128 is generally used to train Temporal CNN models on the TiSeLaC dataset, and 256 for the SITS-TSI dataset. When running time comparisons between models however, the batch size is consistently kept at 128 for fairness.

6.2.2 Selecting a value for dropout

Dropout is a technique for addressing overfitting [55], being the prime mechanism for ensuring generalisation within trained models as reported by [3]. The idea is to temporarily drop random units and their connections from layers within the neural network during training, preventing units from co-adapting too much. This process is done every epoch, and perhaps per mini-batch, thus creating an exponential number of "thinned" models. The presence of neurons is made unreliable, forcing the model to generalise through the sampling of many thinned networks. A parameter $\boldsymbol{p}$ is used to denote the percentage of neurons in a layer that are randomly dropped.

Dropout values between 0.1 and 0.5 were tested on the TiSeLaC data, with the results influencing the range of tested values on the SITS-TSI dataset. Larger values for dropout (>0.3) reduced the accuracy of models due to too much information loss, whereas lower dropouts (<0.15) did not account for over-fitting enough. From Figure 6, it can be shown that values between 0.15 and 0.25 are optimal due to their comparable values. The specific value of 0.182 was experimented with due to it being used as the dropout value for the pre-trained Breizhcrops Temporal CNN [45]. Following the results of this experiment, a dropout of 0.2 is adapted as the default value for future use and for the SITS-TSI dataset.

As can be seen in Figure 7, the over-fitting appears to be more controlled with a dropout of 0.5, but due to higher information loss the validation accuracy by 3.2%. The over-fitting observed with a dropout of 0.2 is not too severe and training can be stopped earlier.

6.3 Overview of results for the TiSeLaC dataset

The performance of each optimised model is given in Table 5. The reported accuracies were produced from predictions on the test set. Despite extensive hyper-parameter optimisations and the employment of procedures to reduce over-fitting, some over-fitting was still observed on the InceptionTime network due to its complexity. The observed test accuracies do however demonstrate the ability of each model to generalise well to unseen data. The training times and number of epochs to train each model are then reported in Table 6.

As evidenced from table 6, the Temporal CNN outperformed the other approaches by at least $\sim$2%. The Time-CNN outperformed the MCDCNN as expected due to it being reported as an improved version of the MCDCNN. Despite being a simpler model than the Time-CNN, the MCDCNN was slower in this case since it had to separate each

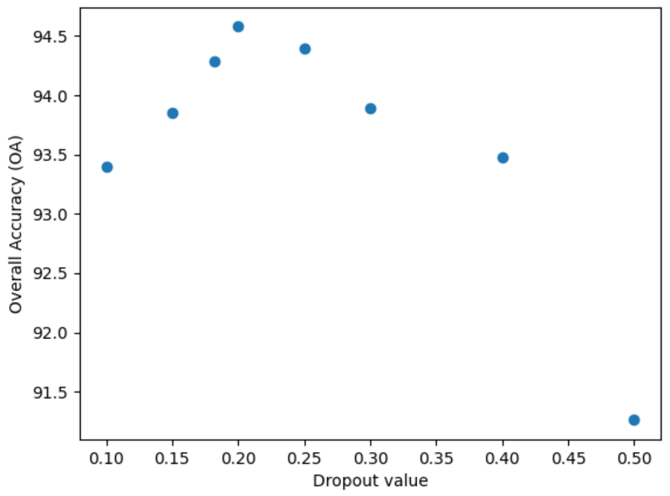


Figure 6: Overall Accuracy of each model as dropout is fine-tuned. Significant performance drops are witnessed at higher values.

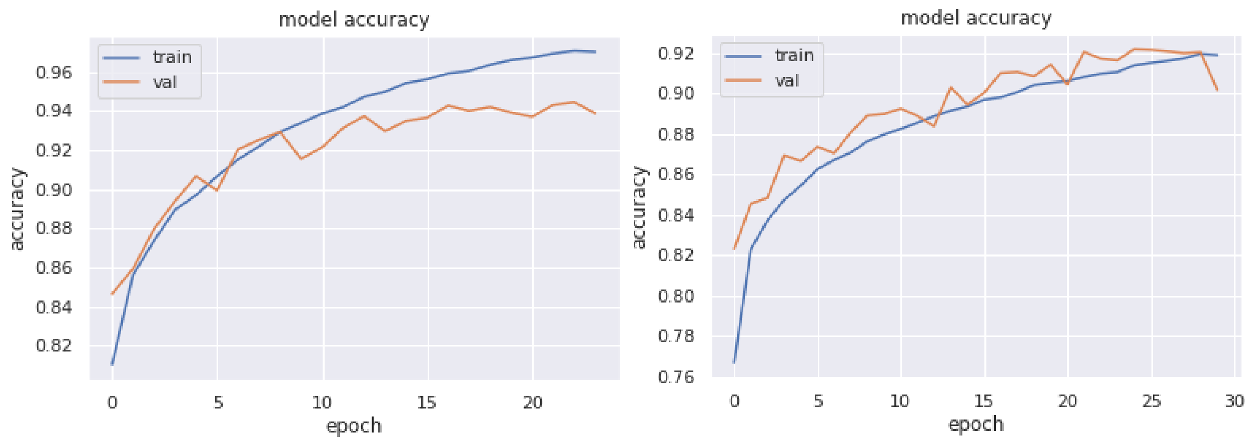


Figure 7: Training curve of the model with 0.2 dropout (L) against the training curve of the model with 0.5 dropout (R).

	Method - TiSeLaC dataset				
	Temporal CNN	Time-CNN	MCDCNN	RNN	InceptionTime
OA	**95.0**	89.2	86.6	92.7	92.2
F1	**94.9**	89.1	86.7	92.6	92.3

Table 5: Best accuracies on the TiSeLaC dataset for each method.

