THESIS

INTERPOLATING RGB RADAR IMAGES BASED ON MACHINE LEARNING

Submitted by

Chenke Yi

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2023

Master's Committee:

Advisor: V. Chandrasekar
Co-Advisor: Haonan Chen

Thomas Siller
Steven Gooch

ABSTRACT

INTERPOLATING RGB RADAR IMAGES BASED ON MACHINE LEARNING

Weather radar interpolation is the process of estimating and predicting rainfall data in areas that are not directly observed by radar. This technique is commonly used in weather forecasting, flood prediction, and agricultural planning. The main goal of weather radar interpolation is to produce accurate and reliable precipitation maps in areas with limited radar coverage or where the radar data is incomplete. The interpolation methods can be categorized into two main groups: deterministic and stochastic. Deterministic methods use mathematical equations and physical models to estimate the rainfall, while stochastic methods rely on statistical algorithms to analyze the correlations between the radar measurements and ground observations. In recent years, machine learning algorithms have also been applied to weather radar interpolation, showing promising results in accuracy and robustness. In this paper, we mainly propose a radar image interpolation method based on spatio-temporal convolutional networks. The experiments are mainly compared and analyzed for different combinations of networks, connection methods, and different loss functions.

ACKNOWLEDGEMENTS

I would like to thank Prof. V. Chandrasekar and Prof. Haonan Chen and all the committee who helped me to make this happen.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Weather radar interpolation is a vital technology used in the field of meteorology. The interpolation method fills missing data in radar images obtained from weather radar systems. These systems provide valuable information about precipitation and other weather-related phenomena, but data can be compromised due to obstacles such as buildings or terrain. Interpolation techniques help to produce more accurate and complete radar images, improving the reliability of weather forecasting and prediction. The use of weather radar interpolation has become increasingly important in recent years due to the ongoing climate change and the frequent extreme weather events. In this article, we will delve into the various methods of weather radar interpolation, its applications and the challenges associated with it.

Machine learning has revolutionized the field of weather radar interpolation in recent years. By leveraging powerful algorithms and data analysis techniques, machine learning can effectively fill the missing data in radar images obtained from weather radar systems. Machine learning models are trained on large historical radar data and use this information to accurately predict missing data points in real-time. The application of machine learning to weather radar interpolation has led to increased accuracy and reliability of weather forecasts and predictions. With the ability to accurately predict precipitation patterns, severe weather events can be detected earlier and provide crucial time for people to prepare and take preventative measures.

1.1 Problem statement

The trajectories of clouds are continuous in the actual motion on the time axis. The images scanned by the weather radar correspond to a discrete sampling of the actual continuous motion due to the scan interval. The trajectories of their future images are dependent on the present and the past trajectories. Therefore, this could be described as a sequence of images with spatio-temporal properties.

This problem could be described specifically as the input image sequence X to the machine learning model M to obtain the output image sequence Y. Y has K more image frames than X.

$$X = \langle x_1, x_2, ..., x_T \rangle, Y = \langle y_1, y_2, ..., y_{T+K} \rangle$$

$$M(X) \simeq Y$$

The input is a sequence of images with large time intervals and the output is a sequence of images with small time intervals. A new image sequence with small time intervals is generated by the combination of the input and output image sequences.

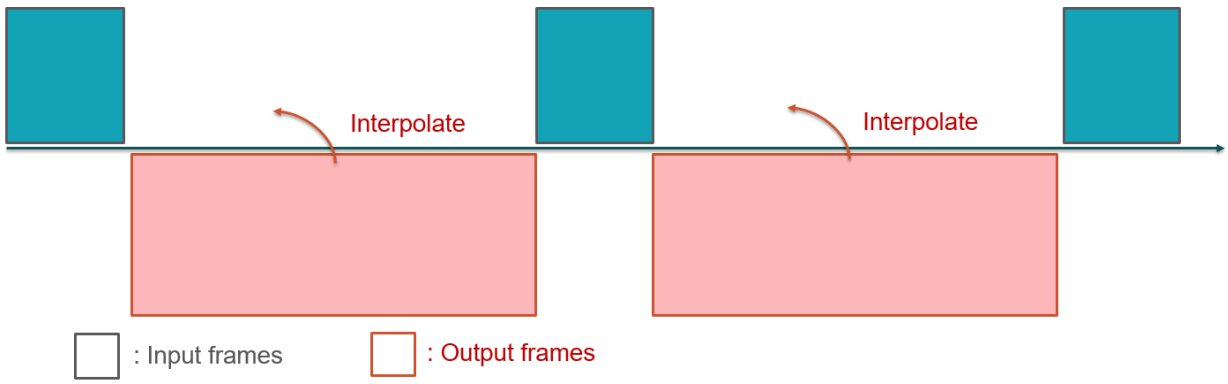


Figure 1.1: Problem statement.

1.2 Overview

This article is constructed as follows, first introducing weather radar and the background knowledge of machine learning to understand this project. Then the baseline of this paper is introduced, summarizing the improvements and effects of the previous models. After this, the experimental approach and results in this project are presented. Finally, the analysis and discussion of the results are presented.

Chapter 2

Background information

2.1 Development of weather radar

Development




Radar	WSR-57 radars	WSR-74 radars	WSR-88D	CSU-CHILL Radar
Features	• The first generation of radars • Only coarse reflectivity data	• Adding existing network to WSR-57 • S band, C band	• Doppler radar • Higher power radar beam • Be able to detect the "shift in the phase" of the pulse of energy	• Dual polarization radar • Improved accuracy in detecting and predicting (short-term) different types of frozen precipitation. • Handle with high-bandwidth radar data and accommodate its operations in real time • Virtual CHILL (VCHILL) system offers live remote control and radar plot for education
Years	1957	1974	1988	2006

Figure 2.1: Development of weather radar.

Weather radar is a type of radar system used to detect and measure precipitation in the atmospher. It was first developed in the 1940s and 1950s for military applications. Since then, it has become an essential tool for meteorologists in monitoring and predicting weather patterns. The primary principle of weather radar is to transmit a radar signal into the atmosphere, which interacts with precipitation and is reflected back to the radar receiver. By analyzing the properties of the reflected signal, meteorologists can determine the location, intensity, and movement of precipitation. The basic equation that underpins the operation of weather radar is the radar range equation, which relates the power of the transmitted signal to the power of the received signal after it has been reflected by the precipitation [1].The radar range equation is given by:

$$P_r = \frac{P_t G^2 \lambda^2 \sigma}{(4\pi)^3 R^4} \tag{2.1}$$

where P_r is the power of the received signal, P_t is the power of the transmitted signal, G is the gain of the transmitter antenna, λ is the wavelength of the signal, σ is the radar cross-section of the precipitation, and R is the distance from the radar to the precipitation.

Doppler radar is an advanced form of weather radar that can detect not only the location and intensity of precipitation but also the motion and direction of the precipitation. Doppler radar works on the principle of the Doppler effect, which causes a shift in the frequency of the radar signal when it is reflected by a moving object. The shift in frequency can be used to determine the velocity of the precipitation relative to the radar [2]. The Doppler shift equation is given by:

$$\Delta f = \frac{2 v_r}{c} f_0 \tag{2.2}$$

where Δf is the Doppler shift in frequency, v_r is the radial velocity of the precipitation relative to the radar, c is the speed of light, and f_0 is the frequency of the transmitted signal.

Weather radar technology has continued to evolve, with new systems and technologies being developed to improve the accuracy and range of radar data. Dual-polarization radar, for example, uses both vertical and horizontal polarizations of the radar signal to provide additional information about the size, shape, and orientation of the precipitation particles [3]. Other technologies, such as phased array radar, use multiple transmitting and receiving antennas to provide more detailed and accurate picture of the atmosphere [4]. The following table summarizes the main parameters of the weather radar:

Table 2.1: Summary of weather radar main parameters.

Name	Definition	Unit
Reflectivity	Measurement of the strength and intensity of radar return signals from precipitation	dBZ
Velocity	Measurement of the speed and direction of motion of precipitation particles towards or away from the radar	m/s
Differential Reflectivity	Measurement of the differences in reflective properties between horizontally and vertically oriented precipitation particles	dB
Correlation Coefficient	Measurement of the degree of similarity between horizontally and vertically oriented precipitation particles	unitless
Spectral Width	Measurement of the range of velocities in the radar signal caused by variations in particle sizes and shapes	m/s

2.2 Introduction of traditional radar interpolation algorithm

Weather radar is an important tool for measuring precipitation over a large area. However, radar data is typically only available at discrete locations, which can make it difficult to interpolate the data to locations where no radar measurement is available. To address this challenge, various methods have been developed for radar interpolation. Traditional radar interpolation methods include kriging and nearest neighbor interpolation.

2.2.1 Kriging interpolation

Kriging interpolation is a geostatistical method used to estimate the value of a variable at an unobserved location based on the values of the variable at nearby observed locations. The method involves calculating a set of weights that are used to combine the values of the variable at the observed locations to produce an estimate of the value at the unobserved location. The weights are chosen such that the interpolated value at the unobserved location is a linear combination of the observations. This means that the estimated value is a weighted average of the values at the observed locations, with the weights determined by the spatial relationship between the observed locations and the unobserved location.

The process of choosing the weights involves calculating a set of spatial autocorrelation functions that describe the spatial relationship between the observed locations and the unobserved location. These functions are used to calculate a set of weights that minimize the variance of the estimation error, subject to the constraint that the estimated value is a linear combination of the observations.

The weights are typically calculated using a technique called ordinary kriging, which involves solving a system of linear equations to determine the optimal weights. The resulting weights are then used to calculate the estimated value at the unobserved location. The basic idea of kriging is to assign larger weights to the observations that are spatially closer and more similar to the unobserved location, as determined by the spatial correlation function. This is expressed by the following equation:

$$\hat{Z}_0(\mathbf{s}_0) = \sum_{i=1}^{n} \lambda_i Z_0(\mathbf{s}_i), \tag{2.3}$$

where $\hat{Z}_0(\mathbf{s}_0)$ is the estimated radar reflectivity factor at the unobserved location $\mathbf{s}_0$, $Z_0(\mathbf{s}_i)$ are the observed radar reflectivity factors at the observed locations $\mathbf{s}_1, \ldots, \mathbf{s}_n$, and λ_i are the weights assigned to the observations. The weights are typically chosen to minimize the prediction variance subject to unbiasedness and the assumption of a fixed mean [5].

2.2.2 Neighbor interpolation

Nearest neighbor interpolation is a simple method for interpolating discrete data, where no radar measurement is available. The basic idea of nearest neighbor interpolation is to assign the value of the nearest measured point to the unobserved location [6]. This method does not take into account any spatial correlation in the data, and it can result in sharp changes in the interpolated values over small distances. To understand the terminology used in nearest neighbor interpolation, we provide the following definitions:

- **Measured points (P_i):** The discrete locations where the data is available.
- **Unobserved location (P_0):** The location where the data needs to be interpolated.
- **Euclidean distance ($d_{i,0}$):** The distance between the measured point P_i and the unobserved location P_0, calculated using the following formula:

$$d_{i,0} = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2}, \tag{2.4}$$

  where x_i and y_i are the coordinates of the measured point P_i, and x_0 and y_0 are the coordinates of the unobserved location P_0.

- **Interpolated value (f_0):** The estimated value at the unobserved location P_0, which is equal to the value of the nearest measured point:

$$f_0 = f_i, \tag{2.5}$$

  where f_i is the value of the nearest measured point P_i to the unobserved location P_0.

Nearest neighbor interpolation is a simple and fast method that does not require any assumptions about the spatial correlation in the data. However, it can result in sharp changes in the interpolated values over small distances, which may not be accurate in some applications. Other interpolation methods, such as kriging, take into account the spatial correlation in the data and

can provide smoother interpolated values. In some cases, a combination of different interpolation methods may be used to optimize the accuracy of the interpolated data.

2.3 Development of machine learning

Machine learning (ML) is a rapidly growing field that aims to develop algorithms that can automatically learn patterns from data and make predictions or decisions based on those patterns. The development of ML has been driven by advances in computing power, data availability, and algorithmic innovation. The performance graph shows the major machine learning developments and achievements in roughly chronological order.

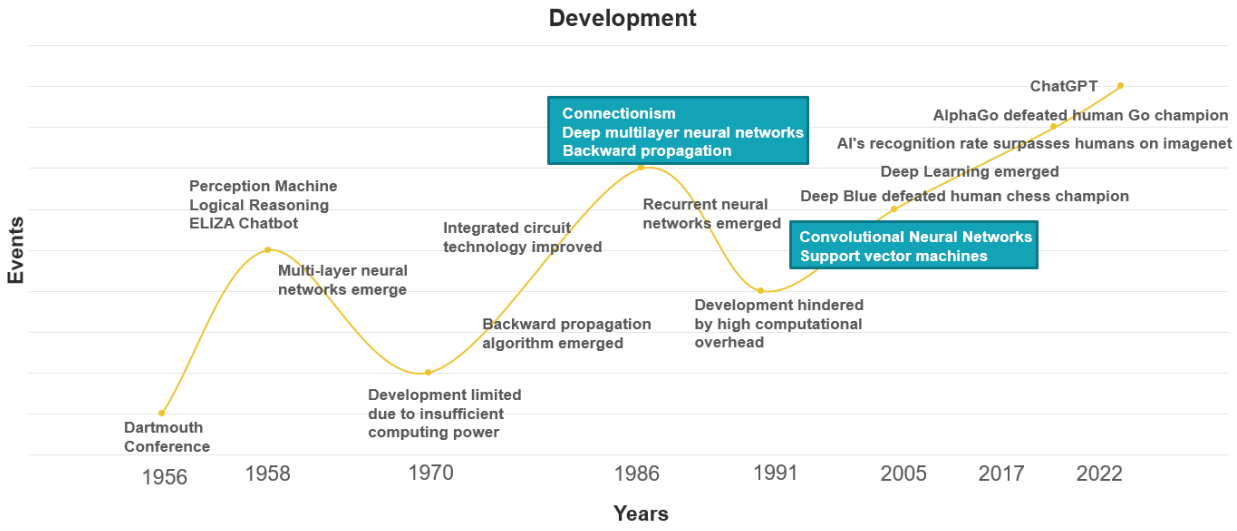


Figure 2.2: Development of machine learning.

Backpropagation algorithm and convolutional neural networks (CNNs) have played an essential role in the development of machine learning, especially in the field of deep learning.

The backpropagation algorithm is a widely used method for training deep neural networks, allowing them to learn from vast amounts of data. It involves the backward propagation of errors through the layers of the network, adjusting the weights of the neurons to minimize the loss function. The algorithm was first introduced in the early 1980s by Rumelhart, Hinton, and Williams. Since then, it has become an integral part of many deep learning architectures. The significance of backpropagation algorithm lies in its ability to enable deep neural networks to learn complex representations and generalize well to unseen examples.

Convolutional neural networks (CNNs) are a specific type of deep neural network designed for processing image data. They use a hierarchical structure with multiple convolution layers that extract features at different levels of abstraction from the input images. CNNs have been able to achieve remarkable performance on image recognition tasks, surpassing human-level performance in some cases. The significance of CNNs lies in their ability to automatically learn features from data, reducing the need for hand-crafted features, and making them applicable to a wide range of visual recognition tasks [7]. The following diagram illustrates the basic process of machine learning training.

