UNIVERSITY OF THE WESTERN CAPE

A Comparative Evaluation of 3D and Spatio-temporal Deep Learning Techniques for Crime Classification and Prediction

by

Tawanda Lloyd Matereke

A thesis submitted in fulfilment for the
degree of Master of Science

in the
Faculty of Natural Sciences
Department of Computer Science

Supervisor: Mehrdad Ghaziasgar
Co-supervisor: Clement Nyirenda

March 2022

Declaration of Authorship

I, Tawanda Lloyd Matereke, declare that this thesis "*A Comparative Evaluation of 3D and Spatio-temporal Deep Learning Techniques for Crime Classification and Prediction*" is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signed: _____

Date: _____

Acknowledgements

This thesis is a compilation of the highly regarded acknowledgements from many people that assisted me through the years. I would first like to thank my supervisor Dr Mehrdad Ghaziasgar and Co-supervisor, Dr Clement Nyirenda for the central role of involvement and the progress meetings during my study.

I would also like to give special thanks to my family and friends for their continuos support and understanding when undertaking my research and my project.

Finally I would like to thank God for letting me through all my diffficulties, and making all things possible.

Declaration of Publications

The following research papers has been ACCEPTED and PRESENTED for publication, and parts of its materials are included in the thesis:

1. **T. L. Matereke**, C. Nyirenda, M. Ghaziasgar, "A Performance Evaluation of 3D Deep Learning Algorithms for Crime Classification", In Proceedings of IEEE Africon 2021, Arusha, Tanzania, ISBN 978-1-6654-1983-3, pp. 106–111
2. **T. L. Matereke**, C. Nyirenda, M. Ghaziasgar, "A Comparative Evaluation of Spatio Temporal Deep Learning Techniques for Crime Prediction", In Proceedings of IEEE Africon 2021, Arusha, Tanzania, ISBN 978-1-6654-1983-3, pp. 100-105.

Abstract

This research is on a comparative evaluation of 3D and spatio-temporal deep learning methods for crime classification and prediction using the Chicago crime dataset, which has 7.29 million records, collected from 2001 to 2020. In this study, crime classification experiments are carried out using two 3D deep learning algorithms, i.e., 3D Convolutional Neural Network and the 3D Residual Network. The crime classification models are evaluated using accuracy, F1 score, Area Under Receiver Operator Curve (AUROC), and Area Under Curve - Precision-Recall (AUCPR). The effectiveness of spatial grid resolutions on the performance of the classification models is also evaluated during training, validation and testing. The classification experiment results show that the 3D ResNet-18 achieved the best performance with an accuracy of 0.9984, F1 score of 0.9942, AUROC of 0.9999 and AUCPR of 0.9933, whereas the 3D CNN achieved an accuracy of 0.9946, F1 score of 0.9764, AUROC of 0.9998, and AUCPR of 0.9766 during testing, with a spatial resolution of 16 pixels.

The crime prediction experiments are carried out using three Spatio-temporal deep learning, i.e., the Spatio Temporal Residual Network (ST-ResNet), the Deep Multi-View Spatio-temporal Network (DMVST-Net), and the Spatio Temporal Dynamic Network (STD-Net). The crime prediction models are evaluated using the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). The prediction experiment results show that the STD-Net achieved the best results of the three approaches RMSE of 0.2870, and MAE of 0.2093. The ST-ResNet and DMVST-Net also showed considerable promise. The ST-ResNet achieved an RMSE of 0.4033 and an MAE of 0.3278 while the DMVST-Net achieved an RMSE of 0.4171 and an MAE of 0.3455.

Future work will include training these algorithms with various crime datasets, which are augmented with external data such as climate and socioeconomic data. Hyperparameter optimization of these algorithms using techniques, such as evolutionary computation, will also be explored.

Contents

List of Figures

List of Tables

Abbreviations

DL	**D**eep **L**earning
ANN	**A**rtificial **N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
RAM	**R**andom **A**ccess **M**emory
SVM	**S**upport **V**ector **M**achine
ST-ResNet	**S**patio-temporal **R**esidual **N**etwork
DMVST-Net	**D**eep **M**ulti **V**iew **S**patio-temporal **N**etwork
STD-Net	**S**patio-temporal **D**ynamic **N**etwork
FC	**F**ully **C**onnected
FCN	**F**ully **C**onvolutional **N**etwork
CuDNN	**CUDA D**eep **N**eural **N**etwork
NCCL	**NVIDIA C**ollective **C**ommunications **L**ibrary
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
TPR	**T**rue **P**ositve **R**ate
FPR	**F**alse **P**ositve **R**ate
AUROC	**A**rea **U**nder **R**eceiver **O**perator **C**haracteristic
RMSE	**R**oot **M**ean **S**quare **E**rror
MAE	**M**ean **A**bsolute **E**rror

Chapter 1

Introduction

1.1 Orientation of the Study

Crime is an ever-present issue in any society. Crime refers to any punishable conduct that the state authorities consider unlawful [11]. Crime is generally classified by criminologists into five primary types [12], namely: violent crime, property crime, white-collar crime, organized crime, and consensual crime. Fig. 1.1 illustrates the major categories of crime and the related examples.

FIGURE 1.1: Major Categories of Crime

Venezuela, Papua New Guinea, and South Africa have the highest crime rates in the world, according to data from 2021 [13]. Venezuela tops the list with the highest crime index of 84.36. The crime originates from corruption, a defective court system, and lax weapons control. Papua New Guinea has the second-highest crime rate in the world,

with a crime index of 80.04 attributable to violent crimes, which are largely driven by the country's social, economic, and political developments. South Africa's present crime index is 77.29. It is the third highest in the world. The causes of the crimes include high levels of poverty, inequality, social isolation, unemployment, and the normalizing of violence.

Crime prevention is one of the distinguishing qualities of a healthy society. One of the most well-known and visible aims of the police and other agencies includes predicting where and what crime will be committed, and crime prediction is essential since it speeds up the process of investigating crimes and decreases crime rates. Predicting crime events consumes a considerable amount of money and effort for police forces. Anticipating crime events helps the police to respond more proactively and distribute patrol resources more effectively [14].

Machine learning is revolutionizing how governments prevent, detect, and respond to crime. Predictive algorithms, such as the PredPol software, created in Santa Cruz [1], are becoming increasingly important to some law enforcement organizations. These algorithms use machine learning coupled with past crime data to forecast crime hotspots before crime happens. This can generally be referred to as *crime forecasting*. Crime forecasting opens up completely new avenues of crime prevention. Fig. 1.2 shows a map of Chicago with forecasted crime hotspots generated by Predpol; the potential hotspots are indicated by squares with red borders. For clarity, a hotspot on the bottom-right of the figure is magnified to clearly demonstrate the area indicated.

Forecasting using machine learning models can be subdivided in two main categories, relating to the nature of the intended target output: (i) classification which involves forecasting a discrete-valued output e.g. the type of crime committed in a specific area; and (ii) regression which involves forecasting a continuous-valued output e.g. the number of crime incidents that will be committed in a specific area. For ease of reference, the task of regression will henceforth be referred to as "prediction" in this thesis. To carry out classification, a machine learning model uses the training set to draw separating boundaries between the intended categories in a given feature space, which can then be used to classify a new unseen sample with the same features into one of the intended categories. In contrast, prediction requires for a line to be fitted to a given continuous-valued target output in a given feature space; the fit-line can then be used to predict the value of a new unseen sample with the same features.

Based on the above discussion, and to provide complete clarity, the following terms used

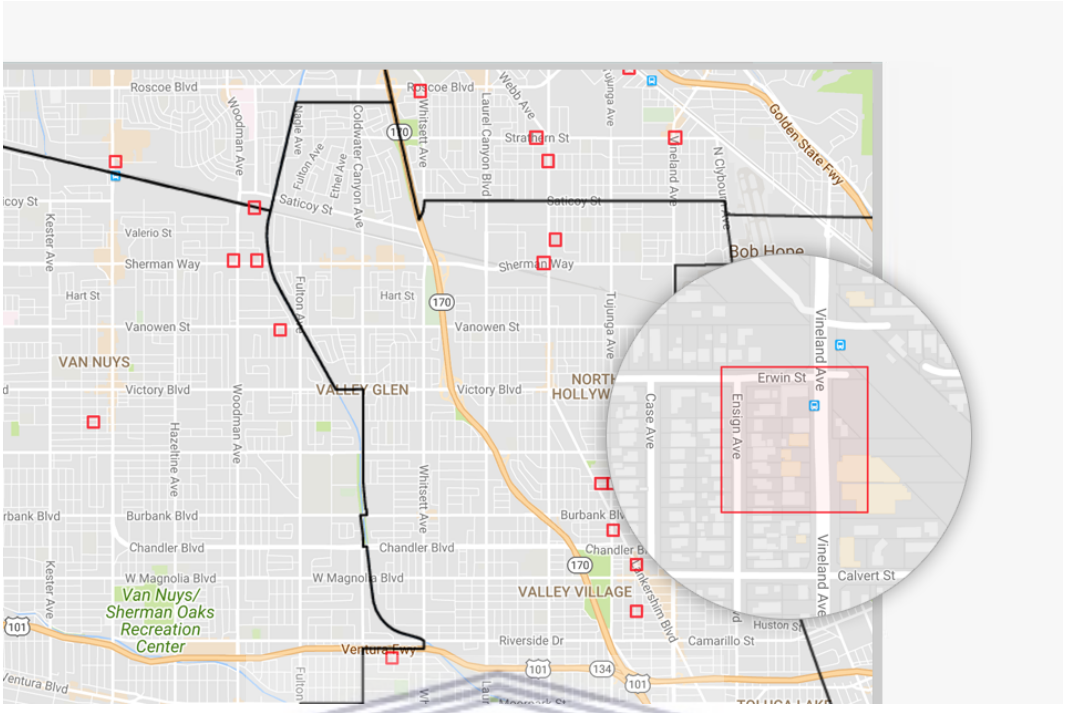


FIGURE 1.2: Crime hotspots (red squares) in Chicago predicted by the PredPol software using machine learning [1]

in this thesis are defined: "crime forecasting" refers to the general task of either classification or prediction of crime incidents, while "crime classification" refers to forecasting a discrete-valued output, e.g. the category of crime committed such as theft, assault, burglary while "crime prediction" refers to forecasting a continuous-valued output, e.g. the number of crime incidents in a given area, the number of thefts in a given time period etc.

A variety of statistical learning approaches such as Auto-Regressive Moving Average, and standard machine learning techniques have been used for crime forecasting. Of the machine learning techniques, Random Forest, Support Vector Machines (SVMs), and Classification and Regression Trees (CART) have been shown to achieve highly accurate results [15]. Deep learning, a subfield of machine learning that is currently the state-of-the-art, has demonstrated superior forecasting accuracy in several domains as compared to standard machine learning approaches. For example, the study in [16] showed that deep learning was able to out-perform SVMs, RF, and K Nearest Neighbours for crime forecasting.

In general, deep learning has shown great promise and state-of-the-art performance in a variety of classification and prediction scenarios, such as image classification, computer vision, speech recognition and financial forecasting. A deep learning model mainly takes the form of a deep neural network (DNN), which is a neural network with numerous hidden layers, usually also including one or more specialized structures for efficient and

effective spatial processing and/or temporal processing.

The specialized structures used in DNNs include: Recurrent Neural Networks (RNNs) which are specifically designed to model temporal dependencies in sequential data, and thereby make accurate predictions [17]; and Convolutional Neural Networks (CNNs), which are superior at mining spatial information and have demonstrated a high accuracy in a variety of fields involving data with spatial dimensions such as images [18]. CNNs are widely used in image processing/recognition and computer vision in general. Traditionally, CNNs were designed to process data with two spatial dimensions, such as images. These CNNs are therefore sometimes referred to as 2D CNNs. Subsequently, 3D CNNs have been proposed. 3D CNNs have the ability to process data with three spatial dimensions such as 3D images and Lidar images, or data with two spatial dimensions and a time dimension such as videos and weather data.

Deep learning algorithms have recently been applied to spatio-temporal data in domains such as taxi demand forecasting, city crowd flow forecasting, and crime forecasting [4, 19–21]. Spatio-temporal data has a time dimension as well as spatial dimensions. Crime data is inherently spatio-temporal in nature whereby a crime incident has spatial dimensions e.g. longitude and latitude where the crime took place, as well as a time dimension i.e. the time and date at which the crime took place or was reported. In domains involving spatio-temporal data, a deep learning model that has both RNN and a 2D CNN component is commonly used [22]; the 2D CNN is typically used to learn spatial characteristics of the data, whereas the RNN is used to learn temporal characteristics of the data [3].

RNNs and CNNs are potentially layered and arranged in many contiguous layers to enhance prediction performance in a given domain based on the spatial and/or temporal nature of the data in the domain. There is great flexibility in the arrangement (also referred to as "architecture") of DNNs. The architecture of a DNN has a direct influence on the capabilities of the DNN, in terms of: (i) predictive performance i.e. how accurately the model can classify or predict data; (ii) type of output i.e. whether the model is designed to achieve classification or prediction; and (iii) domain of application i.e. whether the model applies to static, spatial, temporal and/or spatio-temporal domains. For this reason, innovations in the architecture of DNNs is an active field of research, and has resulted in many unique DNN models that have shown promise in various domains.

The Spatio Temporal Residual Network (ST-ResNet) proposed in [3] has currently shown the most promise specifically for crime prediction according to several studies [19, 21, 22].

Two other spatio-temporal deep learning architectures i.e. the Spatio-temporal Dynamic Network (STD-Net) proposed very recently in [23] and the Deep Multi View Spatio-temporal Network (DMVST-Net) proposed in [4] are also promising architectures for prediction, and have shown greater promise than the ST-ResNet in several spatio-temporal domains unrelated to crime prediction [4, 5, 21]. To the best of our knowledge, these techniques have not been applied to crime prediction, and it is therefore desirable to investigate the application of these approaches to crime prediction. Given that these architectures are specifically designed to cater for spatio-temporal data, they are collectively referred to as "spatio-temporal deep learning" techniques in this thesis. Comparing the three spatio-temporal deep learning techniques is therefore one area of focus of this research.

A second area of focus of this research pertains to crime classification. 3D CNNs were initially proposed for action classification by Ji et al. [2] and were shown to be more easily applicable to the classification of spatio-temporal data, in this case action sequences, than 2D CNNs [24]. Later, Hara et al. [10] proposed merging 3D CNNs into residual neural networks (ResNets) to allow for a much deeper architecture by avoiding overfitting on numerous parameters. One of the models that they developed that showed excellent classification promise was the 3D ResNet-18 (or 3D ResNet in short). Given that these architectures i.e. the 3D CNN and 3D ResNet both make use of 3D CNNs to provide spatio-temporal processing capabilities, they are collectively referred to as "3D deep learning" techniques in this thesis. As such, the second area of focus of this research is to compare 3D deep learning techniques as applied to crime classification.

The Chicago crime dataset will be used in this research [7]. The dataset is constantly updated with crimes committed in Chicago in the United States dating back to 2001. It contains more than 7 million records which include incident information such as location, type, description, date, among others. A total of 32 types of crime are represented and recorded in the dataset. It has been used in a good deal of crime forecasting studies [16, 25–27].

For the task of comparing spatio-temporal deep learning techniques towards crime prediction, the aim is to use the Chicago dataset to train these techniques to predict the number of times a certain crime type will happen in a given area on a training subset of the data, and subsequently evaluate the techniques on a testing subset of the data.

For the task of comparing 3D deep learning techniques towards crime classification, the aim is to use the Chicago dataset to train these techniques to classify whether a

given crime type has taken place at a given location at a given time on a training subset of the data, and subsequently evaluate the techniques on a testing subset of the data.

As part of the classification process, a spatial grid of a given resolution will be used to subdivide the Chicago area into a given number of sub-areas; the goal of classification will then be to predict whether a given crime type will take place in a given sub-area i.e. grid cell. The resolution of the spatial grid, henceforth referred to as "spatial resolution", will likely directly affect the classification accuracy of the 3D deep learning methods used and will be investigated as well.

1.2 Problem Statement

The ST-ResNet is currently the most advanced deep learning technique for crime prediction, according to the findings in [19, 21, 22]. Newer spatio-temporal deep learning methods such as STD-Net and DMVST-Net are available and that have not been applied to crime prediction to the knowledge of the researcher. It is desirable to investigate and compare the application of these techniques to crime prediction.

In terms of crime classification, it is also desirable to investigate and compare the application of 3D CNNs and 3D ResNets to crime classification. As part of this objective, crime maps will be subdivided into a spatial grid using a given spatial resolution. The goal of classification will then be to determine whether a given crime type takes place in each grid cell. The effect of the spatial resolution on classification accuracy needs to be investigated.

1.3 Research Question

Based on the discussion in the Section 1.1, the following over-arching research question can be formulated: "How effective are 3D and spatio-temporal deep learning techniques for crime forecasting?" This main research question can be broken into the following sub-questions:

1. Crime classification:
    1.1. How effective are 3D deep learning techniques for crime classification on the Chicago crime dataset?
    1.2. What is the effect of spatial resolution on the performance of 3D deep learning algorithms for crime classification?

2. Crime prediction:

    2.1. How effective are spatio-temporal deep learning techniques for crime prediction on the Chicago crime dataset?

1.4 Research Objectives

The following objectives are set out in which, once met, will lead to answers to each of the research sub-questions in the previous section, which in turn will finally help provide an answer to the main question:

1. Carry out a literature review to explore various applications of statistical learning, traditional machine learning,3D and spatio-temporal deep learning techniques, with a focus on the reported effectiveness of the techniques in each application. This will demonstrate the effectiveness of the deep learning techniques over statistical and machine learning techniques, as well as providing a context for various applications of these deep learning techniques in the literature.
2. Implement the selected 3D deep learning techniques.
3. Implement the selected spatio-temporal deep learning techniques.
4. Compare the effectiveness of the 3D deep learning techniques implemented in the previous objective towards crime classification on the Chicago crime dataset. This will provide an answer to research sub-question 1.1.
5. Further to the previous objective, analyze and determine the effect of spatial grid resolution on 3D deep learning techniques towards crime classification. This will provide an answer to research sub-question 1.2.
6. Compare the effectiveness of the spatio-temporal deep learning techniques implemented in the previous objective towards crime classification on the Chicago crime dataset. This will provide an answer to the remaining research sub-question 2.1.

Having met all objectives and answered all research sub-questions, an answer will then be provided for the main research question set out in the previous section.

1.5 Contributions

This thesis has made the following important contributions to the field of crime forecasting:

1. It has applied two new spatio-temporal deep learning techniques, namely the DMVST-Net and STD-Net, to the task of crime prediction, which has not been carried out before.

2. Furthermore, these two techniques were compared with a spatio-temporal deep learning technique i.e. ST-ResNet which has been shown to provide state-of-the-art performance for crime prediction. The results showed that the STD-Net out-performs the ST-ResNet, and in so doing, the state-of-the-art in this context has been re-defined in this thesis.

3. The thesis has applied two 3D deep learning techniques, namely 3D CNN and 3D ResNet, to the task of crime classification, which has not been attempted before.

4. The 3D deep learning techniques were both shown to be highly effective for the task of crime classification. The 3D ResNet was further shown to out-perform the 3D CNN.

5. The effect of the spatial resolution size was investigated, and it was established that a smaller resolution size consistently out-performs larger sizes.

1.6 Limitations

The research will not cover criminal profiling, and does not use any information about individuals, or populations and their characteristics, as this information has restricted access by governments.

1.7 Dissertation Outline

The remainder of the thesis is organised as follows.

Chapter 2: *Literature Review.* This chapter reviews relevant research on general forecasting and, where possible, crime forecasting using statistical learning, conventional machine learning, and deep learning methods. In so doing, it aims to explore the effectiveness of each set of techniques. To this end, for each study considered, the research methods including the forecasting technique(s), datasets and assessment measures used, as well as the resulting findings are detailed and analysed. This will help demonstrate the relevance of 3D and spatio-temporal techniques towards meeting research objective 1.

Chapter 3: *3D and Spatio-Temporal Deep Learning.* This chapter provides a detailed description of the selected 3D and spatio-temporal deep learning algorithms implemented and compared in this research. This theoretical background serves as an underpinning

for the implementation of these techniques, in line with objectives 2 and 3, and the experiments subsequently carried out and described in the chapters that follow.

Chapter 4: *Experimental Setup.* This chapter provides details of the experimental setup in terms of the programming language used, the deep learning framework and API, the platform on which the experiments were carried out, the crime dataset used, the data preprocessing steps and the algorithm optimization carried out. In so doing, objectives 2 and 3 are partially met.

Chapter 5: *3D Deep Learning for Crime Classification.* This chapter evaluates the effectiveness of the selected 3D deep learning methods, namely 3D ResNet and 3D CNN, for crime classification. The chapter details the experiment carried out, including experimental design, data preprocessing, algorithm configuration, and evaluation metrics utilized. In so doing, objective 2 is fully met. Also, an analysis of the experiment is carried out, thereby meeting objective 4 and 5. Therefore, the chapter provides an answer to research sub-questions 1.1. and 1.2.

Chapter 6: *Spatio-temporal Deep Learning Techniques for Crime Prediction.* This chapter is similar in structure to Chapter 5. The chapter evaluates the effectiveness of the selected spatio-temporal deep learning methods, namely ST-ResNet, DMVST-Net, and STD-Net, for crime prediction. As with Chapter 5, a detailed description of the experiment carried out, including experimental design, data preprocessing, algorithm configuration and evaluation metrics utilized, is provided. This ensures that objective 3 is fully met. A detailed analysis of the experiment is then carried out, which ensures that objective 6 is met. As such, the chapter provides an answer to the research sub-question 2.1.

Chapter 7: *Conclusion and Future Work.* This chapter summarizes the findings from the experiments conducted and provides an answer to the main research question based on answers to the research sub-questions obtained previously. The chapter also summarizes the contributions made by this research and closes with recommendations for further research.

Chapter 2

Literature Review

This chapter presents prominent research studies that have used statistical learning, traditional machine learning, and the deep learning techniques under consideration in this research i.e. the 3D and spatio-temporal deep learning techniques specified in the previous chapter.

Where possible and relevant, the focus will be placed on applications of these techniques to crime forecasting, but where studies on crime forecasting are limited, other relevant applications involving spatio-temporal data may also be discussed, as necessary e.g. human action recognition, crowd flow prediction, and taxi demand prediction, among others, which are all examples of applications involving spatio-temporal data.

The chapter's structure is as follows: Section 2.1 analyses the studies that focused on relevant applications of statistical models and statistical learning methods; Section 2.2 investigates relevant studies comparing conventional machine learning methods such as Random Forest and Naïve Bayes. Section 2.3 presents studies related to the 3D deep learning techniques selected in this research i.e. 3D CNNs and 3D ResNets; and Section 2.4 details relevant studies that used the spatio-temporal deep learning techniques selected in this research i.e. ST-ResNet, STD-Net and DMVST-Net; finally, the chapter concludes in Section 2.6.

2.1 Statistical Learning Techniques

Li et al. 2018 [28] analysed and predicted urban crime in China on a spatio-temporal scale. They acquired training data from a city's Public Security Bureau. Each entry in the dataset included the following attributes: a record's unique ID; the time the crime occurred; further details such as the name of the person and property. The dataset included 2,708 records that were collected between August 2015 and March 2018. Their

preprocessing methods included the definition of entities with particular meanings derived from the Chinese text.

The entities included location, institution names, criminal occurrences, the goods involved, and monetary values. The Named Entity Recognition (NER) resolved errors in the structured data during its extraction [28]. For crime prediction, they examined the Auto-Regressive Integrated Moving Average (ARIMA), the Global Model (GM), the Pooled Model (PM), and the Hierarchical Model (HM). The rate of error analysis of the prediction results from the ARIMA model was lower than 5% compared to the other three models.

Kumar et al. [29] predicted the yearly crime rate in India using Time Series Models such as ARIMA and Exponential Smoothing. The tests utilized data from India's National Crime Record Bureau spanning the years 1953 to 2013. For time series modelling, the statistical package R was utilized. They used the Box-Jenkins technique for time series modelling. The technique had three phases: model selection, testing, and forecasting. They used exponential smoothing, a technique that is based on current results having a higher weight than older values, which diminish as the observation ages. Using the Mean Absolute Error (MAE), Mean Absolute Squared Error (MASE), and Mean Absolute Percentage Error (MAPE) metrics, the ARIMA model compared to the Holt Linear model.

Alves et al. [30] suggested a random forest regressor to predict crime and quantify the impact of urban variables on homicide in order to understand and predict crime. They discovered that deducing causality from data was difficult. Many linear models predicted crime based connections between crime and urban indicators. However, because of the non-Gaussian distributions and multicollinearity in urban indicators, it was difficult to determine the impact of certain urban indicators on crime.

Alves et al. [30] discovered that machine learning ensemble-based methods were effective for predicting crime. They used a dataset from DATASUS which is the Department of Informatics and Public Health Systems in Brazil. Based on the relationship of the dataset to crime, they chose potential crime prediction factors from the dataset. The prediction factors were child labour, the old population, the female population, the gross domestic product (GDP), illiteracy, family income, the male population, the population, sanitation, and unemployment. In their study, random forest algorithms had two main parameters that controlled the trade-off between bias and variance errors: the number of trees in the forest and the maximum tree-depth. These parameters controlled the trade-off between bias and variance errors in the forest. Stratified k-fold cross-validation

with k set to 10, was used to estimate the validation curves for a range of values of the parameters number of trees and maximum depth, in order to identify the set of parameters that prevented over- and under-fitting of the data.

Agarwal et al. [31] conducted a study on statistical models for crime prediction They utilized crime statistics from Indian states between the years 2001 and 2013 for their research purpose. They used crime statistics from 2001 to 2011 to forecast crimes in 2012 and 2013. There was a comparison between projected values and real crime data from 2012 and 2013. The NCRB website and Open Government Data (OGD) provided the crime dataset. The information acquired from these sites was in Microsoft Excel format, which was then converted to CSV format. The dataset included crime information for districts in India between 2001 and 2013. The crime statistics included many areas such as the number of murders, rapes, thefts, and so on, as well as the total IPC Crimes for each district in India.

The effectiveness of the three statistical models for crime prediction was evalauated. These models are: Weighted Moving Average (WMA), Functional Coefficient Regression (FCR), and Arithmetic-Geometric Progression (AGP). They found that the AGP was most suited for predicting crime, since it achieved the highest accuracy for the dataset of Indian districts.

2.2 Traditional Machine Learning Techniques

Almaw et al. [32] suggested analysing and predicting crime data using ensemble techniques. They developed a model for crime prediction using N-ensemble learning. There are two types of N-ensemble learning techniques: one-ensemble and three-ensemble learning. Denver open data site provided the dataset. The data collected had 19 characteristics and contained 127,799 records during a two-year period. Preprocessing of the data included identifying desirable characteristics, verifying missing numbers, finding outliers, and removing noise from the crime data.

The following methods were compared for classifying data: Naïve Bayes, J48, Random Tree, Bagging, Bagging plus Random Tree, and Stacking (J48 with Random Tree). Accuracy and F1 score evaluated the models' performance. The study provided a list of the outcomes of the experiment for each algorithm. Their research found that the Random Tree outperforms the Naïve Bayes classifier with an accuracy of 82.02%, while Naïve Bayes classifier achieved an accuracy of 53.64%. The J48 achieved an accuracy of 76.8926%. They utilized a method called classifier selection by accuracy to identify

classifiers that were merged to achieve optimal performance. The Naïve Bayes classifier performs poorly in comparison to the J48 and random tree classifiers, it was excluded from creating the ensemble.

Bappee et al. [33] presented a machine learning model for categorizing crime and predicting it using geographical data. The model tried to forecast the association between criminal activity and geographic areas. Their research focused on a crime dataset from Nova Scotia (NS). They concentrated on four distinct types of crime: those involving alcohol, violence, property crime, and motor vehicle crime. Furthermore, they predicted crime using two geographical features: geocoding and crime hotspots. The spatial data for open street map (OSM) was retrieved using a reverse geocoding technique. They formulated crime hotspots characteristics using a density-based clustering method. They compared Linear Regression (LR), Support Vector Machine (SVM), and Random Forest (RF) classifiers, as well as an Ensemble with all prior classifiers. The accuracy was measured using the Area Under Receiver Operator Characteristic (AUROC). Ten-fold cross validation was performed and the data into training and test sets. The Ensemble method achieved the highest accuracy and AUROC values.

Kim et al. [34] compared the KNN classifier to boosted Decision Trees for crime analysis. The study examined crime statistics from the Vancouver Police Department (VPD) spanning 15 years. The VPD crime dataset contained about 560 000 entries, which were collected between 2003 and 2018. They preprocessed the data in two ways. First, every neighbourhood was allocated a unique identification number and every crime, a unique identification number. A binary number was assigned to the neighbourhood and the day of the week. A "1" was recorded when a specific crime occurred in the neighbourhood on that day, and "0" was recorded if there was no crime.

Identical methods and settings were used in all data preparation processes, as well as the identical validation procedures for both data preprocessing processes. They used a 5-fold cross validation procedure to assess the performance of the classifiers. Although the KNN model achieved an accuracy of 40.1% while training for 2209 seconds on the data preprocessed by the first approach, the model achieved an accuracy of 39.9% while training for 101.73 seconds on the data preprocessed by the second approach, which is a significant improvement over the first approach. A boost decision tree obtained an accuracy of 41.9% on the data preprocessed by the first approach after training for 906.63 seconds, whereas a boost decision tree achieved an accuracy of 43.2% after training for 459.26 seconds on the data preprocessed by the second method [34].

Vural et al. [35] proposed a Naïve Bayes method for categorizing criminal activity. They used crime prediction to identify the criminal who was most likely to commit a certain crime incident using the history of crime events together with incident-level crime data. They used characteristics such as the date and place of the event, the crime type, the criminal identification (ID) number, and the names of associates in the incident-level crime data. Actual crime data could not be obtained due to confidentiality laws and regulations, as a result they created a synthetic dataset using the latest techniques to simulate the dataset they needed. They cross validated the suggested system. The experiment used a comparison between Naïve Bayes classifier and a Decision Tree. The Naïve Bayes had an accuracy of 81.66%, the Naïve Bayes algorithm produced the best results.

Khan et al. [36] developed a system for predictive policing on counterfeit street crime such as mobile theft and snatching, as well as simulated outcomes of machine learning models using various algorithm methods and their performance comparison. They gathered data from government databases from the years 2005 to 2013. Their data preprocessing procedures included various steps such as identifying potential and unusable attributes, replacing and filling in missing information in records with a default mean, model-derived or global constant values, removing records with incomplete information, grouping related attributes to obtain more meaningful information, and so on. Google Maps collected coordinates with an approximate error of 20 meters in order to locate, monitor, and map crimes. They also utilized K-means clustering and the Naïve Bayes classifier to predict crimes.