	Method - TiSeLaC dataset				
	Temporal CNN	Time-CNN	MCDCNN	RNN	InceptionTime
Time (s)	125	**60**	88	275	406
# Epochs	30	30	23	30	30

Table 6: Best training times for each method on the TiSeLaC dataset using similar hyper-parameter settings.

spectral channel for individual processing and then recombine them unlike the Time-CNN. The overall simplicity of the MCDCNN architecture displays a respectable accuracy but it is outclassed by every other method as it lacks the complexity to fully utilise the feature space.

The confusion matrices for the Temporal CNN and MCDCNN are depicted in Figure 8. The order of class labels are as follows: urban areas, other built-up surfaces, forests, spare vegetation, rocks and bare soil, grassland, sugarcane crops, other crops and water. By displaying the predictive power of the Temporal CNN against the weakest model (MCDCNN), comparisons can be made on how each model handles under-sampled classes and classes with similar temporal features. It can be shown that both classes are affected by the low frequency of the *other crops* class with both models frequently misclassifying it as *forest*. The MCDCNN also frequently misclassifies *other built-up areas* as *urban areas* due to their similar properties. The overall performance of the Temporal CNN was damaged by the sub-par performance on the under-sampled *other crops* class which was correctly classified 66% of the time compared to an average of 90%+ for the other classes.

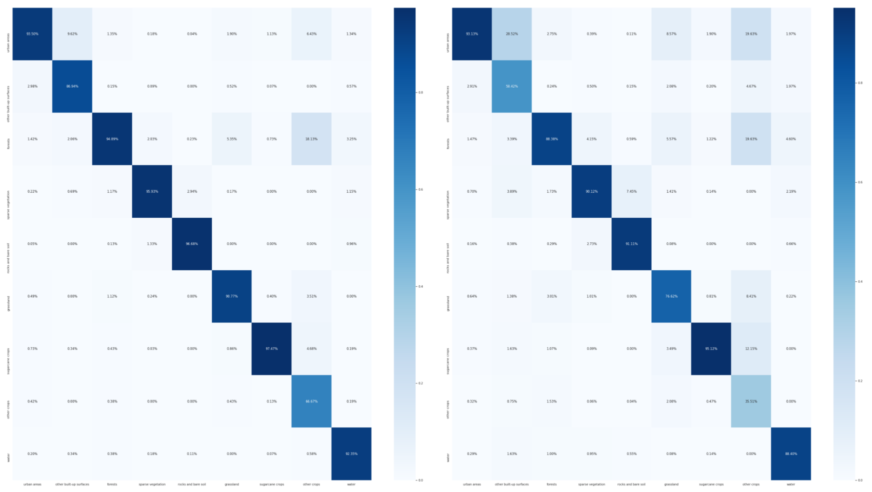


Figure 8: Confusion Matrix for the Temporal CNN on the TiSeLaC dataset (L). Confusion Matrix for the MCDCNN on the TiSeLaC dataset. (R).

A drawback evidenced in the lower performance of the MCDCNN is that the features in the TiSeLaC dataset are not inherently independent of each other as last three features that represent calculated indices are derived from values in the prior seven features. Furthermore, depending on the target associated with a feature vector, many of the channel values may be strongly correlated with each other, reducing the basis of the argument for the independence of features. Given that the core of the methodology is the use of a generic CNN with only minor modifications to process temporal data, it is unsurprising to observe it achieving the lowest performance.

In this context the training time is of lesser concern due to the small dataset size and comparatively low training times observed. Whilst the RNN had the second best accuracy, the long training time is indicative of a poor ability to scale to large data domains. Whilst the InceptionTime network boasted strong performance, in this instance it was outperformed by a more generic solution for speed and accuracy. It could be the case that the InceptionTime network works better within larger data domains due to the complexity of the network, and so a larger number of training samples may be of benefit.

In this context it can be demonstrated that the Temporal CNN works well when there are many features available per data point, with the model seemingly able to learn information encoding within the temporal dimension of the data within a short training time. The performance of the other methods are also respectable. The simpler CNN models of the Time-CNN and MCDCNN proposed for the dataset were both outperformed by the more sophisticated architectures (*i.e.* Temporal CNN, RNN and InceptionTime) by at least 3%. In SITS datasets that contain a multitude of input channels in which labeled data is less abundant, models that can fully leverage the input channels across the temporal domain generally yield better results. Of these specialised architectures, the Temporal CNN performed best with the contemporary RNN architecture narrowly outperforming the next best CNN architecture. The GRU layers of the RNN were able to effectively extract relevant features from the temporal dimension of the dataset. The InceptionTime architecture was also able to do this to a similar degree, but the complexity of the model outstripped the number of available training samples. For a smaller number of training samples, the RNN could learn effective feature representations more adeptly than the InceptionTime architecture, hinting that for the composition of the TiSeLaC dataset, the RNN architecture is more appropriate. The long training time of the RNN does however indicate a potential problem for scalability, especially given its relevant simplicity compared to the InceptionTime network.

As reflected in these results, the optimised Temporal CNN architecture was able to outperform the specialised and simpler architectures. Of the specialised architectures, it also had the lowest training time, demonstrating that high complexity is not essential for state-of-the-art results. Many of the experimental Temporal CNN architectures investigated in the experiments section routinely outperformed the RNN, demonstrating that extensive hyper-parameter optimisations are not a necessity for outstanding performance. These results demonstrate the intrinsic advantages Temporal CNNs possess for efficiently processing multi-spectral SITS data.