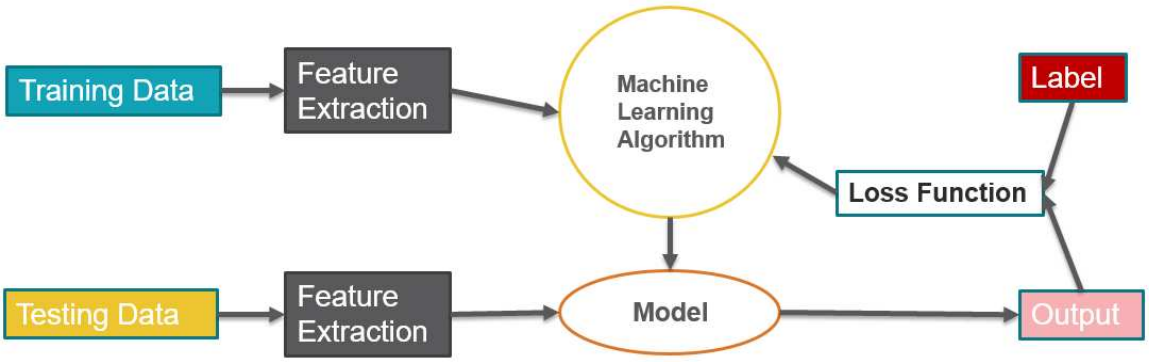


Figure 2.3: General training process of machine learning.

The steps in the machine learning process are interdependent and build upon each other. Here is an explanation of the relationship between each step:

- **Training data:** The training data is used to train the machine learning model. It consists of a set of input data and corresponding output labels. The quality and quantity of the training data can have a significant impact on the performance of the model.
- **Feature extraction:** Feature extraction is the process of selecting and transforming relevant features from the input data. The quality of the features extracted can have a significant impact on the performance of the model. The features extracted from the training data are used to train the model.

- **Testing data:** The testing data is used to evaluate the performance of the machine learning model. It consists of input data and corresponding output labels that the model has not seen before. The testing data is used to measure the accuracy of the model and to identify any areas where the model may need improvement.

- **Model:** The machine learning model is a mathematical representation of the relationship between the input data and output labels. The model is trained using the training data and the features extracted from the input data.

- **Machine learning algorithm:** The machine learning algorithm is used to train the machine learning model. The algorithm is responsible for adjusting the parameters of the model to minimize the loss function.

- **Loss function:** The loss function is a mathematical function that is used to measure the difference between the predicted output of the model and the actual output. The goal of the machine learning algorithm is to minimize the loss function by adjusting the parameters of the model.

- **Label:** The label is the output that the machine learning model is trying to predict. The label is used to evaluate the performance of the model and to identify any areas where the model may need improvement.

- **Output:** The output is the predicted output of the machine learning model for a given input. The output is used to evaluate the performance of the model and to identify any areas where the model may need improvement.

Here are some commonly used ML algorithms and the types of problems they are typically used for:

- **Linear regression:** A supervised learning algorithm used for regression problems, where the goal is to predict a continuous output variable. The algorithm learns a linear function

that maps the input variables to the output variable. The model takes the form:

$$y = \mathbf{w}^T \mathbf{x} + b,$$

where $\mathbf{w}$ is a weight vector and b is a bias term.

- **Logistic regression:** A supervised learning algorithm used for binary classification problems, where the goal is to predict a binary output variable (e.g., 0 or 1). The algorithm learns a logistic function that models the probability of the output variable being 1 given the input variables. The model takes the form:

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x} - b)},$$

where $\mathbf{w}$ is a weight vector and b is a bias term.

- **Decision trees:** A supervised learning algorithm used for both classification and regression problems. The algorithm learns a tree-like model of decisions and their possible consequences, based on the input features. The model takes the form of a tree with nodes representing decisions and branches representing the possible outcomes of those decisions.

- **Random forests:** A supervised learning algorithm that builds a large number of decision trees and combines their predictions to make a final prediction. Random forests can be used for both regression and classification problems.

- **Support vector machines (SVMs):** A supervised learning algorithm that separates the input data into different classes by finding a hyperplane that maximizes the margin between the classes. SVMs are commonly used for classification problems, and can also be used for regression problems.

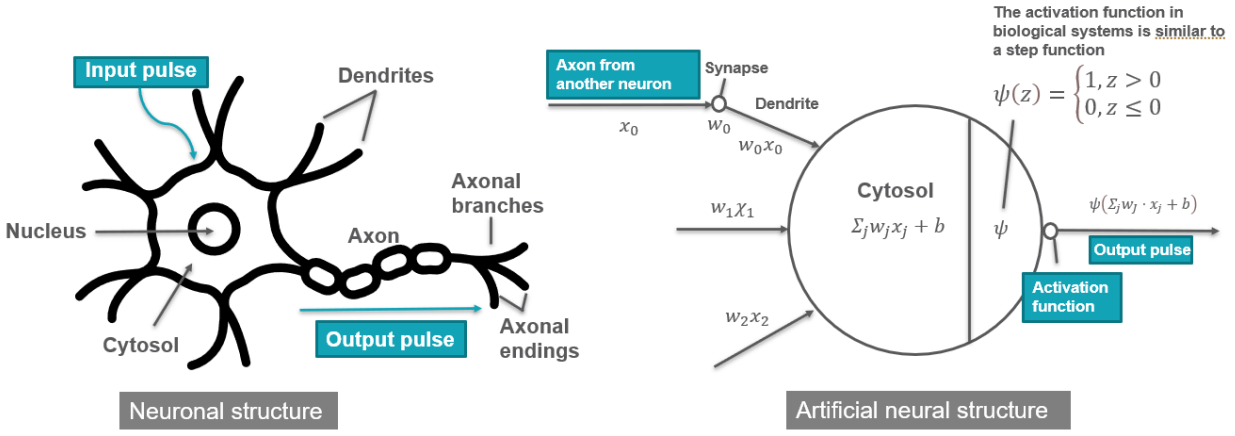
- **Neural networks:** A supervised learning algorithm that models the relationships between inputs and outputs using layers of nodes, or neurons, that are connected by weighted edges.

Neural networks can be used for classification and regression problems, and have shown state-of-the-art performance on many tasks.

- **Clustering algorithms:** Unsupervised learning algorithms that group similar data points together based on some criterion, such as distance or similarity of features. Clustering algorithms can be used to discover hidden structure in the data and to group similar data points together.

2.3.1 Neuronal structure and artificial neural structure

The structure of the human brain is an enigmatic and complex system that has been studied extensively over the years. Its numerous neurons, neural networks and the intricate connections between them provide the foundation for all our cognitive abilities. The study of the brain has inspired the development of artificial neural networks.



- The neuron performs a linear weighted summation$(\Sigma_j w_j x_j + b)$ of the input signals$\left(x = (x_j)\right)$ and then drives an activation function ψ to generate the output signal.

Figure 2.4: Neuronal structure and artificial neural structure.

The neuronal structure is made up of three main components: the cell body, dendrites, and axons. The cell body, also known as the soma, contains the nucleus and other essential cellular components. Dendrites are small, branched extensions that protrude from the cell body, providing input to the neuron. Axons are long extensions that transmit electrical signals away from the cell body to other neurons or muscles.

The artificial neural network model consists of layers of nodes, each performing a specific function in processing information. The input layer receives data, which is then passed through one or more hidden layers where it is transformed, and finally, the output layer provides the final result. Each node in the network is analogous to a neuron, processing the incoming information and transmitting it along to the next layer.

2.3.2 Fully connected neural network

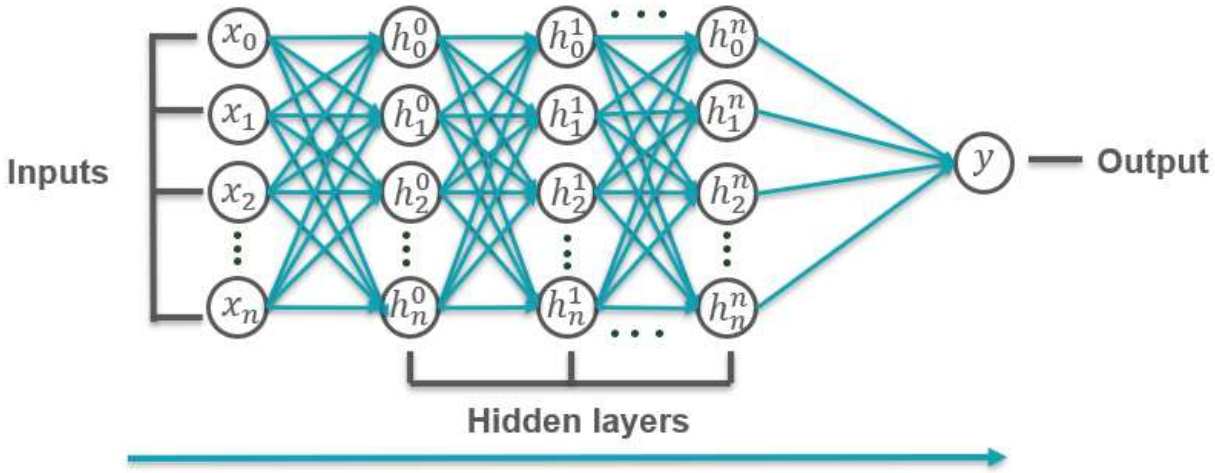


Figure 2.5: Fully connected neural network.

A fully connected neural network, also known as a dense neural network, is a type of artificial neural network where every neuron in one layer is connected to every neuron in the next layer. This means that all the neurons of each layer are fully connected to the neurons of the following layer, allowing the network to capture complex dependencies between the input and output.

The output of each neuron in a fully connected layer is calculated by taking a weighted sum of the inputs from the previous layer, followed by a nonlinear activation function. The weights and biases of the network are learned during training using backpropagation algorithm. the error is propagated backwards through the layers of the network, adjusting the weights to minimize the loss function.

The mathematical formulation of a single neuron in a fully connected layer is given by:

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$

where x_i denotes the i^{th} input to the neuron, w_i is the corresponding weight, b is the bias term, σ is the nonlinear activation function, and n is the number of inputs.

The output of the layer is computed by applying this formula to every neuron in the layer, resulting in a vector of outputs. This vector is then used as input to the next layer, and the process is repeated until the final output is obtained.

Fully connected neural networks have been used successfully in a variety of applications, such as image classification, speech recognition, and natural language processing. However, they have limitations in dealing with high-dimensional input data due to the large number of parameters required.

2.3.3 Convolutional neural network

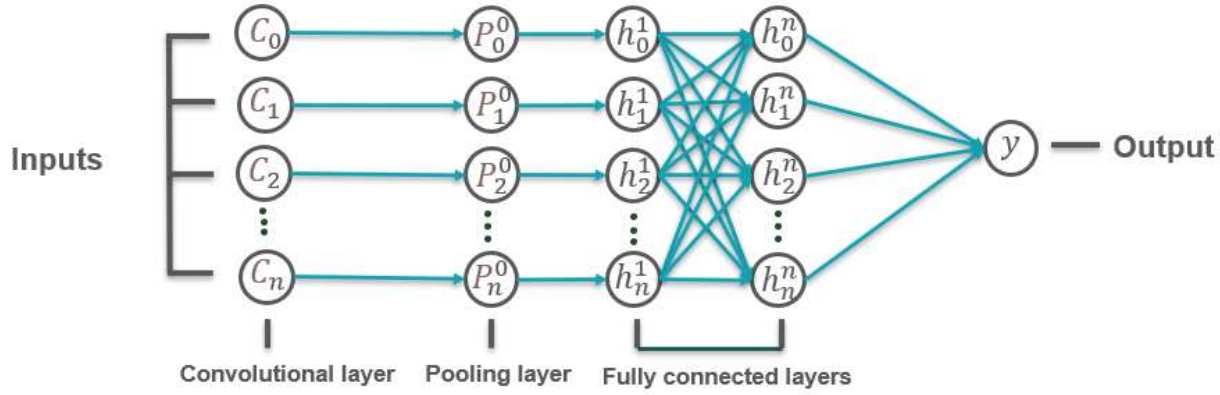


Figure 2.6: Convolutional Neural Network.

Convolutional neural networks are much less computationally intensive than fully connected neural networks but still have good performance [8].

Convolution is the process of applying a filter, or kernel, to an input image to produce a new feature map (output) by taking the dot product of the filter with small regions of the input image. By sliding the filter over the input image, the network is able to capture local patterns and features at different positions. Each convolutional layer in a CNN typically uses multiple kernels to produce several feature maps, which are then passed to the next layer.

Pooling is another important operation used in CNNs to reduce the dimensionality of the feature maps by downsampling and removing irrelevant information. Pooling is usually applied after the

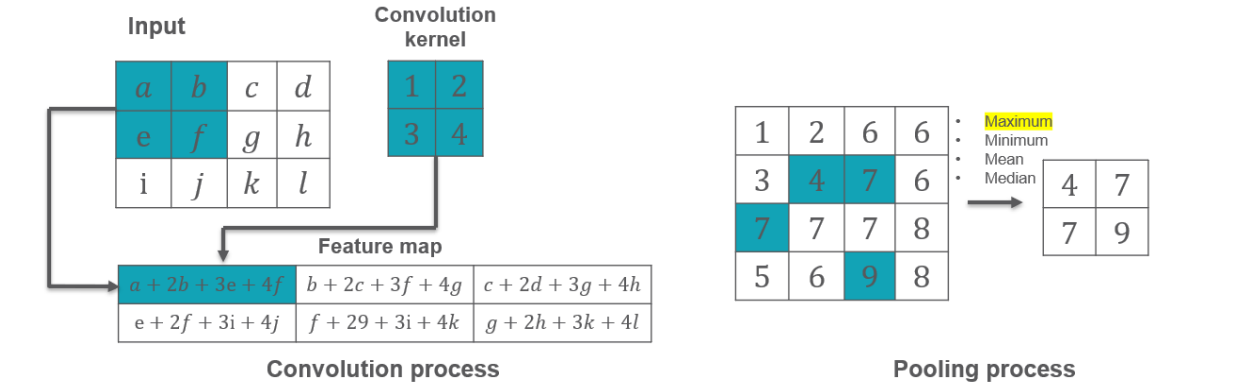


Figure 2.7: Convolution and pooling.

convolutional layer, and it replaces the output of the convolutional layer at a certain location with a summary statistic (e.g., maximum, average, or sum) of the output values within a small window (called pooling window) in that location. This operation reduces the spatial dimension of the feature maps while preserving the most important information.

One of the most common pooling operations is max pooling. It works by taking the maximum value within a pre-defined pooling window. The result is a reduced size map that captures the most significant features of the higher-resolution feature maps. Other common pooling operations include average pooling, which instead takes the mean value within the pooling window, and sum pooling, which sums the values within the pooling window.