Yadav et al. [37] suggested utilizing the Classification and Regression Trees (CART) method to predict the resolution that was provided for crimes that happened in San Francisco between 2003 and 2015. The offences occurred between 2003 and 2015. They obtained information in this dataset from the San Francisco Police Department's Crime Incident Reporting System. The San Francisco crime dataset included both numerical and nominal information. They performed data preparation in order to convert nominal data into numeric data, which was accomplished via the use of a classification method. The study revealed 57 occurrences of redundant or missing data out of a total of 878049 instances.

In addition, Yadav et al. [37] deleted the 57 occurrences from the dataset. When the timestamp column included the date, year, and time of each occurrence of a crime, it was separated into five distinct features as part of the preprocessing phase. These five features were as follows: date, month, year, hour, and minute. They conducted a comparison of the CART methods with the Multi-Layer Preceptron (MLP), the KNN, and

the Gaussian Naïve Bayes procedures. The accuracy and the area under the receiver operating characteristic curve (AUROC) were the two measures that were used to assess the quality of the classifiers. The CART method produced the best accuracy and AUROC.

Matijosaitiene et al. [38] used machine learning techniques to predict motor vehicle theft in Manhattan and New York. They used time series analysis, hotspot analysis, linear regression, elastic-net, support vector machines with radial and linear kernels, decision trees, bagged CART, Random Forest (RF), and stochastic gradient boosting. The dataset was obtained from the New York Open data NYPD Complaint Map," which included 25 variables, 478 805 samples. Theft from a motor vehicle was categorised as larceny. It included both grand and petty larceny, as per New York Penal Law. Comparing thefts from motor cars in Manhattan to other crimes, the annual proportion of theft from motor vehicles throughout Manhattan was less than 6%, with 99.5% of all them occurring on the street. Moreover, they discovered that parking lots, both public and private, accounted for 0.496% of all instances, with the remainder of urban areas accounting for just 0.1%.

In the crime dataset that they used, each committed crime had the following attributes: latitude and longitude, crime type, the date, and time. Police records of the reported crime contained the date and time; circumstances of the crime; address; description of the premises; and other variables. The level of granularity in criminal classification was very fine. In the case of theft from a motor vehicle, there are many classifications: grand larceny from vehicle, petty larceny from vehicle, theft of vehicle accessories, and so on. They did not consider fine granularity, and as a result, they overlooked minor crimes, such as theft of a car accessory or other comparable crimes [38]. The data was grouped by crime categories into larger groupings, such as theft from a motor vehicle, burglary, and so on, using only crime data from the years 2015, 2016, and 2017. They considered crime data from the last two to three years sufficient for conducting crime analytics and generating predictions/forecasts, particularly when the urban planning and development context related to crime [38].

In addition, Matijosaitiene et al. [38] used the R programming language to aggregate the new variable "day of the week". Following that, they conducted data reduction to ensure that only the variables of interest such as: the unique ID, the crime category, the date, and time, the day of the week, and the latitude and longitude. They used time series analysis to determine the pattern in the number of thefts that occurred throughout the day and night. In order to detect geographic patterns in thefts, they utilized ArcGIS with Python for the hot spot analysis (Getis-Ord Gi) in order to find hot spots. This

was accomplished using a Gi-statistic, also known as a *z-score*, for each studied feature or independent variable in the dataset that was imported into ArcGIS software.

Additionally, in the context of the neighbouring features, the computed Gi-statistic uncovers spatial clusters of high (statistically significant positive large *z-score*) or low (statistically significant negative small *z-score* values), as well as statistically significant positive small *z-score* values [38]. They cross validated the dataset repeatedly using 10-fold cross validation in order to choose the best prediction model. They also performed three repetitions, which resulted in a more reliable estimate of the models. R-squared error (RMSE) and root mean squared error (RMSE)(R^2) methods were used to compare the models. The Stochastic Gradient Boosting technique achieved the best performance with RMSE of 0.610 and R^2 of 0.619.

Shi et al. [39] carried out a study on predicting property-related crime using a random forest method based on genetic algorithm optimization. They utilized a significant amount of official data and 75-dimensional characteristics to create large-scale data resources that influence property-related crime. An algorithm known as GA-RF was developed in order to establish a warning task for property-related crime. They merged bagging and random subspace methods in the random forest algorithm. In the process, they created a sophisticated classification method based on the integration of decision trees.

They classified the variables influencing property-related crime as economic, transportation, factors influencing property-related crime population, area, energy, and facility elements, with a total of 75 possible environmental indicators. As the study object, they selected 331 streets and towns in a city [39]. They collected environmental variables linked to the property-related crime of embezzlement via open and professional channels. The data collected included 1,277,814 instances of property-related crime with 75 different feature parameters.

At the same time, the data distribution divided the areas into six levels: highest, higher, high, low, lower, and lowest. They proposed the Genetic Algorithm-Random Forest (GA-RF) algorithm for predicting the trend of property-related crime, using the genetic algorithm to search the decision tree of random forest and generate the most beneficial decision tree combination, to promote the precision of integrated classification. These decision tree knots formed the new integrated classifier. The GA-RF algorithms achieved the best results in terms of accuracy, precision, recall, and F1 score when compared with the following algorithms: RF, XGBoost, SVM bagging, NB, ADABoost, and PAR-RF algorithms.

Alparslan et al. [40] used a combination of unsupervised and supervised learning techniques to predict crime at specific locations and times. They used a crime dataset with about 1.3 million samples. They utilized the 80/20 training and testing sets, from which they received about 838860 samples for training and, 262030 samples for testing. The years covered by the data collection spanned from 2006 to 2015. They used K-Means technique to create yearly clusters of the crime incidents, then stacked the cluster centres. Thereafter, they computed the Euclidean distance between each crime location and the cluster centres and added the distance of the features then trained the KNN, Naïve Bayes, Decision Tree, Random Forest, Logistic Regression, SVM, and MLP on the preprocessed data. For evaluating each model's performance, they used accuracy. The Random Forest algorithm achieved the highest accuracy.

Gupta et al. [41] used data mining and machine learning techniques in the study to predict crime patterns. They got their criminal data from India's National Crime Records Bureau which was collected over two decades. They analysed crime patterns and trends using time series analysis. They predicted future crime rates using machine learning methods. The data visualized, and charted time series using the RapidMiner tool. They compared Support Vector Regressor (SVR), the Decision Tree Regressor, and the Random Forest Regressor. The R^2 metric evaluated the models' performance. The SVR outperformed the other models.

Zaidi et al. [15] used a criminal classification method in crime predictions. They used the Cross-Industry Process for Data Mining (CRISP-DM) approach, which allowed them to repeat stages until they achieved an acceptable result. They used data comprehension, data preparation, modelling, and assessments. They conducted the experiment in collaboration with the Waikato Environment for Knowledge Analysis (WEKA). They used the University of California at Irvine crime and communities dataset, which was acquired from the UCI Machine Learning Research Repository. There were 147 characteristics in the dataset, and there were 2,216 criminal instances.

Zaidi et al. [15] chose 12 characteristics that were most relevant to the data and eliminated rows that had missing values throughout the data preparation phase. They also developed a new characteristic, dubbed "Crime Category," which served as the nominal feature for the output of the forecast. "Crime Category" was limited to just three levels of severity, which were labelled as "Low", "Medium", and "High", respectively. They evaluated the accuracy, precision, recall, and F1 score measures of the Random Forest against those of the SVM to see which was superior. In their study, the Random Forest algorithm produced the best results.

Hossain et al. [42] experimented on crime prediction utilizing spatio-temporal data. The experiment was performed using a crime dataset acquired from the open dataset of the San Francisco Police Department (SFPD). The collection contained statistics on crimes that happened in San Francisco between 1/1/2003 and 5/13/2015. The dataset contained an 8,780,049-row CSV file. They classified the dataset into characteristics such as date, time, and location. The incident's criminal classification was the target designation. The target label correlated with characteristics such as the crime description and resolution.

The San Francisco crime dataset contained 39 distinct kinds of offences which were referred to as classes. Hossain et al. [42] discovered that few crimes occur on a regular basis, while others are very uncommon. Larceny or stealing were the most prevalent crime, occurring 174900 times, while trespassing was the least prevalent crime, occurring six times. They discovered that 14 crime classes were committed more than 10,000 times, whereas 14 crime classes were committed fewer than 2,000 times. This demonstrated the unequal allocation of courses.

Hossain et al. [42] preprocessed the dataset using the Python package *scikit-learn* (sklearn). Certain properties in the CSV files included both text and numeric values. To enable the usage of machine learning on this dataset, they transformed the text characteristics to numeric values. To convert strings to numeric values, they used Python module *NumPy*. They also used string data types for attributes such as "Day", "Category", and "Address" columns. They converted text input to numeric data using the sklearn preprocessing tool.

After sorting items in ascending alphabetical order, the preprocessing program assigned an integer value to each unique item [42]. A Date-time attribute is also a string data type that may be transformed to a date-time object, from which four more properties can be obtained: "Hour", "Date", "Month", and "Year". To prevent overfitting and achieve more realistic accuracy, they divided the data into two parts: testing and training. The training set included all elements in addition to the goal label. The testing dataset included the characteristics used to predict the target label using a machine learning model.

Additionally, Hossain et al. [42] utilized *scikit-learn* model selection package, which included a class called "test train split", which divided the original dataset into a testing and training set. They utilized the default setting for the size of the test dataset, which was 25% of the original dataset. They further compared the accuracy of Decision Tree, KNN, Adaboost, and Random Forest. They used the *scikit-learn* package to balance the

dataset, oversampling and under sampling methods such as SMOTE and the *imblearn* function. The Random Forest method produced the best results, with an accuracy of 73.89% while oversampling the data and 99.16% when undersampling the data.

Kumar et al. [43] conducted an experiment using a KNN algorithm to predict crime. Their model tried to predict the type of crime that will occur, when, where, and at what time the crime occurred. Preprocessing of the crime data included eliminating any null values and superfluous columns. The dataset used was a variation of the original, which was acquired via scraping the police website of Indore, Madhya Pradesh. It was processed several times, and they omitted information such as the police station's name and location, the station's number, the complainant's name and address, and the defendant's name and address. The Extra Trees Classifier function determined the significance of features, which assisted them to ignore superfluous characteristics. The ensemble learning method used an Extra Trees Classifier as anything that takes a given quantity of data and determined the relative significance of each feature individually. The completed dataset had four attributes: the hour, the day of the year, the city's longitude and latitude. They divided the dataset into 80% for training and 20% for testing. They used the KNN classifier for the experiment because it is a supervised machine learning technique which is effective for classification issues [43].

The KNN algorithm works by calculating the distance between a query and all the instances in the data, choosing the closest examples to the query, and then voting on the most frequent label. The technique is non-parametric, which means that it makes no assumptions about the distribution of the main data. To put it simply, the type of data determined the model's structure. They evaluated their model using the accuracy, MAE, and RMSE measures. Their KNN model had an accuracy of 0.9951, an MAE of 0.0064, and a root-mean-square error of 0.0802.

2.3 Conventional Deep Learning Techniques

Stec et al. [16] targeted at using deep neural networks to predict the following day's crime count in a fine-grain city division. They generated predictions using crime statistics from Chicago and Portland, as well as other datasets including weather, demographic data, and public transit. They used the city of Chicago's data portal to acquire the city's crime statistics. The crime statistics for Portland came from the National Institute of Justice's Real-Time Crime Forecasting. The criminal databases included about 6 million entries dated as far back as 2001. The collected crime dataset included many useful variables, including longitude, latitude, beat, and district. Their 'O' model divided the

crime counts into ten bins and predicted the most probable bin for each geographical area on a daily basis.

Stec et al. [16] used sophisticated neural network architectures, including modifications tailored to the geographical and temporal dimensions of the crime prediction issue. They examined four distinct network architectures: Feed Forward Neural Networks (FFN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Recurrent Convolutional Neural Networks (RNN+CNN). They evaluated the performance of the various models using accuracy and Mean Absolute Scaled Error (MASE). The RNN+CNN combination produced the best results, with MASE values of 0.74 and 0.77 for Chicago and Portland, respectively, and accuracy values of 75.6% and 65.3% for Chicago and Portland, respectively.

Seo et al. [44] used Partially Generative Neural Networks (PGNN) as an automated method for identifying and predicting gang-related crimes. They chose PGNN because of its capacity to categorize crimes correctly when there is complete and partial information. They utilized data from the Los Angeles Police Department (LAPD). The dataset included various kinds of crimes committed in 2014, 2015, and 2016. Each crime record included associated modus operandi codes (mocodes), which provided some identification of the crime's distinctive behaviours or characteristics. Seo et al. [44] used a mocode in LAPD crime statistics to identify gang-related crimes. Each crime in the dataset included a collection of categories, textual, and numerical attributes/features collected by police personnel or other analytical procedures. However, certain characteristics were not applied to some crimes, and police officers regarded these as "blank" entries. PGNN was compared with Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), and Neural networks (NN) in their experiment. They used cross validation to determine the best hyperparameters for each machine learning model. They also used AUROC as an assessment measure in the study. The PGNN achieved the best results.

Lim et al. [45] developed a criminal network hidden links prediction model using Deep Reinforcement learning (DRL). They utilized DRL methods to create a hidden or missing link prediction model for a criminal network based on Social Network Analysis (SNA) metrics. DRL methods were trained with relatively little datasets via self-simulation, and were used to the features of a criminal network with many missing or concealed edges. They tested the link prediction DRL model against the gradient boosting machine (GBM) model, both of which were based on a binary classification task.

To link predictions, Lim et al. [45] used the GBM model as the baseline model in contrast to the DRL model. They also utilized the UCINET cocaine smuggling dataset,

which included a dichotomous adjacency matrix of Hartford drug users. The UCINET cocaine smuggling dataset includes four distinct criminal network groupings, namely Juanes, Mambo, Jake, and Acero. The Area Under the Curve (AUC) was chosen as the assessment measure since it is unaffected by class imbalance and is a widely used method for assessing classifier performance. Lim et al. [45] split each criminal group dataset into training and test sets in a 75:25 ratio. Thereafter, they designed both the GBM and DRL link prediction models using the training data.

Their study further used a test set, to evaluate the final model in terms of the binary classification function's performance [45]. The SNA measurements utilized in the feature matrix design. They transferred the trained GBM and DRL link prediction models, the SNA link prediction metrics for the criminal network to a feature matrix. In the three criminal network datasets, the DRL link prediction model outperformed the GBM baseline model.

Lin et al. [46] presented a grid-based crime prediction model based on 84 different kinds of geographical features. They acquired the geographical information by combining crime (theft) statistics for Taoyuan City in Taiwan with the Google Places API. The Taiwan government's open data portal provided statistics for vehicle theft for Taoyuan City. They collected data from January 2015 and April 2018, during that time there were 220 criminal occurrences each month. They compared four machine learning algorithms: Random Decision Forest, Deep Neural Networks (DNN), Support Vector Machine, and K-Nearest Neighbor. Accuracy, precision, and recall methods were used as performance metrics. To validate the algorithm's accuracy, they computed the Mean Average (MA) F1 score, which was utilized as the performance criteria. Across the board, the DNN method exhibited the best performance.

Stalidis et al. [19] proved that techniques based on deep learning outperform the best-performing current conventional methods for crime classification and prediction. They evaluated the efficacy of various parameters in deep learning architectures and provided guidance on how to configure them for better performance in crime classification and ultimately crime prediction. They utilized five distinct datasets from Kaggle that included over a decade's worth of criminal incident records. These five datasets included criminal event records from the police departments of Seattle, Minneapolis, Philadelphia, San Francisco, and the District of Columbia.

Stalidis et al. [19] classified the data into 10 crime categories, including "Homicide", "Robbery", "Arson", "Vice", "Motor Vehicle", "Narcotics", "Assault", "Theft", "Burglary", and "Other", and assigned to each dataset. The "Other" category consisted

of crimes that did not fit into any other categories. They compared 10 algorithms: Combined Cleaning and Resampling (CCR-Boost), Spatio-temporal Residual Networks (ST-ResNet), Decision Trees, Naïve Bayes, Logit Boost, Random Forests, SVM, KNN, and a Multi-Layer Preceptron (MLP). To assess the model's performance, the F1 score, AUROC, Area Under the Precision Recall Curve (AUCPR), and Prediction Accuracy Index (PAI) were employed. According to their tests, the ST-ResNet exhibited the best performance.

Yi et al. [25] used a Neural Network-based Continuous Conditional Random Field (NN-CCRF) machine learning model and presented a method for fine-grained crime prediction. The neural network uses Long Short-Term Memory (LSTM) as the unitary potential, which is combined with a Stacked De-noising Auto-Encoder (SDAE) to learn spatial correlations between areas. Between 2013 and 2015, they gathered 1,072,208 crime records in Chicago and 1,417,083 in New York (NY) City. Each crime record correlated with a timestamp, a location, and a crime category.

To predict a fine-grained crime rate throughout the city, Yi et al. [25] classified crime data into N discrete areas, each of which is a 1 Km by 1 Km grid, resulting in 35 (7×9) and 63 (7×9) grids for Chicago and New York, respectively. To address such data sparsity issues, they grouped comparable crime categories into groups (crime against person or property). Linear Regression (LR), Long Short-Term Memory (LSTM), Conditional Random Field RNN (CRFasRNN), and NN-CCRF were compared. The Route Mean Square Error (RMSE) was used to assess each technique's performance. The NN-CCRF model exhibited the best performance.

Elluri et al. [47] presented models for smart policing that are based on machine learning. They used statistical analysis and machine learning algorithms to predict several crimes types in New York City using 2018 crime statistics. They sourced the dataset from the open data site for New York City. Each year, the crime dataset included approximately 220,000 entries. They integrated meteorological and temporal characteristics such as cloud cover, illumination, and time of day in order to determine the weather's relevance to crime statistics. They preprocessed the data by replacing zero or null values with a "NA" and then extracted the best feature subsets using forward and backward feature selection methods.

They collected 17 characteristics of crime, including meteorological and temporal, for algorithm training and testing following the feature section [47]. They trained and tested conventional algorithms using RapidMiner, a data science tool. They implemented various neural networks using a four-layer architecture consisting of an input layer with 17

input neurons based on the best 17 features, a CNN/RNN layer with 32 neurons based on the algorithm type, a fully connected layer with 256 neurons, and finally an output layer with two neurons for two classes. They used a 20% drop-out for the layer of deep neural networks and a 50% drop-out for the fully connected layer [47].

Lastly, the network was trained in 32-bit micro batches. They utilized binary cross entropy as the loss function for the networks. To improve the network, use the Adam optimizer. They used Python's Keras and TensorFlow libraries for deep learning. The model's performance was evaluated using the AUC and Cohen's Kappa criteria. They used 5-fold cross validation to train and evaluate the models. The SVM, Decision Tree, Neural Network, MLP, Logistic Regression, Random Forest, CNN, Simple RNN, LSTM, and Gated Recurrent Units (GRU) were all compared. They discovered that the LSTM model had the highest AUC and kappa values for prediction. Additionally, they observed that weather-related characteristics had a minimal impact on crime prediction and therefore removed meteorological datasets from the model performance assessment for crime prediction [47].

Wang et al. [48] used an ST-ResNet model for real-time crime predictions on an hourly timeframe. They examined all kinds of crime committed in Los Angeles during the last six months of 2015. There were 104,957 crimes in total. Due to the poor geographical and temporal regularity of the crime data, they conducted both spatial and temporal regularization of the data. Each crime record included information on the crime's start and finish timings, as well as its location. They referred to the time slot as the start time of each event. Geographically, these crimes occurred between the latitude and longitude intervals [33.3427°, 34.6837°] and [-118.8551°, -117.7157°], respectively. The geographical distribution is very diverse.

Wang et al. [48] further discovered that a sizable part of the region had a little crime, $[33.6927°, 34.3837°] \times [-118.7051°, -118.1157°]$. They created a 1616 lattice by partitioning a specific area. They also addressed the lack of spatial regularity in the data preparation phase by using a super resolution method utilizing cubic spline interpolation. In addition, they resolved by a factor of two in each dimension for computing efficiency and substantially enhanced spatial regularity by the cubic spline super resolution.

One advantage of this preprocessing was that it enhances the signal without sacrificing information related to the crime data. They experimented with two distinct deep learning structures: one with convolution layers and one without. In the second model, they used an ensemble of ResNet to learn the time series on each grid, without taking into account the transition of crimes across grids. The first model which recorded crime

dynamics using convolutional layers, was more dynamic. They integrated all characteristics in both networks with the crime data using a parametric-matrix based fusion method. Thereafter, they designed the models using Keras on top of Theano software. Their models included external variables such as weather and vacations. They adopted trend, periodic, and adjacent characteristics due to the periodic pattern and self-exciting property of crimes. These characteristics time intervals were indicated as: weekly, daily, and hourly, respectively. The RMSE was used to assess the models' accuracy. In the top 25 predictions, the ST-ResNet produced the best results.

Wang et al. [22] expanded on their tests on real-time crime prediction in Los Angeles using the ST-ResNet model and suggested several enhancements to the model's accuracy and performance on mobile devices. They utilized the Los Angeles crime dataset, which included information dating back to 2015, as well as external meteorological data obtained from Weather Underground. They used Root-Mean-Square Error (RMSE) as model assessment. They also compared ST-ResNet model with HA, KNN, and ARIMA models. ST-ResNet provided the best findings, with an inaccuracy rate of 0.659% on crime density.

Chun et al. [49] conducted a study to develop a deep learning crime prediction model. They used the DNN on the person's criminal charge history to predict the crime and its nature at an individual level. They utilized a dataset of criminal cases from 1997 to 2017. The dataset had six elements: "Person ID," "Crime Code," "Age", "Gender", "Race," and "Booking Date". The crime dataset included 16,841 distinct individuals, 63,133 arrest records, and 42 distinct crime categories. Their model predicted the amount of crime during a five-year period. They developed their crime prediction deep neural network using TensorFlow. To prevent overfitting, they also used L2 regularisation. For multi-label output, the network utilized a loss function of cross entropy, a learning rate of 0.005, gradient descent as an optimizer, and the softmax loss function. They used accuracy and F1 score measures to compare their model against the SVM. In the multi-class criminal classification exercise, their DNN outperformed the SVM method.

Duan et al. [50] tested deep Convolutional Neural Networks for spatio-temporal crime prediction. In an effort to use deep CNNs for automatically crime-referenced feature extraction, they suggested a new Spatio-temporal Crime Network (STCN). They also utilized the "311 crime dataset" from New York City, which included information from 2010 to 2015. This dataset included information on 10 million complaints. They excluded around 0.12 million complaint data because they lacked locations or were outside the sector limits. As a result, the final dataset included 9.75 million records. They

transformed the dataset input feature maps to 2D image-like arrays.

According to Duan et al. [50], the model captured low-level spatio-temporal relationships for criminal events by running these feature maps through a series of two convolution layers. To prevent the STCN network from overfitting, they created inception blocks and fractal blocks. They compared STCN's performance of three baseline models: SVM, Random Forest, and Shallow Fully Connected Neural Network (SFCNN). TensorFlow 1.01 and CUDA 8.0 were used to construct STCN and SFCNN. The learning parameters used a Gaussian distribution that included the mean and standard deviation (0, 0.02). They used the F1 score and AUC to assess the models. With an F1 score of 0.88 and an AUC of 0.92, their suggested that STCN model outweighed the baseline models.

2.4 3D Deep Learning Techniques

This section details studies that used 3D deep learning techniques, specifically 3D CNNs in Subsection 2.4.1 and 3D ResNet in Subsection 2.4.2.

2.4.1 3D Convolutional Neural Network

Ji et al. [2] developed 3D CNN for automatic human detection in surveillance footage. They utilized the TRECVID 2008 development dataset, which is composed of 49-hour video recordings shot at London Gatwick Airport. They concentrated on the recognition of three action kinds in their experiments (*CellToEar*, *ObjectPut*, and *Pointing*). Each action was categorised as one-against-rest, and additional negative samples were created from actions that did not fall into one of these three categories. During the experiment, they evaluated models using precision, recall, and AUC. In terms of performance, the 3D CNN outperformed 2D CNN in terms of accuracy, recall, and AUC.

Zunair et al. [24] used 3D CNNs for TB prediction. They created a 17-layer 3D CNN with four 3D convolutional layers, two of which had 64 filters, followed by 128 and 256 filters, all with a kernel size of $3 \times 3 \times 3$. Following each convolutional layer comes a max-pooling layer with a stride of 2 and ReLU activation, which is followed by a batch normalization layer. Their feature extraction block was essentially made up of four convolutional, max-pooling, and batch normalization modules. The feature extraction block's final output was flattened and sent to a fully connected layer with 512 neurons. They utilized a 60% effective dropout rate. The output was then routed to a dense layer of two neurons activated with softmax for the binary classification task. They employed a total of 10,658,498 learnable parameters.

The dataset was provided by ImageCLEF Tuberculosis 2019, and it included 335 3D CT chest-scans with annotation of high and low provided by a medical doctor, as well as lung segmentation masks, and clinically relevant metadata that included the following binary measures: disability, relapse, TB symptoms, comorbidity, bacillary, drug resistance, higher education, ex-prisoner, alcoholic, smoking. The dataset had 218 individual chest CT images for training, with the remaining 117 saved for final assessment on the ImageCLEF evaluation platform. The CT images each had a dimension of 512×512 pixels and the depth size ranges from around 50 to 400, and it stores raw voxel intensity in Hounsfield units (HU). The TB prediction was a binary classification problem. Area Under the ROC Curve and accuracy metrics were used to evaluate the models. They reported 73% AUC and binary classification accuracy of 67.5% on the test set, which outperformed other approaches which supported only image information [24].

2.4.2 3D Residual Network

Hara et al. [10] proposed that for action recognition training, 3D residual networks (with 18 layers) be used to learn spatio-temporal properties. The *ActivityNet* and *Kinetics* datasets were used. There were samples from 200 human action classes in the ActivityNet dataset, with an average of 137 untrimmed videos per class and 1.41 activity occurrences per video. The overall length of the video was 849 hours, and there were 28,108 activity occasions. The dataset was divided into three subsets at random: training, validation, and testing, with 50% of the data used for training and 25% for validation and testing. To train their 3D ResNet-18 network, they used the stochastic gradient descent (SGD) approach.

To achieve data augmentation, they randomly produced training samples from videos in the training data during training. They used uniform sampling to select the temporal positions of each sample. Around the selected temporal positions, 16 frame clips were created. The goal of their experiment was to train their 3D ResNet-18 network to the point where, post model training, it could distinguish actions in videos. To generate input clips, they used the sliding window method, in which each video is split into non-overlapped 16 frame clips. Each clip was cropped with the largest scale possible around a central spot. To detect activities in videos, they computed class probabilities for each clip using the training model and averaged them over all clips. The 3D ResNet-18 technique outperformed C3D and ImageNet in terms of classification accuracy.

Zhang et al. [20] suggested a 3D residual network for joint atrophy localisation and Alzheimer's disease diagnosis using self-attention. They acquired their training dataset through the website of the Alzheimer's Disease Neuroimaging Initiative (ADNI) [20].

Their algorithm outperformed 3D VGG-Net, 3D ResNet-18, and 3D ResNet 34 in terms of accuracy, sensitivity, specificity, and AUC.

Akintoye et al. [51] built on the work in [20] by introducing a hybrid parallelization method for increasing the efficiency of a self-attention 3D Residual network (3D ResAttNet) for diagnosing Alzheimer's Disease (AD) by using data from the ADNI website. They compared the 3D ResAttNet with 34 layers against seven state-of-the-art models in a parallel environment over several GPUs, including a 3D CNN. They used speedup, accuracy and training time metrics to evaluate the performance of the models. Their model produced the best classification results.

2.5 Spatio-temporal Deep Learning Techniques

This section details studies that used spatio-temporal deep learning techniques, specifically ST-ResNet in Subsection 2.5.1, DMVST-Net in Subsection 2.5.2 and STD-Net in Subsection 2.5.3.

2.5.1 ST-ResNet

Zhang et al. [3] implemented the ST-ResNet to the prediction of crowd flow at the city level. They used the trajectories and climatic data from Beijing taxicabs, as well as data from New York City bike trajectories. When normalizing the data, they utilized min-max normalization to scale the data, and one-hot coding to translate metadata day of the week, holidays, and weather conditions into a binary vector. The RMSE method was used to evaluate the model. They applied the *tanh* activation function to the final step in the ST-ResNet architecture.

They used Python libraries, TensorFlow and Theano frameworks for the experiments. There are residual units in all the convolutional layers of the network design. The first convolutional layer made use of 64 filters of different shapes and sizes of 3×3. The second convolutional layer used 2 filters of size 3×3. The batch size was 32. They chose 90% of the training data for training each model, and the remaining 10% was used as the validation set [3]. The validation set halted the training method for each model early based on the best validation score obtained from the validation set. When measured in terms of RMSE, the performance of ST-ResNet was compared to that of HA, ARIMA, Seasonal Autoregressive Moving Average (SARIMA), Vector Auto-Regression (VAR), Spatio-temporal Artificial Neural Network (ST-ANN), and Deep Spatio-temporal network (Deep ST). The ST-ResNet demonstrated the achieved the lowest RMSE.

2.5.2 DMVST-Net

Yao et al. [4] used the Deep Multi-View Spatial-Temporal Network (DMVST-Net) to analyse a massive dataset of online taxi requests obtained from Didi Chuxing, one of China's major online car-hailing providers. The dataset contained 1.8 million records and the features included temporal information, such as the average demand value over the previous four years, spatial information, such as the longitude and latitude of region centre intervals, meteorological information, such as weather conditions, and event information, such as holidays. They considered the size of each region and established it as follows: 9×9, which coincided to 6 km $\times$ 6 km grids. The spatial view had 3 layers which had 64 filters, each with a size of 3×3.

In their study, the sequential network had 8 layers, which is equal to 4 hours for the LSTM [4]. The graph embedding output dimension was set to 32. The semantic view's output dimension was set to 6. They employed the Sigmoid function as the activation function for the final prediction network's fully linked layer. ReLU was used to activate functionality in other fully connected layers. Batch normalization was used in the local CNN network. The batch size in the experiment was set to 64. The first 90% of the samples were selected for training each model, and the remaining 10% constituted the validation set for parameter tuning. They also used the early-stop technique in all of their tests. Early-stop round and maximum epoch were set to 10 and 100 in the experiment, according to their values in the study. Model evaluation is based on the mean average percentage error (MAPE) and the rooted mean square error (RMSE). The DMVST-Net model outperformed the following models: HA (Historical average), ARIMA, Linear regression, MLP, XGBoost, and ST-ResNet.