6.4 Overview of results for the SITS-TSI dataset

The Temporal CNN studied in section 6.3 was adjusted for use on the SITS-TSI dataset which was then compared to four other approaches. Prior to analysing the following results, a technical note needs to be raised. Since the SITS-TSI dataset only has one feature channel per timestamp, the MCDCNN essentially acts as a single channel 1D-CNN. This simple model was expected to be outperformed by the other neural networks but the results are still shown to be respectable. All of the neural networks were trained using all ten folds of the cross-validation data for training, validation and testing, accounting for a total of 10 million time series. Tables 7 and 8 provide the results of each approach on the SITS-TSI dataset.

Method - SITS-TSI dataset	**OA**	**F1**
Temporal CNN	96.6	96.7
Time-CNN	93.2	93.2
MCDCNN	88.3	88.3
InceptionTime	**97.9**	**98.0**
RNN	87.4	87.4

Table 7: Best accuracies on the SITS-TSI dataset for each method.

Method - SITS-TSI dataset	**Time (h)**	**# Epochs**
Temporal CNN (1 fold) | (10 folds)	0.181 | 2.99	20 | 20
Time-CNN (1 fold) | (10 folds)	0.124 | 1.413	20 | 20
MCDCNN (1 fold) | (10 folds)	**0.0976** | **0.99**	20 | 20
InceptionTime (1 fold) | (10 folds)	0.630 | 6.3	20 | 20
RNN (1 fold) | (10 folds)	0.857 | 2.14	20 | 5

Table 8: Best training times for each method on the SITS-TSI dataset using similar hyper-parameter settings.

The results on the SITS-TSI dataset highlight that of the neural networks tested, the CNN approaches yielded the best performance, all of them overshadowing the RNN. Each of the neural networks were able to demonstrate their ability to learn unique feature extractors from the data. These features were then able to accurately inform the class predictions of each network for them to make strong predictions.

Of the tested neural networks, the InceptionTime model displayed the best performance with a remarkable 97.9% accuracy over the 24 class labels. The Temporal CNN was also a very strong performer with an accuracy of 96.6% - just 1.3% worse than InceptionTime. Although the InceptionTime model showed the strongest performance, it was by far the slowest of the neural networks, with the Temporal CNN significantly outpacing it. When investigating training times, the MCDCNN was the fastest network on a single fold and all folds due to the relatively low number of parameters it contains. The Time-CNN in this instance was not the fastest since the MCDCNN did not need to separate out each feature channel for convolutional operations as there is only a single feature channel in use. Of the specialised networks, the Temporal CNN is the fastest on a single fold by a significant margin, but does not scale up as well on all ten folds. Although it displays slow training on a single fold, the early stopping mechanism caused the RNN to be faster than the Temporal CNN; though this is a heavily circumstantial occurrence. If more time was available to train further RNN models on all ten folds, it would likely be the case that the Temporal CNN is much faster in general as was seen on a singular fold.

The confusion matrices for the Temporal CNN and InceptionTime models can be found in Figure 12. The confusion matrices for the two models display incredibly strong per-class accuracies, with each model achieving over 92% accuracy for each class and averaging ∼96% for each class overall. Both models achieve 100% accuracy on the final class. Despite prevalent class imbalances, both models are still able to extract features well enough to represent these under-sampled classes.

Given the limited information provided by each individual time series within the dataset, the InceptionTime network was able to most effectively derive valuable feature representations with which to make predictions. The abundance of

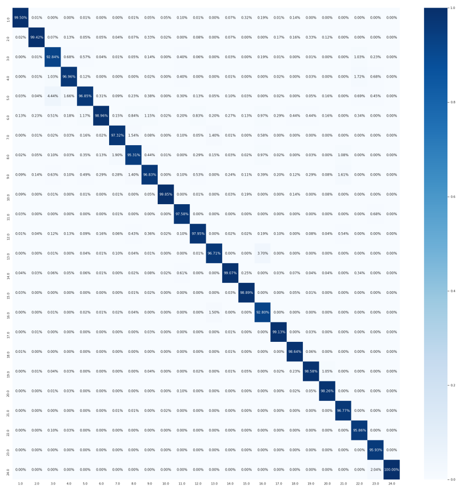
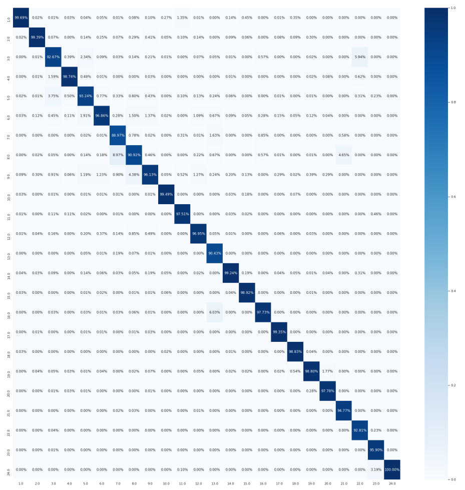



Figure 9: Confusion Matrix for the InceptionTime network on the SITS-TSI dataset (L). Confusion Matrix for the Temporal CNN on the SITS-TSI dataset (R).

training samples was leveraged by the complex architecture which capably generalised to the unseen test data. The same can be said for the Temporal CNN, with the dilated convolutions of the network also deriving effective features from the sparse data per timestamp per sample. Notably, it was able to do this in a much better time-frame than the InceptionTime network, with performance remaining relatively stable as the amount of training data increased.