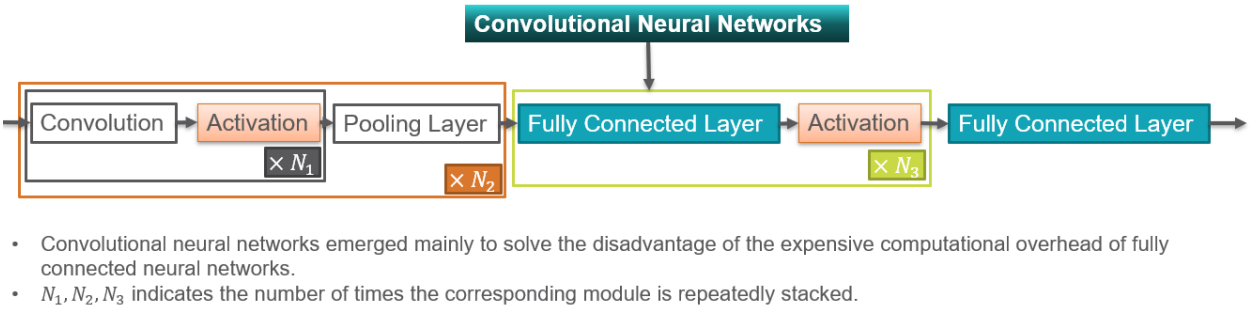


Figure 2.8: Convolutional neural network structure.

The combination of convolution and pooling operations allows CNNs to extract hierarchical representations of visual data, from low-level local features to high-level. CNNs are able to learn and identify patterns, shapes, and objects in images, making them highly effective in various computer vision tasks such as image classification, object detection, and segmentation.

2.3.4 Forward propagation and backward propagation

Forward propagation and backward propagation are two of the key algorithms used in machine learning for training neural networks. In forward propagation, an input is fed into the network, and the output is calculated by performing a series of matrix multiplications and activation functions. Backward propagation is then used to calculate the gradient of the loss function with respect to the network's weights, allowing them to be updated using an optimization algorithm.

Forward propagation

Forward propagation is the process of feeding an input through the neural network to generate an output. This process can be broken down into a series of matrix multiplications and activation functions, with each layer of the network transforming the input in a non-linear fashion. The output of the final layer is the prediction made by the network.

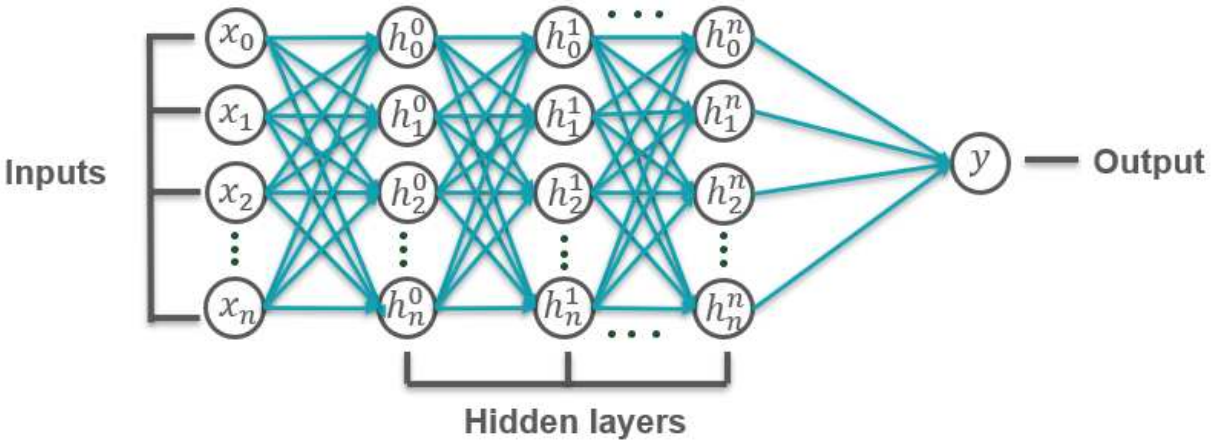


Figure 2.9: Forward propagation.

The outputs of each layer are given by the following equation:

$$a^{[l]} = g^{[l]}(z^{[l]}) = g^{[l]}(W^{[l]}a^{[l-1]} + b^{[l]})$$

where $a^{[l]}$ is the output of the l^{th} layer, $g^{[l]}$ is the activation function of the l^{th} layer, $z^{[l]}$ is the weighted input to the l^{th} layer, $W^{[l]}$ is the weight matrix for the l^{th} layer, $b^{[l]}$ is the bias vector for the l^{th} layer, and $a^{[l-1]}$ is the output of the $(l-1)^{th}$ layer.

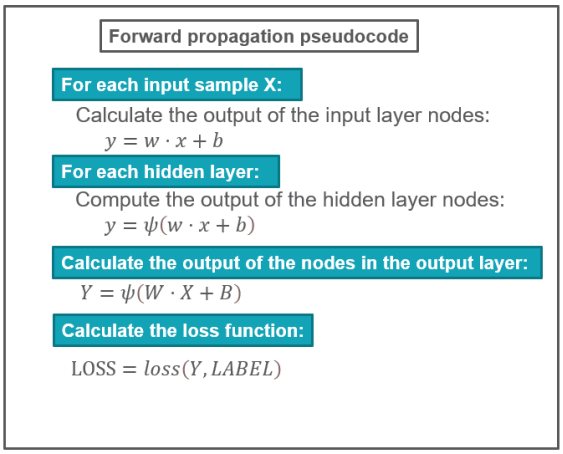


Figure 2.10: Forward propagation pseudocode.

The output of the final layer is then used to calculate the loss function, which measures the difference between the predicted output and the true output. This loss function is used to inform the network of how well it is performing, allowing backward propagation to adjust the network's weights and improve its predictions.

Backward propagation

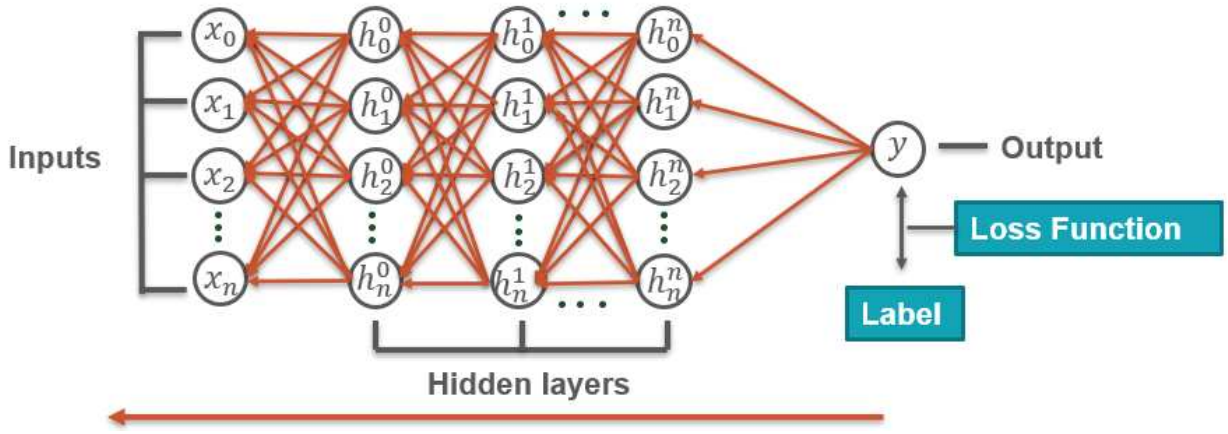


Figure 2.11: Backward propagation.

Backward propagation is used to calculate the gradient of the loss function with respect to the network's weights.

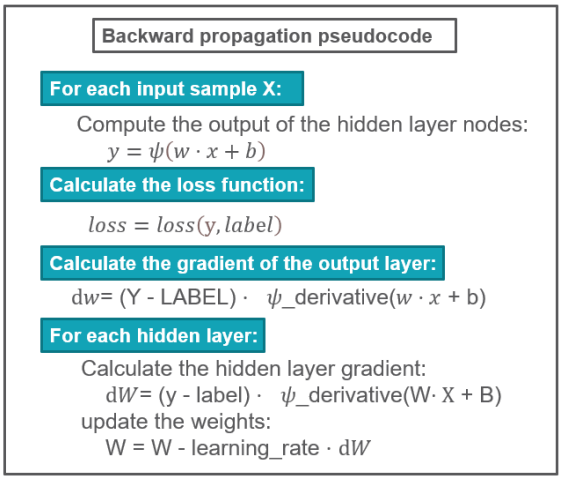


Figure 2.12: Backward propagation pseudocode.

The gradient of the loss function with respect to the output of the network is given by:

$$\frac{\partial L}{\partial a^{[L]}}$$

where L is the loss function and $a^{[L]}$ is the output of the final layer.

This gradient is then propagated backwards through the network using the chain rule, allowing the gradients of the earlier layers to be calculated as well. For each layer l, the gradient with respect to the input to that layer is given by:

$$\frac{\partial L}{\partial z^{[l]}}$$

where $z^{[l]}$ is the weighted input to layer l.

The gradient with respect to the weights and biases is then calculated based on the input gradient and the outputs of the previous layer:

$$\frac{\partial L}{\partial W^{[l]}} = \frac{\partial L}{\partial z^{[l]}} a^{[l-1]T}$$

$$\frac{\partial L}{\partial b^{[l]}} = \frac{\partial L}{\partial z^{[l]}}$$

These gradients can be used to update the network's weights and biases using an optimization algorithm such as stochastic gradient descent.

2.3.5 Gradient descent algorithm

Table 2.2: Summary of common gradient descent algorithms

Year	Reference	Algorithm
1951	[9]	Stochastic Gradient Descent
1964	[10]	Gradient Descent
1970	[11]	Least Squares Gradient Descent
1986	[12]	Resilient Propagation
1992	[13]	AdaMax
1996	[14]	Additive Groves
1998	[15]	Broyden-Fletcher-Goldfarb-Shanno
2003	[16]	Conjugate Gradient
2010	[17]	Heavy-Ball
2015	[18]	Adam

Gradient descent is an optimization algorithm that is commonly used in machine learning to minimize the cost function of a model. The algorithm works by iteratively adjusting the parameters of the model in the direction of the negative gradient of the cost function. The amount of adjustment made in each iteration is controlled by the learning rate, which is a hyperparameter that determines the step size of the algorithm. Gradient descent is an example of a first-order optimization algorithm since it uses only the first derivative (gradient) of the cost function to update the model parameters. However, gradient descent has some limitations, such as being prone to getting stuck in local minimum and requiring a fixed learning rate that may not be optimal for every iteration. These problems are addressed by variants of gradient descent, such as stochastic gradient descent, which uses a random sample of data points for each iteration, and Adam, which adapts the learning rate for each parameter based on historical gradients. One of the recent variants of gradient descent is the fGradient descent, which is proposed in [19]. The fGradient descent algorithm is designed to overcome the limitations of traditional gradient descent by introducing a new feedback mechanism based on the Hessian matrix of the cost function. The algorithm uses the Hessian matrix to estimate the curvature of the cost function and adjust the learning rate and step size accordingly. This allows the algorithm to converge faster and achieve better accuracy than traditional gradient descent. The gradient of a function $f(x)$ is defined as the vector of partial derivatives of the function with respect to each variable:

$$\nabla f(x) = \frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, ..., \frac{\partial f(x)}{\partial x_n}$$

The Hessian matrix of a function $f(x)$ is defined as the matrix of partial derivatives of the function with respect to each pair of variables:

$$\text{Hess}(f)(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) are two commonly used optimization algorithms in machine learning.

SGD is an iterative optimization algorithm that updates the weights of a model during training by computing the gradient of the loss function with respect to the weights and making small adjustments to the weights in the direction of the negative gradient. This process is repeated for number of epochs until the model converges on a set of weights that minimize the loss function.

Adam is an extension to SGD that adapts the learning rate for each weight parameter based on the first and second moments of the gradients. This adaptation allows for a more dynamic and efficient optimization process that can handle more complicated functions and avoid local optima.

The formulas for SGD and Adam are as follows:

SGD:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t) \tag{2.6}$$

Adam:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.7}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.8}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.9}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.10}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2.11}$$

where θ represents the weight parameters, η is the learning rate, J is the loss function, g is the gradient of J with respect to θ, β_1 and β_2 are exponential decay rates for the mean and variance estimations, and ϵ is a small value added for numerical stability.

2.4 Introduction to the sequence modeling in machine learning

Sequence modeling is a subfield of machine learning that focuses on building models capable of processing and analyzing sequences of data, such as time series, speech, text, and DNA sequences. Sequence models are especially useful in applications where the order of the data is crucial, as in the case of predicting future values of a time series or generating coherent sentences in natural language processing.

Two of the most common types of sequence models are recurrent neural networks (RNNs) and long short-term memory (LSTM) networks. RNNs are a type of neural network that are able to process sequences of varying lengths by passing information from one time step to the next through a hidden state. LSTM networks, on the other hand, are a specialized type of RNN that are able to capture long-term dependencies in the data through a gating mechanism that regulates the flow of information.

The core idea behind RNNs is to use the output of the previous time step as an additional input to the current time step, creating a loop that enables the network to maintain a memory of past inputs. Mathematically, RNNs can be expressed as follows:

$$h_t = f(x_t, h_{t-1})$$

where x_t is the input at time step t, h_{t-1} is the hidden state at time step $t-1$, and $f(.)$ is a non-linear activation function that produces the hidden state for the current time step h_t.

One of the main drawbacks of RNNs is that they can be computationally expensive and difficult to train on long sequences due to the vanishing gradient problem. This problem arises when the gradients used to update the network weights during backpropagation become very small.

Long short-term memory (LSTM) is a type of recurrent neural network (RNN) architecture that is designed to overcome the limitations of the traditional RNNs in handling long-term dependencies. LSTMs have been widely used in various applications such as language modeling, speech recognition, and time series prediction.

Another type of sequence model that has gained popularity in recent years is the transformer, which was introduced in the context of natural language processing. Transformers use a self-attention mechanism to compute the importance of each input element relative to every other element, allowing the network to attend to different parts of the input sequence when it produces outputs. Transformations have significantly improved the performance of natural language processing tasks such as machine translation and language modeling.

Chapter 3

Baselines

This section presents the simple and naive models that serve as a starting point for building model in this thesis. The purpose of creating a baseline model is to determine the performance limits of more complex models, which can be used to measure their effectiveness and identify areas of improvement.

3.1 Temporal Convolutional Networks

Temporal Convolutional Networks (TCNs) are a relatively new type of neural network architecture used for processing sequential data, which has shown increasing popularity in recent years as an alternative to traditional recurrent neural networks (RNNs). TCNs are able to process sequences of varying length without losing temporal information, while also benefiting from parallelization and avoiding some of the limitations of RNNs, such as vanishing or exploding gradients. TCNs were first introduced by Bai et al. in their 2018 paper "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling".

3.1.1 Causal convolutions

Causal convolutions are a type of convolution that only considers past and present inputs, and not future inputs. This is particularly useful for processing time-series data, where the output at time t should only depend on inputs up to time t, but not on future inputs.