2.5.3 STD-Net

Ali et al. [5] developed a deep hybrid spatio-temporal neural network (DHSTNet), which is made of recurrent and convolutional networks, to anticipate citywide traffic population flows by using Spatio-temporal patterns. This network consisted of recurrent and convolutional networks. They used two large-scale world datasets, namely TaxiBj and BikeNYC. They used Min-Max normalization approach to standardize the data in these datasets. Each dataset contained information on the weather and flow trajectories.

During the preprocessing step for the TaxiBJ dataset, they separated the entire city map into a number of 32×32 rectangular sections and set the duration of each phase to 30 minutes. Similarly, they partitioned the entire city map into 8×16 rectangular

sections for the BikeNYC dataset and set the duration of each period to 1 hour.

The Min-Max normalization approach standardized the data in these datasets. During the evaluation process, the result was re-normalized in order to provide a normal value, which was then compared to the ground truth. During the preprocessing step, the one-hot coding technique converted the external features to metadata (such as DayOfWeek, Weekend/Weekday), holidays, and weather conditions into a binary vector in the external features. Normalization between min and max to scale the wind speed and temperature between min and max [0, 1]. They also applied MinMax normalization to the existing baselines before comparing them with the DHSTNet [5]. The models applied Keras 2.1.0 and TensorFlow 1.2.1 frameworks.

In the experiment, the batch size was 64. The learning rate was 0.001. They added a drop out layer with a drop out rate of 0.25 drop-out rate in order to alleviate the overfitting problem. RMSE and MAPE methodologies were used to assess the model's overall performance and accuracy. They further evaluated the performance of their model to the following models HA, ARIMA, LinUOTD, XGBoost, MLP, ConvLSTM, STDN, and ST-ResNet. They found that their proposed model outperformed all the baseline models.

Zhang et al. [21] presented a model for attention-based supply and demand in autonomous vehicles. They analyzed data from a Chinese online car-hailing service, and they analysed it. For the city of Beijing, the dataset comprised taxi order data, taxi trajectory data, and weather data. The taxi order data included the request start location, request end location, request start time, and request finish time, as well as the request start and end times. For every 2 seconds, the taxi data trajectory included the following information: position, speed, and direction. For every 30 minutes, the weather data featured information about the weather such as rainy, sunny, and cloudy conditions. They have divided the city into sections 20 km $\times$ 10 km regions.

In addition, each time interval is 30 minutes long. They grouped the information on traffic speed, volume, journal distance, demand, and supply for every 30 minutes. Using the 30-minute time period, they further divided the dataset into training and testing datasets in a 5:3 ratio. They predicted the outcome of the next 30 minutes based on the preceding 2.5 hours. If the data covered several days and historical data for prediction was present, they limited the duration to 5 days. All sizes of convolution kernels were limited to 3×3. The size of each region considered was set as 5×5.

Zhang et al. [21] created 64 batches, and the learning rate was 0.001. The residual network consisted of four layers, whereas the completely linked network consisted of three

levels. They evaluated the model performance using the Mean Average Percentage Error (MAPE), Mean Absolute Error (MAE), and Rooted Mean Square Error (RMSE) approaches. They compared their model's performance to that of baseline models such as ARIMA, LSTM, ConvLSTM, Reduced-ConvLSTM, ST-ResNet, DMVST-Net, STDN, and Reduced-STDN. Among the baseline models, the STDN produced the best results, with a MAPE of 21.08%, a root-mean-square error of 0.1634, and an MAE of 0.1348.

Awan et al. [23] devised a hybrid neural network (STD-Net) to simulate dynamic spatio-temporal correlations for the prediction of urban traffic flows. They analysed traffic flow data from BikeNYC and TaxiBj. Their suggested STD-Net framework consisted of four primary components: temporal proximity, period volume, weekly volume, and external influences. They combined the usage of a hybrid spatial-temporal network (HSTN) to improve the prediction of urban population traffic flows, specifically inflow and outflow. They employed the CNN to model the spatial dependency around the areas in the STD-Net architecture.

Moreover, Awan et al. [23] used an LSTM network to simulate sequential temporal dependency, which resolved vanishing gradient and recurrent neural network problems. Their network's key components comprised a convolutional-LSTM (ConvLSTM) network that substituted the kernel size with a convolutional network. The STD-Net was designed to extract spatio-temporal dependencies in urban crowd flows by focusing on hidden features. This technique accounted for both spatial and temporal information contained in the data. The designed hybrid neural network emphasised on hidden features in order to capture both spatial and temporal information on urban crowd flows.

After loading numerous layers of the deep hybrid model, the result for the model's last layer was stored. They devised a framework for the deep hybrid neural network that separated the DNN into two sub-DNNs [23]. The first sub-DNN used CNN to capture spatial characteristics, while the second used an LSTM network to learn temporal features over time. After that, the three outputs were infused as Z_{fusion}. The Z_{fusion} was also infused with the the external branch as X_{ext} output. As the last step, both the external and the spatial features were infused together.

The activation function used in the last layer of the STD-Net architecture was the *tanh* activation function, which had a value between -1 and 1. To balance the data between -1 and 1, min-max normalization was applied. The metadata for external characteristics changed utilizing one-shot coding schemes such as weather, weekday, holiday, and weekend. Additionally, they normalised these external features using the Min-Max method to balance the temperature and wind and to keep them within the

range [0, 1]. Thereafter, they constructed their STD-Net model in Python on PyTorch using Keras 2.0.1 and TensorFlow 2.1.6. They used two convolutional layers to obtain a reduced file size. i.e., $(4 \times 8 \times 16)$ in all components. They reduced the drop out rate to 0.25 in order to avoid overfitting. They compared their proposed STD-Net model against ten industry-standard models, including HA, SARIMA, ARIMA, ST-ANN, VAR, DeepST, DeepST-CPTM, ST-ResNet, STDN, and MST3D. The root-mean-square error and the MAPE were used to analyse these models. Their models outperformed the baseline models by a factor of 5.34 and a factor of 17.4 respectively.

2.6 Conclusion

For the purposes of classification and prediction, this chapter detailed studies that used statistical and traditional machine learning approaches, as well as selected 3D and spatio-temporal deep learning techniques. Where possible, it also discussed datasets used, including crime datasets, data preprocessing procedures, and metrics for classification and prediction, including those applied specficially to crime forecasting. In addition, the chapter presented the performance of spatio-temporal deep learning in other domains, such as taxi demand prediction.

For crime prediction and other spatio-temporal domains, the experiments cited above revealed that deep learning techniques outperform both statistical learning and classical machine learning techniques, demonstrating superiority of deep learning over statistical and classical machine learning techniques.

Furthermore, it was shown that while the ST-ResNet was frequently used to forecast crime with promising results, and is the current state-of-the-art technique for crime prediction, two recently developed spatio-temporal deep learning techniques. Specifically, the STD-Net and the DMVST-Net significantly out-performed the ST-ResNet when applied to other domains with spatio-temporal data, such as taxi demand forecasting and crowd flow forecasting. This demonstrates the importance of comparing these newer techniques to the ST-ResNet in the domain of crime prediction, to establish the technique that is the best fit for the task.

As such, at this point that research objective 1 has been successfully met.

The next chapter provides a theoretical discussion on the 3D and spatio-temporal deep learning techniques dealt with in this chapter.

Chapter 3

3D and Spatio-Temporal Deep Learning

This chapter provides a detailed description of the selected 3D and spatio-temporal deep learning techniques. These techniques are implemented and compared in this research in the context of crime forecasting. Therefore, there is a need for a deep theoretical understanding of each of these techniques. This will serve as a foundation for the implementation of these techniques as a precursor to their comparison in subsequent chapters, in line with objectives 2 and 3. Section 3.1 describes 3D deep learning techniques, and Section 3.2 discusses spatio-temporal deep learning techniques. The chapter is concluded in Section 3.3.

3.1 3D Deep Learning

This section focuses on providing a theoretical description of the selected 3D deep learning techniques, namely 3D CNNs in Section 3.1.1 and 3D ResNet in Section 3.1.2.

3.1.1 3D CNN

A 3D convolution consists of moving a 3D kernel, with the cube joined together using several adjacent frames. The feature maps in the convolution layer link with different adjacent frames in the previous layer. Then the subsequent temporal information is collected in the next layer. A formal representation of the feature map resulting from a 3D convolution is given by:

$$v_{ij}^{xyz} = tanh\left(b_{ij} + \sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right) \tag{3.1}$$

where $(x,\ y,\ z)$ is the value of the position on the jth feature map in the ith layer; R_i represents the size of the 3D kernel along the temporal dimension, and w_{ijm}^{pqr} is the value

of the kernel at point $(p,\ q,\ r)$, linked to the mth feature map in the previous layer. Fig. 3.1 illustrates a 3D convolution.

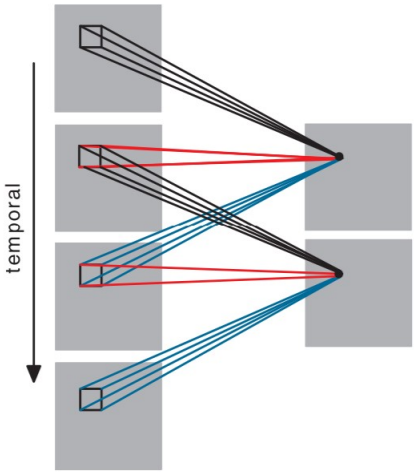


FIGURE 3.1: 3D convolution [2]

This 3D convolution is the basis of the development of 3D CNN architecture. Fig. 3.2 presents an overview of the 3D CNN architecture [2].

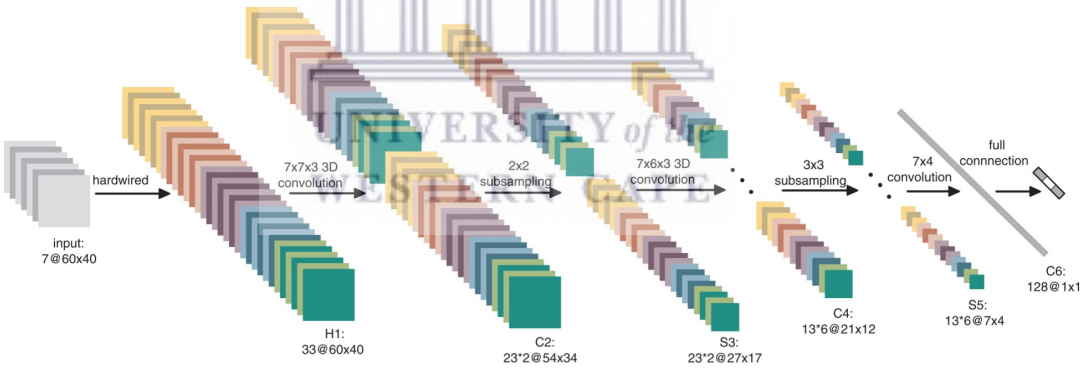


FIGURE 3.2: 3D CNN Architecture [2]

Fig. 3.2 shows that there are seven frames of size 60×40 centred on the current frame as inputs to the 3D CNN model. Multiple channels of information are generated from the input frames using a series of hard-wired kernels. As a result, the second layer consists of 33 feature maps divided into five distinct channels indicated by g, gradient-x, gradient-y, optflow-x, and optflow-y. The g channel contains pixel values of the seven input frames. The gradient-x and gradient-y channels include feature maps generated by calculating gradients in the horizontal and vertical directions on each of the seven input frames, respectively. The optflow-x and optflow-y channels contains the optical flow fields along the horizontal and vertical directions, calculated from adjacent input frames. This hard-wired layer encodes the prior knowledge about these features, and this approach results in excellent performance than random initialization. 3D convolutions

are applied with a kernel size of $7 \times 7 \times 3$ (7×7 on each of the five channels individually, and size 3 in the spatial dimension and size 3 in the temporal dimension). To expand the number of feature maps, each location layered distinct sets of convolutions, resulting in two distinct sets of feature maps in the $C2$ layer, each consisted of 23 feature maps.

The $S3$ layer in Fig. 3.2 indicates how subsampling was performed, then 2×2 subsampling applied to each of the feature maps in the $C2$ layer, which results in the same number of feature maps with a lower spatial resolution. The next convolution layer $C4$ applies convolution with a kernel size of $7 \times 6 \times 3$ on each of the five channels in each of the two sets of feature maps. Three convolutions, with different kernels used at each location to increase the number of feature maps, results in six unique sets of feature maps in the $C4$ layer, each containing 13 feature maps.

The next layer $S5$ applies 3×3 subsampling on each feature map in the $C4$ layer, which results in the same number of feature maps with a lower spatial resolution. At this point, the temporal dimension is very low (3 for g, gradient-x, gradient-y, and 2 for optflow-x and optflow-y). Therefore, this layer only performs convolutions. The size of the convolution kernel is 7×4 so that the sizes of the output feature maps reduces to 1×1.

The $C6$ layer consists of 128 feature maps of size 1×1, and each of them links to all 78 feature maps in the $S5$ layer. The seven input frames transform into a single output frame using several layers of convolution and subsampling as indicated by the $128D$ feature vector which captures the motion data in the input frames. This layer consists of the same number of units as the number of actions, with each unit being linked extensively to each of the 128 units of the $C6$ layer, which is the output layer.

Trainable parameters are randomly determined using the online error back-propagation. Inputs for CNN models are restricted to a few contiguous frames. As a result, there are benefits that can be derived from the encoded high-level motion information which is integrated into 3D CNN models. To this end, the 3D CNN model is regularized by a sample of features as auxiliary outputs, as shown in Fig. 3.3. For each training map, a new feature is created by vector encoding the long-term map information that was not included in the CNN's input frame cube. The CNN learns a feature vector that is similar to this feature. This is accomplished by attaching a number of auxiliary output units to the CNN's final hidden layer and clamping the calculated feature vectors to the auxiliary units during training. This boosts the information included in the hidden layer to be near the high-level motion characteristic as much as possible.

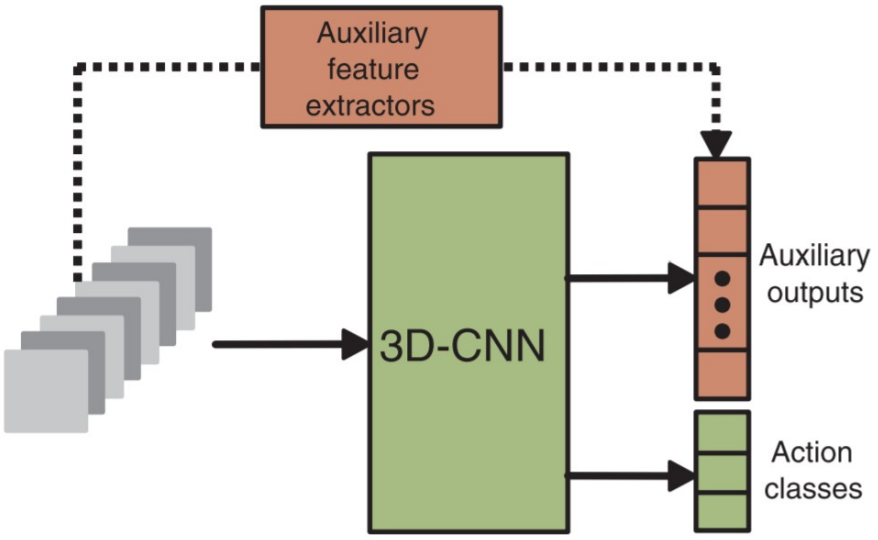


FIGURE 3.3: The regularized 3D CNN Architecture [2]

3.1.2 3D ResNet-18

Residual Neural Networks (ResNets) [52] serve as the foundation for the 3D ResNet-18 [10]. ResNets have shortcut systems that allow a signal to bypass some layers. The connections pass through the gradient flows of networks from later layers to early layers, facilitating the training of extremely deep networks. Fig. 3.4 shows the residual block, which is an element of ResNets. The connections bypass a signal from the top of the block to the lower block. ResNets consist of various residual blocks.

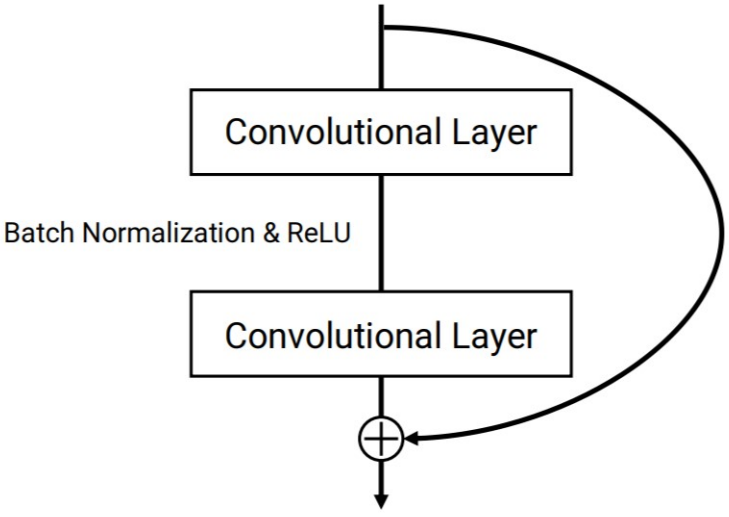


FIGURE 3.4: Residual block (Skip connections pass a signal from the top of the block to the tail. Signals are summed at the tail).

Table 3.1 shows the 3D ResNet-18 architecture. The difference between the 3D ResNet-18 and original ResNets [52] is determined by the number of dimensions of convolutional kernels and pooling. The 3D ResNet-18 mounts 3D convolution and 3D pooling. The sizes of convolutional kernels are $3 \times 3 \times 3$. The 3D ResNet-18 consists of 20 convolution layers. The temporal stride of the first convolution layer is 1, similar to C3D [53]. The network uses inputs such as 16 frame RGB clips. The sizes of input feature are $3 \times 16 \times 112 \times 112$. The convolution layers with a stride of 2 which carries out input down-sampling when the number of feature maps increases. The work in [54] used identity shortcuts with zero-padding (type A) to avoid increasing the number of parameters.

TABLE 3.1: 3D ResNet-18 Architecture [10]

Layer Name	Architecture
conv1	$7 \times 7 \times 7$, 64 stride 1(T) 2(XY)
	$3 \times 3 \times 3$ max pool, stride 2
conv2_x	$\begin{bmatrix} 3 \times 3 \times 3, & 64 \\ 3 \times 3 \times 3, & 64 \end{bmatrix} \times 2$
conv3_x	$\begin{bmatrix} 3 \times 3 \times 3, & 128 \\ 3 \times 3 \times 3, & 128 \end{bmatrix} \times 2$
conv_4x	$\begin{bmatrix} 3 \times 3 \times 3, & 256 \\ 3 \times 3 \times 3, & 256 \end{bmatrix} \times 2$
conv_5x up to conv_20x	$\begin{bmatrix} 3 \times 3 \times 3, & 512 \\ 3 \times 3 \times 3, & 512 \end{bmatrix} \times 2$

3.2 Spatio-Temporal Deep Learning

This section details the architecture of the selected spatio-temporal deep learning techniques as follows: Section 3.2.1 describes the ST-ResNet; Section 3.2.2 discusses the DMVST-Net; and Section 3.2.3 details the STD-Net.

3.2.1 ST-ResNet

The Spatio-temporal Residual Network (ST-ResNet) is a spatial-temporal extension of the ResNet [55], tailored for spatio-temporal prediction. Fig. 3.5 summarizes the ST-ResNet architecture, as proposed by Zhang et al., [3]. The architecture has four main components that model the temporal *trend, period, closeness* and *external* influence.

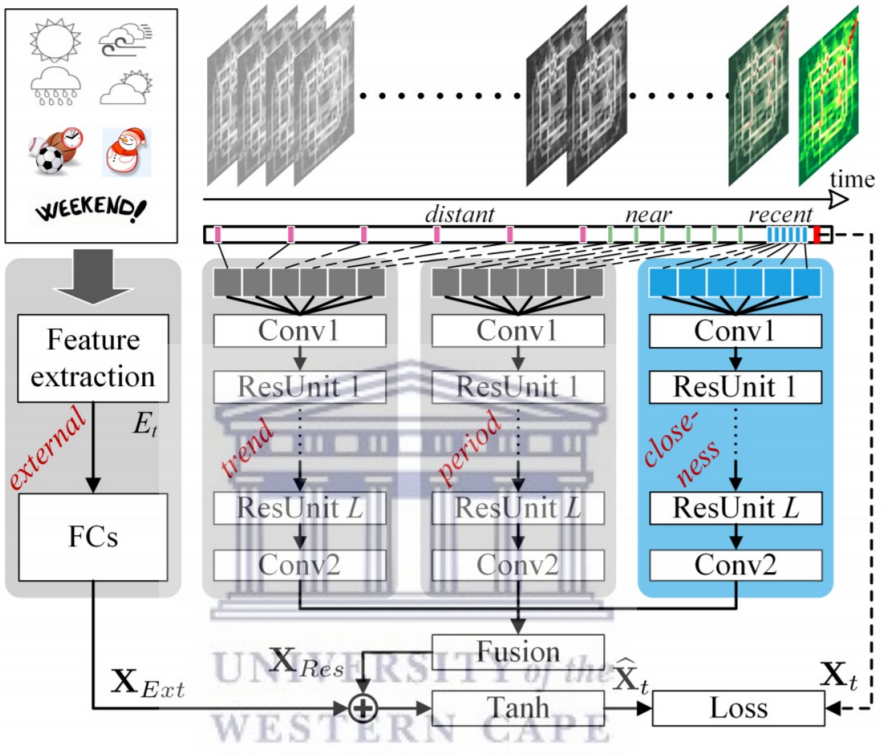


FIGURE 3.5: ST-ResNet Architecture, adopted from Zhang et al., [3]

The *trend, period* and *closeness* components have layers of convolutional neural networks followed by residual unit sequences. These three-component structures encapsulate the spatial interdependence of proximate and distant areas. The *external* component extracts features from external datasets.

The *trend, period, closeness* components include of three subcomponents, that is, a convolution and residual unit, and fusion. These subcomponents are described as follows;

1. **Convolutions *(Conv1 and Conv2)*:** CNNs have shown powerful ability to hierarchically capture the spatial structural information [52]. In order to capture the spatial dependency of crime incidents in any region, the CNN design is multi layered because one convolution only accounts for spatially close dependencies, which are limited by the size of their kernels. The same problem occurs in the video sequence generating task, where the input and output has the same resolution in [56]. To avoid the loss of resolution through sub-sampling while maintaining

distant dependencies, convolutions are one of the methods that can be used [57]. As shown in Fig. 3.6, there are three multiple levels of feature maps that are connected with a few convolutions.

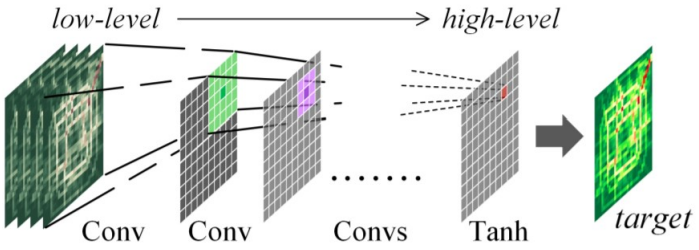


FIGURE 3.6: ST-ResNet Convolutions [3].

Each node in the high-level feature map is dependent on nine nodes in the middle-level feature map, which in turn rely on all nodes in the lower-level feature map, or input maps. This means that a single convolution detects spatially close relationships naturally, but a stack of convolutions detects distant, even spatial or region-level dependencies. The closeness component of Fig. 3.5 adopts a few two-channel flows matrices of intervals in real time to model temporal closeness dependence. Let the real-time fragment be $[X_{t-l_c}, X_{t-l_c+1}, ..., X_{t-1}]$, which is also known as the closeness dependent sequence. They are first concatenated along with the first axis, i.e. time interval, as one tensor $X_c^{(0)} \in \mathbb{R}^{2l_c \times I \times J}$, which is followed by a convolution shown in Fig.ure 3.5 as:

$$X_c^{(1)} = f\left(W_c^{(1)} * X_c^{(0)} + b_c^{(1)}\right), \tag{3.2}$$

where * represents the convolution; f is an activation function; $W_c^{(1)}, b_{c^{(1)}}$ are the trainable parameters in the first layer.

2. **Residual Unit *(ResUnit 1, ..., ResUnit L)*:** Rectified Linear Units (ReLU) and regularization have been used to improve the effectiveness of deep CNN's [58–60]. ResNets are equipped with skip connections, which allow a signal to be transmitted from one layer to another without interruption. Identical shortcut connections, also known as skip connections, are those that pass across the gradient flows of networks from later layers to earlier layers, hence reducing the amount of time required to train very deep networks. Fig. 3.7 illustrates how the residual unit was implemented in the ST-ResNet.

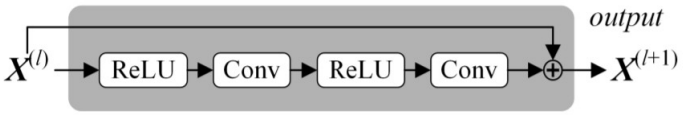


FIGURE 3.7: ST-ResNet Residual Unit [3].

A ResNet can consist of multiple residual units. The skip connections passed through a signal from the top of the block to the tail. Alternatively, in order to construct a deep network in order to capture large spatio-temporal dependencies, it is necessary to build a wider network. In [3] the input size is 32×32, and the kernel size of convolution is maintained to 3×3. In order to design spatio-temporal dependencies, i.e. each node in a high-level layer relies on all input nodes, it needs more than 15 consecutive convolutional layers. Residual learning is extremely successful for training deep neural networks [53]. In the ST-ResNet shown in Fig. 3.5, L residual units are stacked upon the convolutional layers as follows,

$$X_c^{(l+1)} = X_c^{(1)} + F(X_c^{(l)}; \theta_c^{(l)}), l = 1, ..., L, \tag{3.3}$$

where F represents a residual function, i.e., a combination of ReLU and convolution. Batch Normalization (BN)[53] is added before ReLU. A convolutional layer is appended on top of the L^{th} residual unit. With 2 convolutions and L residual units, the output of the closeness component of Fig. 3.5 is $X_c^{(L+2)}$.

The period and trend subcomponents are constructed in the similar fashion to the aforementioned operations. Assuming that there are l_p time intervals from the period fragment and the period is p. Then, the period dependent sequence is $[X_{(t-l_p)\cdot p}, X_{(l_p-1)\cdot p}, ..., X_{t-p}]$, with the convolutional operation and L residual units like in Eqn. 3.2 and 3.3, and the output of the period component is $X_p^{(L+2)}$.

The output of the trend component is $X_q^{(L+2)}$ with the input, $[X_{(t-l_q)\cdot p}, X_{(l_q-1)\cdot q}, ..., X_{t-q}]$, where l_q is the length of the trend dependent sequence and q is the trend span. Note that p and q are actually two different types of periods. In the detailed implementation, p is equal to one-day that describes daily periodicity, and q is equal to one-week that reveals the weekly trend.

3. **Fusion:** The outputs of *trend, period* and *closeness* components are combined using parameter-matrix based fusion to give X_{Res}. Fig. 3.8 shows details of the ST-ResNet fusion step. The parameter matrix allocates unique weights to the

results of each component in each region by using:

$$X_{Res} = W_c \cdot X_c^{(L+2)} + W_p \cdot X_p^{(L+2)} + W_q \cdot X^{(L+2)}, \tag{3.4}$$

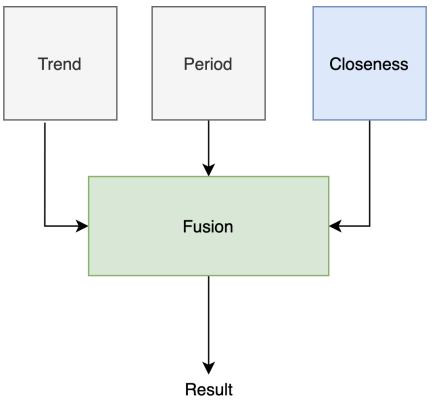


FIGURE 3.8: ST-ResNet Fusion [3]

3.2.2 DMVST-Net

Yao et al., [4] developed the Deep Multi-View Spatio-Temporal Network (DMVST-Net) framework, in order to make spatio-temporal predictions in the domain of taxi demand prediction. In this domain, the DMVST-Net outperformed the ST-ResNet by a significant margin. DMVST-Net is a multi-view model that takes into account several factors such as *spatial*, *temporal* and *semantic* relations at the same time. The DMVST-Net integrated these three views to form a combined network architecture. Fig. 3.9 shows the summary of the network architecture. The network can be described in terms of the spatial view, temporal view, semantic view, predictive component and loss function.

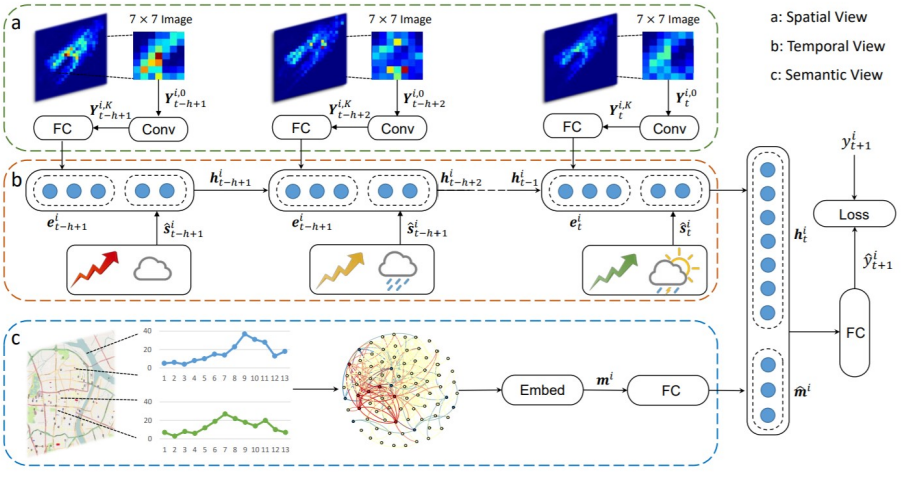


FIGURE 3.9: DMVST-Net Architecture, adopted from Yao et al., [4]

The components of the DMVST-Net can be described as follows:

1. **Spatial View:** The spatial view is composed of a local CNN, which only considers spatially nearby regions. Fig. 3.10 shows the architecture of the spatial component.

