Evaluating Deep Learning Architectures for Predicting Semiconductor Absorption Patterns

Submitted by

Angshuman Aumi	20200205134
Navid Aziz	20200205160
Farhan Rahman	20200205172
MD. Hisbu Zaman	20200105054

In partial fulfilment of the requirement for the degree of
BACHELOR OF SCIENCE

Department of Electrical and Electronic Engineering

AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Spring-2024

Declaration

We hereby declare that this paper titled "**Evaluating Deep Learning Architectures for Predicting Semiconductor Absorption Patterns**" is our original work and has not been presented elsewhere for assessment or award of any other degree or any other publication. Any additional resources have been properly acknowledged and referred.

Angshuman Aumi

____

Navid Aziz

____

Farhan Rahman

____

MD. Hisbu Zaman

____

Mr. Nahyan Al Mahmud

____

Mr. Nahyan Al Mahmud

Assistant Professor

Department of Electrical and Electronic Engineering

Ahsanullah University of Science and Technology

Acknowledgment

We like to convey our profound appreciation to Allah, the Almighty, for granting us the strength, knowledge, and resolve required to initiate and successfully complete this scientific endeavor. We express our profound gratitude to our esteemed advisor, Mr. Nahyan Al Mahmud, Assistant Professor in the Department of EEE at AUST, for his essential mentorship, unwavering support, and insightful analysis, which have been foundational to our research undertaking. His assistance and expertise were essential in shaping this initiative and fostering our growth as aspiring scientists. We extend our profound gratitude to Prof. Dr. Bobby Barua, Head of the Faculty, Department of EEE, AUST. His astute observations and engaging discussions have considerably improved our comprehension and skill, while also providing us with all essential resources for evaluating our work. We extend our sincere appreciation to the distinguished faculty and staff of the Department of EEE at AUST for their essential support in initiating this research project, including a conducive intellectual atmosphere and requisite resources. In closing, we convey our profound gratitude to our families for their steadfast faith, resilience, and unconditional love. Their emotional support and altruistic gifts have been the bedrock that sustained us during the most challenging phases of this endeavor.

Abstract

Semiconductor manufacturing is a fragile and also complicated process which is needed in almost all of the modern electronic devices. With the rise in the requirements of higher performance and size reduction, accurate control, small margins of error and high efficiency are of utmost importance. Since semiconductor device manufacturing is to be accomplished manually, therefore, much can be carried out in the elimination of high human involvement that consumes much time and is inefficient. Machine learning presented here will, therefore, certainly be useful in making the production effective and the manufacturing process optimized. Since we are dealing with machine learning, therefore, neural networks are the first to be mentioned, which are the computational models based on the human brain and can learn the complex patterns in data.

We commenced our research by applying a 1D CNN model that was trained on datasets that depict angles to which the rays are sent in the semiconductor materials during their manufacturing process. This was aimed at modelling a model that would help predict or estimate these angles hence help in production. Our first model which used 1d CNN estimated almost the similar results as we expected. However, to further enhance the model we used RNN and LSTM to make the forecasts, specifically Long Short-Term Memory (LSTM) networks, which work best with sequential data. Then our model went ahead to even predict noisy input data. The results show that our machine learning model can predict the angles of the rays used in the semiconductor fabrication and hence allow the fabrication process to be more efficient. The research, also, maintains the basis of future incorporation of deep learning in the semiconductor manufacturing pipelines, which can bring down the errors to a considerable extent and optimize the performance.

Table of Contents

List of Tables

Table of Figures

Chapter 01

1.1 Introduction

Due to the advent of machine learning and the ever-increasing availability of data, its usage has found its way into other industrial sectors - specifically regarding the optimization of existing production lines. Optimization may be aimed at the better quality of goods or optimization of the manufacturing process. Machine learning can benefit many sections as it allows using the resources available better, using less energy and gaining more yield. The optimization is aided by machine learning, which applies a vast number of methods, such as data transformation (such as PCA) and clustering (such as k-means) to supervised learning (such as decision trees) to more sophisticated models such as Convolutional Neural Networks (CNNs). This paper discusses machine learning in optimization of products and production processes. The paper will focus on applications whereby available data (regardless of the presence of real-time quality feedback) is used to make process parameter adjustments and achieve better results.

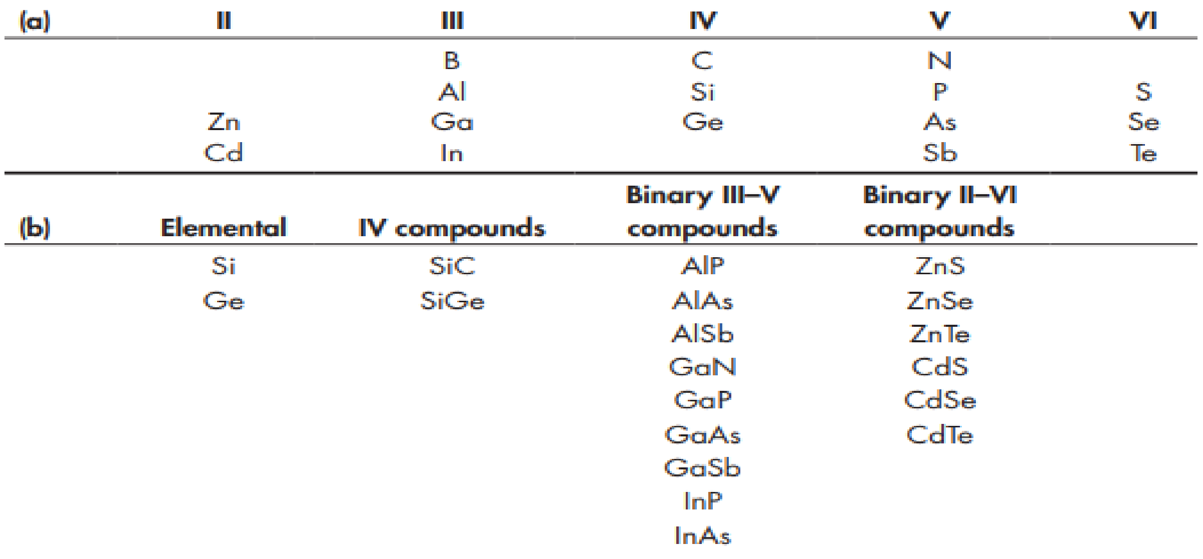
1.2 Motivation

This research is primarily driven by the fact that today there is an imperative of accurate and effective optimization of the production process due to the fact that semiconductors form the heart of all modern electronic gadgets which have become a necessity in our present day to day life, beginning right with every gadget like mobile phones to our computers to medical equipment and electric cars. There is not one single industry that can be left unmarked with semiconductor and electronics. It is there that we get our drive to conduct research on the topic of optimization of semiconductor manufacturing in the industry. This thesis is motivated by the possibility of using machine learning methods, especially neural networks, to improve the precision and efficiency of semiconductor fabrication. We are particularly interested in forecasting the angle with which a ray is shot into the semiconductor materials, which is a process parameter of importance since it affects material behavior and ultimately the quality of the end product.

1.3 Semiconductor and Machine learning

1.3.1 Semiconductor

A semiconductor is a category of materials with electrical conductivity intermediate between that of metals and insulators. Of importance is that the conductivity of these materials can be changed by many orders of magnitude with temperature, optical excitation and impurity content. It is this electrical property variability that makes the semiconductor materials natural candidates to electronic device studies. As a rule, semiconductor materials are some elements of group 2 to group 6. However group 4 materials that include carbon, silicon and germanium are pretty important.



(a)	II	III	IV	V	VI
		B	C	N	
		Al	Si	P	S
	Zn	Ga	Ge	As	Se
	Cd	In		Sb	Te

(b)	Elemental	IV compounds	Binary III–V compounds	Binary II–VI compounds
	Si	SiC	AlP	ZnS
	Ge	SiGe	AlAs	ZnSe
			AlSb	ZnTe
			GaN	CdS
			GaP	CdSe
			GaAs	CdTe
			GaSb	
			InP	
			InAs	

Table 1: Semiconductors of different groups

1.3.2 Structure of some semiconductors

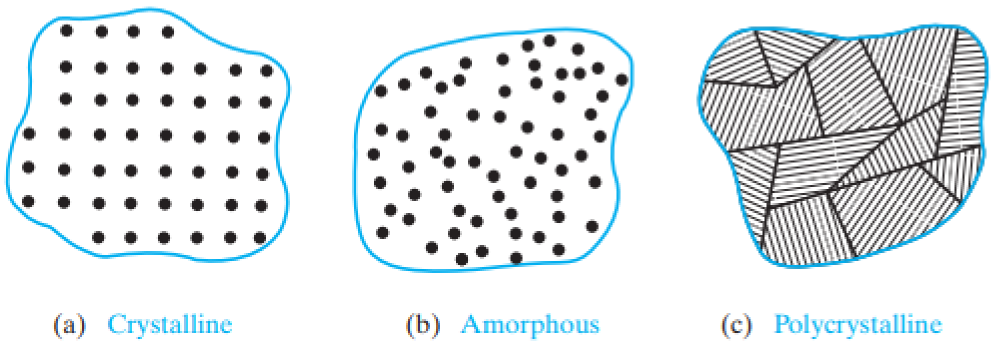


Figure 1: Structure of semiconductor materials

a) **Crystalline**: The periodic arrangement of atoms is there and at whichever point in the crystal we look at it will appear the same in crystalline structure.

b) **Amorphous**: These types of elements have no periodic structure at all and each point looks different.

c) **Polycrystalline**: This type has the structure like crystalline at different parts. There are small regions of crystal materials here.

1.3.3 Machine Learning

Machine learning (ML) is a branch of artificial intelligence that enables systems to learn from provided data and improve their performance on a task. By using ML, we can build models from input data to make predictions or decisions by the system. This data dependent ability makes machine learning especially suitable for complex and dynamic environments like semiconductor manufacturing where it can improve day by day by using the available data.

It has 3 main tasks:
a) Classification
b) Regression
c) Clustering.

Our focus lies in regression, where the goal is to predict a continuous value which in our case, is the angle at which a ray is projected into semiconductor material. Some machine learning approaches:

1.Supervised Learning: In this type of learning, the algorithm learns from both the given input and output data so that it can then predict the output from unknown inputs and these are done by previously learning the patterns.

2.Unsupervised Learning: In this learning the algorithm learns from patterns of unlabeled data. Here input might be given but not the output.

3.Reinforcement Learning: The algorithm is that it learns from mistakes and feedback from humans. It learns by trial and error and doesn’t use any labelled data

Machine learning has been used in recent years a lot due to the larger computational power of current generation computers and the availability of huge datasets. These methods have surpassed traditional statistical techniques in many real-world applications.
In semiconductor manufacturing, ML is used for:

- Defect detection in wafers
- Predictive maintenance of equipment
- Process parameter optimization
- Yield prediction

In our thesis we used a machine learning model to address a highly specific but critical task of predicting ray projection angles. Accurate predictions at this step can significantly reduce waste, improve material usage time, and enhance final device performance

Chapter 02

Literature Review

In the previous chapter we discussed the main objective of this thesis project and also about the semiconductors and its structures. We also discussed Machine learning and how it relates to semiconductors.

This chapter will highlight the literature review on this thesis project. In recent years, deep learning techniques (a machine learning type which uses neural networks to process data at different layers) have shown tremendous promise in the prediction and identification of various production parameters by using available datasets and thus predicting.

2.1 Neural Networks

This is a part of machine learning where the computer works like neurons of the human brain. Each neuron will automatically divide themselves into required tasks and will accomplish it, after that all neurons will pass that info they processed to the next layer, which is hidden, after that the hidden layer will ultimately pass the info to the output which will predict what the thing is actually. So, many neurons together work here to identify what is given to them and this is a continuous training process. And the machine will get better at predicting the specific task as we provide it with a lot of datasets. The more dataset is available the more the machine learns and predicts accurately.