The RNN was the worst performing neural network on this dataset, a stark contrast to the performance seen on the TiSeLaC dataset. This highlights the potential of CNNs for SITS classification in situations where few features are provided, but labeled training data is plentiful. Early stopping prevented the training time of the RNN becoming prohibitively long, converging to an optima after five epochs. In the general case though, the RNN can be expected to scale poorly in high data dimensions due to the costly and complicated training procedure. RNNs, even with contemporary components such as GRUs still have much longer processing times than Temporal CNNs in general [5]. This behaviour is observed even on the relatively short temporal sequences involved within this study, clearly displaying some of the main drawbacks of a pure RNN network for SITS classification. The RNN had per-class accuracies significantly lower than that of the Temporal CNN and InceptionTime networks. It also displayed significant deficiencies on the under-sampled classes.

The InceptionTime network is able to utilise a much greater amount of training data due its complexity, resulting in high performance and the accurate classification of rarer classes. However, this complexity reduces the scalability of the network, with the training time being prohibitively long when compared to the Temporal CNN. Due to the complexity of the InceptionTime network, the model tends to over-fit slightly on smaller datasets with regularisation techniques having little effect. Decreasing the batch size increases the generalisation of the model slightly but at the cost of drastically increasing the training time. The only case in which the network does not over-fit is when it is trained on all ten folds of the SITS-TSI dataset as enough data is provided to match the network complexity. Consequently, the training time begins to sharply increase.

It was also demonstrated that the relatively simple neural network architectures of the Time-CNN and MCDCNN were able to outperform the more traditional RNN. The Time-CNN, being an improvement of the MCDCNN yet again demonstrated the benefits of incorporating temporal processors that are more carefully developed. The MCDCNN, lacking an advanced mechanism for processing the temporal dimension beyond basic convolutions performed $\sim$5% worse than the Time-CNN, but still outperformed the RNN by $\sim$1%. The relatively low number of parameters that the Time-CNN and MCDCNN contains also demonstrates their ability to scale to large data domains.

7 Conclusion and Future Work

Temporal CNNs is extensively explored for use in SITS classification. In this paper, an extensive study is conducted to improve a baseline architecture of Temporal CNNs through the employment of various architectural optimisations, the regularisation techniques, and hyper-parameter tuning. Emphasis was placed on how to design architectures that can leverage the spectral and temporal dimensions within the studied datasets. These experiments were carried out on two SITS datasets that consist of vastly different feature spaces, allowing for a simultaneous investigation into the adaptability of Temporal CNNs for extracting features across diverse feature spaces.

During the testing of the Temporal CNN architecture across the two datasets, it was discovered that Temporal CNNs achieved superior or competitive performance in comparison to each of the benchmark methods, achieving accuracies of 95.02% and 96.58% on the TiSeLaC and SITS-TSI datasets respectively. The only benchmark method to outperform the Temporal CNN was the InceptionTime network on the SITS-TSI dataset. The Temporal CNN still demonstrated competitive performance however, doing so whilst having significantly reduced training times. These results confirmed the ability of Temporal CNNs to effectively utilise the spectral and temporal dimensions when applying convolutions. The impact of the architectural configuration on performance such as the width and depth was studied. The influence of regularisation mechanisms and temporal filter size was also examined. Many of these factors were not heavily affected by the feature space of the dataset used except that of filter size. It was found that model performance is partially dependent on the degree to which the temporal filter fits the observed sequential data. It was also shown that unnecessarily high dropout values that appeared to control over-fitting degenerated the overall performance on the test data and that more moderate values are appropriate.

Overall, these results demonstrate that Temporal CNNs provide state-of-the-art results for high-medium spatial resolution SITS datasets comprised of varying feature spaces and temporal length. The experimental architecture demonstrated a remarkable ability to scale up to larger data domains whilst simultaneously exhibiting minimal over-fitting. These results are in agreement with that of previous studies [40, 3], in which Temporal CNNs can be expected to outperform canonical RNNs and simpler CNN architectures for diverse SITS classification tasks. Accordingly, this study suggests that Temporal CNNs should be considered as a more appropriate starting model for deep learning applications for sequential data in place of RNNs [9].

Future work for this paper would entail a deeper analysis of the mechanisms and regularisation procedures explored here, using a cross-validation testing approach for optimising the architecture on each dataset. There would also be a focus on modifying the network to utilise the spatial dimension of SITS data. Developing such a model would further restrict which datasets the model could utilise but such spatial information is available for the TiSeLaC dataset so future studies could begin to widely incorporate it. By incorporating this feature, models will be able to learn more discriminant features, boosting performance further. Alternatively, research could be done into making the model more robust to noise within the dataset such as cloud cover, reducing the need for excessive pre-processing steps that may diminish the quality of the temporal sampling of the time series.

Applying any recent developments from transfer learning would also help to facilitate model development for a plethora of SITS classification tasks. Currently however, the lack of standardisation in SITS datasets and easily accessible data banks limits the scope of plausible transfer learning tasks. This lack of standardisation in dataset composition and preprocessing steps often means that pre-trained models are incompatible with many available datasets, resulting in severe performance degradation. Overcoming this requires a community effort to create a convention for standardising the composition and pre-processing used when creating public SITS datasets and making them as accessible as possible with accompanying documentation.

References

[1] Patrick Schäfer, Dirk Pflugmacher, Patrick Hostert, and Ulf Leser. Classifying land cover from satellite images using time series analytics. In *EDBT/ICDT Workshops*, pages 10–15, 2018.

[2] Marc Rußwurm, Romain Tavenard, Sébastien Lefèvre, and Marco Körner. Early classification for agricultural monitoring from satellite time series. *arXiv preprint arXiv:1908.10283*, 2019.

[3] Charlotte Pelletier, Geoffrey I. Webb, and François Petitjean. Temporal convolutional neural network for the classification of satellite image time series. *Remote Sensing*, 11(5), 2019.

[4] Valerie J. Pasquarella, Christopher E. Holden, Les Kaufman, and Curtis E. Woodcock. From imagery to ecology: leveraging time series of all available Landsat observations to map and monitor ecosystem state and dynamics. *Remote Sensing in Ecology and Conservation*, 2(3), 2016.