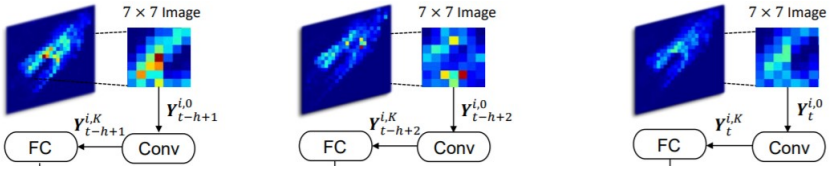


FIGURE 3.10: DMVST-Net Spatial View Component [4]

As shown in Fig. 3.9, at each time interval t, the surrounding neighborhood of location i is denoted by $S \times S$, where S controls the spatial granularity. Zero padding was implemented for location at boundaries of different locations. As a result, an image is expressed as a tensor having one channel $Y_i^t \in \mathbb{R}^{S \times S \times 1}$, for each location i and time interval t. The local CNN takes Y_t^i as input $Y_t^{i,0}$ and feeds it into k convolutional layers. The transformation at each layer k is defined as follows:

$$Y_t^{i,k} = f(Y_t^{i,k-1} \cdot W_t^k + b_t^k) \tag{3.5}$$

where $(\cdot)$ denotes the convolutional operation and $f(\cdot)$ is the rectified linear unit (ReLU) activation function, defined by

$$f(z) = max(0, z); \tag{3.6}$$

W_t^k and b_t^k are two sets of parameters in the k^{th} convolution layer. The parameters $W_t^{1,\ldots,K}$ and $b_t^{1,\ldots,K}$ are shared across all regions $i \in L$ to make the computation tractable. After K convolution layers, a flattened layer is used to transform the output $Y_t^{i,K} \in \mathbb{R}^{S \times S \times \lambda}$ to a feature vector $s_t^i \in \mathbb{R}^{S^2 \lambda}$ for region i and time interval t. At last, a fully connected layer is used to reduce the dimension of spatial representations s_t^i, which is defined as:

$$\hat{s}_t^i = f(W_t^{fc} s_t^i + b_t^{fc}) \tag{3.7}$$

where W_t^{fc} and b_t^{fc} are two learnable parameter sets at time interval t. Finally, for each time interval t, we get the $\hat{s}_t^i \in \mathbb{R}^d$ as the representation for region i.

2. **Temporal View:** The temporal view frames the sequential relationships inherent in demand time series. The temporal view is represented by a Long Short-Term

Memory (LSTM) network. LSTM [61] is a form of neural network structure that enables the modelling of sequential dependencies through the recursive application of a transition function on the input's hidden state vector. It is designed to solve the challenges of the classical Recurrent Neural Network's (RNN) gradient, expanding or disappearing after long sequence training [17]. Fig. 3.11 shows the architecture of the temporal component.

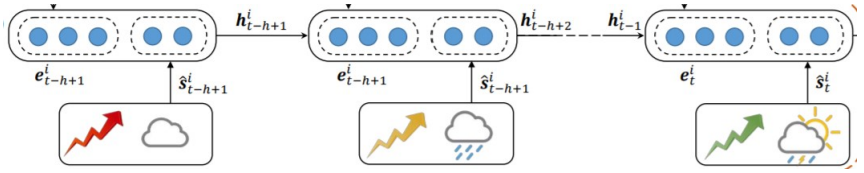


FIGURE 3.11: DMVST-Net Temporal View Component [4]

LSTM learns sequential correlations stably by maintaining a memory cell c_t in a time interval t, which can be regarded as an accumulation of previous sequential information. In each time interval, the LSTM takes the following inputs: (i) temporal information, g^i_t, (ii) the number of time intervals, h^i_{t-1} and (iii) memory cells, c^i_{t-1}. All information accumulates in the memory cell, and then gets activated at the input gate I^i_t. In addition, LSTM has a *forget gate* f^i_t. When the forget gate is activated, the network is capable of erasing the preceding memory cell c_{t-1}. Also, the output gate o^i_t controls the output of the memory cell. In this study, the architecture of LSTM is formulated as follows:

$$
\begin{aligned}
I^i_t &= \sigma\big(W_i g^i_t + U_i h^i_{t-1} + b_i\big),\\
f^i_t &= \sigma\big(W_f g^i_t + U_i h^i_{t-1} + b_f\big),\\
o^i_t &= \sigma\big(W_o g^i_t + U_i h^i_{t-1} + b_o\big),\\
\theta^i_t &= tanh\big(W_g g^i_t + U_i h^i_{t-1} + b_g\big),\\
c^i_t &= f^i_t \cdot c^i_{t-1} + i^i_t \cdot \theta^i_t,\\
h^i_t &= o^i_t \cdot tanh(c^i_t)
\end{aligned}
\tag{3.8}
$$

where $(\cdot)$ represents the Hadamard product and *tanh* is the hyperbolic tangent function. Both of these functions are element-wise. W_a, U_a, b_a $(a \in \{i, f, o, g\})$ are trainable parameters. As shown in Fig. 3.9 the temporal component takes

representations from the spatial view and concatenates them with context features. This is represented by the following equation:

$$g_t^i = \hat{s}_t^i + e_t^i \tag{3.9}$$

where (+) denotes the concatenation operator, therefore, $g_t^i \in \mathbb{R}^{r+d}$.

3. **Semantic View:** It makes logical sense that locations with similar functions would have similar crime patterns; for example, low-density residential regions may have a low rate of house break-ins, whereas high-density residential areas may have a higher rate of theft events. It is possible that similar places are not always adjacent to one another in space. As a result, the graph maps locations sites that show functional (semantic) similarity among regions. Fig. 3.12 shows the architecture of the semantic component.

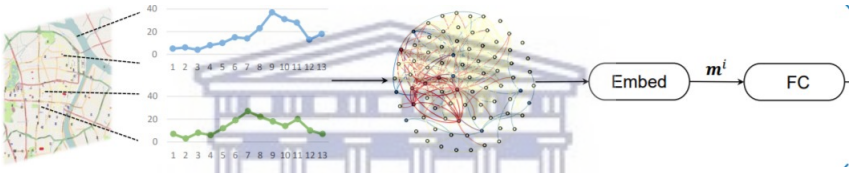


FIGURE 3.12: DMVST-Net Semantic View Component [4]

The semantic graph of location is given by $G = (V, E, D)$, where the set of locations L are nodes $V = L$, $E \in V \times V$ is the edge set, and D is a set of similarity on all the edges. Dynamic Time Warping (DTW) is used to measure the similarity ω_{ij} between node (location) i and node (location) j which is given by

$$\omega_{ij} = exp(-\alpha DTW(i, j)) \tag{3.10}$$

where α is the parameter that controls the decay rate of the distance, and $DTW(i, j)$ is the dynamic time warping distance between the demand patterns of two locations. The demand patterns determined the use of an average weekly demand time series. Full connections are evident on the graph, as they are symbolised by the dots in the middle for every two regions. An embedding method is used on the graph in order to encode each node into a low-dimensional vector while still keeping the structural information. For each node i (location), the embedding method outputs the embedded feature vector m^i. In addition, in order to co-train the embedded m^i with the whole network architecture, the feature vector m^i is fed to a fully connected layer, which is defined as

$$\hat{m}^i = f(W_{fe} m^i + b_{fe}) \tag{3.11}$$

where W_{fe} and b_{fe} are both learnable parameters.

4. **Predictive Component:** This component is used to give a prediction at time $t+1$ given the data has details up until time t. The three views and joined together by concatenating $\hat{m}^i$ with the output h_t^i of LSTM:

$$q_t^i = h_t^i \cdot \hat{m}^i \tag{3.12}$$

The output of LSTM h_t^i contains both effects of temporal and spatial views. Then q_t^i input is fed into the fully connected network to get the final prediction value $\hat{y}_{t+1}^i$ for each region. The final prediction function is defined by

$$\hat{y}_{t+1}^i = f(W_{ff}q^i + b_{ff}) \tag{3.13}$$

where W_{ff} and b_{ff} are learnable parameters. $\sigma(x)$ is a Sigmoid function defined as

$$\sigma(x) = 1/(1+e^{-x}) \tag{3.14}$$

The output of the model is in the range $[0, 1]$, as the demand values adjust. Later, the readjusted prediction is used to get actual demand values.

5. **Loss Function:** The type of loss function is a cost function, which is a mapping "cost" function between events or values of one or more variables and a real number conceptually expressing some measure of the cost of the occurrence. The cost function calculates the distance between the current output of the algorithm and the expected output. The loss function is defined as:

$$L(\theta) = \sum_{t=1}^{N} \left((y_{t+1}^i - \hat{y}_{t+1}^i)^2 + \gamma \left(\frac{y_{t+1}^i - \hat{y}_{t+1}^i}{y_{t+1}^i} \right)^2 \right) \tag{3.15}$$

where θ are all learnable parameters in the DMVST-Net and γ is a hyperparameter.

3.2.3 STD-Net

The Spatio-temporal Dynamic Network (STD-Net) architecture was proposed by Ali et al., [5]. Fig. 3.13 shows an architecture of the STD-Net, that is structured around four primary components: temporal closeness, period volume, weekly volume, and external influences.The architecture is designed to accurately predict a target region with low correlations. Using a CNN to simulate the spatial dependency between nearby regions.

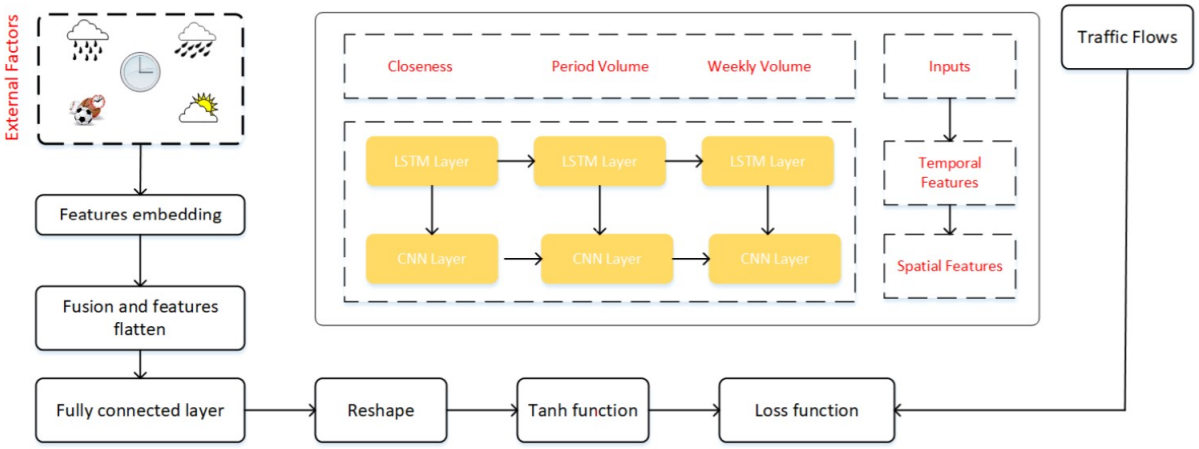


FIGURE 3.13: STD-Net Architecture, adopted from Ali et al., [5]

To simulate the sequential temporal dependency, an LSTM network is used to solve the problems of the typical recurrent neural network and the vanishing gradient problem. The most important components of the STD-Net network are the convolutional-LSTMs (ConvLSTMs), which replace the kernel size in LSTM with convolutional operations. The designed network extracts spatio-temporal dependencies by focusing on hidden features. This technique deals with both spatial and temporal information contained within the data. The hybrid neural network captures both spatial and temporal information by focusing on hidden features. There are two sub-DNNs in the deep hybrid neural network. In contrast to the first DNN, which collects spatial features using CNN, the second DNN learns temporal features over a period of time using LSTM. Furthermore, the first three attributes outputs form a single output Z_{fusion}.

The CNN part of the ConvLSTM is used to extract spatial data. The CNN is quite effective at spatial data extraction, while the input data also contains temporal information, this directly impacts the accuracy of the prediction. Sun et al., [62] proposed the use of a recurrent neural network (RNN) to simultaneously capture the temporal information in order to accurately explore the temporal information.

In the STD-Net, the LSTM is used in conjunction with regular RNN to handle the vanishing gradient problem and to do time series feature exploration. The LSTM determines three distinct components, namely proximity, period, and trend, which are illustrated by various inputs for closeness, period, and trend. The mathematical formulation for the STD-Net model is given by

$$
\begin{aligned}
i_t &= \sigma\big(W_{yi} * Y_{r,t} + W_{di} * Y_{r,t} + W_{ri} \cdot C_{t-1} + b_i\big),\\
f_t &= \sigma\big(W_{yf} * Y_{r,t} + W_{df} * Y_{c,t} + W_{ri} \cdot C_{t-1} + b_f\big),\\
C_t &= f_t \cdot C_{t-1} + i_t \cdot tanh\big(W_{yr} * Y_{r,t} + W_{dr} * Y_{r,t-1} + b_c\big),\\
o_t &= \sigma\big(W_{yo} * Y_{r,t} + W_{ho} * Y_{c,t-1} + W_{ro} \cdot C_t + b_o\big),\\
Y_{r,t} &= o_t \cdot tanh(C_t),
\end{aligned}
\tag{3.16}
$$

where * denotes the convolutional operation and $\cdot$ represents the Hadamard product, and W_{yi}, W_{di}, W_{yf}, W_{df}, W_{yr}, W_{dr}, W_{yo}, W_{ho}, W_{ro}, b_i, b_f, b_c, b_o are training parameters. The variables i_t, f_t, C_t and o_t represented the input, forget, and output gates, respectively. Similarly, the designed structure determines the period volume and weekly volume branch using the same process as previously described. Given that l_p is a time intervals of daily segments and d denotes a daily segment. The daily dependent sequence is given by $[Y_{t-l_p\times p}, Y_{t-l_{p-1}\times p}, ..., Y_{t-1}]$ and the output of the daily segments is $Y_{p,t}^{(L+2)}$. The daily dependent sequence of trend component, $[Y_{t-l_w\times t}, Y_{t-(l_{t-1})\times w}, ..., Y_{t-1}]$, and weekly component output is $Y_{w,t}^{(L+2)}$, where l_w represents the daily dependent sequence length of trend segment and w denotes the trend. The main components of the STD-Net described as follows:

1. **Convolutional and Residual Unit:** The ConvLSTM was designed in such a way that it extracted the spatial and temporal dependencies of any area. It proved the fact that deep convolutional neural networks (DNN) improve the effectiveness of training by using a well-known activation function, such as the ReLU, as well as some regularization procedures that have already been implemented in [58, 63]. Similarly, the DNN explores big location dependencies. The kernel size set is 3×3 and input size is 32×32 for crime data. In the STD-Net model, L residual units with both CNN-LSTM are stacked together. Batch normalization method is used to apply to the network layers [58, 63]. Batch normalization method is applied before residual units (ReLU). Convolutions and ConvLSTM layers are added on top of the residual unit. The output of closeness component with ConvLSTM, convolutions and L residual units is $Y_{Ext(l+2)}$.

2. **Fusion:** The fusion step is the last major step in the STD-Net architecture. Fig. 3.14 shows the steps taken in fusing the components, then passing the results to fully connected layers. The three components (closeness, period volume, and weekly volume) are infused using the parametric based fusion method.

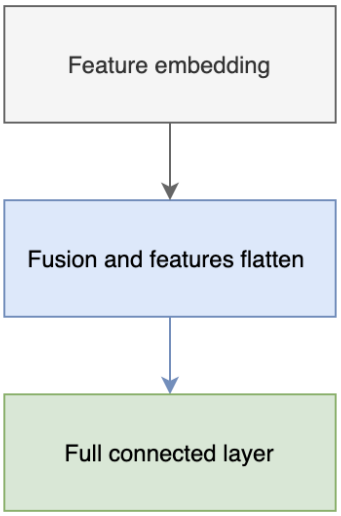


FIGURE 3.14: STD-Net Fusion steps [5]

The parametric fusion method is given by

$$Z_{fusion} = W_r \cdot Y_{r,t} + W_p \cdot Y_{p,t} + W_w \cdot Y_{w,t} \tag{3.17}$$

where $(\cdot)$ represented the Hadamard product, while W_w, W_p, and W_r learnable parameters that modify the degrees which are affected through weekly volume, period volume, and temporal closeness. The result of the three components are combined. At time interval t, the predicted value indicates as $Y_{r,t}$. The $Y_{r,t}$, $Y_{p,t}$, and $Y_{w,t}$ represented closeness, period, and weekly components respectively. The STD-Net model was trained to predict z_t from four properties, i.e., weekly, period, closeness and external components respectively by reducing the value of Mean Squared Error (MSE) between the ground truth and predicted crimes at time interval t.

3.3 Conclusion

This chapter discussed the architectures of the selected deep learning models in detail. This provided an understanding of how each technique works as a theoretical base for the implementation of these techniques in this research.

The 3D deep learning techniques described were the 3D CNNs and 3D ResNet. The spatio-temporal techniques detailed were the ST-ResNet, DMVST-Net and STD-Net.

The next chapter provides details of the experimental setup used to evaluate and compare the deep learning techniques in crime forecasting.

Chapter 4

Experimental Setup

This chapter discusses the setup of the experiments carried out to provide answers to the research questions. As part of the setup to the experiments, the implementation of various components was also carried out. This included implementing the deep learning models as well as preprocessing, splitting and formatting the data in preparation for training and/or testing with the various deep learning models. As such, the chapter details steps carried out to initially address research objectives 2 and 3.

This chapter is organised as follows; Section 4.1 provides information about the programming language that was utilized and why it was selected. Section 4.2 describes the deep learning framework used in the implementation and experiments. Section 4.3 discusses features of the deep learning application program interface and the hardware that was used in the experiments. Section 4.4 describes the Chicago crime dataset used in addition to the data prepossessing, splitting and formatting procedures that were performed. Section 4.6 closes the chapter with a conclusion.

4.1 Programming Language

Python is an interpreted high-level general-purpose programming language. With the usage of considerable indentation, its design philosophy prioritizes code readability. Its language elements and object-oriented approach are intended to assist programmers in writing clear, logical codes for small and large-scale projects [64]. Python is a dynamically typed and garbage-collected programming language. This programming language supports a variety of programming paradigms, including structured (especially procedural), object-oriented, and functional programming. In recognition of its extensive standard library, it is frequently referred to as a "batteries included" language. [65]. Python is characterised by the following features and benefits [66];

- *Independence across platforms*: Unlike other programming languages, developers prefer Python because of its ability to run on many platforms without the need to modify the code. Python is compatible with a variety of operating systems, including Windows, Linux, and macOS, and hence requires little or no customization. Due to the fact that the platforms are fully compatible with the Python, there is little to no need for a Python expert to explain the code of the software. Python's simplicity of executability makes it simple to share software, and it also makes it possible to develop and run standalone applications using the language. Python is the only programming language that can be used to create the software from start to finish. It is advantageous for developers because other programming languages require additional programming languages before a project can be considered complete. Python's platform independence saves time and resources for developers to execute a single project.

- *Consistency and simplicity*: Most software developers who seek simplicity and consistency in their work will find Python quite useful. The Python code is short and legible, making the presenting process much easier. Unlike other programming languages, a developer may write a code quickly and concisely. It enables developers to gather feedback from other developers in the community in order to improve the software or product. Python's simplicity allows beginners to master it quickly and with less effort. Furthermore, experienced developers find it simple to build solid and effective systems, allowing them to focus their efforts on improving their creativity and solving real-world problems with machine learning.

- *Frameworks and libraries variety*: The environment, libraries, and the frameworks are essential components of a proper programming language. Python frameworks and libraries provide a dependable environment that considerably decreases the amount of time spent developing a software. A library is essentially a collection of pre-written codes that developers can utilize to expedite coding when working on large projects, such as web applications. For instance, Python contains a modular machine learning library known as PyBrain, which contains easy-to-use algorithms for machine learning tasks. The best and most dependable coding solutions necessitate a well-structured and well-tested environment, which is provided by Python frameworks and libraries, among other things.

- *Support for Machine Learning*: Because of its higher simplicity and consistency than other programming languages, Python has algorithm for automating operations. Furthermore, the presence of an active Python community makes it simple for developers to discuss projects and provide ideas to improve their codes.

4.2 Deep Learning Framework

Deep learning frameworks provide building blocks for developing, training, and verifying deep neural networks by utilizing a high-level programming interface to communicate with the deep neural network. CUDDA Deep Neural Network (CuDNN), NVIDIA Collective Communications Library (NCCL), and NVIDIA Data Loading Library (DALI) are GPU-accelerated libraries that are widely used for deep learning frameworks such as MXNet, PyTorch and TensorFlow to enable high-performance multi-GPU accelerated training [67]. TensorFlow is a Google-developed open-source library designed primarily for deep learning applications. Additionally, it supports conventional machine learning. TensorFlow originally performed huge numerical computations without regard for deep learning. However, it has recently proved to be extremely useful for developing deep learning models, and as a result, Google open-sourced it. TensorFlow accepts data in the form of multidimensional arrays called tensors. Multidimensional arrays are extremely useful when working with enormous volumes of data. TensorFlow uses nodes and edges in data flow graphs. Due to the graph-based execution mechanism, it is much easier to distribute TensorFlow code across a cluster of computers when using GPUs [6]. TensorFlow framework has the following benefits;

- *C++ and Python Application Program Interface Support*: Prior to the development of libraries, the machine learning and deep learning coding mechanisms were far more sophisticated. This library implements a high-level API, which eliminates the need for sophisticated coding to create a neural network, configure a neuron, or program a neuron. The library keeps track of all these functions. Additionally, TensorFlow integrates with Java and R.

- *CPUs and GPUs Computing Devices Support*: Deep learning applications are quite difficult, requiring a large amount of computing throughout the training process. It takes a long time due to the big data set and involves multiple iterative processes, mathematical calculations, and matrix multiplications, among other things. If these tasks were performed on a standard Central Processing Unit (CPU), they would take much longer. Graphical Processing Units (GPUs) are popular in the context of games, which require a high-resolution screen and image. Originally, GPUs were built for this function. They are, however, used to develop deep learning applications. One of TensorFlow's primary advantages is that it works on both GPUs and CPUs. Additionally, it compiles faster than other deep learning libraries, such as Keras [68] and Torch [69].

When programming with TensorFlow, the primary object that is changed and passed around is the *Tensor*. Tensor is a multidimensional generalization of vectors and matrices. Tensors are arrays of data with variable dimensions and rankings that are sent

into the neural network as input. Throughout deep learning, particularly during the training phase, vast amounts of data are encountered in a highly convoluted manner. Tensors helps if the data is placed, used, or stored compactly, since it has multidimensional arrays. When tensors hold data and are fed into a neural network, it results in the following output, as illustrated in Fig. 4.1.

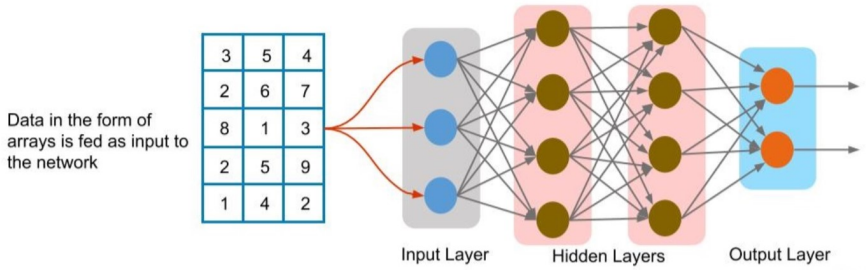


FIGURE 4.1: Output of Neural Network when fed a *Tensor* [6]

A tensor can be defined in terms of *dimensions* and *ranks*. A tensor can be expressed in terms of *1-dimension*, *2-dimension*, or *3-dimension*. Fig. 4.2, 4.3, and 4.4 illustrate the various dimensions.

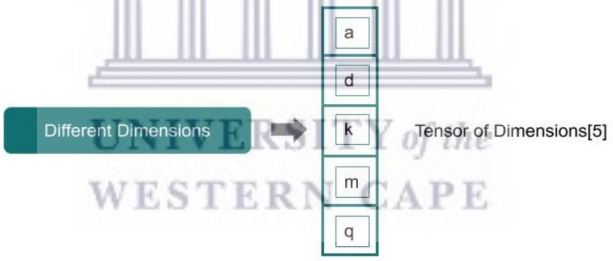


FIGURE 4.2: 1-dimesional Tensor *Tensor* [6]

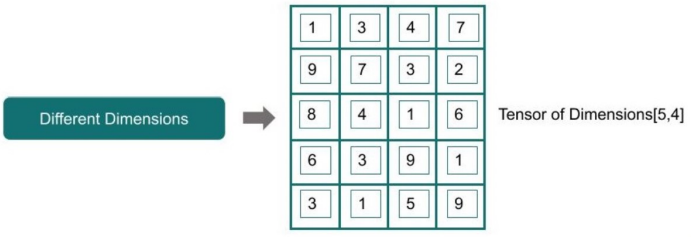


FIGURE 4.3: 2-dimensional Tensor *Tensor* [6]

Tensor ranks are the number of dimensions used to represent the data; this means tensors are simply mathematical objects that can be used to describe physical properties, much like scalars and vectors. They represent the number of dimensions used to represent the data. In fact, tensors are a generalized version of scalars and vectors. It is defined as the

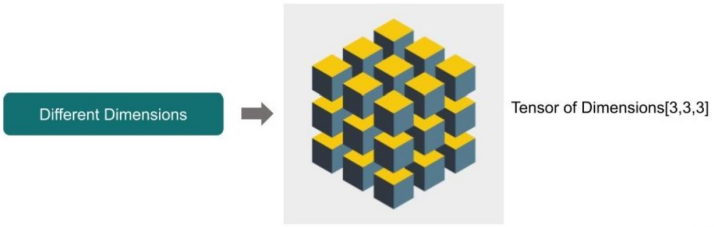


FIGURE 4.4: 3-dimensional Tensor *Tensor* [6]

rank (or order) of a tensor by the number of directions (and thus the dimensionality of the array) that are necessary to describe the tensor. Examples include features that can be fully expressed by a single direction (first rank) and properties that require multiple directions (second rank) of a 3×1 column vector, as well as features that need two directions (second rank tensors), can be expressed with nine digits 3×3 matrix. As such, in general, a n^{th} rank tensor can be discussed by 3n coefficients [70]. In this study, the deep neural networks are trained using 3-dimensional tensors with a *rank* of 3.

4.3 Deep Learning API and Experimental Platform

A deep learning application programming interface (API) is a machine learning library built on top of a deep learning framework, such as TensorFlow, that is intended to reduce the cognitive load associated with developing and training deep learning models [71]. Keras is an open-source Python interface for deep learning. This work uses Keras version 2.4.0, which is supported by the TensorFlow backend.

A hyper threaded Linux virtual machine hosted on Azure Cloud Services [72], with 8 virtual CPUs, is used. Each virtual CPU has 128 GB of RAM, and a processor speed of 2.3 GHz.

4.4 Dataset used in this study

This section describes the dataset used in this work (Section 4.4.1), as well as the data preprocessing (Section 4.4.2) and splitting into training and testing subsets (Section 4.4.3) carried out on the dataset.

4.4.1 Dataset Description

This research used the crime dataset from Chicago Data portal [7]. The dataset consists of crime incident reports dating back to 2001 for the City of Chicago, which has 7.29 million entries that are mapped on a heatmap in Fig. 4.5.

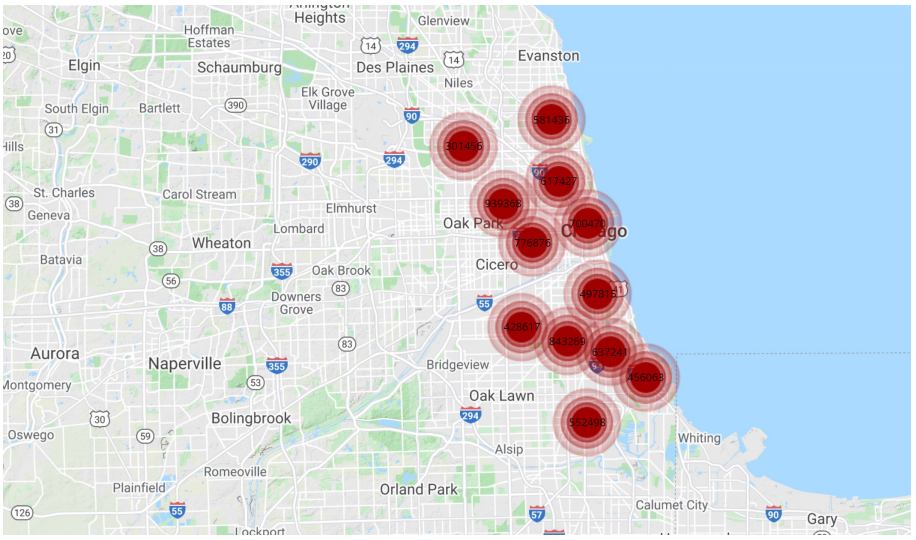


FIGURE 4.5: Heatmap of Incidents in Chicago Crime Dataset [7]

Each reported incident consists of location information about latitude, longitude, X and Y coordinates, district, time and crime category, e.t.c. The dataset has 22 feature columns and 32 unique crime types. The Table 4.1 describes the features and data types of the feature columns in the crime dataset.

TABLE 4.1: Column description of the Chicago crime dataset

Column Name	Description	Type
ID	Unique identifier for the record.	Number
Case Number	Unique Records Division Number	Plain Text
Date	Date when the incident occurred.	Date & Time
Block	The address.	Plain Text
IUCR	Unifrom Crime Reporting code.	Plain Text
Primary Type	The primary description of the IUCR code.	Plain Text
Description	The secondary description of the IUCR code.	Plain Text
Location Description	Description of the location.	Plain Text
Arrest	Indicates whether an arrest was made.	Checkbox
Domestic	Indicates whether the incident was domestic.	Checkbox
Beat	Indicates the beat.	Plain Text
District	Indicates the police district.	Plain Text
Ward	The ward.	Number
Community Area	Indicates the community.	Plain Text
FBI Code	Indicates the FBI crime code.	Plain Text
X Coordinate	The x coordinate of the location.	Number
Y Coordinate	The y coordinate of the location.	Number
Year	Year the incident occurred.	Number
Updated On	Date and time the record was last updated.	Date & Time
Latitude	The latitude of the location.	Number
Longitude	The longitude of the location.	Number
Location	The location.	Location

All feature columns are categorised into three groups namely: *crime type*, *spatial*, and *temporal* features as shown in Table 4.2. The unique identifiers of the crime incident, such as the *ID, Case Number*, feature columns, do not impact crime prediction. Therefore, they are not included in the related features in Table 4.2. The columns that identify the crimes similarly are *Primary Type, IUCR, Description, Domestic, FBI code*. These "crime type" columns are combined to enrich the crime-identifier feature. The *Location Description, Beat, Ward, District, Community Area, X Coordinate, Y Coordinate, Latitude, Longitude, Location, Block*, feature columns all describe the spatial features of the crime incident to a different degree of specificity. Lastly, the *Date, Year, Updated On*, feature columns describe the date and time the crime occurrence happened, as well as the date and time of the documented incident, all of which described the temporal aspects. The overall number of criminal episodes varied, as illustrated in Fig. 4.6, which shows the frequency of criminal incidences in each community area from 2001 to date.