2.2 Convolutional Neural Networks

The hidden layers we talked about form convolutional neural networks. The image provided to a computer might be shifted, or rearranged, or might have different dimensions. Even after all that the computer can recognize what the image is by using CNN. So, its main feature is that it can process the image and tell us about the image.

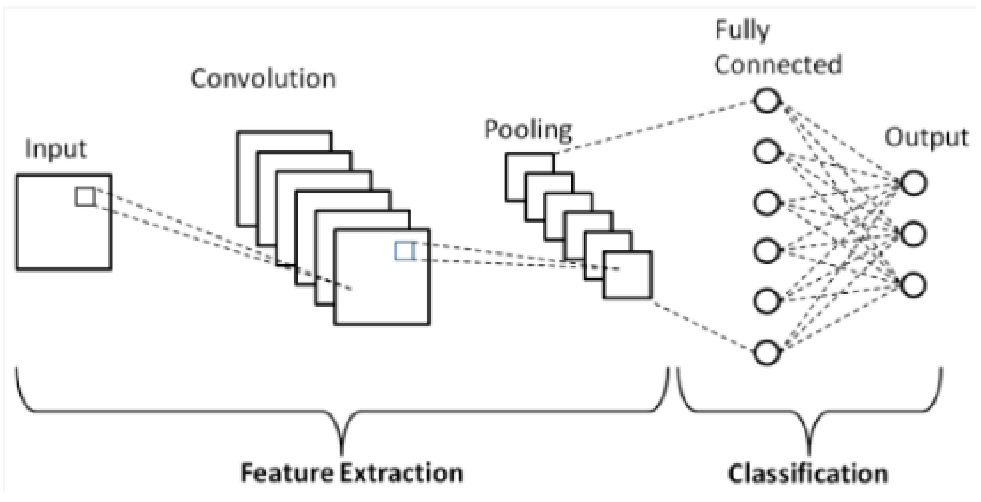


Figure 2: A simple CNN architecture [1]

Especially convolutions are translation invariant, which means that the filters are stationary despite the location. As a result, forward function sees its efficiency grow significantly, a massive reduction of parameters is applied, and the network becomes less dependent on the volatility in the size of the data and simpler to optimize. Unlike standard neural networks, the layers of a CNN consist of neurons arranged in a limited set of dimensions: channels, breadth, height, and, at the most basic level, the number of filters in a two- dimensional scenario. A convolution neural network, similar in concept to an MLP, consists of a sequence of layers, where in each layer the outputs or activations of the prior layer are modified by a distinct differentiable function in the next layer. Although many layers are used in CNNs, they will be described in the following sections, the most frequent building blocks used in CNN architectures are the convolution, pooling, and fully connected layers. These are, in that order, the classification, feature extractor, and dimensionality reduction layers. These layers can be stacked to form a full convolutional layer CNN.

2.3 What our machine learning model will do:

Our model will use CNN to process the dataset we provided it to predict. Our dataset has an angle given a definite absorption rate for four different materials. The machine learning model we designed has to predict the angle at which a ray is to be emitted to a specific material for placing a semiconductor device on another material. This angle, which must be quite accurate, is to be guessed by our model so that it can increase the yield in production and optimize the overall process. We used CNN in this process which was then upgraded to RNN using LSTM. The use of RNN was necessary as we have a limited dataset and have to use the order of the dataset to influence the output.

2.4 Topic Significance:

As electrical engineers with specialization in electronics, we were very much interested in how these semiconductor dependent devices were constructed in bulk quantities. Because semiconductors form the base of all modern electronic devices hence, we were enthusiastic in determining how the manufacturing is done in the initial stage where different materials are fabricated over other materials in the making process. But that involved detailed fabrication processes of the materials. Apart from all these, machine learning has become a trend in the current technological era. So, we thought of incorporating that with the manufacturing optimization process which would definitely reduce time and increase yield by quick prediction as computers are really fast. That made us peek deeper into artificial intelligence/Machine learning and the terms associated with it. Initially we didn’t have any dataset as these types of datasets of semiconductor manufacturing are quite hard to find, still we with the help of our supervisor acquired some data which were fed to our created machine learning models and it could successfully predict the outcomes which were expected of it.

Chapter 03

Theoretical Background

In the previous chapter we discussed literature review about the semiconductors and some of their structures as well as the incorporating of machine learning into semiconductors. In this chapter we will dive deeper into machine learning fundamentals.

3.1 Artificial Neural Networks

All we have known so far about machine learning involving CNN is artificial neural networks. This works by transferring data through layers. There are hidden layers in between which the n forms the Convolutional layers. The hidden layers possess filters which determine the given image parts and pass its prediction to the next layer. But the basic concept is to provide our model an image. The model has to find out what that image means. After the initial tries the machine fails to predict correctly, but then it will have a feedback mechanism which are its penalties. By this feedback mechanism the machine will learn and get better at predicting. This is how artificial neural networks work.

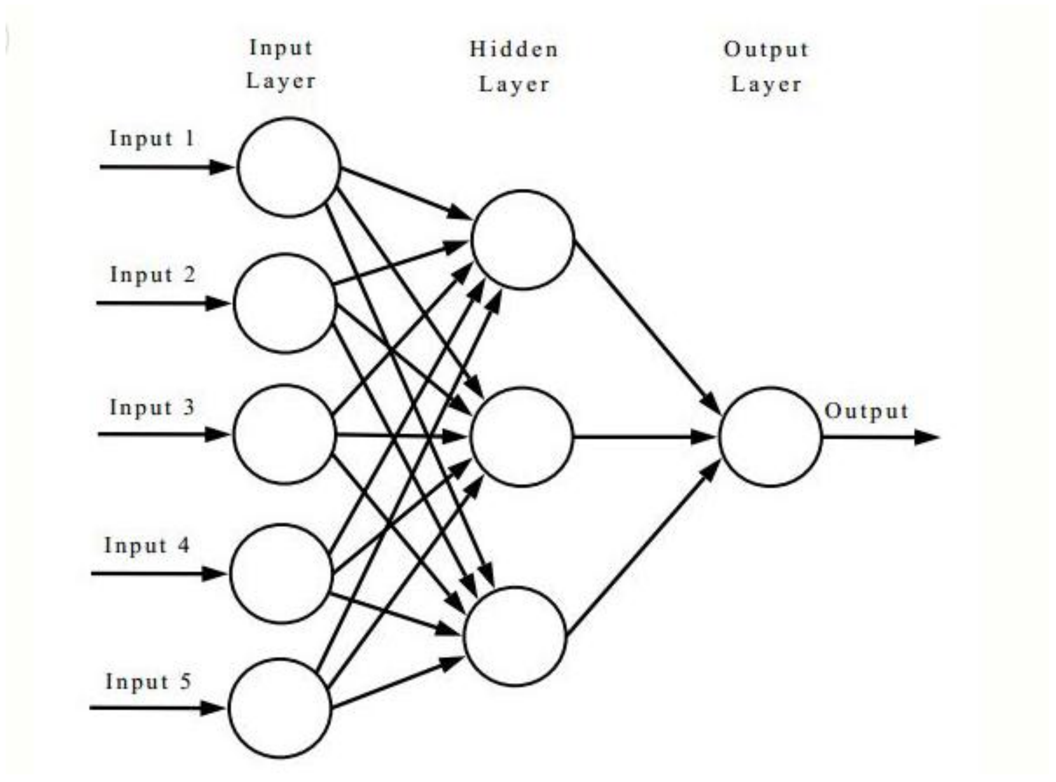
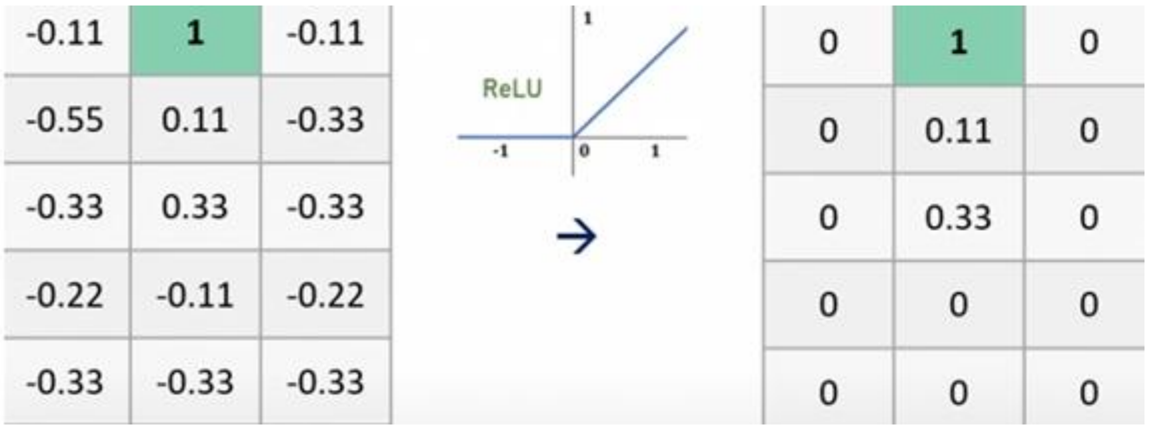


Figure 3: Basic working principle of Neural Network

3.2 ReLU Operation

ReLU means Rectified Linear unit. This brings non-linearity in the model. Which means if any value is less than 0, then it replaces it with 0 if it is greater than 0 it stays that way. This is done to increase efficiency by making the data simple. While training deep learning models, like the one which we designed, there arises a Vanishing gradient problem.



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Figure 4: How ReLU works

So, this is how ReLu works and thus simplifies the dataset. As we see here, all the values which are less than 0 have now become a 0. And any values greater than 0 are basically the same and nothing changes. This simplifies our data and increases efficiency of our model.

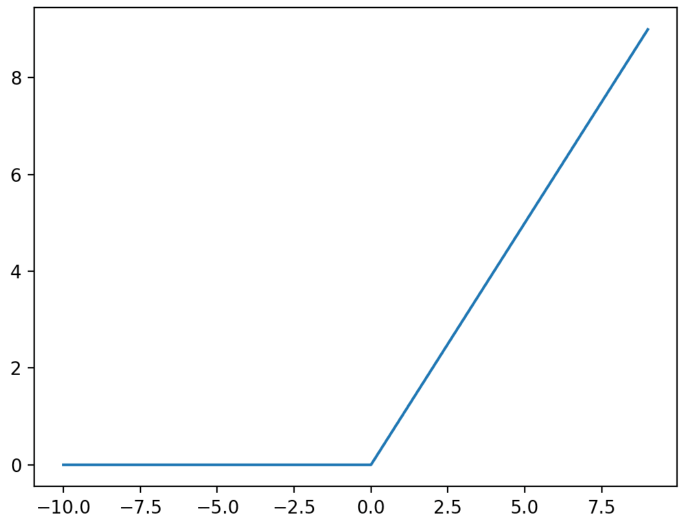
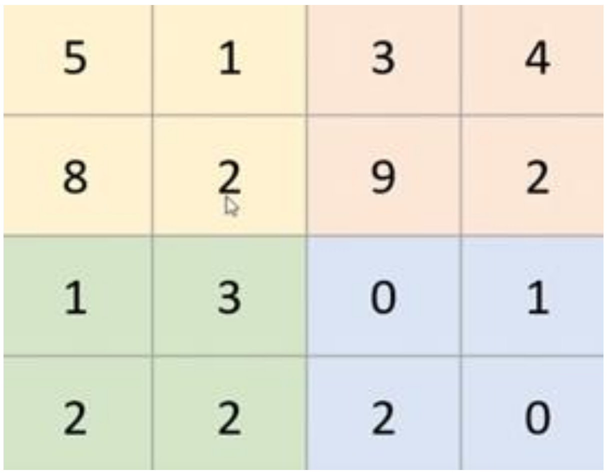


Figure 5: Line Plot of Rectified Linear Activation for Negative and Positive Inputs

3.3 Pooling

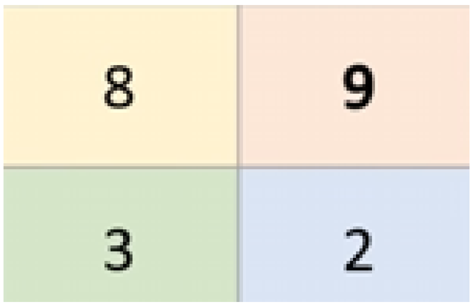
If we are given a table like this:



5	1	3	4
8	2	9	2
1	3	0	1
2	2	2	0

Here each of the Yellow, Pink, Green and Blue are each window. The biggest value of the yellow window is 8. The biggest value of the pink window is 9. The biggest value of the green window is 3 and the biggest of the blue window is 2.