[5] V. Sainte Fare Garnot, L. Landrieu, S. Giordano, and N. Chehata. Satellite image time series classification with pixel-set encoders and temporal self-attention. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2020.

[6] Chang Wei Tan, Geoffrey I. Webb, and François Petitjean. Indexing and classifying gigabytes of time series under time warping. In *Proceedings of the 17th SIAM International Conference on Data Mining, SDM 2017*, 2017.

[7] Bamber J., P. Bates, H. Brindley, B. Evans, Jackson T., Merchant C. Davey, Palmer M., P, and Spencer T Scott. Space-based earth observations for climate security. *COP26 Universities Network Briefing*, 2021.

[8] Steven W. Running. Ecosystem disturbance, carbon, and climate. *Science*, 321(5889):652–653, 2008.

[9] Liheng Zhong, Lina Hu, and Hang Zhou. Deep learning based multi-temporal crop classification. *Remote Sensing of Environment*, 221, 2019.

[10] Jean Daniel Sylvain, Guillaume Drolet, and Nicolas Brown. Mapping dead forest cover using a deep convolutional neural network and digital aerial photography. *ISPRS Journal of Photogrammetry and Remote Sensing*, 156, 2019.

[11] T Trevor Caughlin, Cristina Barber, Gregory P Asner, Nancy F Glenn, Stephanie A Bohlman, and Chris H Wilson. Monitoring tropical forest succession at landscape scales despite uncertainty in landsat time series. *Ecological Applications*, 31(1):e02208, 2021.

[12] Sébastien Rapinel, Cendrine Mony, Lucie Lecoq, Bernard Clément, Alban Thomas, and Laurence Hubert-Moy. Evaluation of Sentinel-2 time-series for mapping floodplain grassland plant communities. *Remote Sensing of Environment*, 223, 2019.

[13] Tao Guo. Satellite image time series simulation for environmental monitoring. In *Multispectral, Hyperspectral, and Ultraspectral Remote Sensing Technology, Techniques and Applications V*, volume 9263, 2014.

[14] François Petitjean, Jordi Inglada, and Pierre Gancarski. Satellite image time series analysis under time warping. *IEEE Transactions on Geoscience and Remote Sensing*, 50(8):3081–3095, 2012.

[15] Cristina Gómez, Joanne C. White, and Michael A. Wulder. Optical remotely sensed time series data for land cover classification: A review, 2016.

[16] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4), 2019.

[17] D. Sushma Reddy and P. Rama Chandra Prasad. Prediction of vegetation dynamics using NDVI time series data and LSTM. *Modeling Earth Systems and Environment*, 4(1), 2018.

[18] Charlotte Pelletier, Silvia Valero, Jordi Inglada, Nicolas Champion, and Gérard Dedieu. Assessing the robustness of Random Forests to map land cover with high resolution satellite image time series over large areas. *Remote Sensing of Environment*, 187, 2016.

[19] Mailys Lopes, Pierre Louis Frison, Sarah M. Durant, Henrike Schulte to Bühne, Audrey Ipavec, Vincent Lapeyre, and Nathalie Pettorelli. Combining optical and radar satellite image time series to map natural vegetation: savannas as an example. *Remote Sensing in Ecology and Conservation*, 6(3), 2020.

[20] Marc Rußwurm and Marco Körner. Self-attention for raw optical Satellite Time Series Classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 169, 2020.

[21] Rose M Rustowicz. Crop classification with multi-temporal satellite imagery. *Final Reports. Univ. Stanford*, 2017.

[22] Reza Khatami, Giorgos Mountrakis, and Stephen V Stehman. A meta-analysis of remote sensing research on supervised pixel-based land-cover image classification processes: General guidelines for practitioners and future research. *Remote Sensing of Environment*, 177:89–100, 2016.

[23] Johan Debayle, Nima Hatami, and Yann Gavet. Classification of time-series images using deep convolutional neural networks. 2018.

[24] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

[25] Dino Ienco, Roberto Interdonato, Raffaele Gaetano, and Dinh Ho Tong Minh. Combining Sentinel-1 and Sentinel-2 Satellite Image Time Series for land cover mapping via a multi-source deep learning architecture. *ISPRS Journal of Photogrammetry and Remote Sensing*, 158, 2019.

[26] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-Scale Convolutional Neural Networks for Time Series Classification. *Multi-scale Convolutional Neural Networks for Time Series Classification*, 2016.

[27] Marc Rußwurm and Marco Körner. Multi-temporal land cover classification with sequential recurrent encoders. *ISPRS International Journal of Geo-Information*, 7(4):129, 2018.

[28] Vivien Sainte Fare Garnot and Loic Landrieu. Lightweight temporal self-attention for classifying satellite images time series. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12588 LNAI, 2020.

[29] Lichao Mou, Lorenzo Bruzzone, and Xiao Xiang Zhu. Learning spectral-spatial-temporal features via a recurrent convolutional neural network for change detection in multispectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 57(2):924–935, 2018.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[31] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1), 2017.

[32] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre Alain Muller, and François Petitjean. InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, 34(6), 2020.

[33] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

[34] Nataliia Kussul, Mykola Lavreniuk, Sergii Skakun, and Andrii Shelestov. Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geoscience and Remote Sensing Letters*, 14(5):778–782, 2017.

[35] Zahraa S. Abdallah and Mohamed Medhat Gaber. Co-eye: a multi-resolution ensemble classifier for symbolically approximated time series. *Machine Learning*, 109(11), 2020.

[36] Colin Lea, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks: A unified approach to action segmentation. In *European Conference on Computer Vision*, pages 47–54. Springer, 2016.