TABLE 4.2: Related Feature Columns in Chicago Crime Dataset

Feature Category	**Related Feature Columns**
Type	Primary Type IUCR Description Domestic FBI code
Spatial	Location Description Beat Ward District Community Area X Coordinate Y Coordinate Latitude Longitude Location Block
Temporal	Date Year Updated On

Furthermore, Fig. 4.7 demonstrates the crime types with the largest number of reported incidences as follows: *theft, battery* and *criminal damage*. This is determined by putting together all the criminal occurrences that occurred between 2001 and 2020. This demonstrates dataset imbalance, because the total number of crime incidents per category is not equal, and as a result, the prediction model does not perform well when predicting.

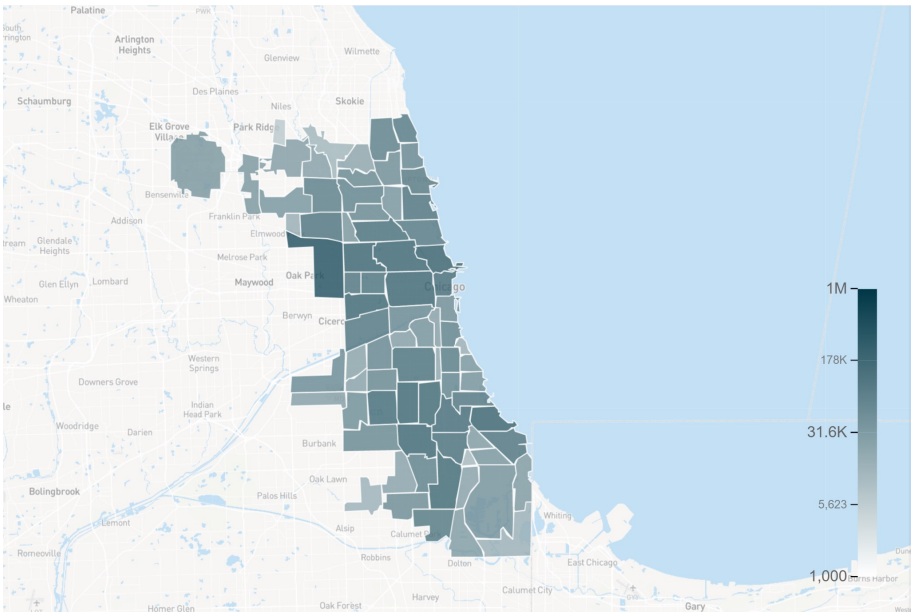


FIGURE 4.6: Incidents Per Location in Chicago Crime Dataset [7]

FIGURE 4.7: Total Number of Incidents in Chicago Crime Dataset [7]

Fig. 4.8 indicates the top three site descriptions such as *street, residence* and *apartments* where the majority of reported crimes occurred. This gives an overview of the effects, of the location description on crime prediction.

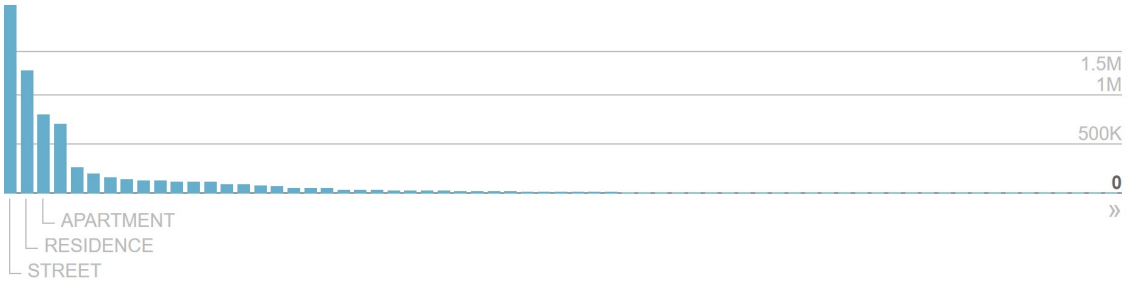


FIGURE 4.8: Total Number of Incidents Per Location Description in Chicago Crime Dataset [7]

4.4.2 Preprocessing

In the data preprocessing phase, two Python libraries are mainly used, namely;

- Pandas: an open-source Python-based data analysis and manipulation tool.
- Sci-kit Learn: an open-source Python-based machine learning library.

The data preprocessing consists of the following steps;

1. Feature selection: This involves reducing the feature space of the dataset by selecting potentially salient features and discarding other features. This is described in Section 4.4.2.1.
2. Drop missing rows: Rows with missing information are discarded in this step. This is discussed in Section 4.4.2.2.
3. Label encoding: In this step, one-hot encoding is applied to the classification data. This is detailed in Section 4.4.2.3.
4. Converting spatio-temporal data into *image-like* data. This is described in Section 4.4.2.4.
5. Feature scaling: This is a procedure applied to data in order to speed up training and increase the likelihood of convergence. This is discussed in Section 4.4.2.5.

4.4.2.1 Feature Selection

The initial step in the data preprocessing phase is feature selection, which helps the deep learning algorithms to train more quickly. Simplifying a model through feature selection decreases its complexity and makes it easier to interpret. Selecting the appropriate subset enhances model accuracy. Therefore, a few related feature columns train the deep learning algorithms. Table 4.3 is a transposed table that shows a sample of two rows of the dataset with all 22 features.

The selected features are *Crime Type, Longitude, Latitude*, and *Date*. The *Crime Type* is the class of the crime which the models predict. The *Longitude* and *Latitude*, are criminal incident areas. The *Date* feature column contains details of the time, day, month, and year when the incident was documented. Table 4.4 shows a sample of the dataset with the selected feature columns.

TABLE 4.3: Sample 2 Rows of Crime Dataset (Transposed) Before Feature Selection

Features	1^{st} Record	2^{nd} Record
ID	12014684	12012127
Case Number	JD189901	JD189186
Date	03/17/2020 09:30:00 PM	03/18/2020 02:03:00 AM
Block	039XX N LECLAIRE AVE	039XX W JACKSON BLVD
IUCR	0820	0910
Primary Type	THEFT	MOTOR VEHICLE THEFT
Description	$500 AND UNDER	AUTOMOBILE
Location Description	STREET	APARTMENT
Arrest	False	False
Domestic	False	True
Beat	1634	1132
District	16	11
Ward	45.0	28.0
Community Area	15.0	26.0
FBI Code	06	07
X Coordinate	1141659.0	1150196.0
Y Coordinate	1925649.0	1898398.0
Year	2020	2020
Updated On	03/25/2020 03:45:43 PM	03/25/2020 03:47:29 PM
Latitude	41.952052	41.87711
Longitude	-87.75466	-87.72399
Location	(41.952051946, -87.754660372)	(41.877110187, -87.723989719)

TABLE 4.4: Sample of Crime Dataset After Feature Selection

Primary Type	Date	Longitude	Latitude
DECEPTIVE PRACTICE	06/25/2020 03:59:00 PM	-87.680412	41.955957
NARCOTICS	01/15/2020 12:22:32 PM	-87.654806	41.791444
OTHER OFFENSE	11/17/2020 04:30:00 PM	-87.681111	41.826677
BATTERY	12/31/2020 05:15:00 AM	-87.688750	41.917656
CRIMINAL DAMAGE	02/08/2020 08:30:00 PM	-87.639608	41.722820
THEFT	09/12/2020 10:50:00 AM	-87.565044	41.753071
ASSAULT	06/30/2020 09:51:00 AM	-87.639963	41.740926
THEFT	07/25/2020 03:00:00 PM	-87.795150	41.789739
BATTERY	05/15/2020 08:07:00 PM	-87.729569	41.975517
ASSAULT	01/26/2020 05:28:00 AM	-87.604816	41.736422
BATTERY	03/01/2020 08:10:00 AM	-87.614347	41.777639
DECEPTIVE PRACTICE	11/30/2020 09:00:00 AM	-87.592465	41.712860
CRIMINAL TRESPASS	01/09/2020 06:27:00 PM	-87.620274	41.893204
OTHER OFFENSE	03/26/2020 11:05:00 AM	-87.798886	41.922909
BURGLARY	06/22/2020 02:43:00 AM	-87.662831	41.689491
WEAPONS VIOLATION	06/20/2020 11:41:00 PM	-87.678200	41.835807
THEFT	02/09/2020 01:15:00 PM	-87.767662	41.887265
BATTERY	06/06/2020 05:00:00 PM	-87.666186	41.904004
MOTOR VEHICLE THEFT	10/10/2020 09:40:00 PM	-87.726638	41.882028
ROBBERY	01/09/2020 07:35:00 PM	-87.670907	41.977065

TABLE 4.5: Sample of Crime Dataset with Missing Information

Primary Type	Date	Longitude	Latitude
NARCOTICS	09/01/2020 09:00:00 AM	NaN	NaN
THEFT	10/08/2020 04:00:00 PM	NaN	NaN
DECEPTIVE PRACTICE	11/24/2020 12:01:00 AM	NaN	NaN
THEFT	09/11/2020 05:00:00 PM	NaN	NaN
THEFT	11/16/2020 09:00:00 AM	NaN	NaN
THEFT	11/06/2020 08:05:00 AM	NaN	NaN
DECEPTIVE PRACTICE	06/03/2020 05:50:00 PM	-87.694878	41.982229
OTHER OFFENSE	12/15/2020 08:34:00 AM	NaN	NaN
DECEPTIVE PRACTICE	08/03/2020 08:40:00 PM	-87.714285	41.871776
ASSAULT	12/01/2020 12:01:00 AM	NaN	NaN
ARSON	10/09/2020 12:49:00 AM	NaN	NaN
OTHER OFFENSE	03/29/2020 05:55:00 PM	NaN	NaN
THEFT	08/20/2020 01:05:00 PM	NaN	NaN
THEFT	03/08/2020 03:00:00 AM	NaN	NaN
DECEPTIVE PRACTICE	07/21/2020 10:00:00 AM	NaN	NaN
DECEPTIVE PRACTICE	03/11/2020 01:40:00 PM	-87.698608	41.745308
DECEPTIVE PRACTICE	08/06/2020 06:10:00 PM	-87.616613	41.754280
DECEPTIVE PRACTICE	06/01/2020 08:00:00 AM	-87.642643	41.937684
DECEPTIVE PRACTICE	09/21/2020 06:40:00 PM	-87.601656	41.779781
THEFT	06/09/2020 05:00:00 PM	NaN	NaN

4.4.2.2 Drop Missing Rows

The following step in the data preprocessing phase includes eliminating all rows with missing data. Table 4.5 indicates a sample of records that lack critical information. When Pandas reads a dataset that has missing data, the default value is *NaN*. The records that lack information are removed since the incidents are documented with location data as shown in Table 4.5.

4.4.2.3 Label Encoding

The dataset is then label encoded to convert categorical variables to numerical values. The LabelEncoder technique, as defined in *sci-kit learn* machine learning library, is used to for this task. The LabelEncoder method is used to convert a label to a value between 0 and, $N_{classes} - 1$ where N is the number of distinct labels. If a label is repeated, it is assigned the same value as assigned earlier. Table 4.6 shows a sample of the dataset after label encoding preprocessing step.

4.4.2.4 Conversion of Spatio-temporal Data to *Image-like* Data

The next stage of data preprocessing is the conversion of the spatio-temporal data to *image-like* format. The data is prepared for training using spatio-temporal deep learning algorithms. These strategies for deep learning are necessary for *image-like* data

TABLE 4.6: Sample of Crime Dataset After Label Encoding "Primary Type" Column

Primary Type	Date	Longitude	Latitude
7	11/05/2020 12:00:00 AM	-87.633829	41.778099
1	10/28/2020 06:30:00 PM	-87.560801	41.755430
4	11/08/2020 11:59:00 AM	-87.752200	41.906760
3	10/03/2020 03:30:00 PM	-87.739415	41.804959
7	04/22/2020 12:03:00 AM	-87.600322	41.750752
4	04/13/2020 02:41:00 PM	-87.625944	41.708186
3	03/04/2020 06:30:00 PM	-87.611462	41.891990
1	06/03/2020 06:00:00 PM	-87.754647	41.933254
7	01/29/2020 12:45:00 AM	-87.603743	41.654377
3	10/01/2020 04:34:00 PM	-87.625817	41.866811
5	10/24/2020 01:45:00 PM	-87.635970	41.722834
7	08/06/2020 10:25:00 AM	-87.663945	41.769472
3	10/05/2020 12:00:00 PM	-87.710010	41.980202
7	07/10/2020 01:00:00 AM	-87.682111	41.771432
3	07/01/2020 06:00:00 AM	-87.587054	41.745510
5	06/18/2020 03:00:00 AM	-87.611231	41.748897
0	03/01/2020 05:25:00 PM	-87.626343	41.867429
0	02/01/2020 04:00:00 AM	-87.663734	41.988105
3	10/02/2020 09:00:00 AM	-87.683287	41.883937
3	06/19/2020 12:00:00 AM	-87.813444	41.937970

because they extract spatial information using CNNs. The *image-like* data would be a five-dimensional NumPy array with the shape: (*batch size, date, latitude, longitude, crime type*).

The batch size refers to the number of training samples used in a single training iteration. The date reflects the crime incident's temporal information, the latitude, and longitude indicate the crime incident's spatial information, and the crime type is the crime that occurred. The following is a summary of the procedures taken to turn the spatio-temporal data into *image-like* data:

1. *Spatial Feature Binning*: Feature binning [73] is a subset of feature engineering that deals with the transformation of a continuous variable to a categorical variable. Binning or discretization is a technique for converting a continuous or numerical variable into a categorical feature. Continuous variable binning adds nonlinearity and tends to improve model performance. It can also be used to find missing numbers or outliers.

   Binning can either be supervised or unsupervised. There are two ways for unsupervised binning, namely: *Equal Width Binning* and *Equal Frequency Binning*. Supervised binning has one technique called *Entropy-based Binning*. The study

uses unsupervised learning binning methods. In unsupervised binning, a numerical or continuous variable transforms into categorical bins without taking the target class label into account [74].

Using unsupervised learning binning techniques, the *Equal Width binning* method generates grids of equal length from the dataset. It classifies the continuous variable into many groups with bins or ranges of equal width, demonstrated by

$$w = \left[\frac{max - min}{x}\right] \tag{4.1}$$

where the categories are $[min, min+w-1]$, $[min+w, min+2w-1]$, $[min+2w, min+3w-1]$, ..., $[min+(x-1)w, max]$. Parameter x is the number of categories; w is the width of a category; and max, min are the maximum and minimum of the list.

Pandas binning method performed the spatial feature binning. Table 4.7 demonstrates a sample of the dataset before spatial feature binning.

TABLE 4.7: Sample of Crime Dataset Before Spatial Feature Binning

Primary Type	Date	Longitude	Latitude
5	10/29/2020 02:00:00 PM	-87.680739	41.765743
7	11/25/2020 11:15:00 PM	-87.606357	41.762215
2	06/08/2020 07:20:00 PM	-87.670976	41.999113
0	08/17/2020 11:30:00 AM	-87.644325	41.948347
0	09/03/2020 04:30:00 PM	-87.789972	41.911574
0	02/15/2020 12:37:00 AM	-87.646580	41.946212
1	05/31/2020 01:00:00 PM	-87.629831	41.739918
2	09/04/2020 01:20:00 PM	-87.637460	41.777216
7	08/06/2020 04:15:00 PM	-87.605225	41.750832
0	09/18/2020 08:30:00 AM	-87.697588	41.930897
7	08/09/2020 12:27:00 AM	-87.563411	41.753584
0	07/17/2020 01:00:00 PM	-87.637131	41.670583
2	04/27/2020 12:15:00 PM	-87.621020	41.737553
7	12/09/2020 06:20:00 PM	-87.704483	41.867562
6	03/06/2020 01:00:00 PM	-87.548047	41.751529
2	03/24/2020 01:22:00 PM	-87.716451	41.859938
5	11/13/2020 10:33:00 PM	-87.722228	41.961017
7	04/08/2020 10:30:00 AM	-87.619329	41.830607
0	10/16/2020 12:41:00 PM	-87.644457	41.903033
3	10/01/2020 05:00:00 AM	-87.632285	41.767969

Table 4.8 shows a sample of the dataset after spatial feature binning is performed, and the grids sizes are, 16×16 i.e. the longitude and latitude are in range (0-16).

TABLE 4.8: Sample of Crime Dataset After Spatial Feature Binning

Primary Type	Date	Longitude	Latitude
8	02/23/2020 06:30:00 AM	8	11
7	11/10/2020 05:00:00 PM	14	3
8	05/02/2020 03:00:00 PM	11	0
3	06/06/2020 02:00:00 PM	10	2
7	04/28/2020 10:01:00 AM	8	8
7	11/02/2020 08:38:00 PM	6	10
1	12/09/2020 11:27:00 PM	9	4
1	10/01/2020 10:00:00 PM	10	13
7	06/23/2020 08:17:00 PM	10	5
3	10/15/2020 08:00:00 AM	9	2
6	07/28/2020 02:53:00 PM	7	9
4	01/02/2020 11:45:00 AM	8	10
0	02/07/2020 09:50:00 PM	11	10
7	03/19/2020 09:51:00 AM	10	5
0	01/16/2020 09:00:00 PM	11	10
0	05/06/2020 12:58:00 PM	12	4
8	01/06/2020 12:00:00 PM	8	9
3	05/05/2020 10:30:00 PM	1	14
7	05/13/2020 05:54:00 PM	6	10
2	07/03/2020 05:45:00 PM	7	11

2. *Generating Input and Output Arrays*: The aim of this process is to establish an input array (image) of shape (*batch size, date, latitude, longitude, crime type*), and an output image of shape (*batch size, date, crime type*). Listing 4.1 is a code snippet which illustrates each incident map dimension corresponding to the features of the final dataset after data preprocessing.

LISTING 4.1: Code Snippet to Represent the Dimensions of a Crime Incident Map

```
import numpy as np
crime_incident_map = np.array(
    [ # batch size
        [ # date
            [ # latitude
                [ # longitude
                    [
                        # crime type
                    ],
                ],
            ],
        ],
    ]
)
```

Listing 4.2 is a Python code snippet which generates the input and output arrays.

LISTING 4.2: Converting Spatio-temporal Data to Image-like Data

```
import numpy as np

inputs = list()
outputs = list()

# NB: batch_items are crime incidents grouped in batches of 30-days
for batch_number, batch in batch_items:

    # Define input and output dimensions
    input_sample = np.zeros((
        date,
        latitude,
        longitude,
        crime_type
    ))

    output_sample = numpy.zeros(crime_type)

    # Populate Numpy arrays with 30-day spatio-temporal data
    for day, batch_crimes in batch:
        for crime, data in batch_crimes.groupby(crime_type):
            for _, row in data.iterrows():
                input_sample[day, row.latitude, row.longitude] += 1
                output_sample[crime] = crime

    inputs.append(input_sample)
    outputs.append(output_sample)

# Expand and combine the 30-day interval data
inputs = [np.expand_dims(d, axis=0) for d in np.array(inputs)]
inputs = np.concatenate(inputs, axis=0)

outputs = [np.expand_dims(d, axis=0) for d in np.array(outputs)]
outputs = np.concatenate(outputs, axis=0)
```

The spatial resolution of the resultant dataset is sufficient for crime analysis at the block level. In order to be consistent with the grid resolution ranges that were employed in [75], the finer block size resolution would have grids of 40×40 cells. Even with a finer resolution, the resulting grids are quite sparse, particularly for crime types with some incidents. Neighbourhoods have a coarser resolution, with cells that are approximately the size of their cell edges $l \approx 800m$, leading to grids of 16×16 cells. Rather than

altering the number of cells to match the actual distances between cities, the research area where the dataset is collected is overlaid with a fixed number of cells. The ultimate goal of crime prediction is to establish a system in which potential hotspots are predicted in advance by analysing recent instances. Thus, incident maps collect historical episodes I of timespan t of 1 day, and for a period T of 30 days. As a result, 30 daily incident maps are utilized to predict future crimes. To aggregate events, the study uses a daily time period x_i so that enough temporal detail can be extracted while the time series populates sufficiently.

4.4.2.5 Feature Scaling

The last step in the data preprocessing phase is *feature scaling*. When neural networks use gradient descent as an optimization strategy, data must be scaled. The gradient descent step size is affected by the feature value, X. The discrepancy in feature ranges will result in distinct step sizes for each feature. It is critical to scale the data before feeding it to the model so that the gradient descent moves smoothly towards the minima and that the gradient descent steps updates features at the same rate. Having features of similar scale helps the gradient descent's convergence. All characteristics will be translated into the range [0,1] in this stage, indicating that the minimum and maximum value of a feature/variable will be determined by 0 and 1, respectively. Fig 4.9 demonstrates how data point can be normalized by Min-Max scaling.

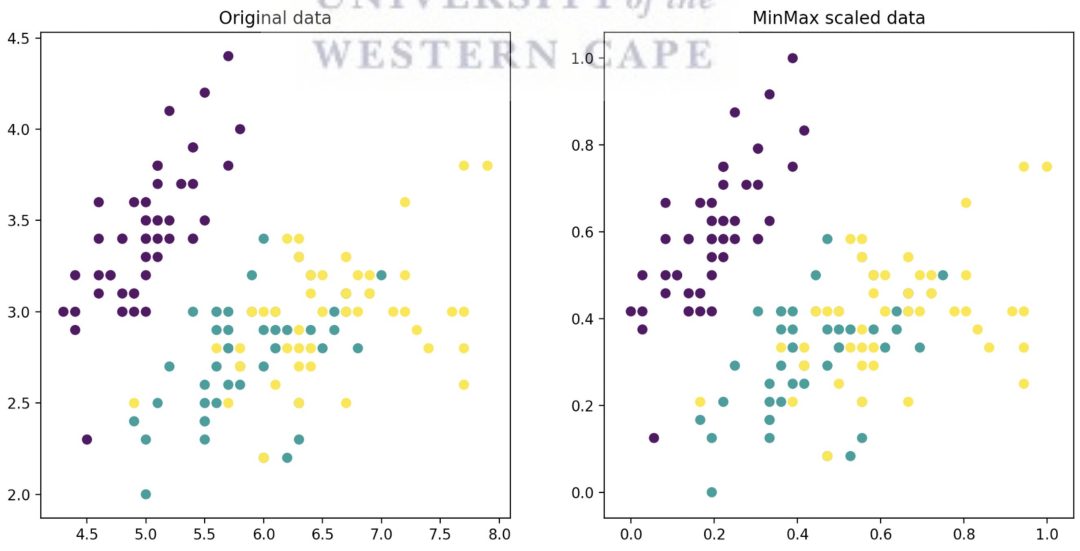


FIGURE 4.9: Min-Max scaling effect on features [8]

Variables assessed at different scales may not contribute equally to the model fitting and learning function and may result in bias. To address this possible issue, feature-wise normalization, such as Min-Max Scaling, is typically utilized prior to model fitting. This is especially important in deep learning techniques, where back-propagation can

be more reliable and even faster when input features are min-max scaled (or scaled in general) versus using the original unscaled data. The Min-Max scaling formula is given by

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.2}$$

where X' is the new value, X is the original value, if X_{max} is the maximum value of the column, and X_{min} is the minimum value of the column. The *MinMaxScalar* from the *sci-kit learn* is used for feature scaling the crime dataset.

4.4.3 Data Splitting

The risk in the training process is that the model will overfit to the training set. For example, the model may learn an extremely specialized function that works well on the training data but fails to generalize to data it has never seen before.

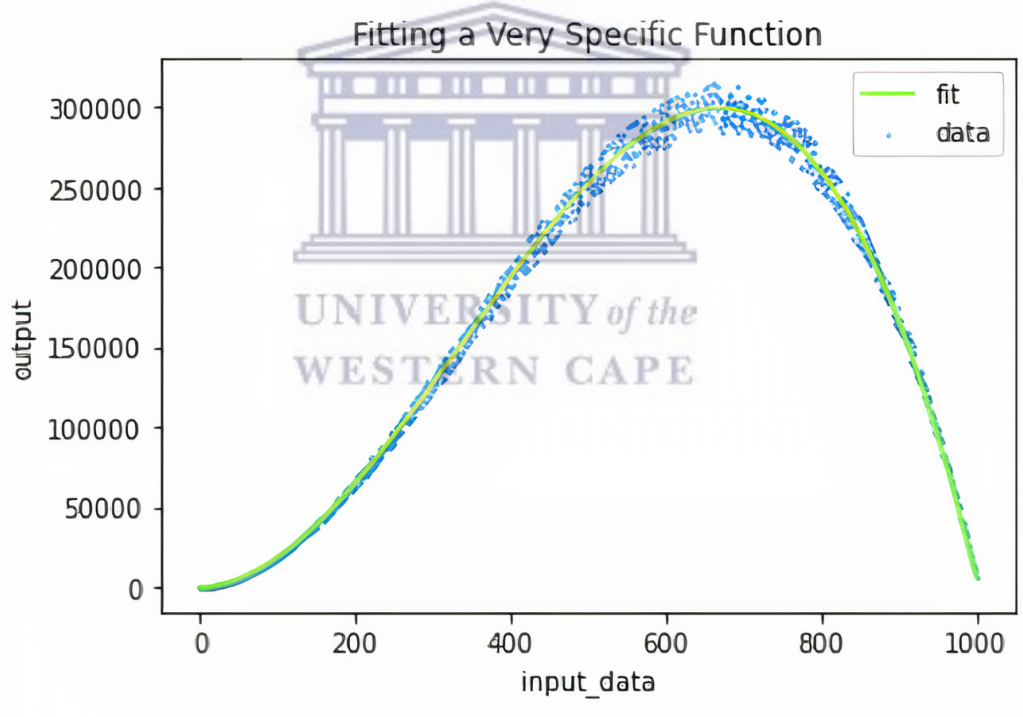


FIGURE 4.10: Example Overfitted Model [9]

If the model is hyper-specified to the training data, the loss function on the training data continues to exhibit decreasing values. The loss function on the held-out validation set, on the other hand, gradually increases. This is seen in the plot in Fig. **??** with two curves displaying the loss function values as training continues. This indicates that the model isn't learning very well and is instead of memorizing the training data. This suggests that the model will not perform well when presented with new data that it has never previously encountered. The train, validation, and testing segments help avoid

overfitting data. The training set is the largest corpus of data in the dataset, and it is used to train the model. It is also known as the test set. It is a separate component of the dataset that is utilized during training to gain an idea of how well the model performs on photos that are not being used in training. During training, it is a usual practice to report validation metrics, such as validation loss, on a continuous basis after each training period.

It is possible to tell when the model has achieved the best performance on the validation set by looking at these measures. After training experiments are complete, and results are recorded, it will be possible to predict how the model will perform on the validation set. In light of the fact that the validation set is utilized in model development, it is critical to maintain a completely independent strong control of data, namely the test set [75].

In this work, the dataset is split as follows: 60% is used for training; 10% for validation; and 30 % for testing. The train_test_split method in *sci-kit learn* was used for data splitting.

4.5 The Optimization Function

In this study, both the 3D and Spatio-temporal deep learning models are trained by using the Adam optimizer [76]. Deep learning relies heavily on stochastic gradient-based optimization. Many deep learning issues can be framed as the optimization of a scalar parameter objective function that requires maximum or minimization. Gradient descent is a relatively efficient optimization method if the function has different parameters, because computing first-order partial derivatives with respect to all parameters has the same processing complexity as evaluating the function [76]. Objective functions are frequently stochastic. Many objective functions, for example, are built on a sum of sub-functions assessed at distinct data sub-samples; in this instance, optimization can be made more efficient by executing gradient steps with regard to individual sub-functions, i.e. stochastic gradient descent (SGD) or ascent.

SGD is an efficient and effective optimization method, playing a key role in many deep learning techniques [63]. Additionally, objectives may have noise sources other than data subsampling, such as dropout regularization [77]. All of these noisy objectives require effective stochastic optimization approaches. The Adam optimizer optimized stochastic objectives with large parameter spaces. Higher-order optimization methods are inappropriate in these instances. Therefore, the discussion in this study is limited to first-order methods. The Adam optimizer is an excellent example of these methods,

and it consumes little memory.

4.6 Conclusion

The experimental design used in this research was described in this chapter. The experimental setup included the programming language utilized, the deep learning framework and API, the experimental platform, dataset analysis, data preparation stages, and a model optimization function for 3D and spatio-temporal deep learning models.

The programming language used to implement the deep learning algorithms was Python. Python was chosen because of its benefits of being independent across platforms, consistency and simplicity, rich frameworks and libraries and support for machine learning.

TensorFlow deep learning framework was used for developing, training the neural networks, under the hood of a Keras deep learning API. The experiments were carried out on a threaded Linux virtual machine hosted on Azure Cloud Services. The crime data was preprocessed using Pandas data analytics and *sci-kit* learn machine learning Python libraries. All of these elements formed part of the implementation in line with research objectives 2 and 3.

The next two chapters describe the procedures followed to further prepare the data ahead of classification or prediction, the process of optimization of the models trained, and the results that were obtained by testing the optimized models.

Chapter 5

3D Deep Learning for Crime Classification

This chapter describes the experimentation carried out to compare the 3D deep learning methods for crime classification. This is in line with research objective 4 which aims to provide an answer to research sub-question 1.1., and research objective 5 which aims to provide an answer to research sub-question 1.2.

The chapter begins by presenting further preprocessing steps that were carried out on the dataset to make the data suitable for crime classification (Section 5.1). The chapter further presents the evaluation metrics used to evaluate the results (Section 5.2), as well as describing the hyperparameter settings used to train the two 3D deep learning methods compared (Section 5.3). Finally, experiments are carried out and the ensuing results and findings are presented (Section 5.4). The chapter is concluded in Section 5.5.