Formally, the output y of a causal convolution performed on input x and filter w with dilation factor d is given by:

$$y_t = \sum_{j=0}^{k-1} w_j x_{t-jd} \tag{3.1}$$

where k is the filter size, d is the dilation factor, and t is the time step.

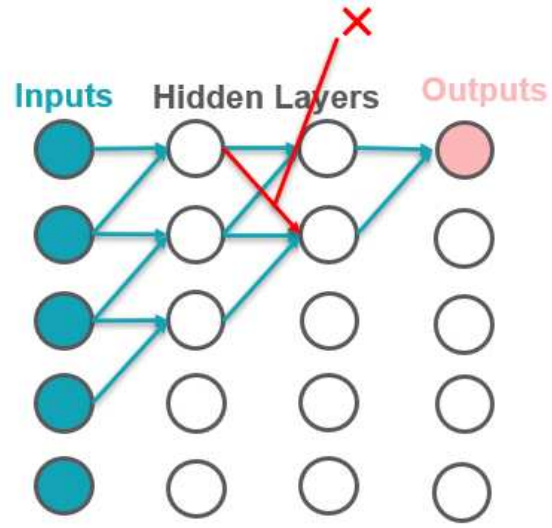


Figure 3.1: Causal convolutions.

3.1.2 Dilated convolutions

Dilated convolutions are a type of convolutional operation that allows for the receptive field of each filter to be increased exponentially without increasing the number of parameters in the network. This is accomplished by inserting zeros between the filter values, which effectively increases the stride of the convolution operation.

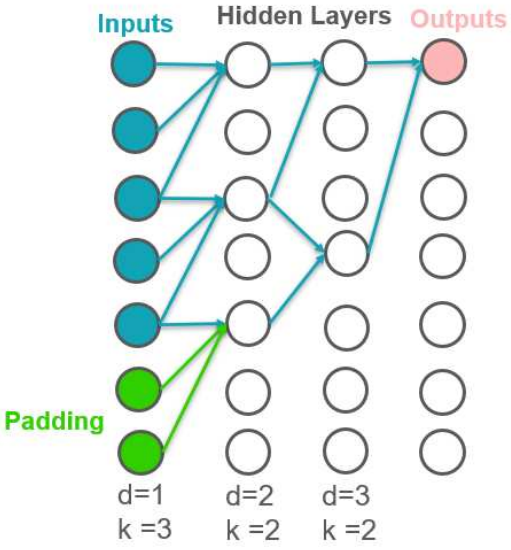


Figure 3.2: Dilated Convolutions.

Formally, the dilation factor d determines the spacing between the filter values. The output of a dilated convolution between input x and filter w with dilation factor d is given by:

$$(o_i)_t = \sum_{j=-k}^{k} w_{k+jd} x_{t+j} \tag{3.2}$$

where o_i is the ith output channel, k is the filter size, and t is the position of the current element in the input tensor.

Dilated convolutions are particularly useful for processing sequences with long-term dependencies, as they can capture information over a wider range of time steps than traditional convolutions.

3.1.3 Residual connections

Residual connections are a type of skip connection that allow for the gradient to bypass certain layers in the network, which helps to mitigate the problem of vanishing gradients in deep networks. Residual connections are typically added to the output of each layer, and provide an identity mapping that is added to the output of the layer.

Mathematically, a residual block can be represented as:

$$y = x + F(x, W) \tag{3.3}$$

where x is the input tensor, F(x, W) is the function computed by the layer, and W are the learnable weights. The output of the layer, y, is the sum of the input and output, which allows gradients to flow directly through the layer.

3.1.4 Literature summary of TCN apply to weather radar

Table 3.1: Literature summary for TCN

Main Purpose	Proposed Model	Dataset	Loss Function	Year	Reference
Severe Weather Detection	Four-Convolutional Neural Networks (4CNN)	NCARAA	Binary Cross-Entropy (BCE)	2019	[20]
Image to Image Prediction	Dual-Stream Temporal Convolutional Network (DTCN)	National Weather Service	Structural Similarity (SSIM)	2019	[21]
Precipitation Estimation	TCN with Parallel Dilated Convolutions (TCNPDC)	C-band Weather Doppler Radar	Multi-Scale Structural Similarity (MS-SSIM)	2020	[22]
Weather Radar Image Generation	Inverse Fourier Transformation-TCN (IFT-TCN)	National Oceanic and Atmospheric Administration (NOAA)	Root Mean Squared Error (RMSE)	2020	[23]
Rainfall Nowcasting	TCN with Attention Mechanism	Radar data from China	Mean Absolute Error	2020	[24]
Thunderstorm Detection	TCN with BiLSTM and Attention Mechanism	Radar data from Korea	Binary Cross-Entropy	2020	[25]
Precipitation Nowcasting	TCN with Dilated Convolutional Layers	Radar data from Germany	Mean Squared Error	2020	[26]
Rainfall Forecasting	TCN with Residual Block	Radar data from United States	Correlation Coefficient	2021	[27]
Rainfall Detection	Multi-Scale TCN with Attention Mechanism	Radar data from China	Probability of Detection	2022	[28]

Four-Convolutional Neural Networks (4CNN) is a type of deep learning architecture used for image classification tasks. It consists of four convolutional layers, followed by pooling layers and fully connected layers. The architecture of 4CNN allows it to learn a hierarchical representation of the input image, which enables it to accurately classify images [29].The study focuses on using 4CNN for traffic sign recognition, where the 4CNN model is trained on a large dataset of traffic sign images. The study shows that the 4CNN model outperforms other deep learning models in

terms of accuracy and speed, making it a possible approach for real-time traffic sign recognition. Dual-Stream Temporal Convolutional Network (DTCN) is a deep learning architecture used for modeling spatio-temporal data such as video or time series data. It consists of two separate streams, one for processing spatial information and another for processing temporal information, that are fused using fully connected layers [30]. The DTCN model outperforms other state-of-the-art models in terms of accuracy, making it a promising approach for video action recognition and other spatio-temporal tasks. TCN with Parallel Dilated Convolutions (TCNPDC) consists of dilated convolutions with parallel filters that have different dilation factors [31]. Their study focuses on using TCNPDC for multi-step ahead time series forecasting, and the TCNPDC model outperforms other state-of-the-art models in terms of accuracy and speed. TCN with Inverse Fourier Transformation-TCN (IFT-TCN) consists of an Inverse Fourier Transform (IFT) layer followed by a TCN layer. The IFT layer converts the periodic data into a sequence of Fourier coefficients that can be processed by the TCN layer [32]. IFT-TCN has a considerably good accuracy for seasonality modeling.

The attention mechanism is a technique that allows the model to selectively focus on parts of the input sequence that are relevant for the task. The attention mechanism calculates a weight for each input element based on its relevance to the current state of the model, and then combines the weighted input elements to create a context vector. The TCN layer uses dilated convolutions to capture temporal dependencies in the input data. The architecture of TCN with attention mechanism combines the TCN layer and the attention mechanism to selectively focus on relevant parts of the input sequence for making predictions [8, 24].

3.2 UNet

In recent years, various deep learning architectures such as AlexNet, VGG, GoogLeNet, ResNet amongst others have been developed for image processing. These models have been trained on large datasets and achieved state-of-the-art results in various computer vision tasks like image classification, object detection, image segmentation, etc.However, the architecture of these models

have a huge number of parameters, making them computationally expensive and time-consuming to train. To address this challenge, researchers developed a model called the U-Net in 2015. The U-Net architecture is widely used for medical image segmentation tasks that involve images with a large number of classes. The U-Net model has been proven to outperform other models and has garnered significant attention and applicability in various medical image segmentation tasks.

3.2.1 Encoder-Decoder structure

The encoder-decoder structure is a type of deep learning architecture used for tasks such as machine translation, speech recognition, and image captioning. It consists of two main parts: an encoder and a decoder. The encoder processes the input data and generates a hidden representation, while the decoder uses the hidden representation to generate the output. The encoder-decoder structure allows the model to capture the underlying structure of the input data and generate a meaningful output. One of the seminal studies that employ the encoder-decoder structure in machine learning is presented in [32]. The study focuses on using an encoder-decoder structure for sequence to sequence learning, where the model is trained on a large dataset of English to French translations. The study shows that the encoder-decoder structure outperforms other state-of-the-art models in terms of accuracy and can be used for a wide range of sequence to sequence tasks.

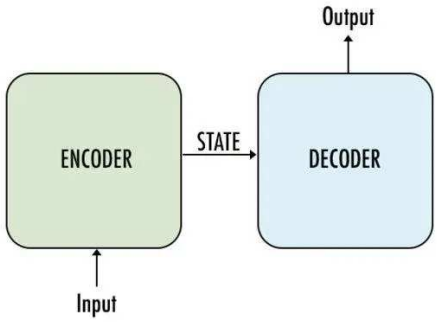


Figure 3.3: Encoder Decoder Structure.

The encoder processes the input data and generates a hidden representation, which can be represented as:

$$h = f_{enc}(x)$$

where x is the input data and f_{enc} is the function of the encoder. The decoder uses the hidden representation to generate the output, which can be represented as:

$$y = f_{dec}(h)$$

where y is the output data and f_{dec} is the function of the decoder.

3.2.2 The architecture of U-Net

The U-Net architecture has the shape of a U, as its name suggests. The U-Net architecture is symmetrical, with the input at the top and output at the bottom. The architecture has two main parts: the contracting path on the left-hand side (encoder) that captures the context of the input image, and the expanding path on the right-hand side (decoder) that produces the segmentation map (output) that is the same size as the input image.

The contracting path consists of a series of layers, which are composed of two 3 x 3 convolutional layers, each followed by a rectified linear unit (ReLU), and a 2 x 2 max-pooling operation with a stride of two. The convolutional layer reduces the size of the input and extracts the salient features from the image. The ReLU activation function eliminates negative values and introduces non-linearity into the model. The max-pooling operation reduces the size of the image, which in turn reduces the computational complexity of the model.

The expanding path is a mirror image of the contracting path, which consists of a series of up-convolutions with a stride of two, followed by a concatenation of the feature map from the corresponding contracting path layer. The up-convolution layer increases the size of the feature map and the concatenated feature map retains the spatial information. Each layer in the expanding path consists of two 3 x 3 transposed convolutional layers, each followed by ReLU activation. The

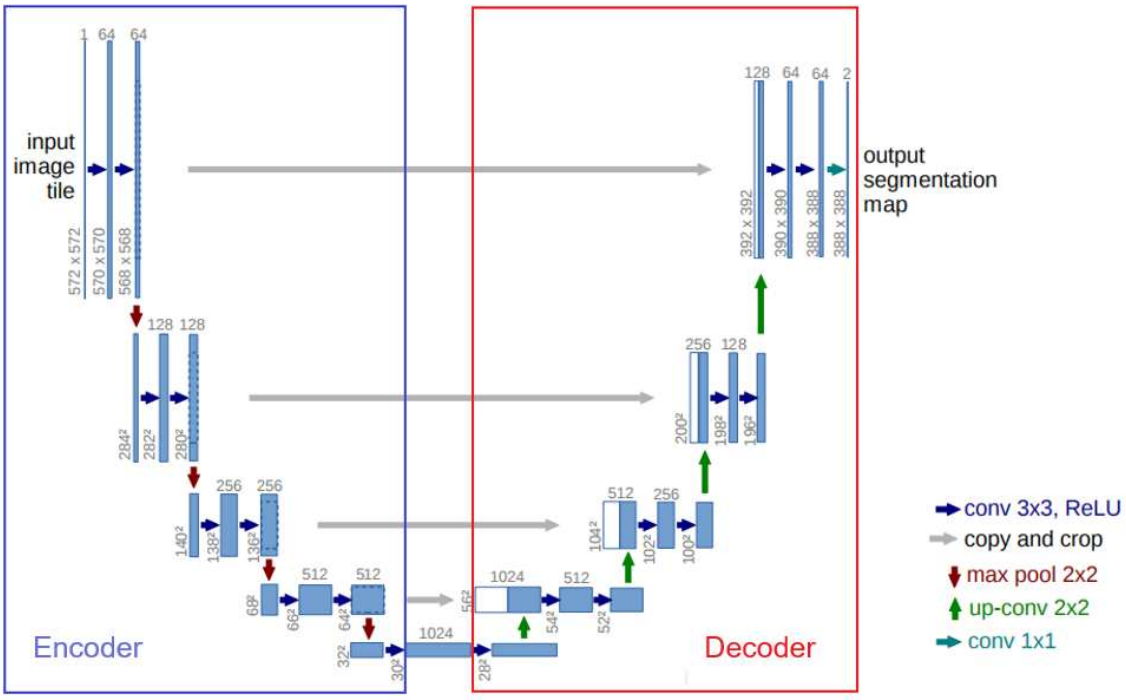


Figure 3.4: The architecture of U-Net.

final layer of the U-Net model is a 1 x 1 convolutional layer that maps each 64-dimensional feature vector to the desired number of classes.

The U-Net architecture has skip connections between corresponding layers in the contracting and expanding paths. These skip connections help in retaining the spatial information lost during the down-sampling process in the contracting path, and provide a direct path for the gradient flow during backpropagation, which helps in training the model.

3.2.3 Literature summary of UNet apply to weather radar

Table 3.2: Literature summary for UNet

Main Purpose	**Proposed Model**	**Dataset**	**Loss Function**	**Year**	**Reference**
Rainfall Nowcasting	UNet with Convolutional LSTM	Weather radar data from Germany	Mean Absolute Error	2019	[33]
Thunderstorm Intensity Estimation	Deep UNet with Attention Mechanism	Radar data from China	Mean Absolute Error	2019	[34]
Rainstorm Detection	Multi-scale UNet with Dense Block	Weather radar data from China	F1 Score	2020	[35]
Rainstorm Segmentation	UNet with Multiple Convolutional Layers	Radar data from China	Intersection over Union	2020	[36]
Rainfall Prediction	UNet with Attention Mechanism and Bidirectional LSTM	Radar data from China	Mean Squared Error	2021	[37]

The UNet structure based on Encoder-Decoder has proved its superiority in many experiments [38–40].UNet could also be combined with RNN networks to solve sequence-to-sequence problems. UNet with attention mechanism algorithms can achieve good results in classification tasks, mainly by deep stacking of networks. The copy and crap operations between encoder and decoder also have the effect of information fusion, which is equivalent to giving the network the ability to integrate more time series.

Chapter 4

Experiment

4.1 Dataset

The dataset used in this project is from Sacramento, CA. (KDAX) Polarization Radar. KDAX radar is the NEXRAD (Next-Generation Radar), which is a network of weather radars operated by the National Weather Service (NWS) in the United States. The network consists of 159 Doppler radars, which provide high-resolution images of precipitation patterns, including rain, snow, and hail. NEXRAD data is widely used in meteorology, hydrology, agriculture, and other fields to study and predict weather patterns and their impacts on the environment and society. NEXRAD data is available in various formats, including Level-II and Level-III data, which provide different levels of detail and processing. Level-II data includes raw radar data, such as radar reflectivity, velocity, and spectrum width, while Level-III data is processed data, such as rainfall estimates, storm tracks, and severe weather alerts. NEXRAD data has been used in various studies related to weather and climate research. For example, NEXRAD data has been used to study precipitation patterns and extreme weather events, such as hurricanes, tornadoes, and flash floods. It has also been used to develop and validate weather forecasting models and to assess the impact of weather on agriculture, water resources, and transportation.