[37] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017.

[38] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2017-May, 2017.

[39] Shunping Ji, Chi Zhang, Anjian Xu, Yun Shi, and Yulin Duan. 3D convolutional neural networks for crop classification with multi-temporal remote sensing images. *Remote Sensing*, 10(1), 2018.

[40] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[41] Misganu Debella-Gilo and Arnt Kristian Gjertsen. Mapping seasonal agricultural land use types using deep learning on sentinel-2 image time series. *Remote Sensing*, 13(2):289, 2021.

[42] Shuchang Zhou, Jia-Nan Wu, Yuxin Wu, and X. Zhou. Exploiting local structures with the kronecker layer in convolutional networks. *ArXiv*, abs/1512.09194, 2015.

[43] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[44] Ceshine Lee. [tensorflow] implementing temporal convolutional networks, 2019.

[45] Marc Rußwurm, Charlotte Pelletier, Maximilian Zollner, Sébastien Lefèvre, and Marco Körner. Breizhcrops: A time series dataset for crop type mapping. *arXiv preprint arXiv:1905.11893*, 2019.

[46] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International conference on web-age information management*, pages 298–310. Springer, 2014.

[48] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Frontiers of Computer Science*, 10(1):96–112, 2016.

[49] Thomas Di Martino. Time series land cover challenge: a deep learning perspective, 2020.

[50] Jason Lines, Sarah Taylor, and Anthony Bagnall. Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1041–1046. IEEE, 2016.

[51] Dino Ienco. Tiselac : Time series land cover classification challenge, 2017.

[52] Guy S Handelman, Hong Kuan Kok, Ronil V Chandra, Amir H Razavi, Shiwei Huang, Mark Brooks, Michael J Lee, and Hamed Asadi. Peering into the black box of artificial intelligence: evaluation metrics of machine learning methods. *American Journal of Roentgenology*, 212(1):38–43, 2019.

[53] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[54] Zhilu Zhang and Mert R Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[56] Nilanjan Dey. *Intelligent speech signal processing*. Academic Press, 2019.

[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[58] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

A Supplementary details on deep learning principles

The purpose of this appendix is to provide additional details to principals relating to deep learning.

Deep learning is a subset of machine learning, in which any neural network with three or more layers is considered a deep neural network (DNN) [56]. Several computational layers are concatenated together, with each layer taking inputs from the previous layer. This is known as a *feed-forward* network. These neural networks attempt to simulate the human brain by "learning" to perform a pre-defined task from a large amount of data by learning representations of the data at various levels of abstraction [56, 57].

In Figure 10 , on the left is an example DNN whereby the neurons in blue represent the inputs, the neurons in green represent neurons in the hidden layers and the purple neurons are the outputs. Each layer is comprised of a certain number of units, or namely neurons [57]. The number of input neurons is dependent on the dimension of the instances in the data, whereas the number of neurons in the output layer is comprised of C neurons for a classification task of C classes [3]. In a regression problem, only a single output neuron is needed. The number of hidden layers used and the number of neurons in each are to be selected by the practitioner and is heavily task dependent.

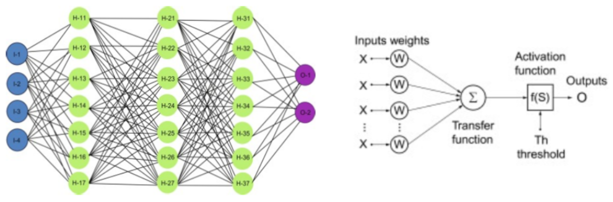


Figure 10: (L) feed forward DNN with three hidden layers. (R) a typical artificial neuron. Image sourced from [56].

Looking at the image in the right of Figure 10 is an example of a typical artificial neuron that are at the core of the neural networks learning process. The outputs of a layer l and the activation map $A^{[l]}$ are obtained via a two-step calculation. First, the outputs of the neurons from the previous layer are received as a linear combination. This linear combination of inputs and respective weights are then passed to a non-linear activation function such as tanh, ReLU [30]. This can be more formally written as [3]:

$$A^{[l]} = g^{[l]}(W^{[l]}A^{[l-1]} + b^{[1]}), \tag{2}$$

where $W^{[l]}$ and $b^{[l]}$ are the weights and biases of the layer l respectively. Both the weights and biases are learnt by the model during training.

The activation function as represented by $g^{[l]}$ introduces non-linear combinations of features within the network. If only linear functions are used, then the final output would simply be a linear combination of the inputs. This could be achieved with just a single layer, making the depth of the network largely irrelevant [3]. Correspondingly, the activation function used within this study is the Rectified Linear Units (ReLU) function that has seen popular use within deep learning applications [30, 3]. It is calculated as: ReLU(z) = max(0, z), which output the input value z if it positive, else it outputs 0.

By stacking several specialised hidden layers together in a neural network, the ability of the network to represent complex functions increases. This is enabled by the use the non-linear activations that are used to form a pattern of active hidden units, subsequently allowing the layers to contain a relatively small number of units, reducing the size of the network overall [3, 56]. Section 5.3 explores the experiments conducted surrounding the optimal model width and depth.

The act of training a neural network relates to minimising a given cost function by finding values of $W = \{W^{[l]}\}_{\forall l}$ and $b = \{b^{[l]}\}_{\forall l}$ [3]. The cost function is used to gauge the fit of the model to the data provided. This process is known as empirical risk minimisation and it is done during when errors are back propagated through the network whilst training occurs, updating the values for the weights of each input such that appropriate values will activate the neuron. The set of weights and biases which are used to uniquely identify a particular output are called the feature kernel and they are randomly initialised at model creation [56]. The cost function $\mathcal{L}$ used is generally defined as the average of the errors made on each individual training instance [3]:

$$\mathcal{J}(\mathbf{W}, \mathbf{b}) = \frac{1}{n} \sum_{\hat{x_i}} \mathcal{L}(\hat{y_i}, y_i). \tag{3}$$

where $\hat{y_i}$ corresponds to the predictions made by the network.