5.1 Data Preprocessing for Crime Classification

Section 4.4 described preprocessing procedures for the crime data. In order to use the data for crime classification, the dataset is further preprocessed by selecting records in the three-year period (2017-2020) for training using the 3D deep learning algorithms. Fig. 5.1 shows the different samples of 10 crime types and their total occurrences. The dataset is filtered for a single crime type, i.e., "Theft", to suit a binary classification problem. All records with dates when "Theft" was reported are considered as the positive class, and the remaining records, when "Theft" was not recorded, are considered as the negative class. Thereafter, label encoding is performed on the crime type, i.e., the positive class was encoded with 1, and 0 for the negative class. Fig. 5.2 indicates results of the dataset for "Theft" as the chosen positive class. Spatial grid resolution ranges are adopted from [19]. These ranges are 16×16, 24×24, 32×32, and 40×40 cells. The

model evaluates past recorded events to predict whether "Theft" incidents happened in a given area in the particular month. Thus, past crime incidents are grouped into incident maps I of time span t of 1 day, and for a period T of 30 days. Therefore, 30 daily incident maps are used as input to forecast "Theft" incidents for the next period. A daily timespan for incidents so that enough temporal details can be extracted while the time series was adequately populated.

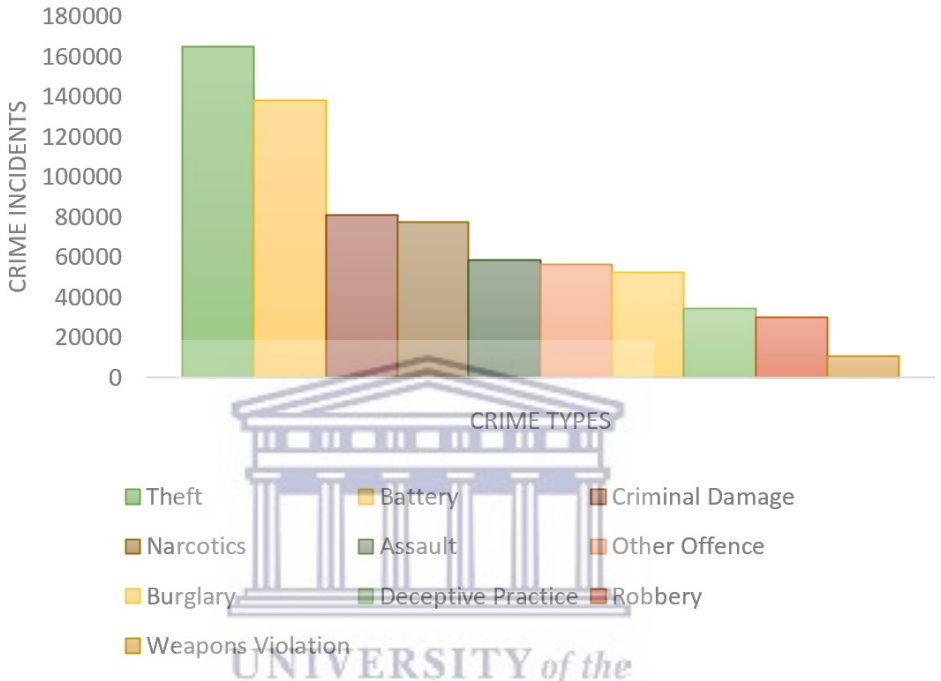


FIGURE 5.1: Total incidents in Chicago crime dataset for period 2017-2020

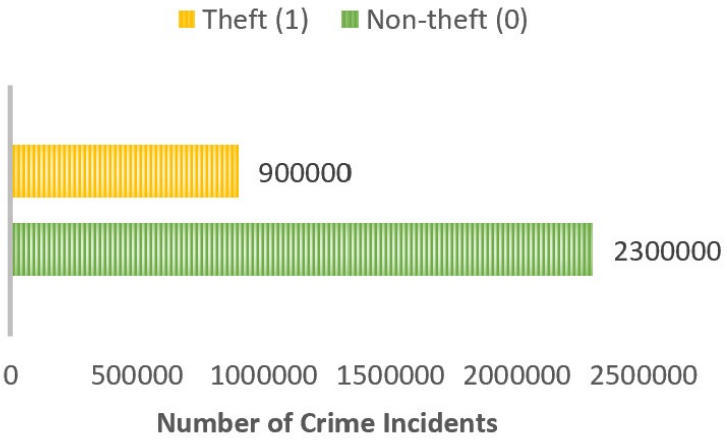


FIGURE 5.2: Total Theft incidents in Chicago Crime Dataset Recorded from 2017 to 2020

5.2 Classification Performance Metrics

The metrics used for crime classification are as follows:

1. *Accuracy*: It is the measurement used to determine which model is best at identifying relationships and patterns between variables in a dataset based on the input, or training, data. Accuracy is calculated using

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

where TP is the True Positive values, TN is the True Negative values, FP is the False Positive values and FN is the False Negative values.

2. *F1 score*: It is the harmonic mean of precision and recall. *F1 score* is preferred over *Accuracy* because it provides a much more objective view of the results, even with class imbalances. *F1 score* is defined as

$$F1\,score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.2}$$

where *Precision* tells us how many out of all instances that were predicted to belong to class X, actually belonged to class X, i.e. the fraction of relevant instances among the retrieved instances. Precision specifies how *reliable* a prediction of the positive class by the classifier is, while recall specifies how *comprehensive* the classifier is in locating all the available instances of the positive class in the test set [78]. *Precision* is defined as

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

Recall is the number of correct positive results divided by the number of positive results that should have been returned, that is, it is an expression of how many instances of class X were predicted correctly. *Recall* is defined as

$$Recall = \frac{TP}{TP + FN} \tag{5.4}$$

3. Area under a Precision-Recall Curve ($AUCPR$): This is a valuable performance metric for imbalanced data in a problem scenario where finding positive examples is important.

4. Area under the Receiver Operating Characteristic ($AUROC$): The area under the ROC curve shows the trade-off between true positive rate (TPR) and false positive

rate (FPR) across different decision threshold values. TPR and FPR are defined as

$$TPR = \frac{TP}{TP + FN} \tag{5.5}$$

and

$$FPR = \frac{FP}{FP + TN} \tag{5.6}$$

respectively.

5.3 Hyperparameter Settings

In the experiments, for both the 3D CNN and the 3D ResNet-18, the following parameters are used: a fixed convolutional filter size of 3×3; 100 training epochs; a batch size of 30; a learning rate of 0.01; and a temporal sliding window length of 30. The *binary cross entropy* function is used as the loss function. The Adam optimizer is used to optimize the loss function for training. The last layer is a dense layer which has 1 output node and uses a sigmoid activation function. The choice of these hyperparameters was guided by [2, 10, 19].

One hyperparameter that is not fixed and is the subject of investigation is the spatial resolution. One objective of this study is to evaluate the effect of different spatial resolutions on the performance of both 3D deep learning models for crime classification on the Chicago dataset. The resolutions that are compared vary from a minimum of $p = 16$ (i.e., a grid of 16×16 cells) to a maximum of $p = 40$ cells per grid, with a step size of $p = 8$ cells.

5.4 Crime Classification Results

This section evaluates the performance of the 3D deep learning methods using the aforementioned metrics in Section 5.2. The results are then compared and the effect of varying the spatial resolution is explored. The models are trained and tested over 4 different spatial resolutions i.e. 16, 24, 32 and 40 pixels. The results are then analysed and concluded. Section 5.4.1 discusses the performance of the 3D algorithms during training and Section 5.4.2 discusses the performance during testing.

5.4.1 Training Results

The 3D algorithms were trained over 100 epochs. The complexity, training loss and validation loss of these algorithms are discussed in the following subsections.

5.4.1.1 Model Complexity

Model complexity refers to the number of features included in a predictive or classification model. This can be determined by assessing model structure or model structure layout [79, 80]. Tables A.1, A.2, A.3 and A.4 in Appendix A show the architecture and complexity details of the 3D CNN model with spatial resolutions of 16, 24, 32, and 40 pixels respectively. Each of these tables show details of the network topology in terms of types of layers, output shapes produced by the layers and the number of parameters that are generated by each layer. The 3D CNN has a total of 17 layers, which consist of 1 input layer 4 convolutional layers, 3 activation layers, 3 dropout layers, 2 max-pooling layers, 1 flatten layer and 2 dense layers.

The input layer passes all the inputs into the network shape. The output shape of the input layer is (*batch_size, latitude, longitude, crime_type*), where latitude and longitude refer to the horizontal and vertical grid index, respectively, within the spatial grid consisting of 16, 24, 32 or 40 cells. The first 2 activation layers have ReLU activation functions, and the last activation layer uses a sigmoid function with an output shape of 1. The max pooling layers reduce the number of training parameters, thereby controlling model overfitting. In max pooling, the maximum value of each kernel in each depth slice is captured and passed on to the next layer. They also make the 3D CNN model invariant to distortion of the input data. To prevent overfitting during training, dropout layers randomly choose a fraction of units and set them to 0 at each update.

The flatten layer converts the input to a lower dimension. The dense layers are regular fully connected neural network layers. These dense layers feed all outputs from the previous layer to all its neurons, with each neuron providing one output to the next layer. The input, activation, max pooling, dropout, flatten, and dense layers do not introduce any trainable or non-trainable parameters to the network architecture. Non-trainable parameters are not optimizable during the learning process using gradient descent. There are no non-trainable parameters for the 3D CNN because all weights of the layers are updated during training with backpropagation. In general, as the spatial resolution increases, the dimensionality of the input shape also increases, thereby causing the output shapes to be different for each 3D CNN architecture. Furthermore, as the spatial resolution increases, the number of parameters in the network also increase. The interested reader may confirm these trends in Tables A.1 to Table A.4 in Appendix A.

Table B.1, B.2, B.3 and B.4 in Appendix B show the architecture and complexity of the 3D ResNet-18 model with spatial resolutions of 16, 24, 32, and 40 pixels respectively. These tables also detail network topology, the output shape of each layer and the

number of parameters that each network layer outputs. The network topology of the 3D ResNet is composed of 67 layers which include: 1 input layer, 20 convolutional layers, 17 batch normalization layers, 17 activation layers, 8 add layers, 1 max pooling layer, 1 average pooling layer, 1 flatten layer and 1 dense layer. Similar to the 3D CNN model, the activation layers all use ReLU activation function in all the layers except the last activation layer, which uses a sigmoid function. The 3D ResNet-18 also has an input, max pooling, flatten a dense layer which perform the same functions as described for the 3D CNN model architecture, and all which introduce no trainable or non-trainable parameters.

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. Hence, the batch normalization layer enables every convolution layer of the network to do learning more independently [81]. This layer normalizes the output of the previous layers. The activations scale the input layer in normalization. These batch normalization layers help the 3D ResNet-18 to efficiently regularized and avoid overfitting [82]. These layers are typically placed before activation layers or after convolution layers to standardize the inputs or outputs. The parameters generated by the batch normalization layer are non-trainable parameters.

The add layer "adds" a list of inputs. It takes as input a list of tensors, all the same shape, and returns a single tensor with the same shape as the inputs. This layer does not generate any parameters. The average pooling layer down-samples the input along its spatial dimensions, i.e. height and width. It does this by taking the average value over an input window for each channel of the input. The average pooling layer does not introduce any parameters.

Increasing the spatial resolution from 16 to 40 pixels has no effect on the number of total parameters, number of trainable parameters and number of non-trainable parameters. It only affects the shape of the inputs and outputs in the network. Therefore, it can be concluded that spatial resolution has negligible effect on the model complexity of the 3D ResNet-18. A high model complexity, in terms of model topology and number of trainable parameters, can enable a model to be efficient and to generally exhibit very a good predictive performance. On the downside, a high complexity can lead to overfitting, i.e. poor predictive performance during testing [80].

Considering the model complexity of the two 3D algorithms in terms of trainable parameters comparatively, the 3D ResNet-18 generally has a higher complexity than the

3D CNN. Also, the 3D CNN's number of trainable parameters is affected more when varying the spatial resolution, whilst the 3D ResNet-18 trainable parameters remain constant. Both the 3D networks have methods of coping with overfitting by the use of layers such as batch normalization, max pooling and dropout layers. Therefore, all things considered, the 3D ResNet-18 is expected to exhibit better performance than the 3D CNN network based on the model complexity. Table 5.1 summarizes the network topology of the two 3D algorithms.

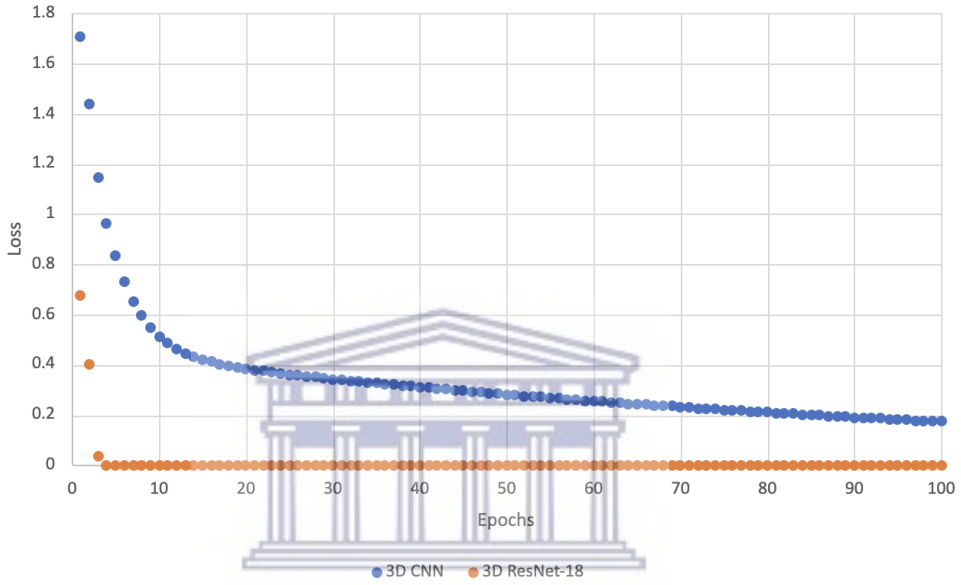
TABLE 5.1: Network Topology of 3D deep learning methods

	Method	
Layer	3D CNN	3D ResNet-18
Input	1	1
Convolutional	4	20
Activation	3	17
Add	0	8
Max Pooling	3	1
Average Pooling	0	1
Dropout	3	0
Batch Normalization	0	17
Flatten	1	1
Dense	2	1
Total Layers	17	67

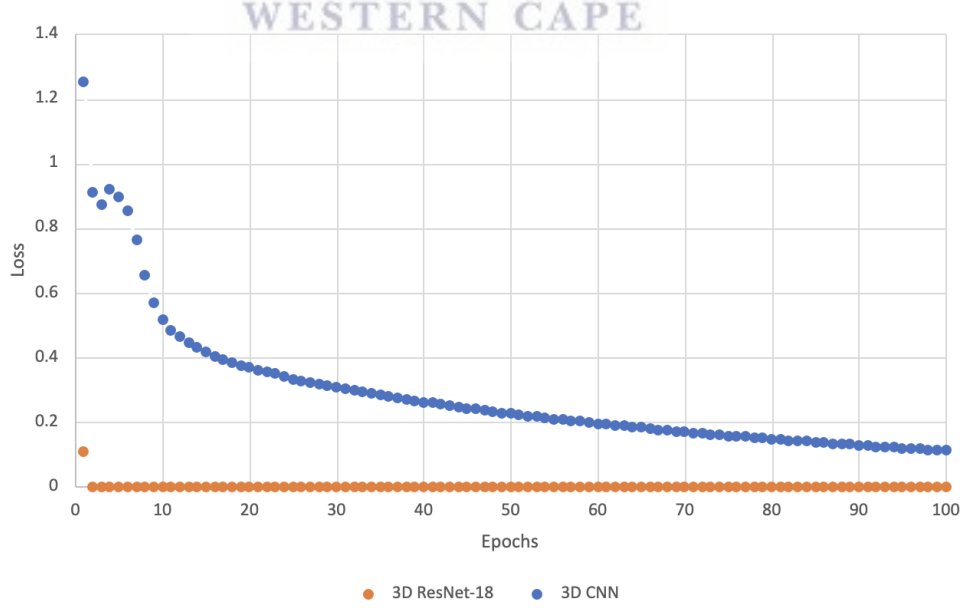
5.4.1.2 Training and Validation Loss

Training loss is the error on the training set of data. The training loss describes how well the model fits the training data. It can be seen that the 3D ResNet-18 converges faster than the 3D CNN in every case on the training cycle. This is because residual networks tend to learn equally or better than regular CNNs [83]. For spatial resolutions 16, 24 and 32 pixels, it is observed that the 3D ResNet-18 converges down to zero and the 3D CNN converges to at or below the 0.2 mark. The models with a spatial resolution of 40 are an exception to this trend, whereby the 3D ResNet-18 and 3D CNN converge to the 0.6 mark. Repeated runs of the training procedure produced a similar outcome. While it is challenging to determine exactly why this takes place, one possible reason for this may be attributed to the larger spatial resolution size which may exceed an implicit threshold in which numerous crimes are being placed into fewer increasingly larger grid cells, resulting is increased relative difficulty of the classification problem. It remains to be seen whether this significantly impacts the testing results, which are discussed in a later section.

Validation loss provides an indication during training of how well the model can perform on unseen data. The validation results show comparisons of the validation losses of the 3D algorithms at different spatial resolutions sizes. Very similar to the training loss curves observed previously, it is evident from these results that the 3D ResNet-18 also generalises better than the 3D CNN. It is also notable that at a spatial resolution of 40 pixels, the validation loss of the two models converge to the 0.6 mark as with the training curves, as has been discussed before.

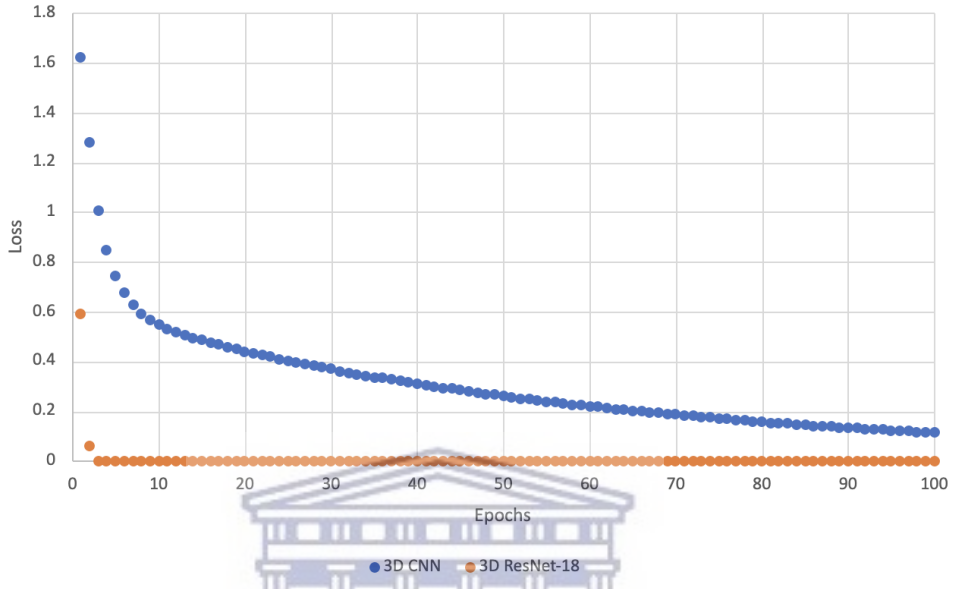


(a) Training loss at 16 pixels of spatial resolution

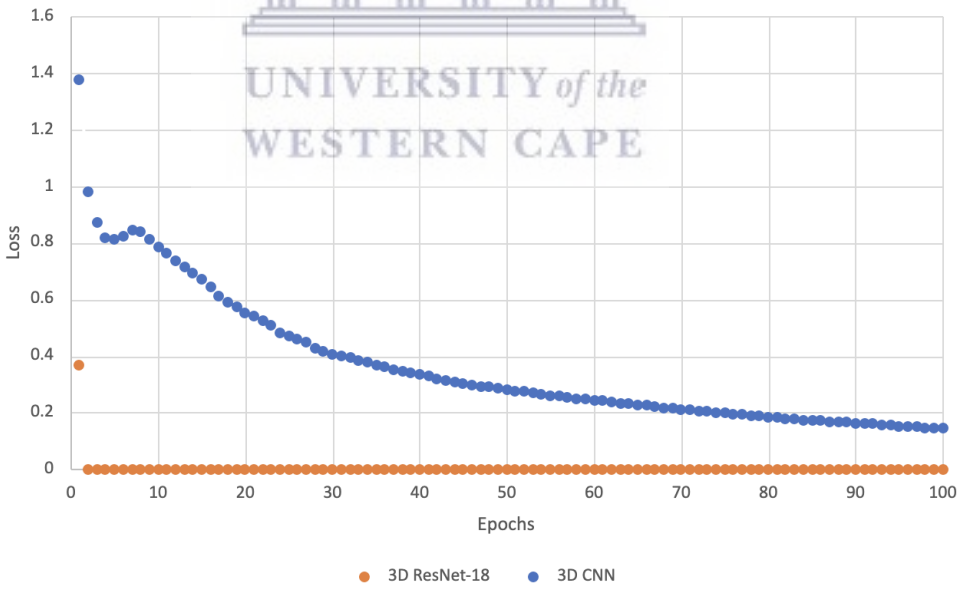


(b) Validation loss at 16 pixels of spatial resolution

FIGURE 5.3: Training and loss of 3D algorithms at 16 pixels of spatial resolution

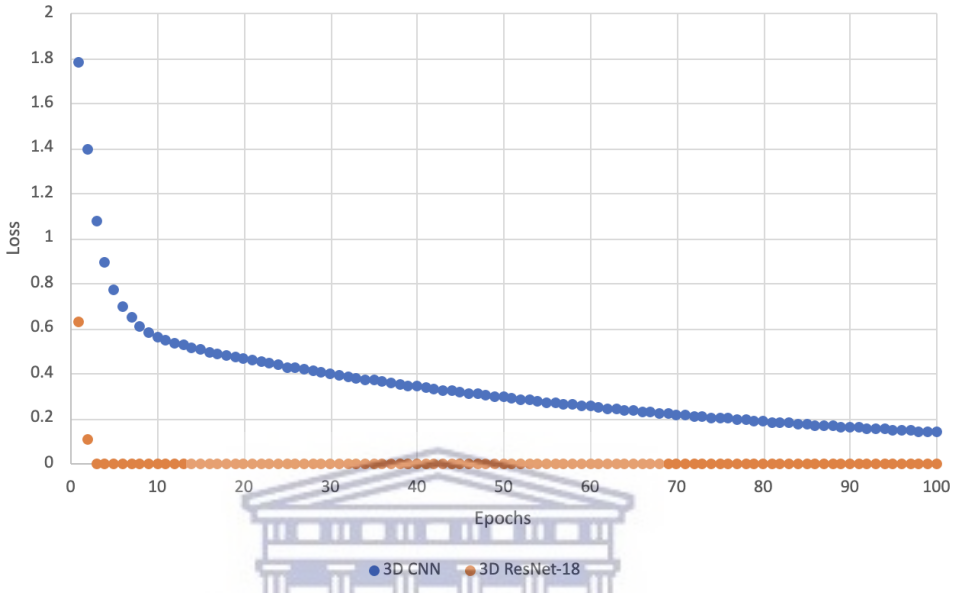


(a) Training loss at 24 pixels of spatial resolution

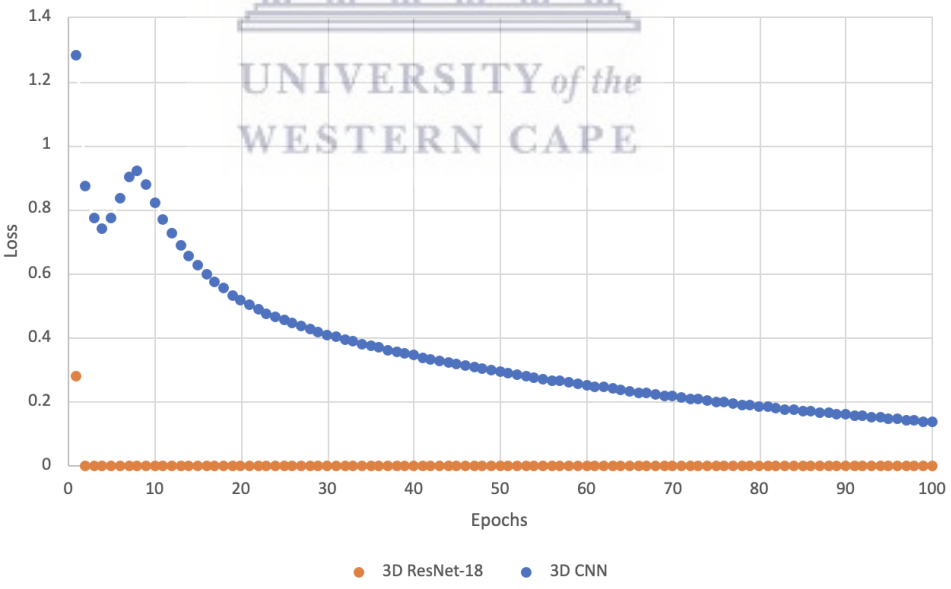


(b) Validation loss at 24 pixels of spatial resolution

FIGURE 5.4: Training and loss of 3D algorithms at 24 pixels of spatial resolution

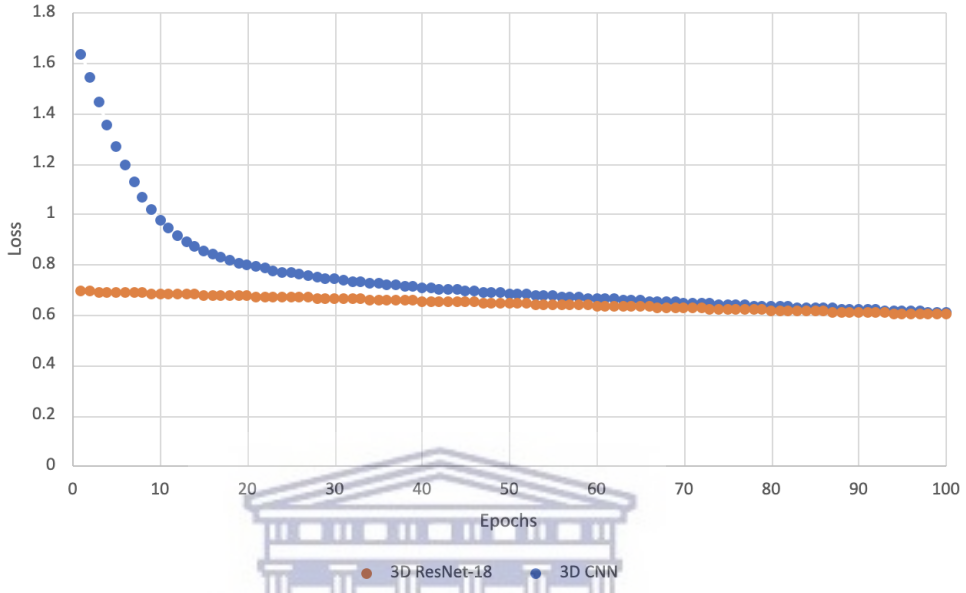


(a) Training loss at 32 pixels of spatial resolution

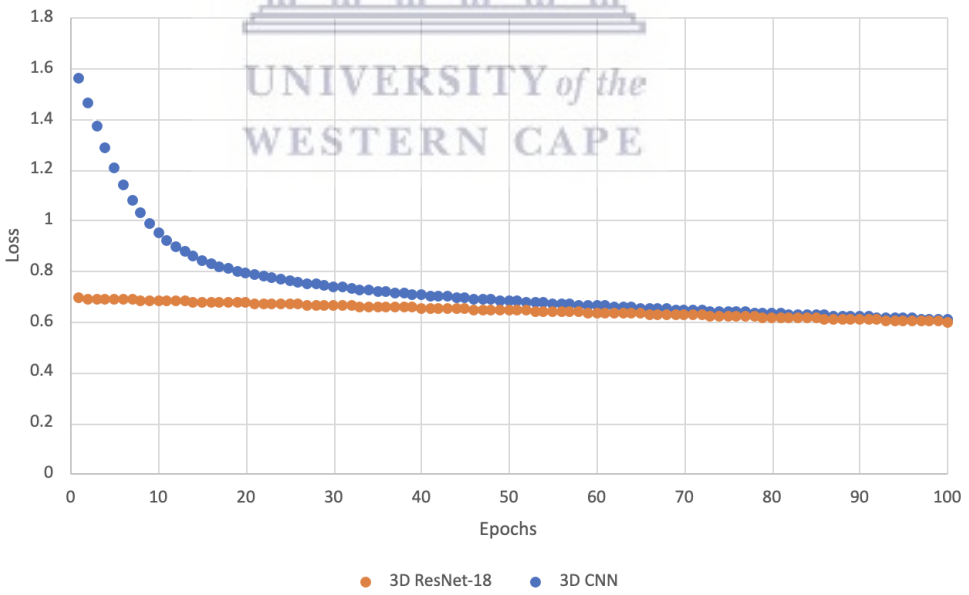


(b) Validation loss at 32 pixels of spatial resolution

FIGURE 5.5: Training and loss of 3D algorithms at 32 pixels of spatial resolution



(a) Training loss at 40 pixels of spatial resolution



(b) Validation loss at 40 pixels of spatial resolution

FIGURE 5.6: Training and loss of 3D algorithms at 40 pixels of spatial resolution

5.4.1.3 Training Time

The time taken to train each of the 3D algorithms at different spatial resolutions is shown in Fig. 5.7. It is evident that the 3D ResNet-18, which is the more complex model, takes more time to train than the 3D CNN. Furthermore, increasing the spatial resolution increases the training time of both models.

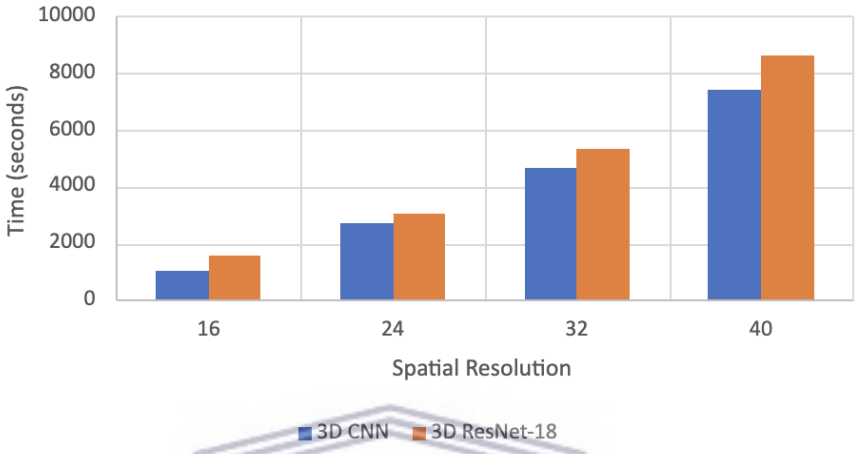


FIGURE 5.7: Training Time For 3D Algorithms

5.4.2 Testing Results

The testing results describe the accuracy, F1 score, AUCPR and AUROC results of the 3D algorithms at different spatial resolutions. Table 5.2 shows the accuracy results for the 3D algorithms at different spatial resolutions. The accuracy of the winning algorithm is highlighted in bold.