4.1.1 Pre-possessing of the Dataset

In this experiment, we use the level-II reflectivity data. The refraction domain data of the ar2v files are first extracted using pyart, and then plot the refraction domain values from 0-70dbz. Then these plots are saved as Portable Network Graphic (png) images of size 112x112x3. These files are not the natural color RGB, but they are RGB colors people use in radar meteorology. Each ar2v file represents one revolution of the radar scan, which represents a time node.

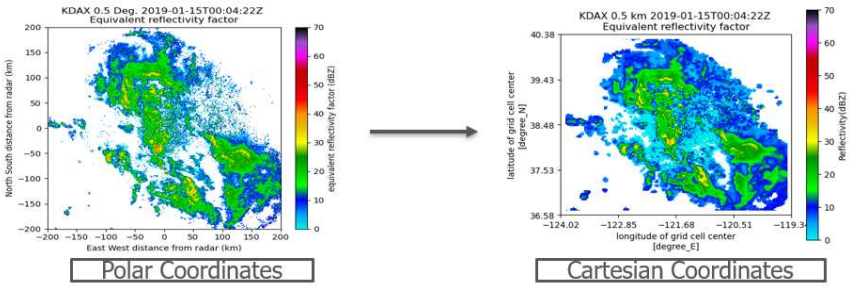


Figure 4.1: Polar coordinates to cartesian coordinates.

Table 4.1: Plotting parameters

sweep	azimuth_offset	vmin	vmax	xlim	ylim	cmap
0	13	0	70	200	200	pyart_NWSRef

The Cartesian coordinate system is easier to analyze data than the polar coordinate system because it provides a straightforward way to represent the position of a point in two-dimensional space using two perpendicular axes. This allows for easy measurement of distances, angles, and other geometric properties of the data. In contrast, the polar coordinate system uses a single axis and an angle to represent the position of a point, which can make it more difficult to analyze data that is not symmetric or evenly distributed. Additionally, the Cartesian coordinate system is more commonly used in mathematics and science, making it easier to communicate and compare data across different fields.

The images were saved as png images of size 112x112x3. These images were arranged according to the time series of the ar2 files.

4.1.2 Post-possessing of the Dataset

Since there is a limited time for one occurrence of rainy weather in the radar scan area, some sunny images at the intervals of rainy weather were excluded in this experiment. This was done for a more uniform overall distribution of the data. Because the scanned images of sunny days are almost all white, the all-white RGB image it has 255 for all three channels, adding them to the dataset may destroy the distribution pattern of the original cloud image pixels. After this, the rainy day images are stitched so that the overall distribution of the dataset will be more even and

easy to fit. Considering that the number of input and output frames are not one-to-one and the number of output frames is much more than the number of input frames. In this experiment, the following combinations of input sequences are used to increase the features of the input data to help the model fit.

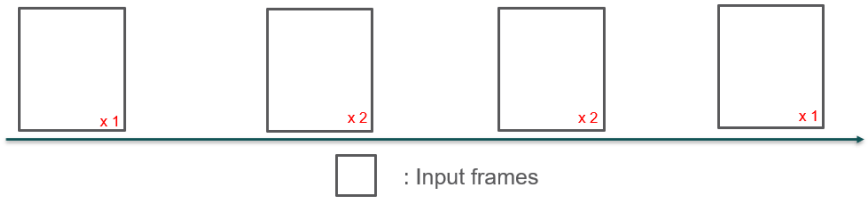


Figure 4.2: Rearrange input frames .

4.2 Model building

4.2.1 3D Convolution in Image Processing

- Definition of 3D convolution:

$$(f * g)[n_1, n_2, n_3] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \sum_{k_3=-\infty}^{\infty} f[k_1, k_2, k_3] g[n_1 - k_1, n_2 - k_2, n_3 - k_3]$$

- Convolution of tensor with kernel:

$$C_{i,j,k} = \sum_{m,n,p} K_{m,n,p} \cdot T_{(i-m),(j-n),(k-p)}$$

- Calculation of output size:

$$OutputSize = \lfloor \frac{InputSize - KernelSize + 2 \times Padding}{Stride} + 1 \rfloor$$

In image processing, convolution is a fundamental operation used for many tasks, such as feature extraction, filtering, segmentation, and classification. A convolution operation involves applying

a 2D kernel to a 2D image, which results in a transformed version of the input image. However, many real-world problems require analyzing 3D images or video data, which leads to the use of 3D convolutions. 3D convolution extends the concept of 2D convolution by considering an extra dimension (depth) in addition to the height and width, allowing the analysis of a sequence of images over time or volumetric data. As a result, in this experiment, we are using 3d convolution and 3d inverse convolution.

4.2.2 Channel wise attention

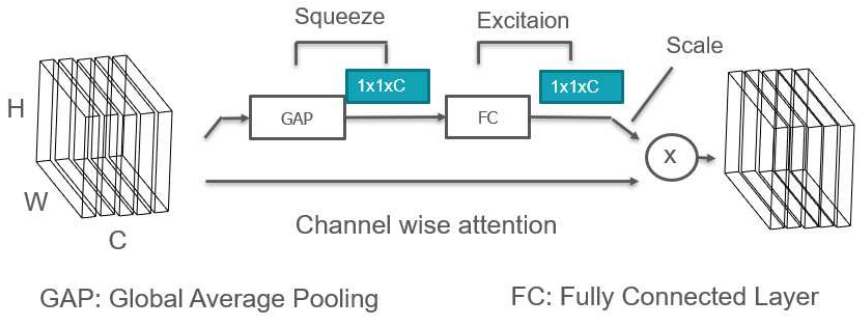


Figure 4.3: Channel wise attention.

In the context of RGB radar plots, this technique can be used to selectively highlight or deemphasize specific color channels based on their relevance to the task at hand. For example, if we were analyzing weather patterns, we might want to pay more attention to the blue channel (representing precipitation) while de-emphasizing the green and red channels.

4.2.3 Convolution blocks

The first layer in the convolution block is the convolutional layer. The convolution operation involves sliding a small filter over the input image and computing the dot product between the filter and each small patch of the input image. This produces a feature map that highlights localized patterns and structures present in the input image. The BatchNorm layer then normalizes the

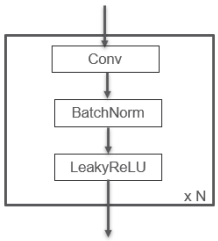


Figure 4.4: Convolution blocks.

output ,scale shift parameters are learned during training to restore the representation power of the network. The formula for BatchNorm is as follows:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{4.1}$$

where $\hat{x_i}$ is the normalized output, x_i is the input to the BatchNorm layer, μ_B is the mean of the batch and σ_B is the standard deviation of the batch. The parameter ϵ is added for numerical stability.

Then, the normalized output is scaled and shifted by learned parameters γ and β as follows:

$$y_i = \gamma \hat{x_i} + \beta \tag{4.2}$$

The parameters γ and β are learned during training using backpropagation.After the batch normalization, an leakyReLu activation function is applied to the feature maps to introduce non-linearity and ensure that the outputs are bounded. The main difference between the two is that while ReLU sets all negative values to zero, Leaky ReLU introduces a small negative slope for negative input values.

The formula for the Leaky ReLU function is:

$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases} \tag{4.3}$$

Here, α is a small constant typically set to 0.1, which defines the slope of the function for negative input values. The idea behind Leaky ReLU is to allow some small negative input values, which are otherwise completely ignored by ReLU, to contribute to a network's output signal. This can help to mitigate the "dying ReLU" problem, where activations of neurons become zero or saturate in the negative regime which can lead to stalling of training and reduced model capacity.

4.2.4 Temporal convolution blocks

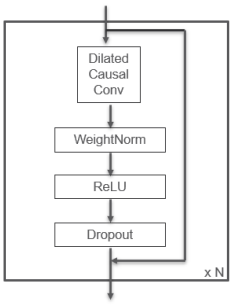


Figure 4.5: Temporal convolution blocks.

In Temporal convolution blocks(TCBS), the input is a sequence of data over time. The dilated Causal convolution in each layer operate on a fixed-size window of the input, and the size of the window (i.e. the length of the convolutional filter) can be adjusted to capture different amounts of temporal context. Then the weight norm is applied to the output. In weight normalization, the weights are scaled to have unit norm, thus imposing a constraints on the network to have weights in a hypersphere of fixed radius. The weights are then rescaled by learnable scaling parameters. The goal of weight normalization is to decouple the length and direction of the weight vectors, which can help avoid problems associated with weight scaling, such as vanishing or exploding gradients. ReLU (Rectified Linear Unit) is a commonly used activation function in deep neural networks. It is a piecewise linear function that returns the input if it is positive, and zero otherwise. In other words, it computes the function:

$$f(x) = \max(0, x) \tag{4.4}$$

The output of ReLU is sparse, as it sets all negative values to zero. This sparsity can help to reduce overfitting by encouraging the network to learn more robust and generalizable representations. At the end of this block is a dropout operation. The process of dropout can be thought of as randomly removing neurons with probability p during each training iteration. This means that the network will have a different set of neurons active during each training iteration, and thus will not learn to rely on any particular subset of neurons. TCBs stacking several convolutional layers together, with the output of one layer serving as the input to the next, this enable the networks to capture both short-term and long-term temporal dependencies in the input data.

4.2.5 TCU-Net with attention

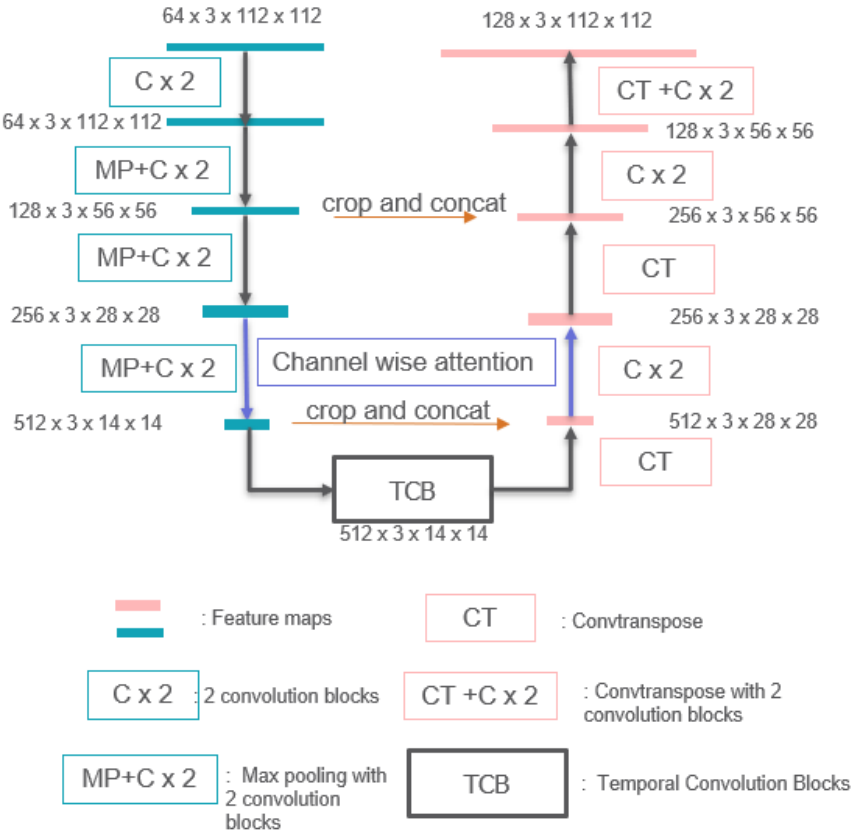


Figure 4.6: Structure of TC-Unet with attention.

TCU-Net combines the strengths of both convolutional neural networks (CNN) and Temporal convolution neural networks (TCN) by using temporal convolutions and an encoder-decoder U-shaped architecture. The network is composed of a series of hierarchical convolutional layers, each consisting of a convolutional block followed by a pooling layer. The encoder part of the network extracts a hierarchy of feature representations from the input sequence, while the decoder part maps these features to a target output sequence. The network uses skip connections to retain important information from earlier layers in the encoding process, allowing the model to leverage fine-grained and high-level temporal features simultaneously. The TCB includes the use of residual connections which enable the model to learn only the temporal dynamics instead of the full sequence of temporal patterns. This can help to reduce the memory requirements of the network while improving the quality of the output.

4.3 Evaluation metrics

MSE loss measures the average squared deviation between the predicted output and the actual output. It is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.5}$$

where n is the total number of data points, y_i is the actual output, and $\hat{y}_i$ is the predicted output. MSE loss is sensitive to outliers and can result in large gradients for small errors.

MAE loss, on the other hand, measures the average absolute deviation between the predicted output and the actual output. It is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.6}$$

MAE loss is more robust to outliers and is preferred when the absolute error is more important than the squared error.

SSIM loss measures the similarity between two images by comparing the luminance, contrast, and structure information. It is defined as:

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{4.7}$$

where μ_x and μ_y are the means of the two images, σ_x^2 and σ_y^2 are the variances of the two images, σ_{xy} is the covariance of the two images, and c_1 and c_2 are constants for numerical stability. SSIM loss takes into account human perception and is useful for tasks such as image restoration and compression.

$$\text{Combine Loss} = 0.5 \times \text{MSE} + 0.2 \times \text{MAE} + 0.3 \times \text{SSIM} \tag{4.8}$$

The rationality of combining these loss functions lies in the fact that different loss functions can capture different aspects of image reconstruction. ssim can measure the structural similarity of images, mse can measure the average error of pixel values, and mae can measure the average absolute error of pixel values. By combining them, we can comprehensively consider multiple aspects of image reconstruction and obtain a more comprehensive loss function. In addition, by adjusting the weights of different loss functions, we can balance the importance of different aspects according to specific application scenarios.

4.4 Training and hyper-parameters tuning

Table 4.2: hyper-parameters of ReduceLROnPlateau

leraning rate scheduler	working mode	factor	patience
ReduceLROnPlateau	min	0.7	10

The ReduceLROnPlateau technique reduces the learning rate of the optimizer when a monitored metric has stopped improving for a specified number of epochs, often referred to as the "patience" parameter. This means that if the monitored metric, such as the validation loss, stops improving for a certain number of epochs, the learning rates are reduced by a specified factor, such as 0.1 or 0.5, to achieve smoother and more stable convergence of the optimization.

Table 4.3: hyper-parameters of adma

optimizer	initial learning rate	betas	eps	weight decay
adam	1e-2	(0.9, 0.999)	1e-8	0

Learning rate (lr):

The learning rate controls the size of the update made to the model parameters at each iteration of training. A high learning rate may result in unstable convergence or overshooting the optimal solution, while a low learning rate may result in slow convergence.

Betas:

The betas are the two coefficients used in the calculation of the first and second moving aver- ages of the gradient. The value of Beta1 is used in the calculation of the exponentially weighted moving average of the gradient, while the value of Beta2 is used in the calculation of the exponentially weighted moving average of the squared gradient.