In multi-class classification problems, the loss function $\mathcal{L}(\hat{y_i}, y_i)$ generally used is the categorical crossentropy loss function [54] and is defined as follows [3]:

$$\mathcal{L}(\hat{y_i}, y_i) = - \sum_{y \in \{1,\ldots,C\}} 1\{y = y_i\} log(p(y|x_i) = -log(p(y_i|x_i)) \tag{4}$$

where $p(y_i|x_i)$ is a representation of the probability that the model predicts the actual class y_i of instance i in the last layer of the network. For networks using categorical crossentropy as the loss function, the last layer will be a Softmax layer that contains scores for each possible output class and returns the class with the highest score [3].

Now that the core components of a deep neural network have been introduced, the benefits of neural networks can be fully understood. However, the designing of effective networks requires considerable expertise when deciding factors such as the architecture configuration, related hyper-parameter values and optimisation techniques [3]. With appropriate decisions made for each of these, the reward is a model that can learn effective features from the data rather than requiring manual feature engineering [30]. Developing models with such large numbers of parameters however creates issues with over-fitting, the phenomena where the model learns the noise within the training data and subsequently fails to generalise to unseen testing data [58]. Measures to mitigate this issue were experimented with extensively within this study.

B Additional results from Temporal CNN experiments

This appendix is concerned with documenting any extra results of interest generated during the researching of the optimal Temporal CNN architecture.

Results for finding the optimal width and depth on the TiSeLaC dataset						
NB_CONV_LAYERS	NB_CONV_UNITS	NB_FC_UNITS	BATCH_SIZE	DROPOUT	FILTER_SIZE	OA
2	64	128	128	0.2	5	91.64
2	64	256	128	0.2	5	92.62
2	64	512	128	0.2	5	93.16
2	128	256	128	0.2	5	93.28
2	256	256	128	0.2	5	94.13
3	64	64	128	0.2	5	90.9
3	64	128	128	0.2	5	92.64
3	64	256	128	0.2	5	92.81
3	64	512	128	0.2	5	91.33
3	128	128	128	0.2	5	93.57
3	128	256	128	0.182	5	93.37
3	128	256	128	0.2	5	95.02
3	128	256	128	0.2	5	94.24
3	128	256	64	0.182	5	94.05
3	128	512	128	0.25	5	94.02
3	256	256	128	0.182	5	94.22
3	256	256	128	0.2	5	94.58
3	512	256	128	0.2	5	93.48
3	512	512	128	0.2	5	93.36
3	512	1024	128	0.2	5	93.02
3	512	2048	128	0.2	5	93.54
3	1024	256	128	0.2	5	93.83
4	64	256	128	0.2	5	92.8
4	64	512	128	0.2	5	92.58
4	128	256	128	0.2	5	93.02
4	128	512	128	0.2	5	94
4	256	256	128	0.2	5	94.44
4	256	256	128	0.182	5	93.37
5	64	256	128	0.2	5	91.62
5	128	256	128	0.2	5	91.94
5	128	256	128	0.3	5	92.91
5	256	256	128	0.2	5	92.94

Table 9: Full list of experiments ran when finding the optimal architecture of the Temporal CNN for the TiSeLaC dataset.

Filter size experiments on TiSeLaC dataset						
NB_CONV_LAYERS	NB_CONV_UNITS	NB_FC_UNITS	BATCH_SIZE	DROPOUT	FILTER_SIZE	OA
3	128	256	128	0.2	3	93.94
3	128	256	64	0.3	3	94.72
3	128	256	128	0.182	5	94.37
3	128	256	128	0.2	5	95.02
3	128	256	128	0.2	7	93.93
3	128	512	128	0.182	7	93.39

Table 10: Full list of experiments ran when finding the optimal filter size of the Temporal CNN for the TiSeLaC dataset.

Results for finding the optimal width and depth on one fold of the SITS-TSI dataset						
NB_CONV_LAYERS	NB_CONV_UNITS	NB_FC_UNITS	BATCH_SIZE	DROPOUT	FILTER_SIZE	OA
2	128	256	256	0.2	5	87.05
3	64	256	256	0.15	5	87.08
3	64	256	256	0.2	5	86.95
3	64	256	256	0.5	5	84.03
3	128	256	128	0.25	5	86.84
3	128	256	128	0.2	5	87.16
3	128	256	128	0.25	5	86.88
3	128	256	128	0.3	5	86.88
3	128	512	128	0.3	5	87.02
3	256	128	128	0.2	5	86.87
3	256	256	128	0.2	5	87.13
3	256	256	128	0.25	5	86.94
3	256	256	128	0.3	5	86.8
4	128	256	128	0.2	5	87.11
4	256	256	128	0.35	5	86.92
5	128	256	128	0.2	5	87.1

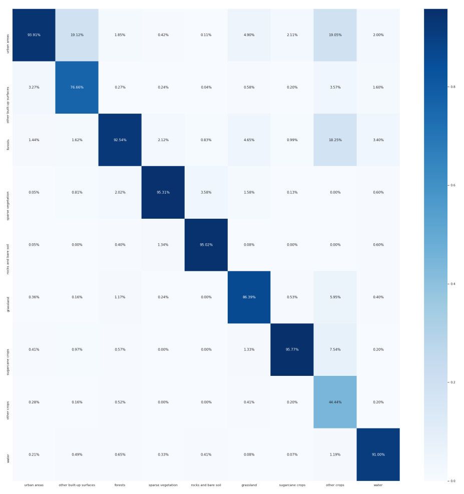
Table 11: Full list of experiments ran when finding the optimal architecture of the Temporal CNN for the SITS-TSI dataset.