TABLE 5.2: Accuracy for 3D Deep Learning Algorithms

Spatial Resolution	Accuracy	
	3D CNN	3D ResNet-18
16	0.9946	**0.9984**
24	0.9926	**0.9961**
32	0.9922	**0.9945**
40	0.9910	**0.9920**

From the accuracy results, it is immediately observed that all of the results are near-perfect and very close. Considering the convergence graphs of the models with spatial resolution 40 observed previously, this indicates that the relevant models are still able to provide excellent testing results. This is a very encouraging result.

Comparing the two 3D deep learning methods for each spatial resolution size, it is evident that the 3D ResNet-18 exhibits the best accuracy across all spatial resolutions. In

terms of spatial resolution, the highest accuracies for both models is achieved with a spatial resolution size of 16 pixels, and as the spatial resolution increases, the accuracy of both models is decreases, although by only very minute amounts. This contradicts how most CNN networks perform for image classification, i.e. what should be observed is for a higher spatial resolution size to result in higher accuracy, but these results show the opposite [84]. However, this decrease in crime classification performance as the spatial resolution increases is consistent with those in [19].

Table 5.3 shows the F1 score results for the 3D algorithms at different spatial resolutions. The F1 score of the winning algorithm is highlighted in bold. The F1 score results confirm that the 3D ResNet-18 exhibits the best F1 score across all spatial resolutions. In terms of spatial resolution, the highest F1 scores for both models is achieved with 16 pixels, and as the resolution is increases the F1 score for both models decreases by a very small amount. The F1 score results are consistent with those in [19].

TABLE 5.3: F1 score for 3D Deep Learning Algorithms

Spatial Resolution	F1 Score	
	3D CNN	3D ResNet-18
16	0.9764	**0.9942**
24	0.9731	**0.9888**
32	0.9659	**0.9933**
40	0.9525	**0.9913**

Table 5.4 shows the AUCPR results for the 3D algorithms at different spatial resolutions. The AUCPR of the winning algorithm is highlighted in bold. Once again, it is observed that the AUCPR results of the 3D ResNet-18 are best across all spatial resolutions. Also, in terms of spatial resolution, the greatest AUCPR for both models is achieved with 16 pixels, and as the spatial resolution is increasing the AUCPR for both models is decreasing.

TABLE 5.4: AUCPR for 3D Deep Learning Algorithms

Spatial Resolution	AUCPR	
	3D CNN	3D ResNet-18
16	0.9766	**0.9933**
24	0.9734	**0.9888**
32	0.9666	**0.9933**
40	0.9526	**0.9913**

Table 5.5 shows the AUROC results for the 3D algorithms at different spatial resolutions. The AUROC of the winning algorithm is highlighted in bold. The AUROC results confirm finally that the 3D ResNet-18 has dominated the 3D CNN at all the spatial resolutions. Also, it is observed that the AUROC for the 3D ResNet-18 is the same across all spatial resolutions. From these results it can be seen that the spatial resolution only affects the AUROC of the 3D CNN, although only by a minute amouont, i.e. when the spatial resolution increases, the AUROC decreases by a very small amount. The best results for the 3D CNN are achieved when the spatial resolution is 16 pixels.

TABLE 5.5: AUROC for 3D Deep Learning Algorithms

Spatial Resolution	AUROC	
	3D CNN	3D ResNet-18
16	0.9998	**0.9999**
24	0.9997	**0.9999**
32	0.9985	**0.9999**
40	0.9966	**0.9999**

Based on the training results, it is evident that the 3D ResNet-18 is a far more complex model than the 3D CNN and this complexity helps it to learn and validate better than the 3D CNN as shown by the training and validation loss results. The accuracy, F1 score, AUCPR and AUROC for both models obtained are very high which indicates that both models perform very well for the binary classification problem at hand. In addition, the 3D ResNet-18 has shown better results in terms of F1 score, AUCPR and AUROC, than the 3D CNN at different spatial resolutions. Although increasing the spatial resolution made the feature maps sparser, it is evident from the results that there is a very slight deterioration in the performance of both models. The performance results of the 3D ResNet-18 over the 3D CNN are consistent with those observed in other domains [20, 51].

5.5 Conclusion

This chapter presented the experimentation carried out to compare the 3D deep learning methods for crime classification. This included a description of: the further preprocessing steps taken to prepare the dataset for crime classification, the metrics used to evaluate the models, and the hyperparameter settings used to train the models.

The 3D CNN and 3D ResNet were trained and tested with four spatial resolution sizes. Two comparisons were carried out: (i) a comparison of the effectiveness of the two 3D

deep learning methods; and (ii) a comparison of the effect of the different spatial resolution sizes on classification. In so doing, research objectives 4 and 5 were successfully achieved. In response to research sub-question 1.1., it is stated that the results showed that both 3D deep learning techniques were very effective on the Chicago dataset, but the 3D ResNet exhibited a comparatively better effectiveness than the 3D CNN.

In response to research sub-question 1.2., it is stated that, of the four spatial resolution sizes compared, i.e. squares of sides 16, 24, 32 and 40, all the resolution sizes resulted in comparable and near-perfect classification performance, but a clear and consistent trend was observed whereby a smaller spatial resolution results in a marginally better classification performance than a larger spatial resolution size which is consistent with the results in [19].

As such, both research sub-questions have been answered. The next chapter focuses on the comparison of three spatio-temporal deep learning methods for crime prediction.

Chapter 6

Spatio-temporal Deep Learning for Crime Prediction

This chapter describes the experimentation carried out to compare the spatio-temporal deep learning methods for crime prediction, in line with the final research objective 6 which aims to provide an answer to the final research sub-question 2.1.

The chapter starts with a description of the appropriate evaluation metrics used to evaluate the results in this experiment (Section 6.1), in addition to a description of the hyperparameter settings used to train the three spatio-temporal deep learning methods compared (Section 6.2). The chapter then describes the experiments and the ensuing results and findings (Section 6.3). The chapter is concluded in Section 6.4.

6.1 Crime Prediction Metrics

The metrics used for crime prediction are described as follows;

1. Root Mean Square Error (RMSE): It is a quadratic scoring rule which measures the average magnitude of the error. RMSE refers to the difference between forecast and corresponding observed values, which are each squared and then averaged over the sample. Finally, the square root of the average is taken. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE is most useful when large errors are particularly undesirable. RMSE was calculated by using

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(\frac{d_i - f_i}{\sigma_i}\right)^2} \tag{6.1}$$

where n is the number of observations, d_i is the actual observations time series, f_i is the estimated time series, and σ_i is the standard deviation.

2. Mean Absolute Error (MAE): It measures the average magnitude of the errors in a set of forecasts, without considering their direction. It measures accuracy for continuous variables. The equation was given in the library references. When expressed in words, the MAE is the average over the verification sample of the absolute values of the differences between the forecast and the corresponding observation. The MAE is a linear score, which means that all the individual differences are weighted equally in the average. MAE was calculated by using

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{6.2}$$

where n is the number of observations, y_i is the actual value and x_i is the predicted value.

6.2 Hyperparameter Settings

In the experiments, the hyperparameters are based on the performance on the validation set. Table 6.1 has the details for the hyperparameters used for all the spatio-temporal algorithms. The choice of these hyperparameters was guided by [4, 5, 19].

TABLE 6.1: Hyperparameter Settings for Spatio-temporal Deep Learning Techniques

Parameter	Value
Convolution Kernel Size	3×3
Convolution Filters	64
Spatial Grid Size	16×16
CNN Dropout	0.5
Temporal Sliding Window	30
LSTM Dropout	0.5
Learning Rate	0.01

For spatial information, the convolution kernel sizes to 3×3 with 64 filters. The size of each neighbourhood considered was set as 16×16. The number of layers we set as $K = 3$. The fixed length of the temporal sliding window was a value of 30, that is, the spatio-temporal networks would extract information of crime incidents that were reported in a period of 30 days. The batch size in our experiment was set to 64. Learning rate was set as 0.001. Both dropout and recurrent dropout rate in LSTM layers was set as 0.5. Early stopping callbacks are used to stop the training when the training loss in the next training step is greater than the previous step for a set threshold. λ is set as 0.5 to balance start and end volume. The models were trained over 100 epochs.

6.3 Crime Prediction Results

In this section, the results of the performance of the ST-ResNet, DMVST-Net and STD-Net for crime prediction based on the Chicago crime dataset are provided. Section 6.3.1 discusses the performance of the spatio-temporal algorithms during training and gives an analysis of the training and validation losses. Section 6.3.2 gives an analysis of the performance of the spatio-temporal models on the test set using the aforementioned metrics in Section 6.1.

6.3.1 Training and Validation Results

This section evaluates the performance of the spatio-temporal algorithms during training over 10 epochs. The decision to use 10 epochs was arrived at by following the work in [3, 4]. Furthermore, preliminary training showed that the losses converge after two epochs. Fig. 6.1, 6.2, and 6.3 show results of the training and validation losses for the different spatio-temporal models. In these graphs, the green line depicts the loss during training, while the yellow line depicts the loss during validation.

From the graphs, it is evident that loss during training drops in each training iteration, hence the ST-ResNet, the DMVST-Net and the STD-Net algorithms are all able to interpret the data points well during training. Likewise, the validation curve for each model converges to a lower value after each epoch.

Another observation is that the validation losses are lower than the training losses for all the models. This indicates that all models are not overfitting the training data and might perform well during testing with new data. It was also evident that the STD-Net converges to the lowest training and validation losses after training over 100 epochs. On the other hand, the validation losses are all below 0.2 for all algorithms, therefore the models are able to generalise well.

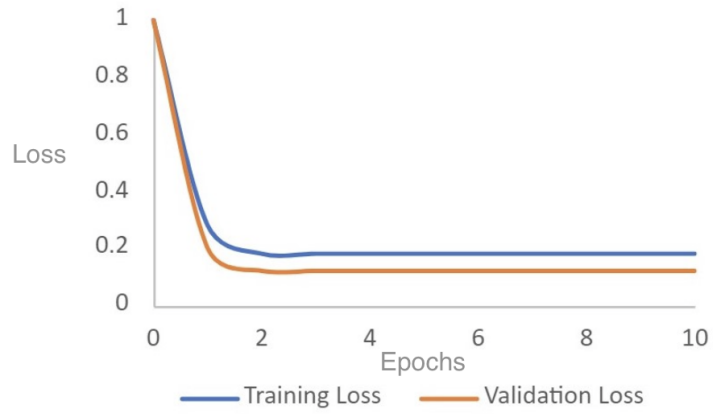


FIGURE 6.1: ST-ResNet Training and Validation Loss

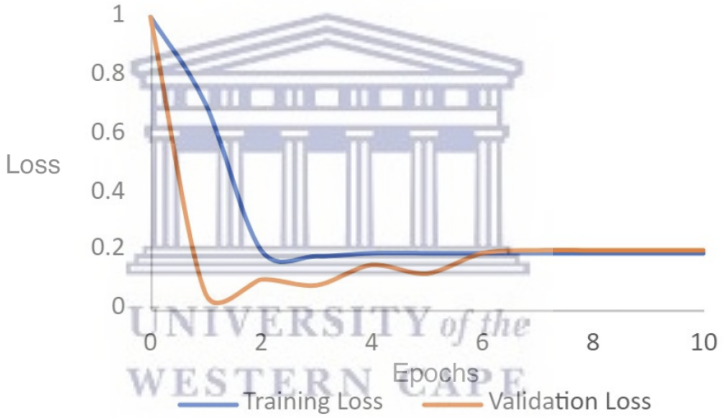


FIGURE 6.2: DMVST-Net Training and Validation Loss

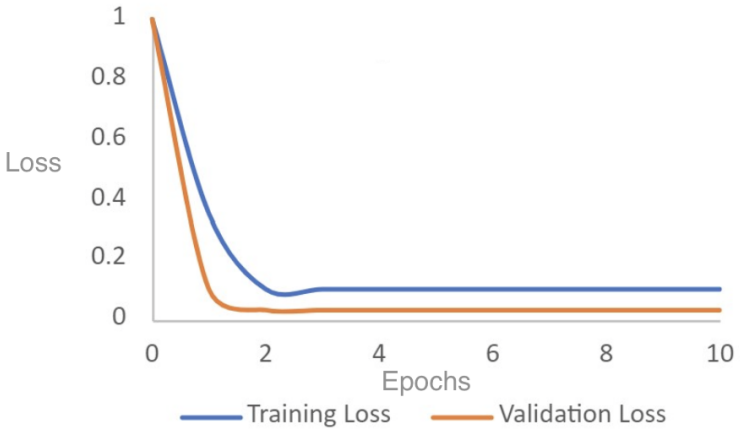


FIGURE 6.3: STD-Net Training and Validation Loss

6.3.2 Test Results

Fig. 6.4 shows the MAE achieved by the spatio-temporal models. Since MAE is the magnitude of difference between the prediction of an observation and the true value of that observation, it can help to identify the model which will perform better for predicting crime. A good MAE is considered to be one with a very low value. The STD-Net achieved the lowest MAE with a value of 0.2093, followed by the ST-ResNet with a MAE of 0.3278 and lastly the DMVST-Net with 0.3455. This indicates that a given prediction output (in number of crimes) by all three of the models has a very low error e.g. for the STD-Net, a given predicted output n can be represented as $n \pm 0.2093$ crime incidents, which indicates that the output is correct to a fraction of a single crime, which is a very confident result. The same is true for the ST-ResNet and DMVST-Net.

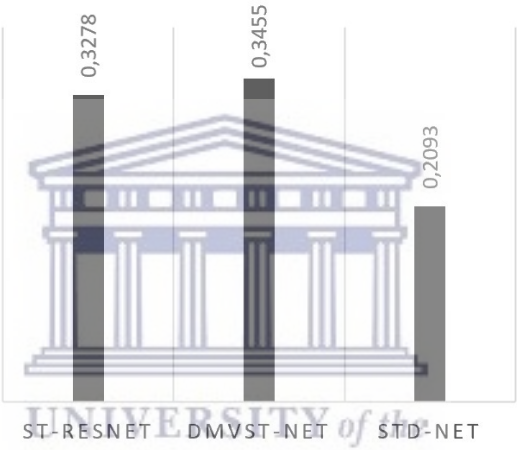


FIGURE 6.4: Comparison of MAE for Crime Prediction for the Spatio-temporal Deep Learning Techniques

Fig. 6.5 shows the RMSE achieved by the spatio-temporal models. Since RMSE is the standard deviation of prediction errors, a low RMSE would indicate that the models are accurate at predicting crime. It should also be noted that the RMSE differs from the MAE in that it is more sensitive to outliers and will therefore penalize outliers more. From the RMSE results, it is observed that all three models have very low values; the STD-Net once again obtained the lowest value of 0.287, followed by the ST-ResNet with 0.4033 and lastly the DMVST-Net with 0.4171. This is a very encouraging result that indicates that all three models are robust to outliers.

The STD-Net once again emerges as the most accurate model, whereby a given predicted output n by the STD-Net can be represented as $n \pm 0.287$ crime incidents, clearly demonstrating an accurate output.

FIGURE 6.5: Comparison of RMSE for Crime Prediction for the Spatio-temporal Deep Learning Techniques

From the MAE and RMSE results, it is noticeable that the MAE results for all the models are lower but almost equal to their corresponding RMSE results. This is attributed to the fact that the RMSE is more sensitive to outliers than the MAE. All the spatio-temporal models have similar structures and components. All these models have achieved low MAE and RMSE results. However, the STD-Net was clearly the best performing algorithm based on the MAE and RMSE.

The STD-Net achieves the best performance results due to its ability to explicitly model spatial dependency and sequential temporal dependency better than the ST-ResNet and DMVST-Net. These results are consistent with the results achieved in the prediction of citywide traffic crowd flows [5] and supply-demand prediction for autonomous vehicles [21].

6.4 Conclusion

This chapter presented the experimentation carried out to compare the spatio-temporal deep learning methods for crime prediction. This included a description of the metrics used to evaluate the models and the hyperparameter settings used to train the models.

The three spatio-temporal deep learning techniques, i.e. the ST-ResNet, DMVST-Net and STD-Net, were trained and tested on the Chicago crime dataset. Then, the three techniques were compared in terms of effectiveness towards crime prediction. In so doing, the final research objective 6 was successfully achieved.

In response to the final research sub-question 2.1., it is stated that all three spatio-temporal techniques successfully predicted crime on the Chicago dataset with low error values; the ST-ResNet, which is considered as the state-of-the-art for crime prediction, was out-performed by the STD-Net with a noticeable reduction in error; the DMVST-Net, while providing low error, did not out-perform either the ST-ResNet or the STD-Net. Noting that STD-Net and DMVST-Net have not, to the knowledge of the researcher, been applied to crime prediction, these results and findings are consistent with the results found in related studies [3, 5, 21, 23] of the same three spatio-temporal deep learning techniques compared in other spatio-temporal prediction domains.

The STD-Net outperforms the ST-ResNet and DMVST-Net due to its ability to sufficiently model spatial dependency and sequential temporal dependency, this is consistent with results achieved in other spatial-temporal domains [5, 21].

The next chapter concludes the thesis with an answer to the main research question posed in Chapter 1.

Chapter 7

Conclusion and Future Work

This research aimed to investigate the use of deep learning techniques for crime forecasting. Effective crime forecasting can be very beneficial to law enforcement. One application of this is to help assign law enforcement resources to crime hotspots and potentially prevent crime incidents before they occur.

Given the prevalent success of deep learning techniques in a variety of domains, including spatio-temporal forecasting domains, this research mainly aimed to investigate the question posed as "How effective are 3D and spatio-temporal deep learning techniques for crime forecasting?"

It was not known how effective 3D deep learning techniques, specifically the 3D CNN and 3D ResNet, are in the context of crime classification. Also, it was desired to investigate the effect of various spatial resolution sizes when preparing the dataset for classification in the domain of crime classification. These problems were represented in research sub-questions 1.1. and 1.2.

Furthermore, while the ST-ResNet spatio-temporal deep learning technique has been applied to crime prediction in the literature with state-of-the-art results, it was not known how this compares to other newer spatio-temporal deep learning techniques, specifically the DMVST-Net and STD-Net, which have been highly successful in other spatio-temporal forecasting domains in the literature. This problem was represented in research sub-question 2.1.

7.1 Conclusions

A series of experiments were carried out to investigate and answer the aforementioned sub-questions, towards obtaining an answer to the main research question.

In response to research sub-question 1.1., it was stated that the results showed that both 3D deep learning techniques were very effective on the Chicago dataset, but the 3D ResNet exhibited a comparatively better effectiveness than the 3D CNN.

In response to research sub-question 1.2., it was stated that, of the four spatial resolution sizes compared, i.e. squares of side 16, 24, 32 and 40, all the resolution sizes resulted in comparable and near-perfect classification performance, but a clear and consistent trend was observed whereby a smaller spatial resolution results in a marginally better classification performance than a larger spatial resolution size.

In response to the final research sub-question 2.1., it was stated that all three spatio-temporal techniques successfully predicted crime on the Chicago dataset with low error values; the ST-ResNet, which was previously considered as the state-of-the-art for crime prediction, was out-performed by the STD-Net with a noticeable reduction in error; the DMVST-Net, while providing a sufficiently low error, did not outperform either the ST-ResNet or the STD-Net. Noting that STD-Net and DMVST-Net have not, to the knowledge of the researcher, been applied to crime prediction, these results and findings are consistent with the results found in related studies [3, 5, 23] of the same three spatio-temporal deep learning techniques compared in other spatio-temporal prediction domains.

Having obtained these findings, a response to the main research question is stated as: the 3D and spatio-temporal deep learning techniques investigated are all highly effective for crime forecasting, but the 3D ResNet is most effective for crime classification and the STD-Net is most effective for crime prediction; and finally, the results indicate that a smaller spatial resolution enhances the effectiveness of deep learning techniques for crime forecasting.

It is the hope of the researcher that these results and findings will be of benefit to other researchers in the field or related fields, and that this research can help facilitate the uptake of these technologies in law enforcement to help mitigate the rampant crime that is endemic in South Africa and the world at large.

The next section concludes this chapter and the thesis as a whole with directions for future work.

7.2 Directions for Future Work

The following directions for future work:

- Explore the possibility of adding external data to these frameworks, similar to worldly semantics, socioeconomic, climate, road guides and focal points in the region, and evaluate how this impacts the performance of the 3D and spatio-temporal models for crime forecasting.
- Evaluate the ability of the models to generalise across different crime datasets.
- Explore optimization of hyperparameters by using techniques such as evolutionary computation.
- Evaluate the ability of 3D deep learning model to perform multi-class crime classification using different crime datasets.

Bibliography

[1] "Predpol – the predictive policing company," https://www.predpol.com/, (Accessed on 10/05/2021).

[2] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[3] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 1655–1661.

[4] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and Z. Li, "Deep multi-view spatial-temporal network for taxi demand prediction," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 2588–2595. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16069

[5] A. Ali, Y. Zhu, Q. Chen, J. Yu, and H. Cai, "Leveraging spatio-temporal patterns for predicting citywide traffic crowd flows using deep hybrid neural networks," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, 2019, pp. 125–132.

[6] "What is tensorflow: Deep learning libraries and program elements explained," https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow, (Accessed on 09/28/2021).

[7] "City of Chicago — Data Portal — City of Chicago — Data Portal," https://data.cityofchicago.org/, (Accessed on 09/28/2021).

[8] "Everything you need to know about min-max normalization: A python tutorial — by serafeim loukas — towards data science," https://towardsdatascience.com/

everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79, (Accessed on 01/07/2022).

[9] "Overfitting vs. underfitting: A complete example — by will koehrsen — towards data science," https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765, (Accessed on 01/06/2022).

[10] K. Hara, H. Kataoka, and Y. Satoh, "Learning spatio-temporal features with 3d residual networks for action recognition," *CoRR*, vol. abs/1708.07632, 2017. [Online]. Available: http://arxiv.org/abs/1708.07632

[11] D. Bottrell and A. France, *Bourdieurian Cultural Transitions: Young People Negotiating 'Fields' in Their Pathways into and Out of Crime*. London: Palgrave Macmillan UK, 2015, pp. 99–112. [Online]. Available: https://doi.org/10.1057/9781137377234_8

[12] "Types of crime," https://saylordotorg.github.io/text_social-problems-continuity-and-change/s11-02-types-of-crime.html, (Accessed on 10/05/2021).

[13] "Crime rate by country 2021," https://worldpopulationreview.com/country-rankings/crime-rate-by-country, (Accessed on 10/05/2021).

[14] "Crime prediction technologies," https://www.ict.org.il/Article/2408/Crime_Prediction_Technologies#gsc.tab=0, (Accessed on 11/06/2021).

[15] N. A. S. Zaidi, A. Mustapha, S. A. Mostafa, and M. N. Razali, "A classification approach for crime prediction," in *Communications in Computer and Information Science*. Springer International Publishing, 2020, pp. 68–78. [Online]. Available: https://doi.org/10.1007/978-3-030-38752-5_6

[16] A. Stec and D. Klabjan, "Forecasting crime with deep learning," 2018.

[17] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *Artificial Neural Networks — ICANN 2001*. Springer Berlin Heidelberg, 2001, pp. 87–94. [Online]. Available: https://doi.org/10.1007/3-540-44668-0_13

[18] H. Lee and H. Kwon, "Going deeper with contextual cnn for hyperspectral image classification," *IEEE Transactions on Image Processing*, vol. 26, no. 10, pp. 4843–4855, 2017.

[19] P. Stalidis, T. Semertzidis, and P. Daras, "Examining deep learning architectures for crime classification and prediction," *CoRR*, vol. abs/1812.00602, 2018. [Online]. Available: http://arxiv.org/abs/1812.00602

[20] X. Zhang, L. Han, W. Zhu, L. Sun, and D. Zhang, "An explainable 3d residual self-attention deep neural network for joint atrophy localization and alzheimer's disease diagnosis using structural mri," *IEEE journal of biomedical and health informatics*, vol. PP, 2021.

[21] Z. Zhang, Y. Li, H. Dong, Y. You, and F. Zhao, "Attention-based supply-demand prediction for autonomous vehicles," *CoRR*, vol. abs/1905.10983, 2019. [Online]. Available: http://arxiv.org/abs/1905.10983

[22] B. Wang, P. Yin, A. L. Bertozzi, P. J. Brantingham, S. J. Osher, and J. Xin, "Deep learning for real-time crime forecasting and its ternarization," *Chinese Annals of Mathematics, Series B*, vol. 40, no. 6, pp. 949–966, Nov. 2019. [Online]. Available: https://doi.org/10.1007/s11401-019-0168-y

[23] N. Awan, A. Ali, F. Khan, M. Zakarya, R. Alturki, M. Kundi, M. D. Alshehri, and M. Haleem, "Modeling dynamic spatio-temporal correlations for urban traffic flows prediction," *IEEE Access*, vol. 9, pp. 26 502–26 511, 2021.

[24] H. Zunair, A. Rahman, N. Mohammed, and J. P. Cohen, "Uniformizing techniques to process ct scans with 3d cnns for tuberculosis prediction," in *Predictive Intelligence in Medicine*, I. Rekik, E. Adeli, S. H. Park, and M. d. C. Valdés Hernández, Eds. Cham: Springer International Publishing, 2020, pp. 156–168.

[25] F. Yi, Z. Yu, F. Zhuang, and B. Guo, "Neural network based continuous conditional random field for fine-grained crime prediction," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 4157–4163. [Online]. Available: https://doi.org/10.24963/ijcai.2019/577

[26] G. Mohler, "Marked point process hotspot maps for homicide and gun crime prediction in chicago," *International Journal of Forecasting*, vol. 30, no. 3, pp. 491–497, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207014000284

[27] J. Q. Yuki, M. M. Q. Sakib, Z. Zamal, K. M. Habibullah, and A. K. Das, "Predicting crime using time and location data," in *Proceedings of the 2019 7th International Conference on Computer and Communications Management*, ser. ICCCM 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 124–128. [Online]. Available: https://doi.org/10.1145/3348445.3348483

[28] Z. Li, T. Zhang, Z. Yuan, Z. Wu, and Z. Du, "Spatio-temporal pattern analysis and prediction for urban crime," in *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, 2018, pp. 177–182.

[29] M. Kumar, S. Athulya, M. Minu M.B., V. Vinodini M.D., A. Lakshmi K.G., S. Anjana, and T. Manojkumar, "Forecasting of annual crime rate in india: A case study," in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2018, pp. 2087–2092.

[30] L. Alves, H. Rebeiro, and F. Rodregues, "Crime prediction through urban metrics and statistical learning," *Elsevier*, 2018.

[31] S. Agarwal, L. Yadav, and M. K. Thakur, "Crime prediction based on statistical models," in *2018 Eleventh International Conference on Contemporary Computing (IC3)*, 2018, pp. 1–3.

[32] A. Almaw and K. Kadam, "Crime data analysis and prediction using ensemble learning," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, June 2018, pp. 1918–1923.

[33] F. K. Bappee, A. Soares Júnior, and S. Matwin, "Predicting crime using spatial features," in *Advances in Artificial Intelligence*, E. Bagheri and J. C. Cheung, Eds. Cham: Springer International Publishing, 2018, pp. 367–373.

[34] S. Kim, P. Joshi, P. Kalsi, Singh, and P. Taheri, "Crime analysis through machine learning," *IEEE*, 2018.

[35] M. Vural, Sait and M. Gök, "Criminal prediction using naïve bayes theory," *Springer*, 2017.

[36] J. Khan, Rasheed, M. Saeed, F. Siddiqui, Ahmed, N. Mahmood, and Q. Arifeen, "Predictive policing: A machine learning approach to predict and control crimes in metropolitan cities," *USJICT*, 2019.

[37] N. Yadav, A. Kumar, R. Bhatnagar, and V. Verma, Kumar, "City crime mapping using machine learning techniques," *Springer*, 2019.

[38] I. Matijosaitiene, A. McDowald, R. Bhatnagar, and V. Juneja, "Predicting safe parking spaces: A machine learning approach to geospatial urban and crime data," *MDPI*, 2019.

[39] T. Shi, G. He, and Y. Mu, "Random forest algorithm based on genetic algorithm optimization for property-related crime prediction," in *2019 International Conference on Computer, Network, Communication and Information Systems*

(CNCI 2019). Atlantis Press, 2019/05. [Online]. Available: https://doi.org/10.2991/cnci-19.2019.73

[40] Y. Alparslan, I. Panagiotou, W. Livengood, R. Kane, and A. Cohen, "Perfecting the crime machine," 2020.

[41] A. Gupta, A. Mohammad, A. Syed, and M. N., "A comparative study of classification algorithms using data mining: Crime and accidents in denver city the USA," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 7, 2016. [Online]. Available: https://doi.org/10.14569/ijacsa.2016.070753

[42] S. Hossain, A. Abtahee, I. Kashem, M. M. Hoque, and I. H. Sarker, "Crime prediction using spatio-temporal data," in *Computing Science, Communication and Security*, N. Chaubey, S. Parikh, and K. Amin, Eds. Singapore: Springer Singapore, 2020, pp. 277–289.

[43] A. Kumar, A. Verma, G. Shinde, Y. Sukhdeve, and N. Lal, "Crime prediction using k-nearest neighboring algorithm," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 2020, pp. 1–4.

[44] S. Seo, H. Chan, J. Brantingham, J. Leap, P. Vayanos, M. Tambe, and Y. Liu, "Partially generative neural networks for gang crime classification with partial information," in *AIES '18: Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. New York, NY, USA: Association for Computing Machinery, 2018/12, pp. 257–263.

[45] M. Lim, A. Abdullah, N. Janjhi, and M. Supramaniam, "Hidden link prediction in criminal networks using the deep reinforcement learning technique," 2018.

[46] Y.-L. Lin, M.-F. Yen, and L.-C. Yu, "Grid-based crime prediction using geographical features," *ISPRS International Journal of Geo-Information*, vol. 7, no. 8, 2018. [Online]. Available: https://www.mdpi.com/2220-9964/7/8/298

[47] L. Elluri, V. Mandalapu, and N. Roy, "Developing machine learning based predictive models for smart policing," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2019, pp. 198–204.