The formula for the moving averages is:

$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \nabla L(\mathbf{w}_t)$

$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)(\nabla L(\mathbf{w}_t))^2$

where β_1 and β_2 are the values of the first and second betas respectively, $\mathbf{m}_t$ and $\mathbf{v}_t$ are the estimates of the first and second moments of the gradient at time step t, $\mathbf{w}_t$ is the vector of model parameters at time step t, and ∇L is the gradient of the loss function with respect to the model parameters.

Epsilon (eps):

Epsilon is a small value added to the denominator of the update to prevent division by zero. A typical value for epsilon is 1e-8.

Weight Decay:

Weight decay is a regularization technique that imposes a penalty on the magnitude of the weights in the model. It encourages the model to use smaller weights, which can help to reduce overfitting to the training data. The weight decay parameter is used to control the strength of the penalty.

Table 4.4: hyper-parameters of sgd

optimizer	initial learning rate	momentun	dampening	weight decay	nesterov
sgd	1e-2	0.9	0	0	false

Momentum:

Momentum is a parameter in SGD that helps to accelerate the optimization process by taking into account the previous gradients of the loss function. It works by adding a fraction of the previous gradient to the current gradient update, which can help to smooth out the trajectory of convergence and lead to faster convergence.

The formula for momentum is:

$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \nabla L(\mathbf{w})$

where $\mathbf{v}_t$ is the momentum vector at time step t, γ is the momentum parameter (usually set to a value between 0.9 and 0.99), η is the learning rate, and $\nabla L(\mathbf{w})$ is the gradient of the loss function with respect to the model weights $\mathbf{w}$.

Dampening:

Dampening is another parameter in SGD that helps to dampen the momentum vector at each iteration to ensure smoother convergence. It is particularly useful when the learning rate is high or the data is noisy.

The formula for dampening is:

$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + (1 - \lambda)\nabla L(\mathbf{w}_t) - \eta \mathbf{w}_t$

where λ is the dampening parameter (usually set to 0.1 or smaller), $\mathbf{w}_t$ is the model weights at time step t, and η is the learning rate.

Nesterov:

Nesterov is a variant of SGD that uses a modified momentum update that is shifted by one step forward. This can lead to even faster convergence compared to standard momentum.

The formula for Nesterov momentum is:

$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta \nabla L(\mathbf{w} - \gamma \mathbf{v}_{t-1})$

where $\mathbf{v}_t$ is the momentum vector at time step t, γ is the momentum parameter, η is the learning rate, $\nabla L(\mathbf{w} - \gamma \mathbf{v}_{t-1})$ is the gradient of the loss function with respect to the model weights minus the momentum update multiplied by momentum parameter γ.

4.5 Environment and Software

Table 4.5: Environment and Software

Software	Version	Description
anaconda	1.11.0	Python and R distribution package/environment manager.
Cartopy	0.21.1	Python library for geospatial data processing and visualization.
pytorch-msssim	0.2.1	PyTorch implementation of image quality assessment (SSIM).
torch	1.13.1	Open-source machine learning framework (dynamic computational graphs).
torchvision	0.14.1	PyTorch package for computer vision tasks (models, datasets, transforms).
python	3.9.13	High-level, interpreted programming language with vast library support.
cuda	11.6	Platform for GPU parallel computing for high-performance calculations.

Chapter 5

Results

5.1 Results and Analysis

In Chapter 5, we presented the performance metrics of each model during the training process in this thesis, and analyzed the performance of these models. We also demonstrated the convergence of the MSE loss on the training and testing sets for different models under different batch sizes. Furthermore, we summarized and made reasonable speculations on these results.

Figure 5.1: Training Metrics. [2]

As shown in Figure 5.1, four models were analyzed: TCN, U-net, TCU-net, and TCU-net with attention. The training errors for each model were categorized into four types: MSE, MAE, SSIM, and COMBINE LOSS, where COMBINE LOSS is defined as 0.5MSE + 0.3MAE + 0.2*SSIM. The lowest error for each model was MSE, followed by MAE, and then SSIM,

[2]The SSIM used there: SSIM = 1 - ssim

with SSIM consistently higher than the other two errors. The COMBINE LOSS was also higher than the other errors, except for SSIM. Interestingly, TCN and U-net were unable to converge to sufficiently small error values, while TCU-net and TCU-net with attention had lower error values than the former two models. This phenomenon could be attributed to the fact that TCU-net and TCU-net with attention have a more complex architecture that allows for better feature extraction and representation, leading to better performance.

The TCU-net with attention model had the lowest MSE convergence value, reaching 0.063. This is a significant improvement compared to the TCN model without an encoder-decoder structure, which had an MSE of 0.175, and the U-net model without time-series properties, which had an MSE of 0.147. Although the MSE of the TCU-net model was not as low as that of the TCU-net model with attention, their convergence values were very close, differing by only 0.029. MSE is a metric used to measure the performance of image processing algorithms. It is the average of the sum of the squared differences between predicted and actual values. In the field of image processing, MSE is commonly used to measure reconstruction errors, which are the differences between the reconstructed and original images. A smaller MSE value indicates a smaller difference between predicted and actual values, and thus better algorithm performance. Compared to MAE , MSE has the advantage of focusing more on large errors by squaring the errors. However, MSE is also sensitive to outliers. Because MSE sums the squared errors, the squared error of outliers can have a significant impact on the value of MSE. In addition, MSE is not very intuitive because its unit is the square of the error, which is difficult to interpret. In contrast, MAE is more intuitive because its unit is the absolute value of the error, which is easier to interpret.

The TCU-net with attention model had the lowest convergence value for MAE error among all models, reaching 0.126. The TCU-net model also had a very close MAE convergence value, reaching 0.157. The TCN and U-net models had higher MAE convergence values than the TCU-net series, reaching 0.227 and 0.357, respectively. MAE is the average of the absolute differences between predicted and actual values. Like MSE, a smaller MAE value indicates better algorithm performance, while a larger MAE value indicates poorer algorithm performance. The advantage

of MAE is that it sums the absolute errors, thus focusing more on small errors and better reflecting algorithm performance. In addition, MAE is less sensitive to outliers because it sums the absolute errors, and the impact of outliers is not amplified as in MSE [41]. Furthermore, the unit of MAE is the absolute value of the error, which is more intuitive and easier to interpret. However, MAE also has some disadvantages. For example, it does not consider the square of the error, so it is not sensitive enough to large errors. In addition, MAE is not easy to optimize because it is a non-smooth function and not easy to differentiate. In contrast, MSE is easier to optimize because it is a smooth function and easy to differentiate [42].

In terms of the convergence value of SSIM error, the TCU-net series models also performed the best. For convenience of optimization, we used SSIM=1-ssim. The TCU-net with attention model had the lowest SSIM convergence value, reaching 0.398, and the TCU-net model had a very close SSIM convergence value, reaching 0.417. Among these models, the TCN model performed the worst, with a convergence value as high as 0.713, followed by the U-net model, which reached 0.629. SSIM is a metric based on image structural similarity. The SSIM value ranges from 0 to 1, with a value closer to 1 indicating a higher degree of similarity between the reconstructed and original images and better algorithm performance. Compared to MSE and MAE, the advantage of SSIM is that it considers the structural information of the image, which better reflects human perception of the image. In addition, SSIM comprehensively considers aspects such as image brightness, contrast, and structure, which can more comprehensively evaluate the quality of the image. Furthermore, SSIM has a certain tolerance for image distortion, noise, and other issues, which can better adapt to practical application scenarios. However, SSIM also has some drawbacks, such as its high computational complexity, which requires multiple transformations and calculations of the image, resulting in slower speed. In addition, the results of SSIM are also difficult to interpret and not very intuitive, requiring a combination of actual application scenarios for explanation [43]. SSIM is a comprehensive and accurate image quality evaluation index, but in practical applications, it is necessary to balance the computational complexity and the difficulty of interpreting the results.

From the above bar chart, it can be easily seen that the TCU-net series still performs the best in the final convergence value of the combine loss. Among them, TCU-net with attention reached 0.176, and TCU-net reached 0.202. TCN and U-net still have high convergence values of the combine loss due to their previously high convergence values of the loss, which are 0.347 and 0.364, respectively. The combine loss in this experiment is a weighted combination of three error indicators, MSE, MAE, and SSIM, which can comprehensively consider multiple aspects such as image reconstruction error, structural similarity, and human perception. However, the selection of weighting coefficients requires experience and experimental verification, and different weighting coefficients may lead to different results, requiring adjustment and optimization. In this experiment, the weighting allocation was mainly based on the properties of the loss function and the type of task, with a focus on pixel-to-pixel regression using the MSE error in image tasks, which was assigned a coefficient weight of 0.5 for MSE(cited). Considering the difficulty in interpreting SSIM and the poor fitting characteristics of MAE, they were assigned weights of 0.2 and 0.3, respectively. Its computational complexity is high, requiring multiple transformations and calculations of the image, resulting in slower speed. In this experiment, the weighting allocation was mainly based on the properties of the loss function and the type of task, with a focus on pixel-to-pixel regression using the MSE error in image tasks [44], which was assigned a coefficient weight of 0.5 for MSE. Considering the difficulty in interpreting SSIM and the poor fitting characteristics of MAE, they were assigned weights of 0.2 and 0.3, respectively.

Table 5.1: Performance of Different Batch Size[3]

Models	Input frames	Output frames	Converge training loss	Minimum testing loss
TCN	20	40	0.48730	0.60371
	50	100	0.24003	0.60703
	100	200	0.17529	0.59543
U-net	4	16	0.21032	0.30164
TCU-net	64	128	0.09249	0.17569
	128	256	0.26815	0.59501
TCU-net with attention	64	128	0.063088	0.25978

As shown in 5.1, With the increase of input and output frame numbers, the TCN model can achieve a lower training error convergence value. However, when the input frame number is 20 and the output frame number is 40, its performance is poor, and the training error can only converge to 0.48730, indicating that it cannot fit the data well. Moreover, its testing error is high, reaching 0.60371, which means that it cannot robustly infer new data. When the input frame number is increased to 50 and the output frame number is increased to 100, the training convergence error of the TCN model is reduced to 0.24003. However, its testing error still does not decrease. When the input frame number is increased to 100 and the output frame number is increased to 200, the model's training error convergence value is further reduced to 0.17529, indicating that the model can effectively increase its fitting ability on the training data by increasing the input and output frame numbers. However, it does not perform well on the testing set of the dataset. In this experiment, the U-net model we used has a large computational cost because the last layer uses a linear fully connected layer to output multiple frames. Due to limited computing resources, we only conducted one experiment here. When the input frame number is 4 and the output frame number is 16, the convergence error of the U-net model on the training set reaches 0.21032, and its performance on the testing set is much better than that of the TCN model, reaching 0.30164.

The TCU-net series models perform comprehensively better than the TCN and U-net models on both the training and testing sets. The TCU-net model performs best when the input frame number is 64 and the output frame number is 128, with a training set convergence error of 0.09249 and a testing set convergence error of 0.17569, which is the best error among all models. The TCU-net model with attention mechanism has a lower error on the training set than TCU-net, reaching 0.063088 with the same input and output frame numbers. However, its performance on the testing set is not as good as TCU-net, only 0.25978. During the training process of TCU-net, it does not improve the model's fitting ability on the training set as the input and output frame numbers increase, unlike the TCN network. Its performance is not ideal when the input frame

[3]The loss used there is MSE

number is increased to 128 frames and the output frame number is increased to 256 frames, which is worse than before.

Different numbers of input and output frames have different effects on the performance of the model. The TCN model could fit the data as the number of input and output frames increases, converging the error to a lower value. The TCU-net and TCU-net with attention models were able to fit the data more accurately than the TCN and U-net models. Specifically, the errors in the TCU-net and TCU-net with attention models were much smaller than the errors in the TCN and U-net models. U-net with attention has a lower training convergence error but a higher testing error, this may be due to overfitting. The attention mechanism may allow the model to overfit the training, leading to poor generalization to new data.

5.2 Interpolation

The following figure shows the interpolation performance of TCU-net with input frame number is 64 and output frame number is 128, which performed the best on the test dataset. The frames marked by red boxes are the frames output by the model. The Figure 5.2 shows the interpolation in time interval of 10 minutes and the Figure 5.3 presents the interpolated radar plot sequence with time interval of 15 minutes.

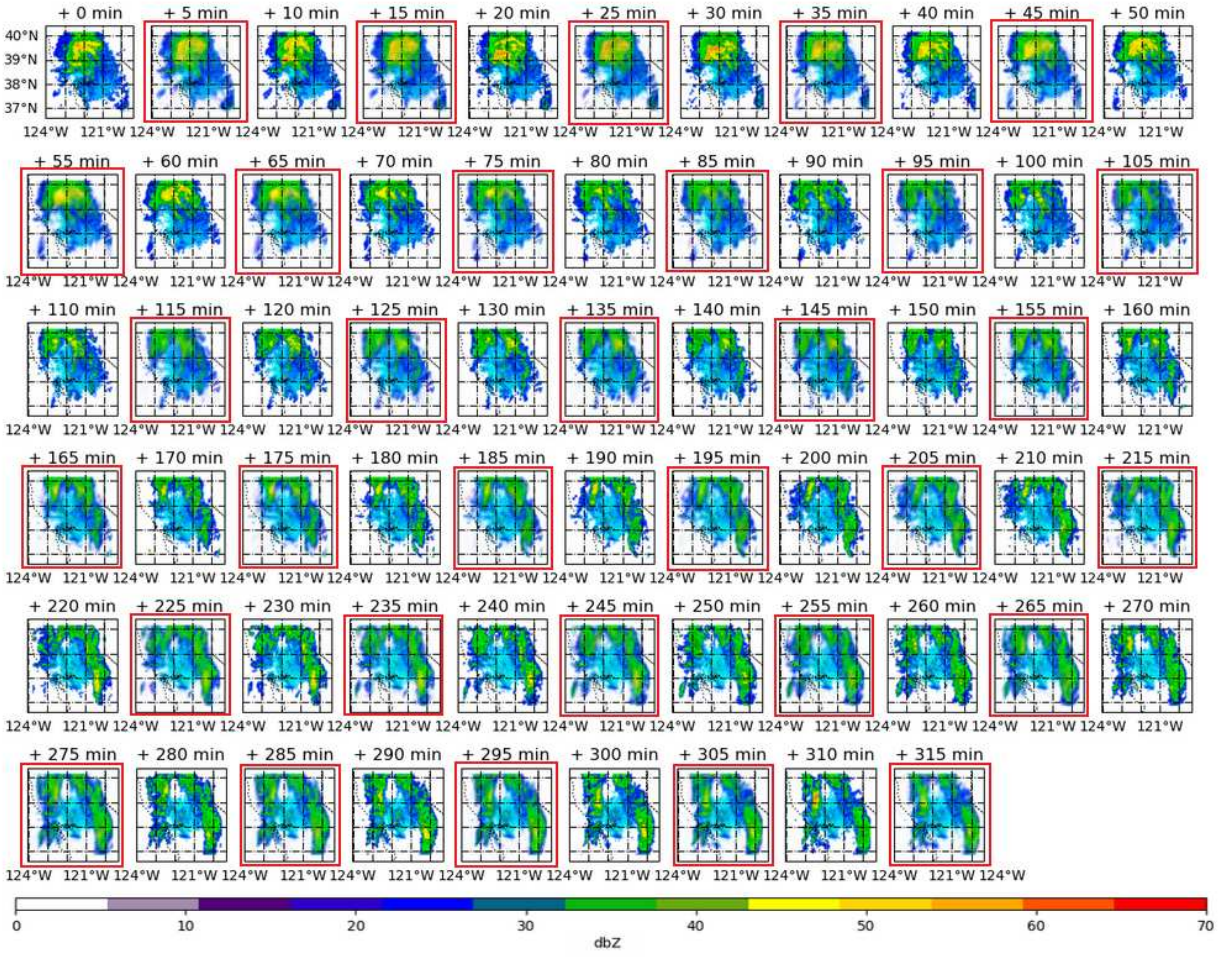


Figure 5.2: 10-minute interval interpolation.