Now the POOLING mechanism will create a new window, that window will have the largest values of the previous windows. And will look like this now:



8	9
3	2

This simplified new window will serve the same task as the complicated one before, and this will be easier for our machine to handle and process thus increasing efficiency. This mentioned POOLING approach is called Max pooling. Though there is average pooling which is done by doing the average of the numbers in a window and then adding it to the new window. But generally max pooling is used.

Benefits of pooling:

1) Lowers the calculation.
2) Reduce the dimension.
3) As there are fewer parameters, it reduces overfitting.
4) The model becomes tolerant to variations and distortions.

Overfitting: This is the condition when a machine learning model is trained overly well, so well that the model cannot then generalize the new data and the purpose is not served. If new data is provided the model, then cannot predict the new data.

3.4 Overall Operation of Artificial Neural Network

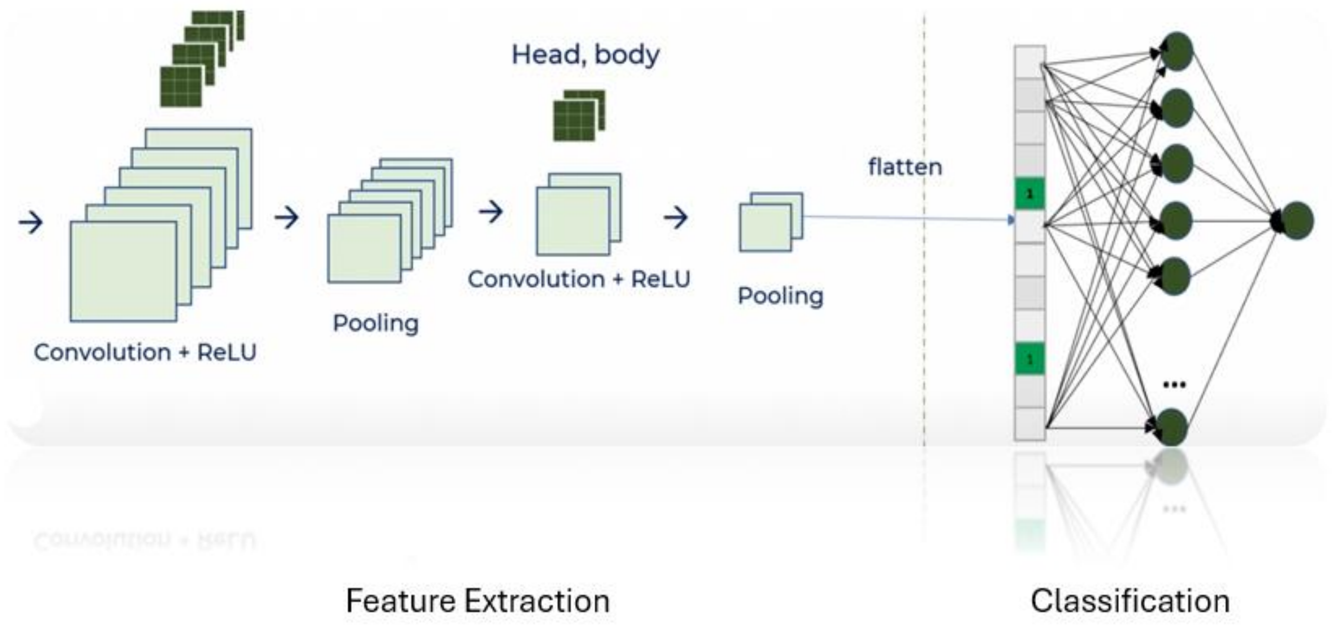


Figure 6: Overall operation of ANN

3.5 Problems faced when using CNN

While the use of CNN is very obvious, this network faces some problems when the thickness of the letter or any image or anything is changed. It then faced a problem to identify the thing. But this problem can be overcome by using more samples or more data with various thicknesses and rotations. But if there are no samples like this then "**data augmentation**" is done.

Data Augmentation: This is done when we do not have sufficient data that we need to train the model. Suppose, we don't have the data with various thickness and rotated angles, so we can use the data we have and edit those to make them thicker or thinner, or rotate them as we want, and by this we are getting more data to present to our model and train it.

3.5.1 Why we are not using ANN

a) Computation to be performed which is too hectic to the machine.
b) When dealing with images, it treats the local pixels the same as pixels far apart.
c) It is sensitive to the position of an object in an image.

3.6 Functions

In machine learning, a function is a mathematical bridge between input data and the corresponding output. It helps the model interpret input values and generate predictions by learning patterns and relationships in the provided dataset. These functions, which form the backbone of the algorithms like neural networks or decision trees, thus guide the training and validation process, and enable the machine to make accurate and reliable decisions.

There are various types of functions in machine learning they are:

3.6.1 Linear Function

In machine learning, a linear function refers to models like linear regression. It represents a method by which the machine learns to predict outputs from input data by establishing a linear relationship among the variables

3.6.2 Non-Linear Functions

Neural networks rely on nonlinearities, as without them, the network would only be capable of computing simple linear functions of its input. The choice of nonlinearity plays a crucial role in determining how efficiently the network can be trained. Different types of activation functions are derived from these nonlinearities.

3.6.3 Basis Functions

Basis functions simplify the process of capturing non-linear patterns in data while keeping the model linear in its parameters. By combining multiple basis functions linearly, we can construct complex models using linear regression techniques. There are several types of basic functions available for this purpose.

3.6.3.1 Polynomial Basis Function

This function creates a set of linearly independent polynomials which can be used to represent any polynomial function of a certain degree.

$$\emptyset_j(x) = x^j$$

3.6.3.2 Gaussian Basis Function

These objects do not require a normalizing coefficient and have a bell curve which helps us to model machine learning parameters.

$$\emptyset j(x) = \exp \frac{\{(x - \mu_j)\}^2}{2s^2}$$

3.6.3.3 Sigmoidal Basis Function

This is used in machine learning and neural networks to model non-linear data relationships. This is based on the sigmoid function and has a ‘s’ shaped curve.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

3.6.4 Activation Function

Activation functions determine how artificial neurons process and transmit information within a neural network. Depending on the task, different types of activation functions can be used. These functions are typically non-linear and, in many cases, continuously differentiable. This differentiability is essential for training neural networks using gradient descent, as it allows the computation of gradients needed to update model parameters. Mostly used function are listed in Table [4].

Function Name	Function Equation	Function Derivative
Sigmoid	$f(x) = 1/(1 + e^x)$	$f'(x) = f(x)(1 - f(x))$
Hyperbolic Tangent	$f(x) = tanh(x) = \frac{1}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Soft sign activation	$f(x) = \frac{1}{1 + \|x\|}$	$f'(x) = \frac{1}{(1 + \|x\|)^2}$
Rectified Linear Unit (ReLU)	$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
Leaky Rectified Linear Unit (Leaky ReLU)	$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$	$f(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases}$
Parameterized Rectified Linear Unit (PReLU)	PReLU is the same as leaky ReLU. The difference is $\propto$ can be learned from training data via backpropagation	
Randomized Leaky Rectified Linear Unit	$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases}$

Soft Plus	$f(x) = ln(1+e^x)$	$f\,'(x) = \dfrac{1}{1+e^x}$
Exponential Linear Unit (ELU)	$f(x)\begin{cases}\alpha(e^x-1), & x<0\\ x, & x\geq 0\end{cases}$	$f'(x) = \begin{cases}\alpha x + f(x), & x<0\\ 1, & x\geq 0\end{cases}$
Scaled Exponential Linear Unit (SELU)	$f(x) = \lambda\begin{cases}\alpha(e^x-1), & x<0\\ x, & x\geq 0\end{cases}$	$f'(x) = \lambda\begin{cases}\alpha x\lambda + f(x), & x<0\\ x, & x\geq 0\end{cases}$

Table 2: Activation Function [4]

3.6.5 Loss function

It is the difference between the predicted value and the actual value which is available as the dataset. This is a parameter for measuring the accuracy of the created network. By using these loss functions we can understand how good or bad our model is performing by seeing the difference between predicted and original values.
There are many types of loss function used in various motives in machine learning.
The table shows the loss functions used for different motives. [5]

Task	Error type	Loss function
Regression	Mean-Squared Error	$\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$
	Mean-absolute Error	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i-\hat{y}_i\rvert$
Classification	Cross Entropy Log loss	$-\frac{1}{n}\sum_{i=1}^{n}[y_i\log(\hat{y}_i)+(1-y_i)\log(1-\hat{y}_i)]=\frac{1}{n}\sum_{i=1}^{n}p_i\log q_i$
	Hinge Loss	$\frac{1}{n}\sum_{i=1}^{n}\max(0,1-y_i\hat{y}_i)$
	KL divergence	$D_{KL}(p\lvert\rvert q)=\sum_{i=1}^{n}p_i(\log(\frac{p_i}{q_i}))$

Table 3: Loss Function **[**6**]**

3.7 Machine Learning based on learning types

Machine Learning algorithms can be divided according to the forms of learning:

a) Supervised: Supervised learning involves training a model on labeled data in the form of input and output target variables. So, output is present here.

b) Unsupervised: unsupervised learning uses unlabeled data which means data without outputs

c) Semi supervised/reinforcement learning: Reinforcement learning is training a model to perform a sequence of decisions in order to obtain a final reward or prevent punishment. And the punishment is returned as the education process.

3.7.1 Tasks performed by ML

Classification: The given data set is divided into training set and test set. The model is completely trained with the training data and then applied to the test data. After that the model is applied to totally new data not previously introduced which is a form of supervised learning.

Regression: In this type of learning, the motive is to find a curve best fitting to the input curve. This is also a supervised learning where continuous values are predicted to plot the best fitting curve.

Clustering: This type of unsupervised learning groups data in terms of their similarity to each other. For unlabeled data this helps a lot.

Anomaly detection: Identifying unusual or unexpected events or patterns in data points, such as stochastic process drift or equipment failure events in semiconductor manufacturing.

The choice of algorithm may depend on the type of data and the problem that we need to solve. Some algorithms are made to do just a single thing, while others, like Decision Trees (DT) and Random Forests (RF), can be changed to do both regression and classification.

Application of ML in the manufacturing process:

3.8 Advanced Process Control [7]

APC is a method used in semiconductor manufacturing to enhance process performance by applying data analytics and control strategies. It involves gathering data from various sensors and systems throughout the production line, analyzing this information, and making real-time process adjustments based on the analysis. APC plays a significant role in improving the efficiency and quality of key manufacturing steps such as wafer fabrication, lithography, etching, and deposition. The primary goal is to reduce process variability and minimize defects in the final product. By implementing APC, manufacturers can achieve higher productivity, better yield rates, and lower operational costs.