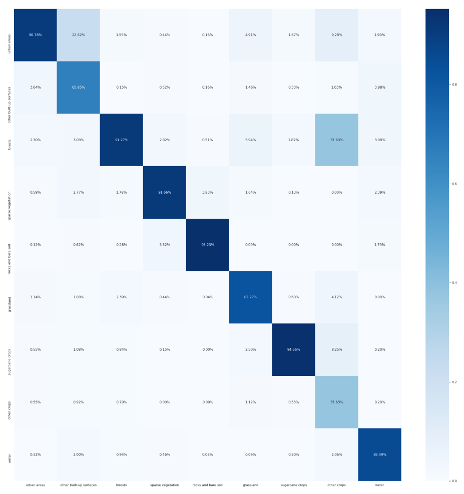
Accuracies found whilst training on all 10 folds of SITS-TSI						
NB_CONV_LAYERS	NB_CONV_UNITS	NB_FC_UNITS	BATCH_SIZE	DROPOUT	FILTER_SIZE	OA
3	64	256	256	0.2	5	90.15
3	64	256	256	0.2	5	89.42
3	256	256	256	0.2	7	96.58

Table 12: Full list of results obtained when training on all ten folds of the SITS-TSI dataset.

	Method - SITS-TSI dataset				
	Temporal CNN (1 fold) | (10 folds)	Time-CNN (1 fold) | (10 folds)	MCDCNN (1 fold) | (10 folds)	InceptionTime (1 fold) | (10 folds)	RNN (1 fold) | (10 folds)
OA	**87.2** | 96.6	86.5 | 93.2	85.8 | 88.3	86.4 | **98.2**	85.5 | 87.4
F1	**87.2** | 96.7	86.6 | 93.2	85.8 | 88.3	86.5 | **98.1**	85.5 | 87.4

Table 13: All results of training the models on the first cross-validation fold versus all ten. It can be observed that each method gains accuracy when trained on all ten folds instead of just one, demonstrating that when more data is provided each model can take advantage of it. Of these methods, the Temporal CNN, Time-CNN and InceptionTime networks were most able to utilise the extra training data.





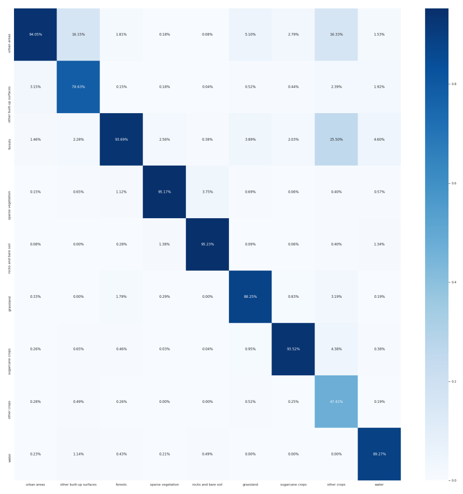
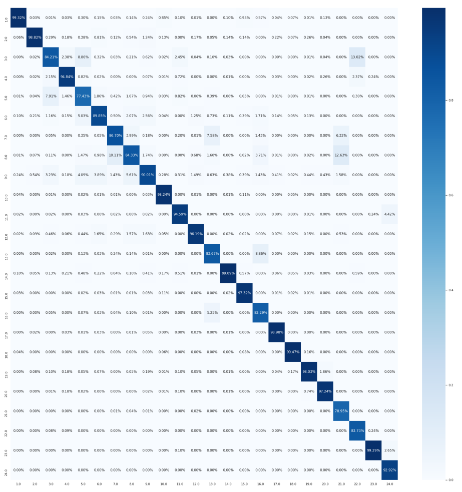
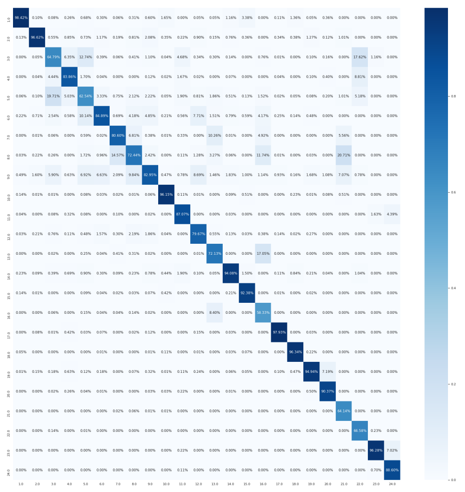


Figure 11: Confusion Matrix on TiSeLaC dataset using a) InceptionTime network b) Time CNN c) RNN





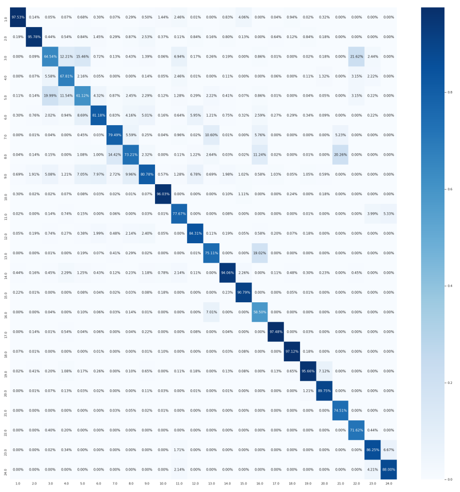


Figure 12: Confusion Matrix on SITS-TSI dataset using a) Time CNN b) MCDCNN c) RNN