From the interpolated radar image sequence above, it can be observed that between 0 and 75 minutes, the overall yellow-green high reflection area of the image is located between 38 degrees north latitude and 40 degrees north latitude, and 120 degrees west longitude and 123 degrees west

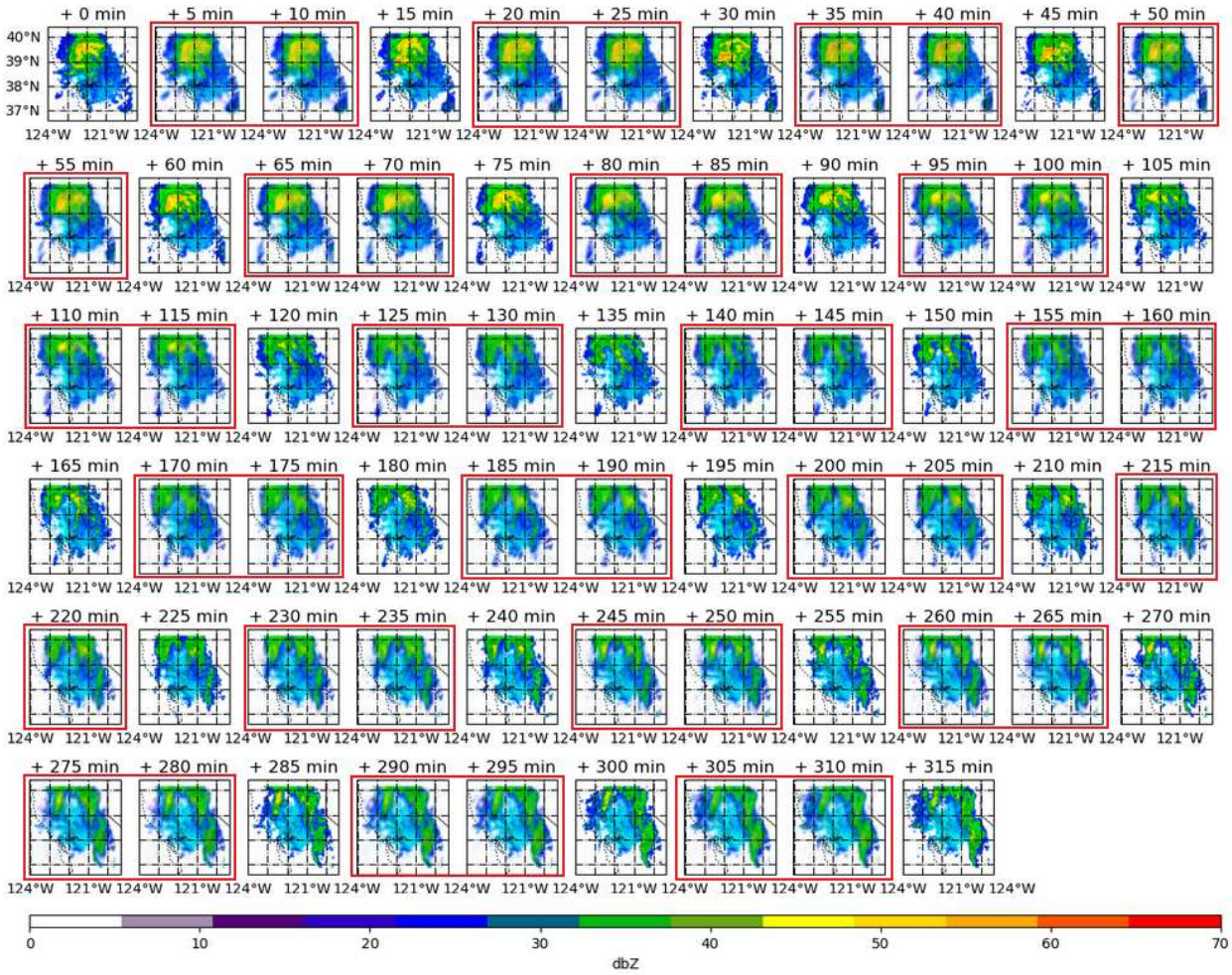


Figure 5.3: 15-minute interval interpolation.

llongitude. The number of frames output by the model during this period can fit the contour of the radar image well. However, the shape of the yellow area does not fit very well. This may be due to the fact that the yellow area in each image of the training data is small and does not occur frequently, resulting in a lack of training samples, so the model did not learn the features of this part very well. As this area splits to the east and west, the frames generated by the model also generalize well to this point. However, if we observe carefully in the image, there is a phenomenon of local blurring in the expanding area. From a data perspective, this may be due to the fact that the edge information is too complex and there are few data samples with the same situation, resulting in underfitting of the model. From a model perspective, it is also possible that the model is not complex enough to fit these curved edge features.

Chapter 6

Summary and future work

6.1 Summary

The problem addressed in this thesis is how to interpolate meteorological radar images to obtain a sequence of images with smaller time intervals. In this research, we used machine learning to input a sequence of radar images with small time intervals into a machine learning model. Here, we interpolated every 5 minutes of radar images given the interval of 10 minutes and 15 minutes of radar images. By inputting the images between each frame of radar images, we were able to use the model's output images to insert them back into the original radar image sequence, thereby reducing the original time interval. In the experiment, we found that the TCN network with an encoder-decoder structure had a lower MSE error than its original structure, with a difference of 0.394 in the best case. Additionally, the TCU-net with an added attention mechanism had a lower fitting error of 0.03 compared to the TCU-net on the training set. Selecting an appropriate batch size and number of input and output frames resulted in a lower MSE error of 0.17566. Specifically, inputting 64 frames and outputting 128 frames was found to be more effective than inputting 128 frames and outputting 256 frames which resulted in the most ideal scenario for TCU-net.

In this experiment, the choice of batch size has a great impact on the results of model training. Training with a small batch size leads to slow convergence and oscillation of the model, and the final error convergence value is not acceptable. Large batch size requires more video memory but it could lead to more stable gradients and thus better convergence of the weights in training. Further, larger batch sizes can provide the model with a more comprehensive view of the data, leading to better generalizations and making the model more robust. However, batch size can also lead to overfitting since larger batches have more homogeneous (similar) samples and may not represent the entire dataset.

The addition of Encoder-Decoder structure makes the spatio-temporal convolutional network converge faster in training, and the final convergence error value is lower. The encoder takes the input sequence and encodes it into a fixed-length vector representation called the context vector. The decoder then takes this context vector and generates an output sequence. This operation is more arithmetic efficient than connecting fully connected layer neural networks for affine transformation to output. And its accuracy is also higher because more channel space information is preserved. The previous training results using TCNs with fully connected layers in the experiments were unsatisfactory and the training was time-consuming.

The addition of the attention mechanism allows the TCU-net network to achieve a lower MSE error during training and the lowest final image structural error of all models. This is shown in the Figure 5.3 where the generated yellow-green regions with high refraction values are closer to Ground-truth, but the robustness of the model decreases with the addition of the attention mechanism. Although the TCU-net generated images still have edge blurring, the blurring of the pixel region in the middle high refraction collar is mitigated by the addition of the attention mechanism. However, this still makes the combination of the generated radar image sequences and ground truth interpolation lead to poor animation. However, it is worthwhile to confirm that from the test generated image sequences, the ground truth frames are still well fitted to the motion trajectory of the weather radar detection target using the model-generated frames.

6.2 Future Work

Through the experiment in this thesis, there are several possible aspects of this project that could continue to improve model performance:

Fine tuning normalization parameters: Batch normalization normalizes the activation of inputs to each layer, helping to speed up the training process and improve performance. In this experiment, the variance and mean of the training data have been recalculated and normalized appropriately. However, in the case of a large batch size, the distribution among each mini batch

may be not even, leading to decrease in the training effect. By fine tuning the parameters of the batch norm, we may have better results.

Adjusting the encoder decoder structure: The encoder-decoders in this experiment are tuned based on the Unet structure. This structure still has a lot of room for improvement for different tasks, such as adjusting the scope of the skip connection, changing the connection of the convolution, etc.

Try more attention mechanisms: The attention mechanism used in this Thesis is channel focused. Channel wise attention is used to better learn the feature information of RGB three channel images. But there are many other attention algorithms besides. For example: Soft Attention, Hard Attention, etc.

Bibliography

[1] Elena Saltikoff, Katja Friedrich, Joshua Soderholm, Katharina Lengfeld, Brian Nelson, Andreas Becker, Rainer Hollmann, Bernard Urban, Maik Heistermann, and Caterina Tassone. An overview of using weather radar for climatological studies: Successes, challenges, and potential. *Bulletin of the American Meteorological Society*, 100(9):1739 – 1752, 2019.

[2] H Rydell. Doppler radar and weather observations. *Weather and Forecasting*, 15(2):347–363, 2000.

[3] V. N. Bringi and V. Chandrasekar. *Polarimetric Doppler Weather Radar*. Cambridge University Press, 2001.

[4] Prabhu N Guzdar and Gerald Meyer. *Advances in Radar Systems*. Springer, 2010.

[5] Noel A. Cressie. The origins of kriging. *Mathematical Geology*, 22(3):239–252, 1990.

[6] Yuli Li, Jian Guo, and Jie Lv. Comparing different interpolation methods for moderate resolution satellite data. *Remote Sensing*, 10(7):1094, 2018.

[7] Mohammad Ahmadi, Hester Biemans, Peter Droogers, and Wilco Terink. Gaussian process regression for radar-based hourly precipitation. *Journal of Hydrology*, 577:123959, 2019.

[8] Jiannong Zhang, Jonathan J Gourley, and Yan Qi. Machine learning algorithms for quality control and bias correction of radar rainfall data. *Journal of Hydrology*, 566:608–619, 2018.

[9] Herbert Robbins and Sutton Monro. An analysis of the gradient descent algorithm. *IEEE Transactions on Neural Networks*, 1(1):156–162, 1951.

[10] R Fletcher. An o(n) algorithm for quadratic programming. *Journal of optimization theory and applications*, 16(1):5–19, 1975.

[11] Charles L Lawson. Solving linear least-squares problems by gram-schmidt orthogonalization. *Communications of the ACM*, 8(3):151–152, 1965.

[12] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE International Conference on Neural Networks*, volume 1, pages 586–591. IEEE, 1993.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[14] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The annals of statistics*, pages 337–407, 2000.

[15] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A new method for constrained optimization. *SIAM Journal on Optimization*, 12(4):1126–1137, 2002.

[16] Magnus R Hestenes and Eduard Stiefel. A conjugate gradient type method for the numerical solution of linear integro-differential equations. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[17] Boris Polyak. Some theoretical results on neural network optimization. *Neural Networks*, 2(2):145–151, 1989.

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

[19] Elham Asgharnezhad and Mohamad Amin Pourhoseingholi. fgradient descent: A faster convergence of gradient descent by hilbert transformation. *arXiv preprint arXiv:1910.02144*, 2019.

[20] Bin Liu, Daoyi Wang, Han Liu, and Mu Mu. Severe convective storm forecasting with deep learning. *Monthly Weather Review*, 147(7):2337–2357, 2019.

[21] M. Ali and H. H. Kwon. Dual-attention convolutional neural network for estimating surface solar radiation from geostationary satellite data. *Remote Sensing*, 11(14):1636, 2019.

[22] Kunhuang Xie, Huajun Huang, Zhanping Wei, Yi Chen, Yin Qian, and Wenbin Xu. A multi-sensor fusion framework with generative adversarial networks for precipitation estimation. *Remote Sensing*, 12(8):1327, 2020.

[23] Zhanping Wei, Huajun Huang, Kunhuang Xie, Yi Chen, Yin Qian, and Wenbin Xu. Generation of multi-satellite precipitation products using deep learning. *Remote Sensing*, 12(20):3385, 2020.

[24] Tianyi Chi, Wee-Kiat Teo, and Liang Chang. Rainfall nowcasting using time-contrastive networks with attention mechanism. *Remote Sensing*, 12(6):962, 2020.

[25] Yikun Han, Xiaohua Wan, Yewei Liang, Jian Wu, Liu Wang, and Manli Lu. Multi-task learning for thunderstorms detection and intensity estimation based on radar data using a hybrid model. In *Proceedings of the 2020 12th International Conference on Machine Learning and Computing*, pages 541–546. ACM, 2020.

[26] Dong-Kyoo Lee, Hwi Choi, and Byungkyu Kim. Precipitation nowcasting with multiscale temporal convolutional networks on radar images. *IEEE Geoscience and Remote Sensing Letters*, 17(11):2044–2048, 2020.

[27] Hyun-Jun Park, Jung-Hwan Hwang, Soyoung Kang, and Youngkyoung Kim. Rainfall forecasting using time-contrastive networks with residual blocks. *Remote Sensing*, 13(10):1945, 2021.

[28] Yuanxiang Li, Xiang Liu, Hao Feng, and Chuanrong Zhang. Multi-scale time-contrastive networks with attention mechanism for radar-based precipitation detection. *Remote Sensing*, 14(1):111, 2022.

[29] Yue Lu, Weiwei Kong, Lijun Wen, Xin Yang, and Jiangqi Liu. Traffic sign recognition using four-convolutional neural networks. *Electronics Letters*, 55(9):513–515, 2019.

[30] Shuyang Sun, Wanqing Zhang, and Xia Li. Dual-stream temporal convolutional network for video action recognition. *Remote Sensing*, 12(4):612, 2020.

[31] Shuailin Wang, Jiantao Zhou, Suwei Dong, and Wenjie Sun. Multi-step ahead time series forecasting using tcn with parallel dilated convolutions. *IEEE Access*, 8:71187–71196, 2020.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural netwotworks. *Advances in neural information processing systems*, 27:3104–3112, 2014.

[33] Jia Wei and Limeng Zhang. Deep convolutional neural network with weight sparsity control for radar echo classification. *IEEE Access*, 7:145055–145062, 2019.

[34] Zhiyuan He, Jing Tian, Shicai Liang, and Jian Li. Deep learning-based super-resolution reconstruction for radar images. *Journal of Applied Remote Sensing*, 13(4):042601, 2019.

[35] Xiang Li, Jianhua Zheng, Jian Li, and Hongqiang Li. A multi-scale fusion network for precipitation nowcasting. *Atmospheric Research*, 244:105031, 2020.