3.9 Overview of popular machine learning methods:

Task	Model	Description	Limitation
Regression	Linear Regression	Linear relationship between independent variables X (weights w, bias b) and target Y, of the form Y = b + wX.	Not appropriate choice for highly non-linear data.
	Polynomial Regression	y = anx n + an−1x n−1 + ... + a1x + a0, more flexible regarding non-linear data.	High degree polynomials result in overfitting, computationally complex
	Regularization Techniques	Add a penalty term (λ Pp j=1 w 2 j for Ridge and λ Pp j=1 |wj | for Lasso regression) to the least squares objective function to shrink the model coefficients towards zero and prevent overfitting. Elastic Net combines Ridge and Lasso penalties into the objective function.	Likely to ignore features for highly correlated features and only focus on one of them.
	Support Vector Regression (SVR)	It can deal with complex non-linear data. It consists in finding a function that fits the training set, while minimizing the error on unseen data. It fits the best line within a threshold value a, satisfying −a < Y − wX − b < a.	Need careful consideration for choice of kernel function, model complexity, and scalability to large dataset.
	Tree-based models	Decision tree (DT) where each node represents a test, each branch the outcome, each leaf a decision. Attributes with smallest Entropy E(S) = Pc i=1(−pilog2pi), pi probability of an event i of state S, or largest Information Gain IG(S, X) = E(S) − E(S, X), are selected.	They can suffer from overfitting and poor generalization for noisy data, mitigation techniques include pruning, regularization and ensemble methods.
Classification	Logistic Regression	Standard probabilistic model used to predict binary outcomes, the probability depends on a linear measure of the sample, P (yi = +1) = 1 (1+e−β T xi) , where the parameter β maximizes the likelihood function over the training samples [10].	Assuming a linear relationship between variables and independent and identically distributed (i.i.d.) observations, may not perform well when the assumptions are violated.
	Naive Bayes	It infers the probability of belonging to a class based on the assumption of i.i.d. attributes [11]. This simplifies the probabilities, P (yi|x1, ..., xn) = P (x1, ..., xn|yi) · P (yi), making the algorithm faster and more scalable.	Performance affected by highly correlated input features and class imbalance.
	Support Vector Machines (SVM)	Based on structural risk minimization criterion, it improves the generalization ability of the model by minimizing ‖w‖ and finding a hyperplane, w T x + b = 0, that separates the input data into classes with the maximum possible margin [12].	Sensitivity to the choice of the kernel function and the selection of the regularization parameter.
Clustering	k-means	It minimizes the function J = Pm i=1	Highly dependent on the

		PK k=1 wik‖x i − µk‖ 2 to assign each data point to a spherical cluster [13]	initial conditions and not well-suited for noisy data.
	Hierarchical clustering	It builds a hierarchy of nested clusters by iteratively merging or splitting them. It is more flexible and captures non-spherical cluster shapes [14].	Computational complexity and sensitivity to outliers.
	Density-based Spatial Clustering (DBSCAN)	It views clusters as high-density regions separated by low density areas. It is based on center-based approach, where the density is estimated by counting the number of points within a specified radius ϵ [15].	Scalable for large datasets, but can be computationally expensive
Neural Networks (NN)	Feedforward NN (FNN)	Processing units organized in input, hidden and output layers. Iterative process consisting of forward propagation (s(j) = w(j)x(j − 1) + b(j), x(j) = f(s(j)), with w weights, x features, f() activation function), and backward propagation (w k ← w (k + 1) − η ←−∇E(w k), with w k parameters, η learning rate, E() cost function) [16].	It can be computationally expensive for large amounts of training data and it risks overfitting.
	Deep NN (DNN)		
	Recurrent NN (RNN)		
	Multilayer Perceptron (MLP)		
	Convolutional NN (CNN)		

Table 4: Overview of popular Machine Learning methods [7]

3.10 CNN (Convolutional neural networks) working procedure

Deep learning is a subset of Machine learning that is capable of thinking and making decisions like humans besides processing them. The word deep in this case appears since whatever the model applies, it applies a numerous number of layers in it to learn and train itself and the deeper the pattern the more complicated issues that the model would be able to solve. The ability to learn deep has become the default today since it can learn to behave as humans and even super humans. Since getting tired of doing something for a long period is possible in humans, these machines can do it and never get tired using deep learning.

Why deep learning is so incredible:

a) Speech conversion to text
b) Understand emotions from videos or voice recordings
c) Can detect anomalous activities
d) Working with specific object in a image and detect them

Convolutional Neural Networks (CNN) is one of the primary instruments of deep learning. These are famous since they are able to identify the patterns and make predictions that are near to the output. The CNNs use the back-propagation method to assess the input and give the corresponding output.

3.10.1 Advantages of CNN:

a) CNN has many processes while more significant are the feature classification and extraction. They combine these two into a single learning body. And the model can learn to optimize the features from our given dataset.
b) The sparse connections and shared weights enable CNNs to scale to large amounts of input data (e.g. images or sequences) much better than traditional Multi-Layer Perceptrons (MLPs), where each neuron is connected to every other neuron in the network. That way it becomes more powerful and uses less memory
c) While the dataset may be modified to make it simple to process, the CNN can handle the modification and work efficiently
d) CNN are not affected by the size of the dataset

In a standard neural network (MLP), every neuron just computes scalars, or numbers. But pictures have 2D data (like width and height), and therefore each neuron in a convolutional neural network (CNN) only computes a small 2D patch of the image, not the whole image.

CNN likewise makes use of 2D blocks referred to as kernels or filters as weights.

The filters generate a fresh 2D image, called a feature map, which highlights the important image contents like edges, corners or patterns.

We shall be able to assume a small 24 x 24 grayscale (black and white) picture. To categorize the image into one of two categories, CNN uses these steps:

3.10.2 Forward Transmission:

a) **Input Image**: The CNN is given a 24 x 24-pixel image.
b) **First Convolution Layer**: The first convolution layer scans the image to match patterns (which can be an edge or a texture and so on) with small filters. This results in feature maps, which are downsized copies of the initial picture that emphasize the essential details.
c) **Activation Function**: To assist the network to learn complex patterns, the convolutional outcome of each is passed through an activation function like ReLU (which we have already discussed), which adds non-linearity.
d) **Pooling Layer**: This layer decreases the dimension of the feature map (e.g. max-pooling) keeping only the most significant information. Consequently, the data becomes smaller and the model runs faster. This step can reduce the feature maps to 7×7.
e) The processes b-d is continued in order to obtain more features
f) **Fully Connected Layer**: After the final pooling, the result is very low, down to 1x1 values in some cases (as in standard MLPs), and so they are put through a fully connected layer. This layer combines all of the previously learned features in order to make final prediction.
g) **Output Layer**: Lastly, the network produces a prediction, in this case the image's classification into one of the two groups.

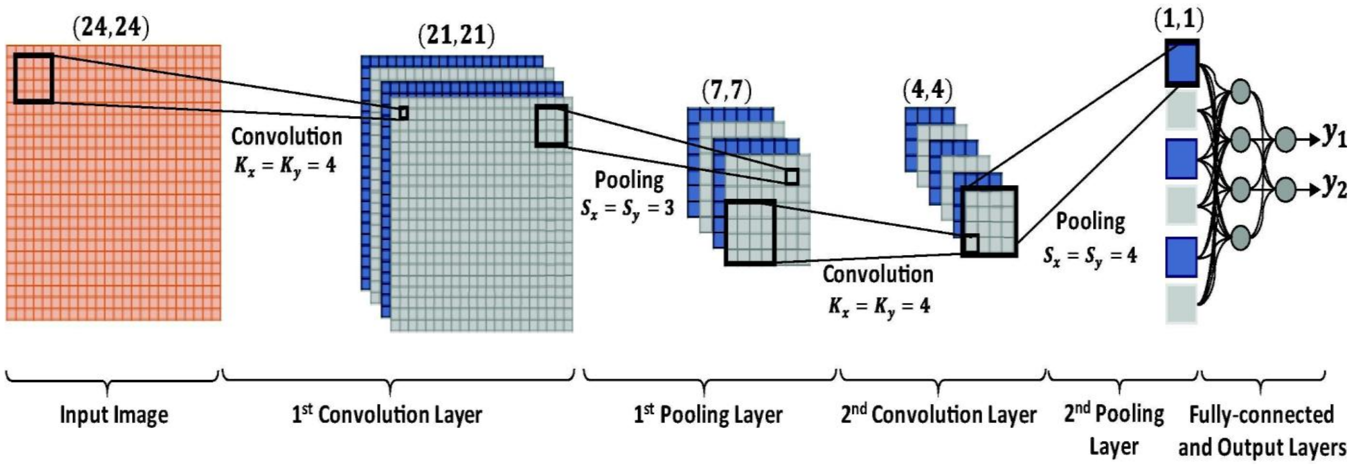


Figure 7: The illustration of a CNN with 2 convolution and one fully-connected layer [17]

As we mentioned earlier, CNNs learn by supervised and unsupervised learning. Supervised learning is the process by which CNNs (Convolutional Neural Networks) learn by being trained on labeled data.

Backpropagation (BP) is the primary algorithm used to train them. Backpropagation (BP), which functions as a kind of feedback system that gradually improves the network, is typically used when training a CNN. And by that the CNN predicts (e.g., classifies an image). By comparing that prediction to the right response (referred to as the loss), we can determine whether it is accurate or not. Backpropagation determines the contribution of each weight (parameter) to the error by iterating through the network. To lower the error the following time, we then slightly modify the weights.

3.10.3 Optimization Techniques for Weight Updates:

CNNs use gradient descent techniques to effectively modify the weights. These instruct the model on how to "learn" from its errors, as we mentioned earlier (like penalties).

Stochastic Gradient Descent (SGD): The fundamental technique that uses small data batches to make updates.

SGD with momentum: Quicken learning because it remembered the previous steps.

Adam, RMSProp, and AdaGrad: These are more intelligent algorithms that modify learning according to the actions of each parameter.

3.10.4 Classic CNNs:

LeNet: One of the first CNNs, it was used to recognize numbers, like postal codes.

AlexNet: The first deep CNN that started the deep learning revolution. It shocked the world with how well it did in a big image competition.

Both have a similar core structure, but AlexNet is much more powerful because it has many more layers and millions of parameters than LeNet

CNN is used so widely because:

Weight Sharing: All neurons in a layer use the same small filter, which means fewer parameters and faster training.

Limited Connectivity: Each neuron looks only at a small region of the image, not the whole thing and this helps the network focus on local patterns like edges or textures. And consume less memory.

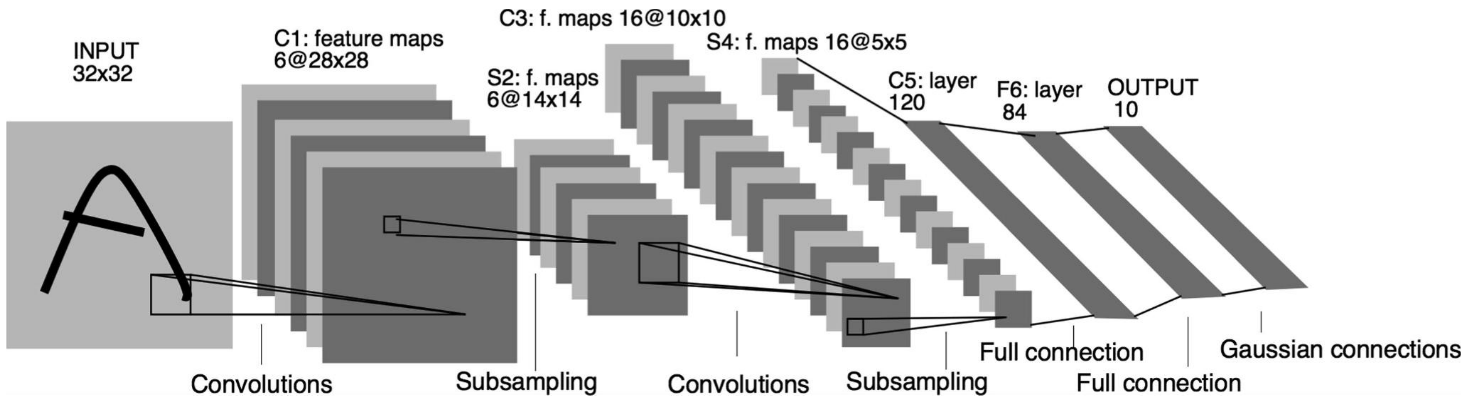


Figure 8: The configuration of the ancestor of CNNs, the "LeNet". The figure is taken from [18]. There are two interleaved convolutional and pooling layers followed by three (two hidden and one output) fully-connected layers. The output layer is composed of 10 Radial Basis Function (RBF) neurons each of which computes the Euclidean distance between the network output and ground truth label for 10 classes. [17]

Chapter 04

Deep Learning Architecture

In this angle-to-absorption task—where each absorption value is contingent upon a succession of incident angles—1D CNNs, vanilla RNNs, and LSTMs have complementing advantages that render them especially appropriate:

i. 1D CNN: Convolutional filters traverse the angle sequence to autonomously identify local motifs or patterns (e.g., minor angular changes) that significantly affect absorption. In contrast to conventional dense networks, CNNs significantly decrease the number of parameters and concentrate on the most informative temporal intervals.

ii. RNN: A simple RNN may sequentially process angles and utilize its hidden state to capture the influence of recent history on immediate absorption. It is lightweight and expedites training on comparatively modest scientific datasets.

iii. LSTM: The LSTM's gated architecture is proficient at retaining or discarding information throughout extended sequences, which is essential when retention relies on both immediate and remote angle fluctuations. Its architecture alleviates the vanishing-gradient issue that affects standard RNNs.