[48] B. Wang, D. Zhang, D. Zhang, P. J. Brantingham, and A. L. Bertozzi, "Deep learning for real time crime forecasting," 2017.

[49] S. A. Chun, V. A. Paturu, S. Yuan, R. Pathak, V. Atluri, and N. R. Adam, "Crime prediction model using deep neural networks," in *Proceedings of the 20th Annual International Conference on Digital Government Research*. ACM, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3325112.3328221

[50] L. Duan, T. Hu, E. Cheng, J. Zhu, and C. Gao, "Deep convolutional neural networks for spatiotemporal crime prediction," in *Int'l Conf. Information and Knowledge Engineering*, 2017.

[51] S. B. Akintoye, L. Han, X. Zhang, H. Chen, and D. Zhang, "A hybrid parallelization approach for distributed and scalable deep learning," *CoRR*, vol. abs/2104.05035, 2021. [Online]. Available: https://arxiv.org/abs/2104.05035

[52] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[54] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," *CoRR*, vol. abs/1511.05298, 2015. [Online]. Available: http://arxiv.org/abs/1511.05298

[55] C.-H. Yu, W. Ding, M. Morabito, and P. Chen, "Hierarchical spatio-temporal pattern discovery and predictive modeling," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 4, pp. 979–993, 2016.

[56] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," 2016.

[57] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2015.

[58] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[60] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, p. 807–814.

[61] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[62] C. Sun, "Fundamental q-learning algorithm in finding optimal policy," in *2017 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2017, pp. 243–246.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[64] "A Python Book: Beginning Python, Advanced Python, and Python Exercises," https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html, (Accessed on 09/28/2021).

[65] "About Python™ — Python.org," https://www.python.org/about/, (Accessed on 09/28/2021).

[66] "Python (m."

[67] "Deep learning frameworks — nvidia developer," https://developer.nvidia.com/deep-learning-frameworks, (Accessed on 09/28/2021).

[68] "Keras: the python deep learning api," https://keras.io/, (Accessed on 09/28/2021).

[69] "Torch — Scientific Computing for Luajit." http://torch.ch/, (Accessed on 09/28/2021).

[70] "Doitpoms - TLP Library Tensors in Materials Science - what is a tensor?" https://www.doitpoms.ac.uk/tlplib/tensors/what_is_tensor.php, (Accessed on 09/28/2021).

[71] "3 Keras Design Patterns Every ML Engineer Should Know — by Dimitris Poulopoulos — Towards Data Science," https://towardsdatascience.com/3-keras-design-patterns-every-ml-engineer-should-know-cae87618c7e3, (Accessed on 09/28/2021).

[72] "Create your azure free account today — microsoft azure," https://azure.microsoft.com/en-gb/free/search/?&ef_id=CjwKCAjw-sqKBhBjEiwAVaQ9a32pzCCry_O4N30SZK8_u6aYt0DIBkzkQXggHCdMM8j5B6otj4MSVhoC4vcQAvD_BwE:G:s&OCID=AID2200255_SEM_CjwKCAjw-sqKBhBjEiwAVaQ9a32pzCCry_

O4N30SZK8_u6aYt0DIBkzkQXggHCdMM8j5B6otj4MSVhoC4vcQAvD_BwE:G:s&gclid=CjwKCAjw-sqKBhBjEiwAVaQ9a32pzCCry_O4N30SZK8_u6aYt0DIBkzkQXggHCdMM8j5B6otj4MSVhoC4vcQAvD_BwE, (Accessed on 09/28/2021).

[73] T.-H. Nguyen, T.-T. Phan, C.-T. Dao, D.-V.-P. Ta, T.-N.-C. Nguyen, N.-M.-T. Phan, and H.-N. Pham, "Effective disease prediction on gene family abundance using feature selection and binning approach," in *IT Convergence and Security*, H. Kim and K. J. Kim, Eds. Singapore: Springer Singapore, 2021, pp. 19–28.

[74] "Feature engineering — deep dive into encoding and binning techniques — by satyam kumar — towards data science," https://towardsdatascience.com/feature-engineering-deep-dive-into-encoding-and-binning-techniques-5618d55a6b38, (Accessed on 10/03/2021).

[75] C.-H. Yu, W. Ding, M. Morabito, and P. Chen, "Hierarchical spatio-temporal pattern discovery and predictive modeling," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 4, pp. 979–993, 2016.

[76] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2015.

[77] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[78] M. Ghaziasgar, J. Connan, and A. B. Bagula, "Enhanced adaptive skin detection with contextual tracking feedback," in *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*. IEEE, 2016, pp. 1–6.

[79] X. Hu, W. Liu, J. Bian, and J. Pei, "Measuring model complexity of neural networks with curve activation functions," 2020.

[80] M. Höge, "On the Way to Appropriate Model Complexity," in *AGU Fall Meeting Abstracts*, vol. 2016, Dec. 2016, pp. NG13A–1683.

[81] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Proceedings of the 32nd international conference on neural information processing systems*, 2018, pp. 2488–2498.

[82] S.-H. Wang, K. Muhammad, J. Hong, A. K. Sangaiah, and Y.-D. Zhang, "Alcoholism identification via convolutional neural network based on parametric relu,

dropout, and batch normalization," *Neural Computing and Applications*, vol. 32, no. 3, pp. 665–680, 2020.

[83] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," *arXiv preprint arXiv:1603.08029*, 2016.

[84] C. F. Sabottke and B. M. Spieler, "The effect of image resolution on deep learning in radiography," *Radiology: Artificial Intelligence*, vol. 2, no. 1, p. e190015, 2020.

Appendix A

3D CNN Architectures

Table A.1: 3D CNN Architecture with Spatial Resolution of 16 Pixels

3D CNN Architecture (16 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 16, 16, 1)	0
conv3d_1	3D Convolution	(None, 30, 16, 16, 32)	896
activation_1	Activation	(None, 30, 16, 16, 32)	0
conv3d_2	3D Convolution	(None, 30, 16, 16, 32)	27680
activation_2	Activation	(None, 30, 16, 16, 32)	0
max_pooling3d_1	3D Max Pooling	(None, 10, 6, 6, 32)	0
dropout_1	Dropout	(None, 10, 6, 6, 32)	0
conv3d_3	3D Convolution	(None, 10, 6, 6, 64)	55360
activation_3	Activation	(None, 10, 6, 6, 64)	0
conv3d_4	3D Convloution	(None, 10, 6, 6, 64)	110656
activation_4	Activation	(None, 10, 6, 6, 64)	0
max_pooling3d_2	3D Max Pooling	(None, 4, 2, 2, 64)	0
dropout_2	Dropout	(None, 4, 2, 2, 64)	0
flatten_1	Flatten	(None, 1024)	0
dense_1	Dense	(None, 512)	524800
dropout_3	Dropout	(None, 512)	0
dense_2	Dense	(None, 1)	513
Total Parameters			719,905
Trainable Parameters			719,905
Non-trainable Parameters			0

TABLE A.2: 3D CNN Architecture with Spatial Resolution of 24 Pixels

3D CNN Architecture (24 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 24, 24, 1)	0
conv3d_1	3D Convolution	(None, 30, 24, 24, 32)	896
activation_1	Activation	(None, 30, 24, 24, 32)	0
conv3d_2	3D Convolution	(None, 30, 24, 24, 32)	27680
activation_2	Activation	(None, 30, 24, 24, 32)	0
max_pooling3d_1	3D Max Pooling	(None, 10, 8, 8, 32)	0
dropout_1	Dropout	(None, 10, 8, 8, 32)	0
conv3d_3	3D Convolution	(None, 10, 8, 8, 64)	55360
activation_3	Activation	(None, 10, 8, 8, 64)	0
conv3d_4	3D Convloution	(None, 10, 8, 8, 64)	110656
activation_4	Activation	(None, 10, 8, 8, 64)	0
max_pooling3d_2	3D Max Pooling	(None, 4, 3, 3, 64)	0
dropout_2	Dropout	(None, 4, 3, 3, 64)	0
flatten_1	Flatten	(None, 2304)	0
dense_1	Dense	(None, 512)	1180160
dropout_3	Dropout	(None, 512)	0
dense_2	Dense	(None, 1)	513
Total Parameters			1,375,265
Trainable Parameters			1,375,265
Non-trainable Parameters			0

TABLE A.3: 3D CNN Architecture with Spatial Resolution of 32 Pixels

3D CNN Architecture (32 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 32, 32, 1)	0
conv3d_1	3D Convolution	(None, 30, 32, 32, 32)	896
activation_1	Activation	(None, 30, 32, 32, 32)	0
conv3d_2	3D Convolution	(None, 30, 32, 32, 32)	27680
activation_2	Activation	(None, 30, 32, 32, 32)	0
max_pooling3d_1	3D Max Pooling	(None, 10, 11, 11, 32)	0
dropout_1	Dropout	(None, 10, 11, 11, 32)	0
conv3d_3	3D Convolution	(None, 10, 11, 11, 32)	55360
activation_3	Activation	(None, 10, 11, 11, 32)	0
conv3d_4	3D Convloution	(None, 10, 11, 11, 32)	110656
activation_4	Activation	(None, 10, 11, 11, 32)	0
max_pooling3d_2	3D Max Pooling	(None, 4, 4, 4, 64)	0
dropout_2	Dropout	(None, 4, 4, 4, 64)	0
flatten_1	Flatten	(None, 4096)	0
dense_1	Dense	(None, 512)	2097664
dropout_3	Dropout	(None, 512)	0
dense_2	Dense	(None, 1)	513
Total Parameters			2,292,769
Trainable Parameters			2,292,769
Non-trainable Parameters			0

TABLE A.4: 3D CNN Architecture with Spatial Resolution of 40 Pixels

3D CNN Architecture (40 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 40, 40, 1)	0
conv3d_1	3D Convolution	(None, 30, 40, 40, 32)	896
activation_1	Activation	(None, 30, 40, 40, 32)	0
conv3d_2	3D Convolution	(None, 30, 40, 40, 32)	27680
activation_2	Activation	(None, 30, 40, 40, 32)	0
max_pooling3d_1	3D Max Pooling	(None, 10, 14, 14, 32)	0
dropout_1	Dropout	(None, 10, 14, 14, 32)	0
conv3d_3	3D Convolution	(None, 10, 14, 14, 32)	55360
activation_3	Activation	(None, 10, 14, 14, 32)	0
conv3d_4	3D Convloution	(None, 10, 14, 14, 32)	110656
activation_4	Activation	(None, 10, 14, 14, 32)	0
max_pooling3d_2	3D Max Pooling	(None, 4, 5, 5, 64)	0
dropout_2	Dropout	(None, 4, 5, 5, 64)	0
flatten_1	Flatten	(None, 6400)	0
dense_1	Dense	(None, 512)	3277312
dropout_3	Dropout	(None, 512)	0
dense_2	Dense	(None, 1)	513
Total Parameters			3,472,417
Trainable Parameters			3,472,417
Non-trainable Parameters			0

Appendix B

3D ResNet-18 Architectures

TABLE B.1: 3D ResNet-18 Architecture with Spatial Resolution of 16 Pixels

3D ResNet-18 Architecture (16 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 16, 16, 1)	0
conv3d_1	3D Convolution	(None, 15, 8, 8, 64)	22016
batch_normalization_1	Batch Normalization	(None, 15, 8, 8, 64)	256
activation_1	Activation	(None, 15, 8, 8, 64)	0
max_pooling3d_1	3D Max Pooling	(None, 8, 4, 4, 64)	0
conv3d_2	3D Convolution	(None, 8, 4, 4, 64)	110656
batch_normalization_2	Batch Normalization	(None, 8, 4, 4, 64)	256
activation_2	Activation	(None, 8, 4, 4, 64)	0
conv3d_3	3D Convolution	(None, 8, 4, 4, 64)	110656
add_1	Add	(None, 8, 4, 4, 64)	0
batch_normalization_3	Batch Normalization	(None, 8, 4, 4, 64)	256
activation_3	Activation	(None, 8, 4, 4, 64)	0
conv3d_4	3D Convolution	(None, 8, 4, 4, 64)	110656
batch_normalization_4	Batch Normalization	(None, 8, 4, 4, 64)	256
activation_4	Activation	(None, 8, 4, 4, 64)	0
conv3d_5	3D Convolution	(None, 8, 4, 4, 64)	110656
add_2	Add	(None, 8, 4, 4, 64)	0
batch_normalization_5	Batch Normalization	(None, 8, 4, 4, 64)	256
activation_5	Activation	(None, 8, 4, 4, 64)	0
conv3d_6	3D Convolution	(None, 4, 2, 2, 128)	221312
batch_normalization_6	Batch Normalization	(None, 4, 2, 2, 128)	512
activation_6	Activation	(None, 4, 2, 2, 128)	0
conv3d_8	3D Convolution	(None, 4, 2, 2, 128)	8320
conv3d_7	3D Convolution	(None, 4, 2, 2, 128)	442496
add_3	Add	(None, 4, 2, 2, 128)	0
batch_normalization_7	Batch Normalization	(None, 4, 2, 2, 128)	512
activation_7	Activation	(None, 4, 2, 2, 128)	0
conv3d_9	3D Convolution	(None, 4, 2, 2, 128)	442496
batch_normalization_8	Batch Normalization	(None, 4, 2, 2, 128)	512
activation_8	Activation	(None, 4, 2, 2, 128)	0
conv3d_10	3D Convolution	(None, 4, 2, 2, 128)	442496
add_4	Add	(None, 4, 2, 2, 128)	0
batch_normalization_9	Batch Normalization	(None, 4, 2, 2, 128)	512
activation_9	Activation	(None, 4, 2, 2, 128)	0
conv3d_11	3D Convolution	(None, 2, 1, 1, 256)	884992
batch_normalization_10	Batch Normalization	(None, 2, 1, 1, 256)	1024
activation_10	Activation	(None, 2, 1, 1, 256)	0
conv3d_13	3D Convolution	(None, 2, 1, 1, 256)	33024
conv3d_12	3D Convolution	(None, 2, 1, 1, 256)	1769728
add_5	Add	(None, 2, 1, 1, 256)	0
batch_normalization_11	Batch Normalization	(None, 2, 1, 1, 256)	1024
activation_11	Activation	(None, 2, 1, 1, 256)	0
conv3d_14	3D Convolution	(None, 2, 1, 1, 256)	1769728
batch_normalization_12	Batch Normalization	(None, 2, 1, 1, 256)	1024
activation_12	Activation	(None, 2, 1, 1, 256)	0
conv3d_15	3D Convolution	(None, 2, 1, 1, 256)	1769728
add_6	Add	(None, 2, 1, 1, 256)	0
batch_normalization_13	Batch Normalization	(None, 2, 1, 1, 256)	1024
activation_13	Activation	(None, 2, 1, 1, 256)	0
conv3d_16	conv3d_16	(None, 1, 1, 1, 512)	3539456
batch_normalization_14	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_14	Activation	(None, 1, 1, 1, 512)	0
conv3d_18	3D Convolution	(None, 1, 1, 1, 512)	131584
conv3d_17	3D Convolution	(None, 1, 1, 1, 512)	7078400
add_7	Add	(None, 1, 1, 1, 512)	0
batch_normalization_15	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_15	Activation	(None, 1, 1, 1, 512)	0
conv3d_19	3D Convolution	(None, 1, 1, 1, 512)	7078400
batch_normalization_16	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_16	Activation	(None, 1, 1, 1, 512)	0
conv3d_20	3D Convolution	(None, 1, 1, 1, 512)	7078400
add_8	Add	(None, 1, 1, 1, 512)	0
batch_normalization_17	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_17	Activation	(None, 1, 1, 1, 512)	0
average_pooling3d_1	3D Average Pooling	(None, 1, 1, 1, 512)	0
flatten_1	Flatten	(None, 512)	0
dense_1	Dense	(None, 1)	513
Total Parameters			33,171,329
Trainable Parameters			33,163,521
Non-trainable Parameters			7,808

TABLE B.2: 3D ResNet-18 Architecture with Spatial Resolution of 24 Pixels

3D ResNet-18 Architecture (24 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 24, 24, 1)	0
conv3d_1	3D Convolution	(None, 15, 12, 12, 6)	22016
batch_normalization_1	Batch Normalization	(None, 15, 12, 12, 6)	256
activation_1	Activation	(None, 15, 12, 12, 6)	0
max_pooling3d_1	3D Max Pooling	(None, 8, 6, 6, 64)	0
conv3d_2	3D Convolution	(None, 8, 6, 6, 64)	110656
batch_normalization_2	Batch Normalization	(None, 8, 6, 6, 64)	256
activation_2	Activation	(None, 8, 6, 6, 64)	0
conv3d_3	3D Convolution	(None, 8, 6, 6, 64)	110656
add_1	Add	(None, 8, 6, 6, 64)	0
batch_normalization_3	Batch Normalization	(None, 8, 6, 6, 64)	256
activation_3	Activation	(None, 8, 6, 6, 64)	0
conv3d_4	3D Convolution	(None, 8, 6, 6, 64)	110656
batch_normalization_4	Batch Normalization	(None, 8, 6, 6, 64)	256
activation_4	Activation	(None, 8, 6, 6, 64)	0
conv3d_5	3D Convolution	(None, 8, 6, 6, 64)	110656
add_2	Add	(None, 8, 6, 6, 64)	0
batch_normalization_5	Batch Normalization	(None, 8, 6, 6, 64)	256
activation_5	Activation	(None, 8, 6, 6, 64)	0
conv3d_6	3D Convolution	(None, 4, 3, 3, 128)	221312
batch_normalization_6	Batch Normalization	(None, 4, 3, 3, 128)	512
activation_6	Activation	(None, 4, 3, 3, 128)	0
conv3d_8	3D Convolution	(None, 4, 3, 3, 128)	8320
conv3d_7	3D Convolution	(None, 4, 3, 3, 128)	442496
add_3	Add	(None, 4, 3, 3, 128)	0
batch_normalization_7	Batch Normalization	(None, 4, 3, 3, 128)	512
activation_7	Activation	(None, 4, 3, 3, 128)	0
conv3d_9	3D Convolution	(None, 4, 3, 3, 128)	442496
batch_normalization_8	Batch Normalization	(None, 4, 3, 3, 128)	512
activation_8	Activation	(None, 4, 3, 3, 128)	0
conv3d_10	3D Convolution	(None, 4, 3, 3, 128)	442496
add_4	Add	(None, 4, 3, 3, 128)	0
batch_normalization_9	Batch Normalization	(None, 4, 3, 3, 128)	512
activation_9	Activation	(None, 4, 3, 3, 128)	0
conv3d_11	3D Convolution	(None, 2, 2, 2, 256)	884992
batch_normalization_10	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_10	Activation	(None, 2, 2, 2, 256)	0
conv3d_13	3D Convolution	(None, 2, 2, 2, 256)	33024
conv3d_12	3D Convolution	(None, 2, 2, 2, 256)	1769728
add_5	Add	(None, 2, 2, 2, 256)	0
batch_normalization_11	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_11	Activation	(None, 2, 2, 2, 256)	0
conv3d_14	3D Convolution	(None, 2, 2, 2, 256)	1769728
batch_normalization_12	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_12	Activation	(None, 2, 2, 2, 256)	0
conv3d_15	3D Convolution	(None, 2, 2, 2, 256)	1769728
add_6	Add	(None, 2, 2, 2, 256)	0
batch_normalization_13	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_13	Activation	(None, 2, 2, 2, 256)	0
conv3d_16	conv3d_16	(None, 1, 1, 1, 512)	3539456
batch_normalization_14	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_14	Activation	(None, 1, 1, 1, 512)	0
conv3d_18	3D Convolution	(None, 1, 1, 1, 512)	131584
conv3d_17	3D Convolution	(None, 1, 1, 1, 512)	7078400
add_7	Add	(None, 1, 1, 1, 512)	0
batch_normalization_15	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_15	Activation	(None, 1, 1, 1, 512)	0
conv3d_19	3D Convolution	(None, 1, 1, 1, 512)	7078400
batch_normalization_16	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_16	Activation	(None, 1, 1, 1, 512)	0
conv3d_20	3D Convolution	(None, 1, 1, 1, 512)	7078400
add_8	Add	(None, 1, 1, 1, 512)	0
batch_normalization_17	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_17	Activation	(None, 1, 1, 1, 512)	0
average_pooling3d_1	3D Average Pooling	(None, 1, 1, 1, 512)	0
flatten_1	Flatten	(None, 512)	0
dense_1	Dense	(None, 1)	513
Total Parameters			33,171,329
Trainable Parameters			33,163,521
Non-trainable Parameters			7,808

TABLE B.3: 3D ResNet-18 Architecture with Spatial Resolution of 32 Pixels

3D ResNet-18 Architecture (32 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 32, 32, 1)	0
conv3d_1	3D Convolution	(None, 15, 16, 16, 6)	22016
batch_normalization_1	Batch Normalization	(None, 15, 16, 16, 6)	256
activation_1	Activation	(None, 15, 16, 16, 6)	0
max_pooling3d_1	3D Max Pooling	(None, 8, 8, 8, 64)	0
conv3d_2	3D Convolution	(None, 8, 8, 8, 64)	110656
batch_normalization_2	Batch Normalization	(None, 8, 8, 8, 64)	256
activation_2	Activation	(None, 8, 8, 8, 64)	0
conv3d_3	3D Convolution	(None, 8, 8, 8, 64)	110656
add_1	Add	(None, 8, 8, 8, 64)	0
batch_normalization_3	Batch Normalization	(None, 8, 8, 8, 64)	256
activation_3	Activation	(None, 8, 8, 8, 64)	0
conv3d_4	3D Convolution	(None, 8, 8, 8, 64)	110656
batch_normalization_4	Batch Normalization	(None, 8, 8, 8, 64)	256
activation_4	Activation	(None, 8, 8, 8, 64)	0
conv3d_5	3D Convolution	(None, 8, 8, 8, 64)	110656
add_2	Add	(None, 8, 8, 8, 64)	0
batch_normalization_5	Batch Normalization	(None, 8, 8, 8, 64)	256
activation_5	Activation	(None, 8, 8, 8, 64)	0
conv3d_6	3D Convolution	(None, 4, 4, 4, 128)	221312
batch_normalization_6	Batch Normalization	(None, 4, 4, 4, 128)	512
activation_6	Activation	(None, 4, 4, 4, 128)	0
conv3d_8	3D Convolution	(None, 4, 4, 4, 128)	8320
conv3d_7	3D Convolution	(None, 4, 4, 4, 128)	442496
add_3	Add	(None, 4, 4, 4, 128)	0
batch_normalization_7	Batch Normalization	(None, 4, 4, 4, 128)	512
activation_7	Activation	(None, 4, 4, 4, 128)	0
conv3d_9	3D Convolution	(None, 4, 4, 4, 128)	442496
batch_normalization_8	Batch Normalization	(None, 4, 4, 4, 128)	512
activation_8	Activation	(None, 4, 4, 4, 128)	0
conv3d_10	3D Convolution	(None, 4, 4, 4, 128)	442496
add_4	Add	(None, 4, 4, 4, 128)	0
batch_normalization_9	Batch Normalization	(None, 4, 4, 4, 128)	512
activation_9	Activation	(None, 4, 4, 4, 128)	0
conv3d_11	3D Convolution	(None, 2, 2, 2, 256)	884992
batch_normalization_10	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_10	Activation	(None, 2, 2, 2, 256)	0
conv3d_13	3D Convolution	(None, 2, 2, 2, 256)	33024
conv3d_12	3D Convolution	(None, 2, 2, 2, 256)	1769728
add_5	Add	(None, 2, 2, 2, 256)	0
batch_normalization_11	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_11	Activation	(None, 2, 2, 2, 256)	0
conv3d_14	3D Convolution	(None, 2, 2, 2, 256)	1769728
batch_normalization_12	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_12	Activation	(None, 2, 2, 2, 256)	0
conv3d_15	3D Convolution	(None, 2, 2, 2, 256)	1769728
add_6	Add	(None, 2, 2, 2, 256)	0
batch_normalization_13	Batch Normalization	(None, 2, 2, 2, 256)	1024
activation_13	Activation	(None, 2, 2, 2, 256)	0
conv3d_16	conv3d_16	(None, 1, 1, 1, 512)	3539456
batch_normalization_14	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_14	Activation	(None, 1, 1, 1, 512)	0
conv3d_18	3D Convolution	(None, 1, 1, 1, 512)	131584
conv3d_17	3D Convolution	(None, 1, 1, 1, 512)	7078400
add_7	Add	(None, 1, 1, 1, 512)	0
batch_normalization_15	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_15	Activation	(None, 1, 1, 1, 512)	0
conv3d_19	3D Convolution	(None, 1, 1, 1, 512)	7078400
batch_normalization_16	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_16	Activation	(None, 1, 1, 1, 512)	0
conv3d_20	3D Convolution	(None, 1, 1, 1, 512)	7078400
add_8	Add	(None, 1, 1, 1, 512)	0
batch_normalization_17	Batch Normalization	(None, 1, 1, 1, 512)	2048
activation_17	Activation	(None, 1, 1, 1, 512)	0
average_pooling3d_1	3D Average Pooling	(None, 1, 1, 1, 512)	0
flatten_1	Flatten	(None, 512)	0
dense_1	Dense	(None, 1)	513
Total Parameters			33,171,329
Trainable Parameters			33,163,521
Non-trainable Parameters			7,808

TABLE B.4: 3D ResNet-18 Architecture with Spatial Resolution of 40 Pixels

3D ResNet-18 Architecture (40 Pixels)			
Layer	**Type**	**Output Shape**	**No. Parameters**
input_1	Input	(None, 30, 40, 40, 1)	0
conv3d_1	3D Convolution	(None, 15, 20, 20, 6)	22016
batch_normalization_1	Batch Normalization	(None, 15, 20, 20, 6)	256
activation_1	Activation	(None, 15, 20, 20, 6)	0
max_pooling3d_1	3D Max Pooling	(None, 8, 10, 10, 64)	0
conv3d_2	3D Convolution	(None, 8, 10, 10, 64)	110656
batch_normalization_2	Batch Normalization	(None, 8, 10, 10, 64)	256
activation_2	Activation	(None, 8, 10, 10, 64)	0
conv3d_3	3D Convolution	(None, 8, 10, 10, 64)	110656
add_1	Add	(None, 8, 10, 10, 64)	0
batch_normalization_3	Batch Normalization	(None, 8, 10, 10, 64)	256
activation_3	Activation	(None, 8, 10, 10, 64)	0
conv3d_4	3D Convolution	(None, 8, 10, 10, 64)	110656
batch_normalization_4	Batch Normalization	(None, 8, 10, 10, 64)	256
activation_4	Activation	(None, 8, 10, 10, 64)	0
conv3d_5	3D Convolution	(None, 8, 10, 10, 64)	110656
add_2	Add	(None, 8, 10, 10, 64)	0
batch_normalization_5	Batch Normalization	(None, 8, 10, 10, 64)	256
activation_5	Activation	(None, 8, 10, 10, 64)	0
conv3d_6	3D Convolution	(None, 4, 5, 5, 128)	221312
batch_normalization_6	Batch Normalization	(None, 4, 5, 5, 128)	512
activation_6	Activation	(None, 4, 5, 5, 128)	0
conv3d_8	3D Convolution	(None, 4, 5, 5, 128)	8320
conv3d_7	3D Convolution	(None, 4, 5, 5, 128)	442496
add_3	Add	(None, 4, 5, 5, 128)	0
batch_normalization_7	Batch Normalization	(None, 4, 5, 5, 128)	512
activation_7	Activation	(None, 4, 5, 5, 128)	0
conv3d_9	3D Convolution	(None, 4, 5, 5, 128)	442496
batch_normalization_8	Batch Normalization	(None, 4, 5, 5, 128)	512
activation_8	Activation	(None, 4, 5, 5, 128)	0
conv3d_10	3D Convolution	(None, 4, 5, 5, 128)	442496
add_4	Add	(None, 4, 5, 5, 128)	0
batch_normalization_9	Batch Normalization	(None, 4, 5, 5, 128)	512
activation_9	Activation	(None, 4, 5, 5, 128)	0
conv3d_11	3D Convolution	(None, 2, 3, 3, 256)	884992
batch_normalization_10	Batch Normalization	(None, 2, 3, 3, 256)	1024
activation_10	Activation	(None, 2, 3, 3, 256)	0
conv3d_13	3D Convolution	(None, 2, 3, 3, 256)	33024
conv3d_12	3D Convolution	(None, 2, 3, 3, 256)	1769728
add_5	Add	(None, 2, 3, 3, 256)	0
batch_normalization_11	Batch Normalization	(None, 2, 3, 3, 256)	1024
activation_11	Activation	(None, 2, 3, 3, 256)	0
conv3d_14	3D Convolution	(None, 2, 3, 3, 256)	1769728
batch_normalization_12	Batch Normalization	(None, 2, 3, 3, 256)	1024
activation_12	Activation	(None, 2, 3, 3, 256)	0
conv3d_15	3D Convolution	(None, 2, 3, 3, 256)	1769728
add_6	Add	(None, 2, 3, 3, 256)	0
batch_normalization_13	Batch Normalization	(None, 2, 3, 3, 256)	1024
activation_13	Activation	(None, 2, 3, 3, 256)	0
conv3d_16	conv3d_16	(None, 1, 2, 2, 512)	3539456
batch_normalization_14	Batch Normalization	(None, 1, 2, 2, 512)	2048
activation_14	Activation	(None, 1, 2, 2, 512)	0
conv3d_18	3D Convolution	(None, 1, 2, 2, 512)	131584
conv3d_17	3D Convolution	(None, 1, 2, 2, 512)	7078400
add_7	Add	(None, 1, 2, 2, 512)	0
batch_normalization_15	Batch Normalization	(None, 1, 2, 2, 512)	2048
activation_15	Activation	(None, 1, 2, 2, 512)	0
conv3d_19	3D Convolution	(None, 1, 2, 2, 512)	7078400
batch_normalization_16	Batch Normalization	(None, 1, 2, 2, 512)	2048
activation_16	Activation	(None, 1, 2, 2, 512)	0
conv3d_20	3D Convolution	(None, 1, 2, 2, 512)	7078400
add_8	Add	(None, 1, 2, 2, 512)	0
batch_normalization_17	Batch Normalization	(None, 1, 2, 2, 512)	2048
activation_17	Activation	(None, 1, 2, 2, 512)	0
average_pooling3d_1	3D Average Pooling	(None, 1, 1, 1, 512)	0
flatten_1	Flatten	(None, 512)	0
dense_1	Dense	(None, 1)	513
Total Parameters			33,171,329
Trainable Parameters			33,163,521
Non-trainable Parameters			7,808