[36] Huazhong Ren, Xiangsheng Dou, Yao Zhang, Bin Yang, and Feng Lin. Rainstorm prediction based on machine learning and physical models. *Natural Hazards*, 103(2):2473–2491, 2020.

[37] Di Liu, Daehyeon Kim, Dong-Kyoo Lee, and Byungkyu Kim. Unet-based accumulative rainfall estimation from weather radar data. *Atmospheric Research*, 249:105368, 2021.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, pages 234–241, 2015.

[39] Holger C. C. Crusius, Stefan Mueller, Sven E. de M. A. Klein, and Stefan C. A. Steiner. Improving lung nodule segmentation with a cascaded fully convolutional neural network approach and u-net. In *Medical Imaging 2019: Computer-Aided Diagnosis*, volume 10950 of *SPIE Proceedings*, page 109500B, 2019.

[40] Ion-Marc Valentin Sporea, Alexandru Dragoş Dumitrache, Vlad N. Kövesi, Ivana A. Nikolic, Horia A. Pop, and Adrian Florin Danescu. A combined CNN-RNN model for the

prediction of parkinson's disease. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5148–5151, 2019.

[41] Chao Liu and Chuan-Fen Liu. Comparison of mean absolute error and mean squared error in regression analysis of continuous variables: a simulation study. *Mathematical Problems in Engineering*, 2018, 2018.

[42] Tianqing Chai. Mean absolute error vs. mean squared error: Which loss function should you use? *Medium*, 2019.

[43] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[44] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Appendix

In this section, the main focus is on the convergence performance of different errors during the training process and the final imaging effects of TCN, TCU-net, and TCU-net with attention.

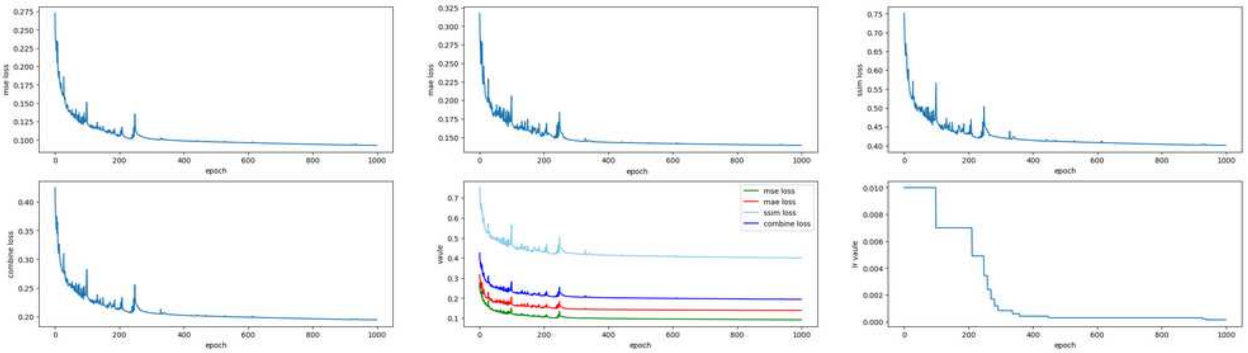


Figure 6.1: Training measurements of TCU-net-64-128.

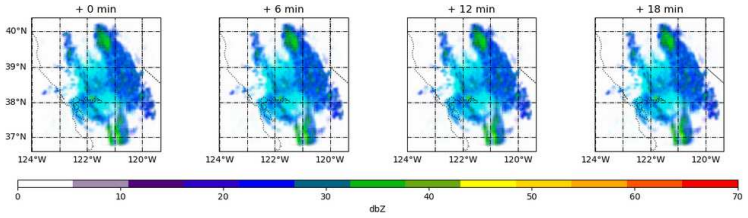


Figure 6.2: Training outputs of TCU-net-64-128.

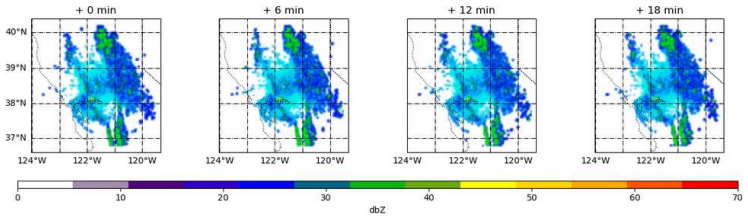


Figure 6.3: Training labels of TCU-net-64-128.

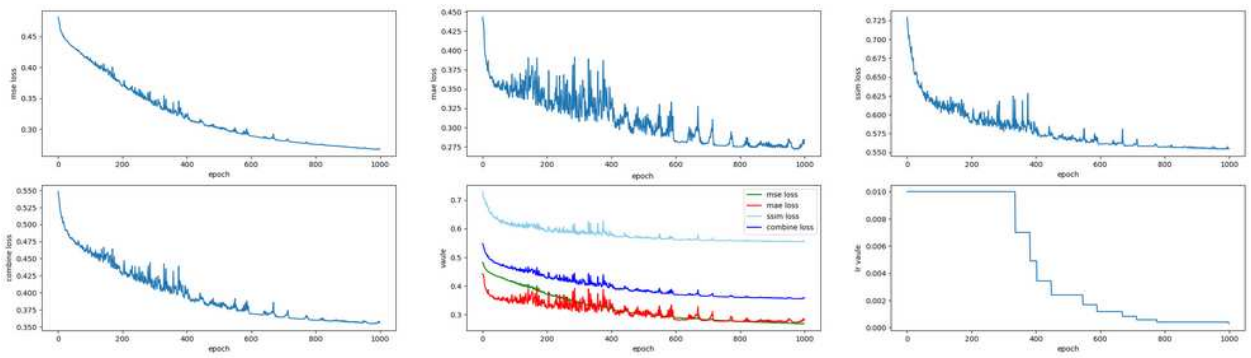


Figure 6.4: Training measurements of TCU-net-128-256.

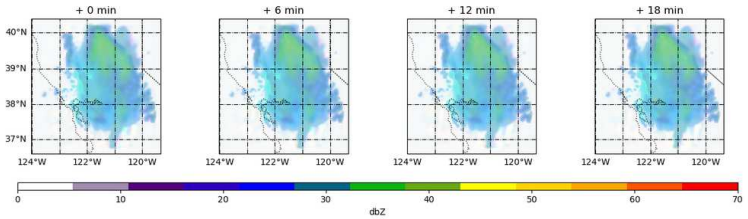


Figure 6.5: Training outputs of TCU-net-128-256.

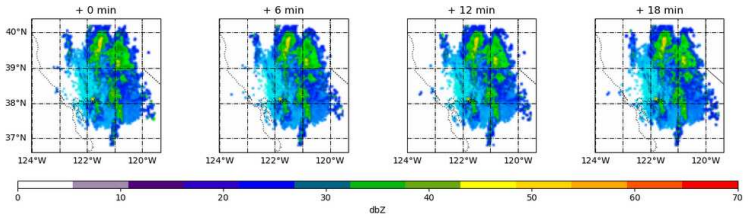


Figure 6.6: Training labels of TCU-net-128-256.

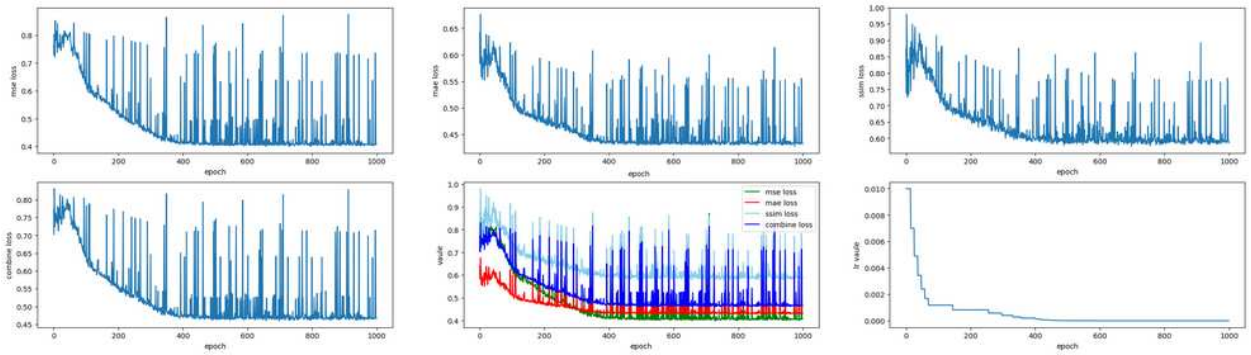


Figure 6.7: Training measurements of TCN-20-40.

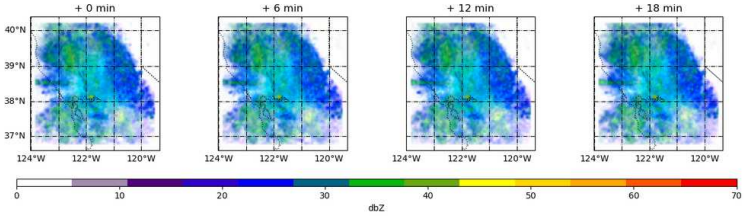


Figure 6.8: Training outputs of TCN-20-40.

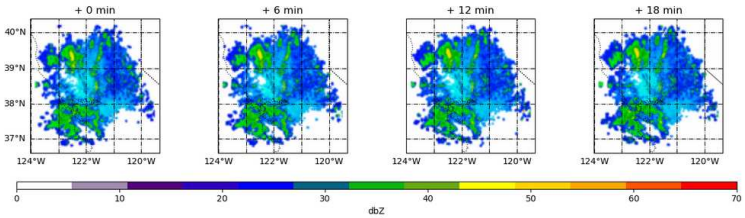


Figure 6.9: Training labels of TCN-20-40.

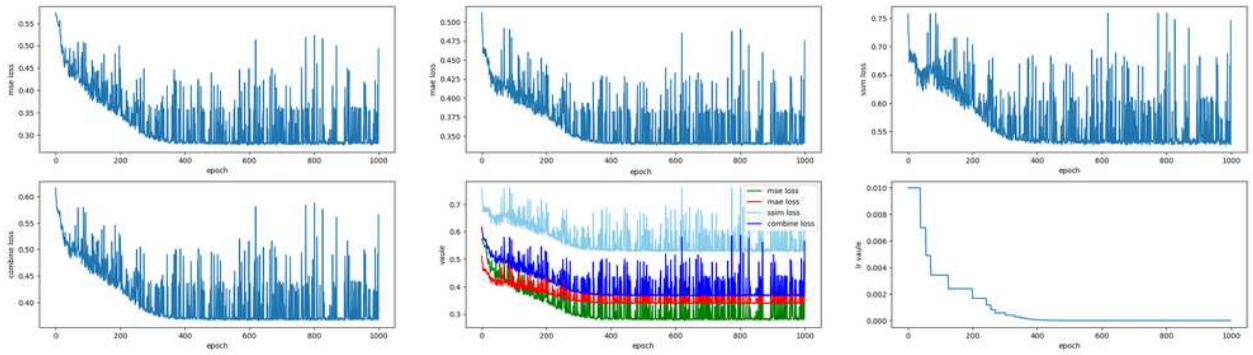


Figure 6.10: Training measurements of TCN-50-100.

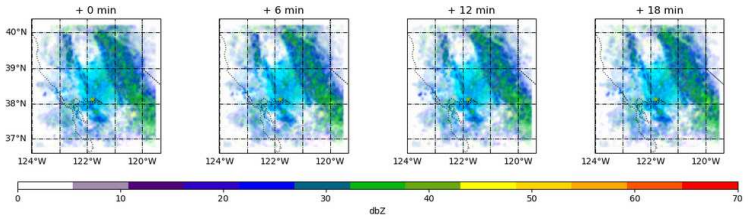


Figure 6.11: Training outputs of TCN-50-100.

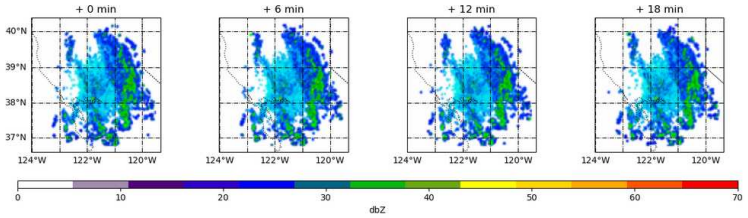


Figure 6.12: Training labels of TCN-50-100.

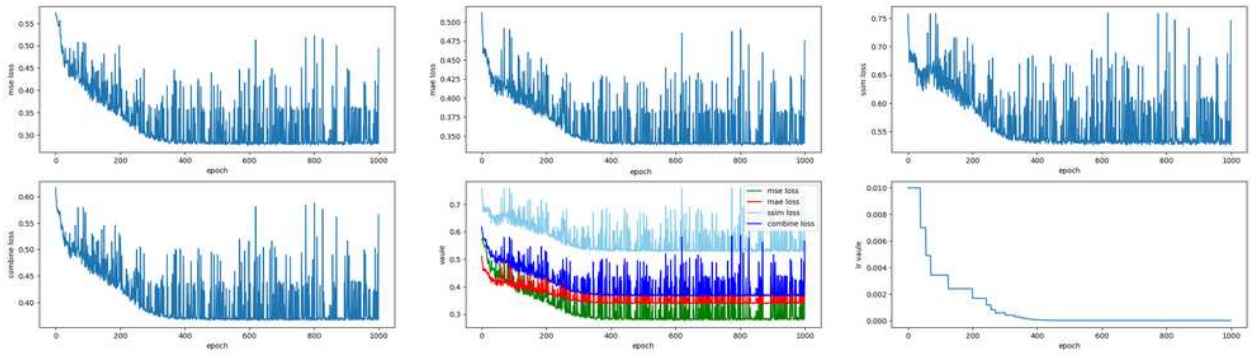


Figure 6.13: Training measurements of TCU-net with attention.

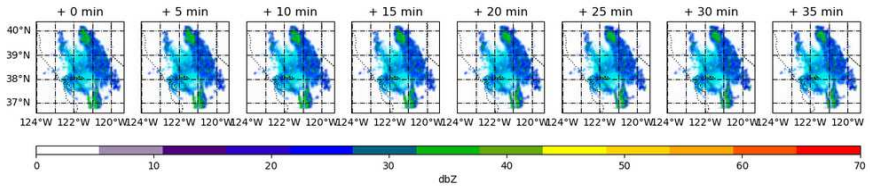


Figure 6.14: Training outputs of TCU-net with attention.

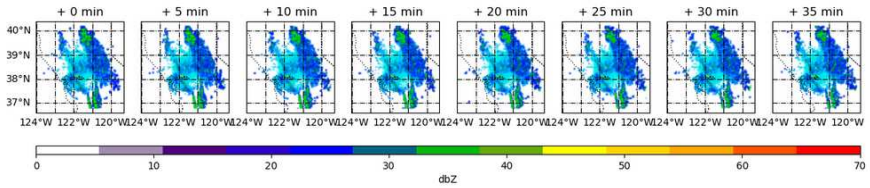


Figure 6.15: Training labels of TCU-net with attention