Alternative methods, such as fully linked networks, do not possess inherent temporal structure, whilst more sophisticated sequence models, like Transformers, would be excessively parameterized for our moderately large dataset. These three topologies achieve a pragmatic equilibrium of representational capacity, training efficiency, and resilience in our angle-absorption series.

4.1 1D CNN

Normally the neural networks we talked about are the 2D neural networks which work with images which associate width and height. But for 1D (like time-series, ECG signals, audio waveforms, sensor outputs, or in our case — ray angle sequences), it makes more sense to use **1D CNNs** instead.1D CNNs are simpler, faster, and cheaper than 2D CNNs when working with 1D data because of the faster computations and need for lesser calculation. Also, the models become smaller as they are working with minimal layers which contain one or two hidden layers. Also, as it is one dimensional so the need for a GPU is removed and work can be done only by the CPU of the computer.

1D CNN works well for us because we have a limited amount of data (labelled) and the signal is quite unpredictable as it varies a lot. **The working procedure of 1D CNN are**:

4.1.1 Input Layer

The first step is to put the raw 1D data directly into the input layer. This input is just a string of numbers that usually show a signal over time or space. In our study, it could refer to sequential data from semiconductor experiments, including intensity values or measurement points associated with ray angles.

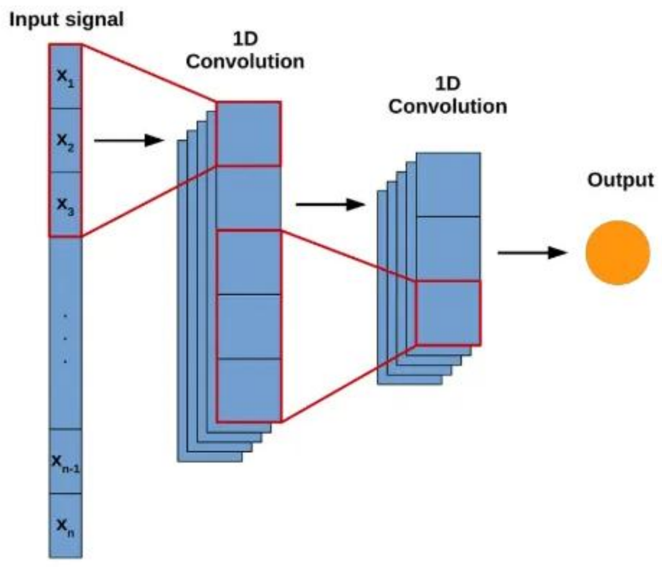


Figure 9: 1D convolutional layer

4.1.2 Convolutional Layer

Once the data is received, it is passed through one or more convolutional layers which contain filters. These layers are made to find small details in the signal. Each neuron in the convolutional layer uses a small 1D filter, which is also called a kernel. This filter convolves over the input signal. After that it does a weighted sum of the nearby input values (like a moving average) at every step of this sliding operation. This helps the network focus on small patterns in the data, like spikes, trends, or shapes that repeat. These patterns could help tell different types or categories of signals apart. There are several filters that work at the same time, and each one finds a different pattern. The output of each filter is a new 1D array called a feature map. This shows where that pattern shows up in the input signal.

4.1.3 Activation Function

After convolution, each value in the feature map is passed through a non-linear activation function, such as ReLU (Rectified Linear Unit) (discussed earlier)

4.1.4 Pooling

Pooling, sometimes known as subsampling, comes next and shrinks the size of every feature map while still preserving most significant information. In 1D CNNs, this typically involves choosing the average or maximum value from tiny portions of the feature map—that is, every 2 or 4 values. And by doing this we can cut the computation count in the upcoming layer and prevent overfitting as well as strengthen the model to withstand little signal fluctuations.

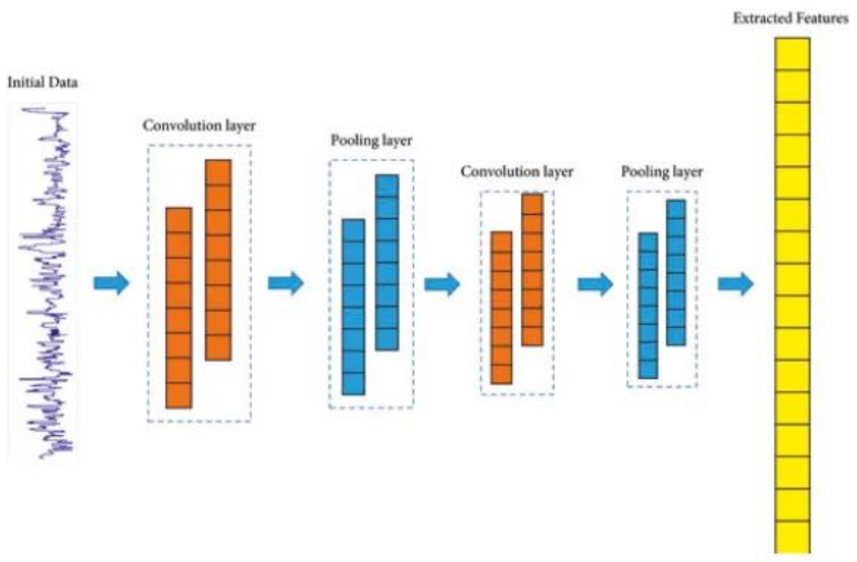


Figure 10: Pooling Subsampling inside the convolutional layer

4.1.5 Stacking CNN layers

One convolutional layer's output can be passed into another, enabling the network to learn more abstract, higher-level features. For example:

a) The first layer might find abrupt signal change or edges.
b) The second layer might pick up patterns created by several edges.
c) The third layer might pick out worldwide trends or forms.

Every new layer pick more intricate pattern from the one before it.

4.1.6 Multilayer Perceptron Layer

Once the feature extraction is finished, the last feature maps are flattened into a single vector and passed into fully connected layers, functioning like a conventional neural network (also known as a Multi-Layer Perceptron or MLP). These layers combine the extracted features and perform the actual prediction whether that is a classification (e.g., “class A” or “class B”) or a regression (e.g., predicting the precise angle of a ray).

4.1.7 Output layer

At last, the output layer generates the result. The neurons number which is used depends on the task type:

a. One neuron each class (softmax activation) used for classification
b. Regarding regression: A single neuron producing a numerical value under linear activation

Depending on the method used in this thesis, the output could show a classification label or a projected angle value.

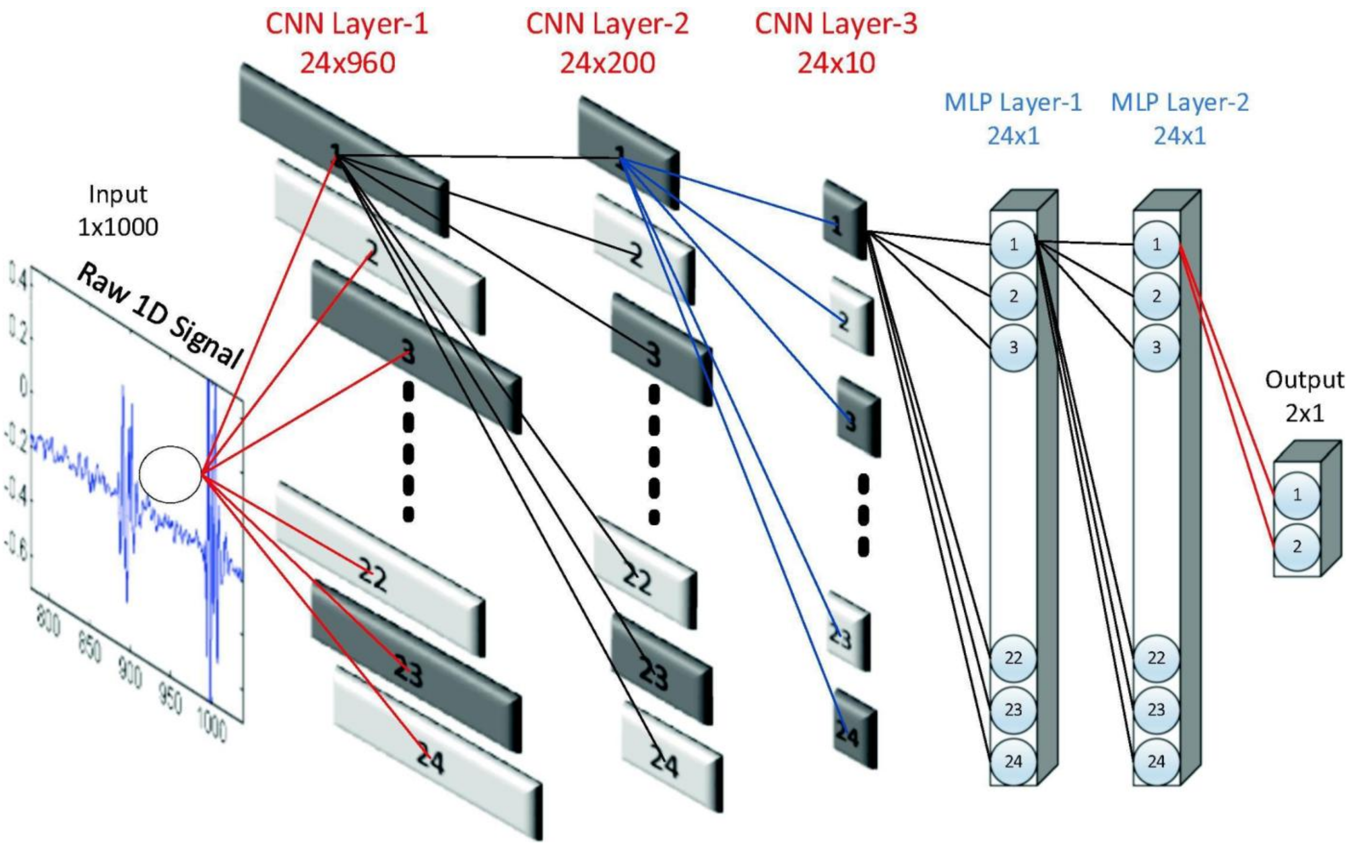


Figure 11: A sample 1D CNN configuration with 3 CNN and 2 MLP layers.[18]

4.2 RNN

Recurrent Neural Networks (RNNs): Architecture & Mechanisms

Recurrent Neural Networks (RNNs) were established based on the inherent drawback of the conventional neural networks, particularly Feedforward Neural Networks (FNNs), in terms of incapability of dealing with sequential data. The RNNs are structured to handle sequential data as compared to the FNNs which employ hidden layers to compute individual inputs without regard to the other inputs in a sequence and context.

The greatest weakness of FNNs is that they cannot learn relationships among sequential elements, and hence they cannot be applied to problems of language modelling, machine translation, speech recognition, time series analysis, and other problems requiring sequential processing. An example of this will be found in a sentence where you have to guess the next word not only based on what word appears in context but based on recalling all the words that appeared before it in context. RNNs solve this issue by introducing a memory capability that allows information to flow through time steps.

The fundamentals and the components:

4.2.1 Recurrent Connectivity Mechanism

RNNs transcend the restrictions of the vintage neural networks by the addition of the recurrent connections that enable the exchange of the information amongst the time steps. This connection is repeated and it allows the network to possess internal memory, where the output of a step is feedback as input to the next stage. This enables the model to take in information in earlier steps and operate it in the current calculation, and can discover temporal dependencies, and efficiently handles inputs of varying length.

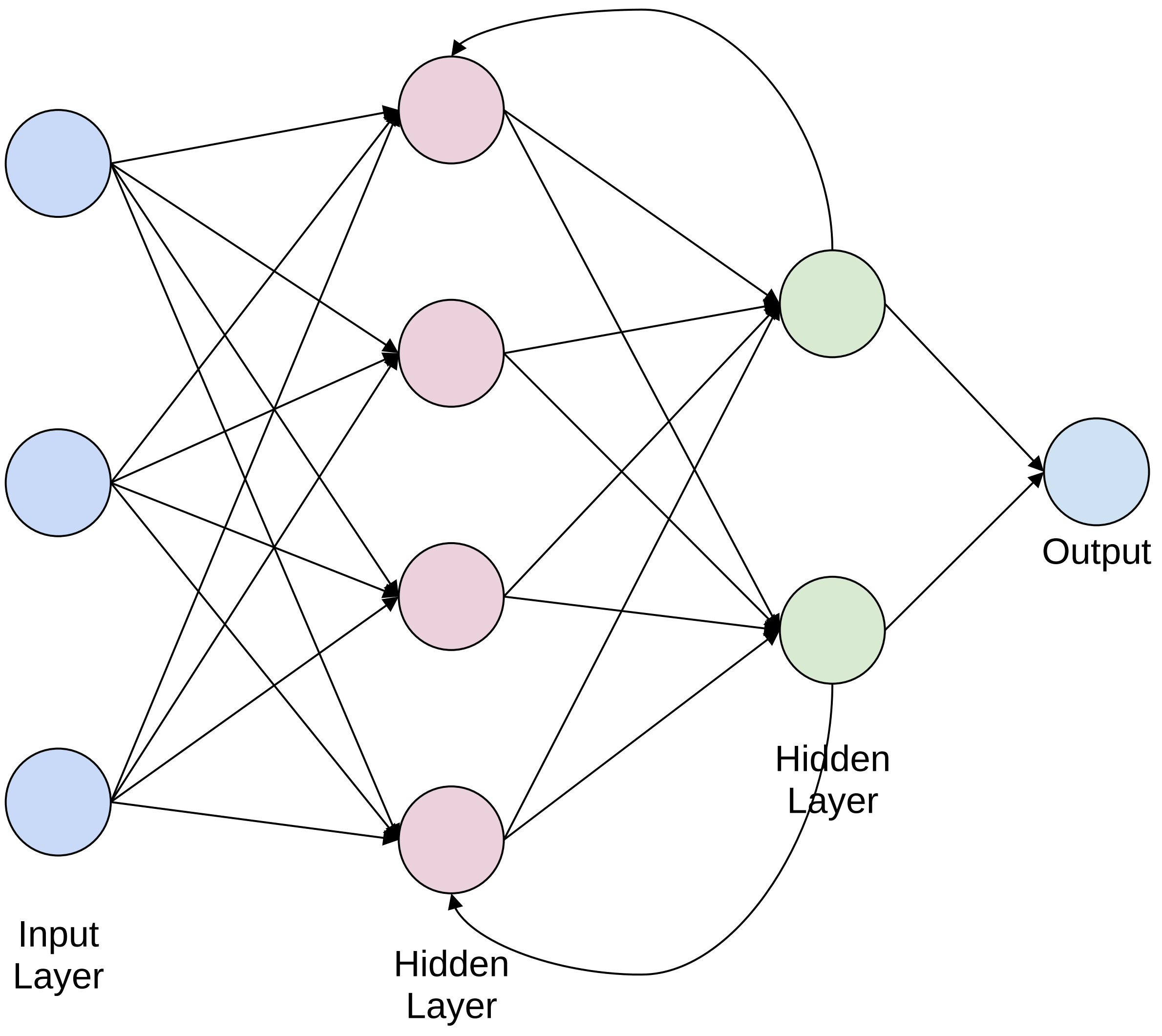


Figure 12: Architecture of RNN

4.2.2 Fundamental Components
There are two simple parts of the RNN architecture:

4.2.2.1 Recurrent Neurons:
The major processing component in RNNs is the recurrent unit, which has a hidden state, which stores past input sequence information. Such units can also "remember" information presented at previous stages by feeding back their hidden state, and in this way, they can encode dependencies over time.

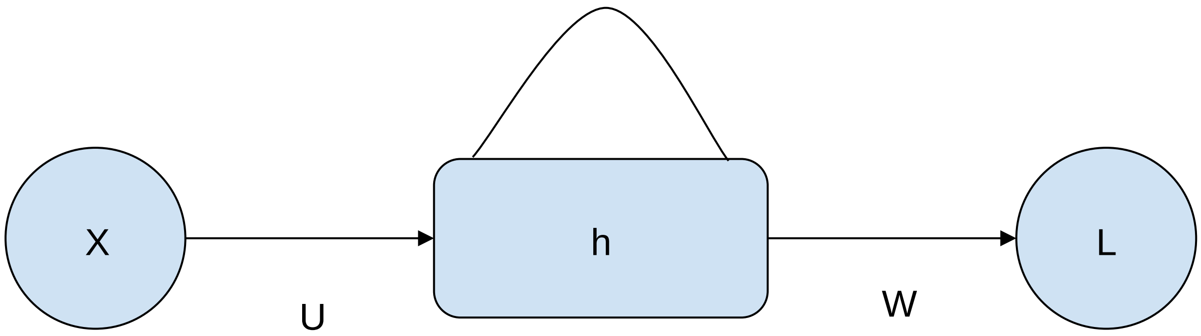


Figure 13: Recurrent Neuron

4.2.2.2 Parameter Sharing: The other key aspect of RNNs structure is parameter sharing among all time steps. The same parameters (U, V, W) are applied equally throughout the network where:

U is the weight matrix which controls the input layer to the hidden layer connection.

W is the weight matrix of the recurrent connections (hidden layer to itself)

V is the weight matrix that controls the hidden to output connections layer

It is this pass parameters forward help that is useful in causing the RNNs to efficiently learn temporal relationships, and process sequential data due to the capacity to maintain the previous input information in the present hidden state.

4.2.2.3 Mathematical Formulation

Forward Propagation

At each time step t, the hidden state a_t is determined by the current input x_t, the preceding hidden state a_{t-1}, and the model parameters:

$$a_t = f(a_{t-1}, x_t;\ \theta)$$

This can be elaborated as follows:

$$a_t = f(U \times x_t + W \times a_{t-1} + b)$$

Where a_t denotes the concealed state at time step t - x_t signifies the input at time step t - θ represents the collection of trainable parameters (weights and biases)
-U ∈ θ is the input-to-hidden weight matrix
-W ∈ θ is the hidden-to-hidden (recurrent) weight matrix
-V ∈ θ is the hidden-to-output weight matrix
-b ∈ θ is the bias vector for the hidden layer.

-The activation function is represented by the letter f, also known as tanh or ReLU.

The output at each time step t is calculated as:

$$\hat{y_t} = g(V \times a_t + c)$$

Progression Through Time

Unrolling RNN is the way to enlarge recurrent structure with time. With there being a finite number of T time steps the calculation can be envisioned as a series of layers that are all connected together of which one of the layers is the time step. This unrolling demonstrates the flow of information between time steps and enables time-backpropagation.

E.g. at T=4 time steps the hidden states are defined by

$$\begin{aligned} a_1 &= f(U \times x_1 + W \times a_0 + b) \\ a_2 &= f(U \times x_2 + W \times a_1 + b) \\ a_3 &= f(U \times x_3 + W \times a_2 + b) \\ a_4 &= f(U \times x_4 + W \times a_3 + b) \end{aligned}$$

Backpropagation Through Time (BPTT)

Training RNNs requires a particular type of backpropagation called Backpropagation over Time (BPTT), which propagates faults backward over all time steps to change the network parameters.
Forward Pass
During the forward pass, the RNN processes the input sequence from t=1 to t=n:

$$\begin{aligned} a_t &= U \times x_t + W \times a_{t-1} + b \\ a_t &= \tanh(a_t) \\ \hat{y_t} &= softmax(V \times a_t + c) \end{aligned}$$

The loss is estimated by comparing expected outputs with actual targets:

$$L(y, \hat{y}) = (^1/_T) \times \sum\nolimits_{t=1} T\, (y_t - \hat{y_t})^2$$

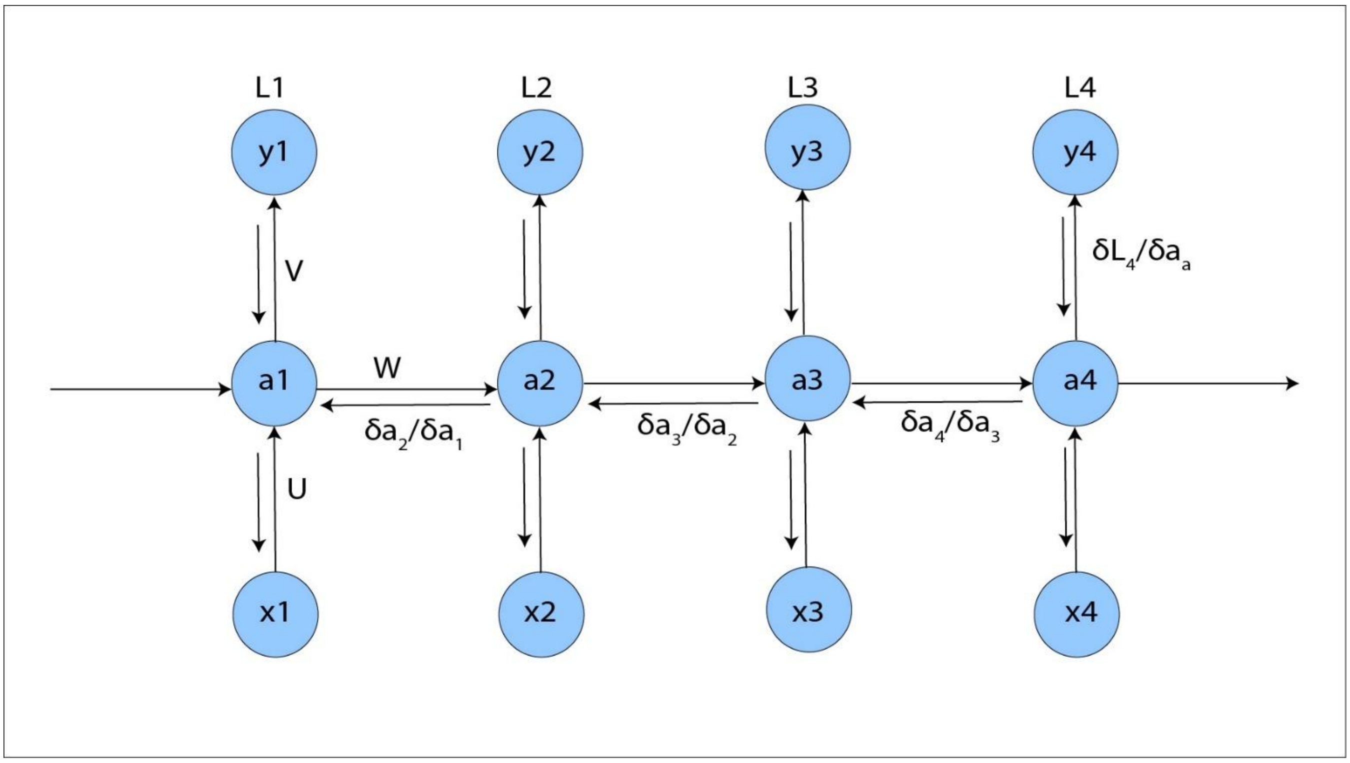


Figure 14: Backpropagation Through Time (BPTT)(2)

Backward Pass
The backward pass identifies the gradients of the loss function with regards to the network parameters at each time step. The point is that due to the recurrent connections, the gradients should consider the impact of the parameters of all the previous time steps.

For parameter W at time step t=4:

$$\begin{aligned}\partial L_4/\partial W = {} & \left(\partial L_4/\partial \hat{y}_4 \times \partial \hat{y}_4/\partial a_4 \times \partial a_4/\partial W\right) + \left(\partial L_4/\partial \hat{y}_4 \times \partial \hat{y}_4/\partial a_4 \times \partial a_4/\partial a_3 \times \partial a_3/\partial W\right) \\ & + \left(\partial L_4/\partial \hat{y}_4 \times \partial \hat{y}_4/\partial a_4 \times \partial a_4/\partial a_3 \times \partial a_3/\partial a_2 \times \partial a_2/\partial W\right) \\ & + \left(\partial L_4/\partial \hat{y}_4 \times \partial \hat{y}_4/\partial a_4 \times \partial a_4/\partial a_3 \times \partial a_3/\partial a_2 \times \partial a_2/\partial a_1 \times \partial a_1/\partial W\right)\end{aligned}$$

Similar equations apply for parameters U and V, with gradients accumulated throughout all time steps.

Constraints and Obstacles

Vanishing and Exploding Gradients: The main problem with simple RNNs is that they are vulnerable to gradient problems as they are back propagated through time:

The Vanishing Gradient Problem: gradients extended over time, can very easily become very small, and therefore the network might not be capable of learning long-term dependencies. This can be seen when the gradients are so minimal as to not update the parameters accordingly resulting in the network forgetting events that occurred in earlier time steps.

The Exploding Gradient Problem: On the other hand, gradients can grow exponentially, which

makes training unstable. The huge gradients can cause huge changes to the parameters of the model which can cause it not converging to the optimal answer.
Long-term Dependency Limitation: Basic RNNs struggle to capture dependencies across several time steps due to the vanishing gradient problem. This constraint greatly hinders their performance on tasks requiring long-term memory, such as document-level language modeling or long sequence prediction.

Advanced RNN Variants

To solve these restrictions, various sophisticated RNN architectures have been developed:

Long Short-Term Memory (LSTM): Introduces gating mechanisms (forget, input, and output gates) to control information flow and minimize disappearing gradients.

Gated Recurrent Unit (GRU): A streamlined variant of LSTM characterized by a reduced number of parameters, employing update and reset gates to regulate information flow.

Bidirectional RNNs: Process sequences in both forward and backward orientations to capture context from both past and future time steps.

4.3 LSTM

Long Short-Term Memory (LSTM) networks are an advanced variant of Recurrent Neural Networks (RNNs) designed to address significant issues inherent in conventional RNNs. RNNs are robust; yet, they struggle with long-term dependencies in sequential data due to issues such as vanishing and exploding gradients. During the training of deep recurrent neural networks by backpropagation through time, the gradients may either diminish to zero (vanishing) or escalate uncontrollably (exploding). This complicates the ability to capture long-range linkages within the data.

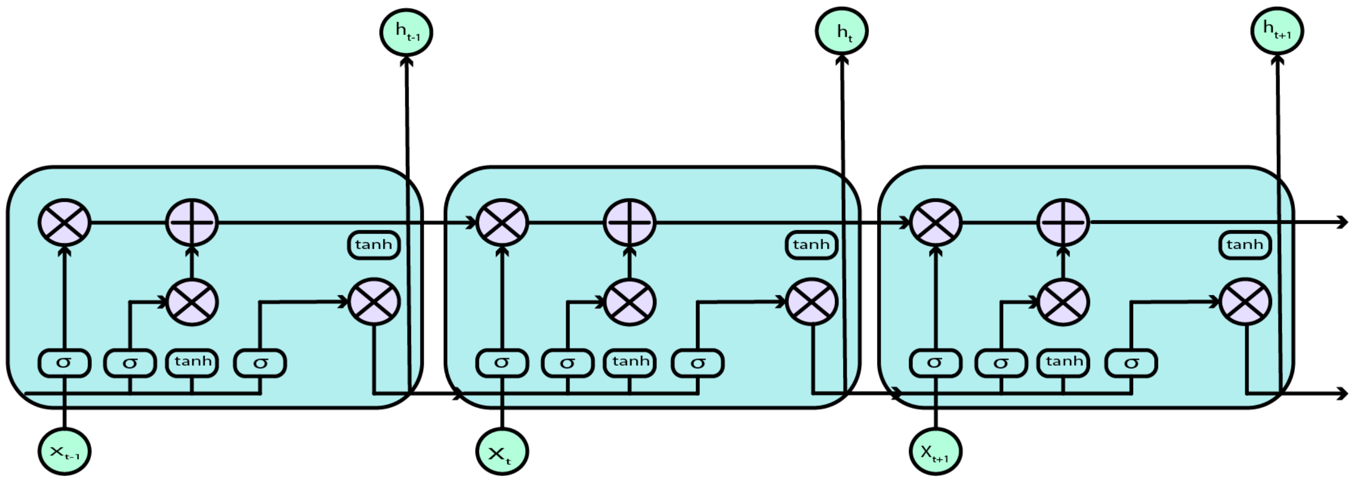


Figure 15: Architecture of LSTM

LSTMs were developed to address these issues by incorporating a memory cell that enables prolonged information retention. LSTM networks excel in tasks requiring the retention of previous inputs, such as language modeling, speech recognition, and time series forecasting. This occurs due to their capacity to retain and discard information over time.

LSTM cells possess a more intricate architecture than conventional RNNs. An LSTM's core comprises a gated unit with four primary components: the memory cell, input gate, forget gate, and output gate. These components collaborate to determine whether information should be retained, discarded, and produced at each time step.

4.3.1 Forget Gate: The forget gate ascertains which components of the preceding cell state should be eliminated. It accepts two inputs: the previous state h_{t-1} and the current input x_t. The output of the forget gate is a value ranging from 0 to 1, with 0 indicating "forget" and 1 indicating "retain". The formula for the forget gate is:

$$f_t = \sigma(W_f - [h_{t-1}, x_t] + b_f)$$

where σ represents the sigmoid activation function, and W_f and b_f are the weight and bias terms, respectively.

4.3.2 Input Gate: The input gate determines the extent to which fresh information is incorporated into the memory cell. The process is regulated by a sigmoid function, while the potential values for the memory cell are generated using a tanh function. The formulas for the input gate and the candidate memory cell C_t are:

$$i_t = \sigma(W_i - [h_{t-1}, x_t] + b_i)$$
$$C_t = \tanh(W_c \times [h_{t-1}, x_t] + b_c)$$

where i_t is the input gate output and C_t is the candidate cell state.

4.3.3 Output Gate: The output gate regulates which segment of the memory cell is presented as the output of the LSTM. It employs both the sigmoid and tanh functions to modulate the output. The equation for the output gate is:

$$o_t = \sigma(W_o - [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \times \tanh(C_t)$$

where o_t represents the output gate and C_t denotes the current cell state. The ultimate concealed state h_t is the product of the interplay between the output gate and the memory cell.

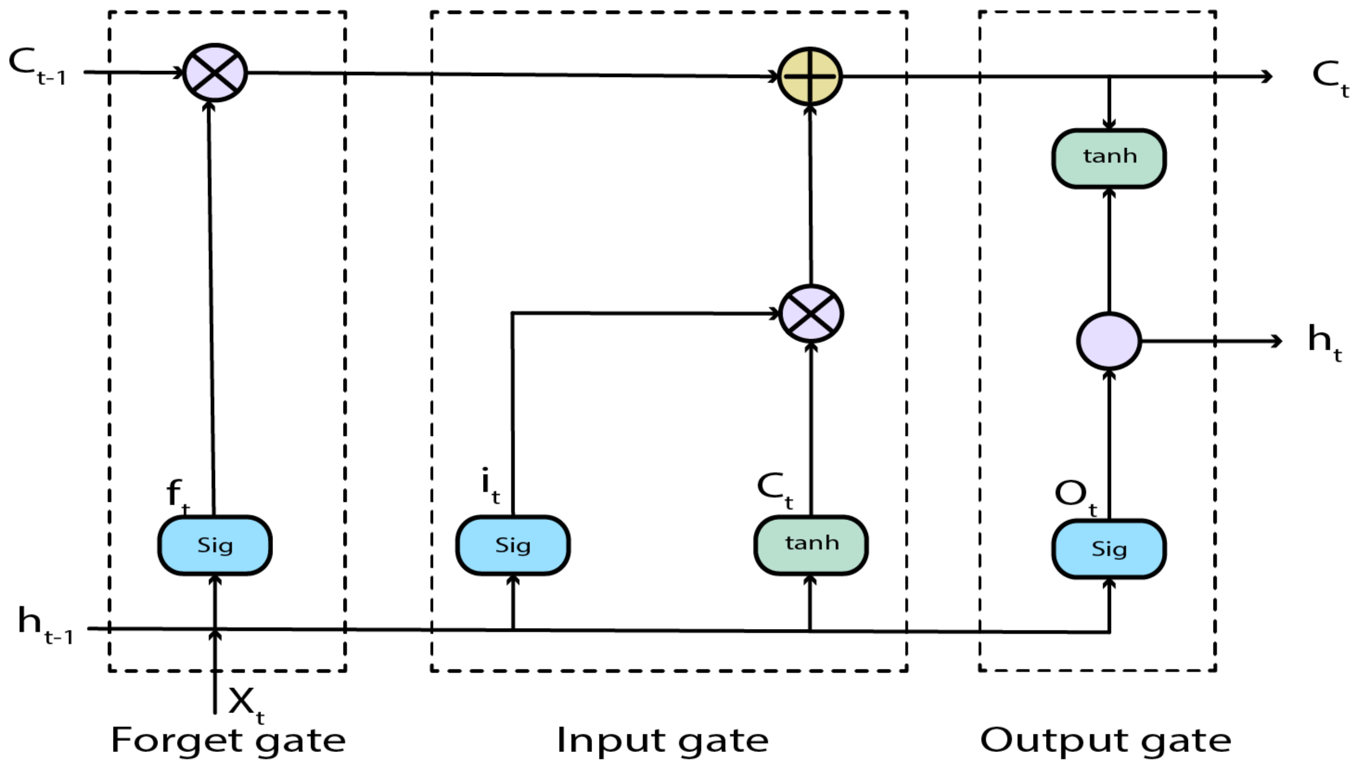


Figure 16: LSTM Model

4.3.4 Variants of LSTM

Numerous alterations to the conventional LSTM architecture have been suggested over time to enhance efficiency and performance. Several significant variants encompass:

Peephole Connections: Developed by Gers and Schmid Huber, peephole connections enable the gates to access the cell state at each time step, hence enhancing the decision-making capabilities of the gates.

Coupled Input and Forget Gates: Certain variants amalgamate the input and forget gates to optimize the process and diminish architectural complexity.

Gated Recurrent Units (GRUs) are a streamlined variant of Long Short-Term Memory (LSTM) networks, wherein the input and forget gates are consolidated into a singular update gate. Although GRUs possess fewer parameters and exhibit greater computational efficiency, they may not successfully capture long-range relationships compared to LSTMs.

4.3.5 Utilizations of LSTM Networks

LSTM networks have emerged as the preferred solution for numerous applications necessitating comprehension of sequential data. Notable applications encompass:

Language Modeling and Text creation: LSTMs anticipate the subsequent word in a sequence by utilizing the context of preceding words, rendering them crucial for applications like speech recognition, text creation, and sentiment analysis.

Machine Translation: An LSTM encoder-decoder system is employed in translation jobs, with the encoder handling the source language and the decoder producing the translated output.

Voice and Handwriting Recognition: LSTMs can learn the temporal dynamics of voice or handwriting sequences, rendering them effective for transcription tasks.

Music Generation: Similar to text, LSTMs are employed to forecast musical notes sequentially, facilitating the creation of new compositions based on acquired patterns.

Chapter 05

Methodology

In the previous chapter we discussed the architectural models that were used in this project. In this chapter we will discuss the methodology we followed throughout the whole thesis project.

5.1 Overview

The detailed flowchart below serves as the foundation for our thesis methodology. Each step in the methodology flowchart is discussed in detail in following sections.

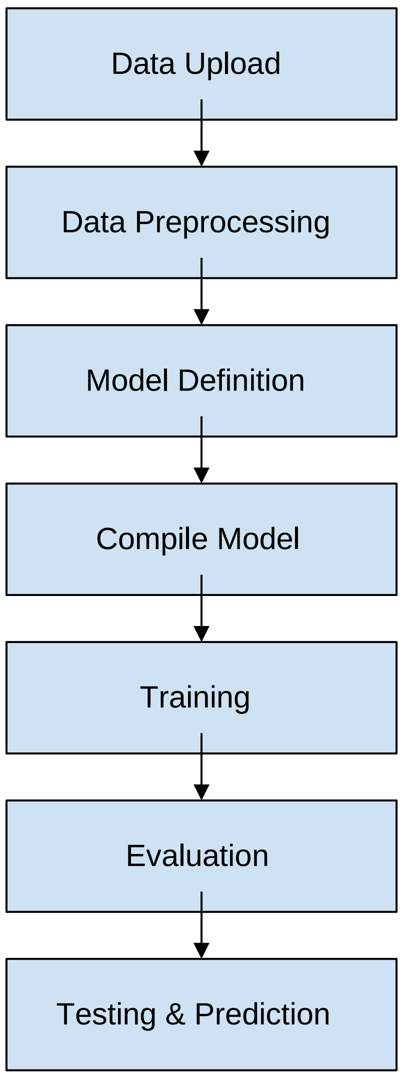


Figure 17: Methodology

5.2 Data Upload

All raw inputs are now derived from two sources. Initially, **angle_data.csv** has the recorded incidence angles (in degrees) at consistent intervals for each experiment. Secondly, we preserve an individual absorption file for each material—e.g. **data_materialA.csv, data_materialB.csv,** etc.—each containing spectra at three different sample thicknesses. We import the relevant CSV for our selected material and extract the row corresponding to the precise thickness being analyzed. In Python, both files are imported into Pandas Data Frames (`df_angles` and `df_absorp`) using `pd.read_csv()`, with meticulous designation of headers and index columns to guarantee accurate alignment of angle and absorption sequences.

5.3 Data Pre-processing

Upon loading, each Data Frame is subjected to fundamental cleaning.

To address missing or NaN values, we utilize df.isnull().sum() to identify gaps in the angle series, subsequently employing linear interpolation (df.interpolate()) to impute these deficiencies, thus maintaining a smooth variation.

Normalization: To render both features comparable, angles are divided by 90° (resulting in a range of [0, 1]), whereas absorption values undergo min-max normalization:

$$X_{norm} = \frac{(X - X_{min})}{(X_{max} - X_{min})}$$

This prevents a single feature from dominating the network during training.

5.4 Model Definition

We employed three deep learning architectures to evaluate their performance and provide a comparison among them. These are 1D CNN, Simple RNN, and LSTM.

Simple RNN: A singular Simple RNN layer with N hidden units and a dense output layer.

An individual LSTM layer, proficient in acquiring long-range relationships, is succeeded by a dense regression head. We encapsulate both into functions (build_rnn() and build_lstm()), parameterising hidden size, activation, and dropout.

5.5 Compile Model

Every model is assembled with:

Loss: Mean Squared Error (MSE), suitable for continuous-value regression.

Optimizer: Adam, configured with an initial learning rate of 0.001.

We observe the Mean Squared Error on the validation subset to inform early stopping.

5.6 Training

Models are trained for a maximum of 300 epochs with:
Batch size: 32
Implement early-stopping callback with a patience of 10 to terminate training if validation loss fails to improve.
Utilize Model checkpoint to preserve the optimal weights.
This method often converges between 40 to 60 epochs, achieving a balance between underfitting and overfitting.

5.7 Evaluation

After training, we load the optimal weights and analyses performance on the validation set. We document:
Validation MSE and MAE (Mean Absolute Error)
Loss curves are presented across epochs to illustrate convergence and identify potential overfitting.

5.8 Testing & Prediction

Ultimately, we implement the learnt model on the test sequence. The anticipated absorption values are re-normalized to their original units and graphed alongside the actual measurements. This visual comparison, accompanied with error data (RMSE, R^2), finalizes the methodology—illustrating how both RNN and LSTM encapsulate the fundamental physics represented in the angle-to-absorption mapping.

Chapter 06

Result Analysis & Comparison

In the previous chapter we discussed the methodology we followed. In this chapter we will discuss the evaluation result and analyses all the results and lastly make comparison of each model.

During the result analysis phase of our thesis, we carefully examine the outcomes of our deep learning experiments in great detail to extract significant insights and make conclusive conclusions. We analyzed the different compositions of zinc, magnesium and molybdenum alloy. And evaluated the different compositions. Each composition name is in aX_bY_cZ format. Here a = zinc, b = magnesium, c = molybdenum & X, Y and Z are thickness in nm. Below is the outline that will be followed.

1. Performance Evaluation.
2. Comparison.

6.1 Performance Evaluation

6.1.1 Material Evaluation – Composition 1 (a0_b30_c1):

Prediction Accuracy: The material's predicted absorption curve, when processed through these models, shows a high degree of accuracy with the LSTM model, closely following the clean curve even in noisy conditions. It means that in an environment, where noise could corrupt data, the behavior of materials (absorption behavior) could be safely estimated using the right model (LSTM).

Noise Effect: LSTM model is a robust alternative to predict the behavior of the material in noisy environments where the data is subject to change due to either measurement or environmental ambiguity. Under such cases, the CNN and RNN models are less reliable, CNN is hard to remove the high-frequency noise, and RNN is more inconsistent in prediction.

Long-Term Behavior: LSTM model has a significant advantage in terms of accuracy and noise sensitivity in cases where the long-term behavior of the material has to be predicted or in cases where the dataset has a temporal dependency (e.g. material behavior over time). CNN and RNN, in their turn, would require additional preprocessing and adjusting to reduce the noise, and thus would be less reliable to use in such tasks.

6.1.1.1 Curve Prediction Using CNN:

The CNN model prediction is illustrated by the green dashed curve. Operatively, it approximates the overall trend of the original curve satisfactorily with certain discrepancies around the values of 60 degrees and 70 degrees. This model appears to give reasonably close predictions as compared to the clean curve, but certain high frequency noise still exists in the predicted curve. It reduces much of the noise, yet small variations are still there. The model reproduces the larger trend nicely but it does not remove the noise completely particularly in the noisy areas 75 to 85 degrees.

Even though CNN models achieve a high score in the general trend, there are still certain noisy outliers on our predicted curve. At the lower end, around 60-70 degrees especially. It works well in reducing the noise but does not exactly trace the original clean curve.

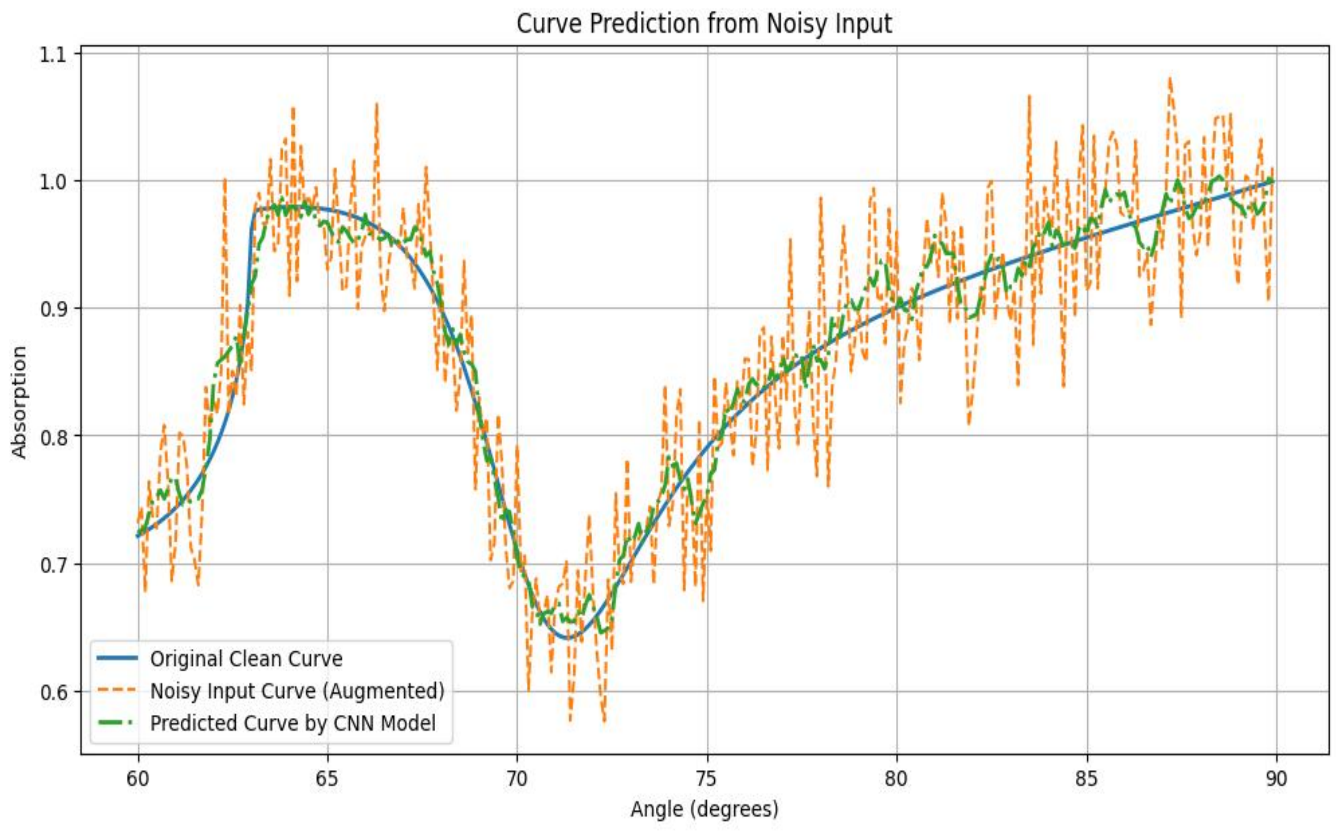


Figure 18: Curve Prediction using CNN (Composition 1)

6.1.1.2 Curve Prediction Using LSTM:

The LSTM smooths the noise better than the CNN model. The predicted curve closely follows the original clean curve despite certain slight distortions at the end. It seems that LSTM as a sequential model has more solid predictions over time than CNN model. It maintains the overall trend and low pass filters the high frequency noise, but it can slightly overfit the curve in the range of higher angles. The LSTM model tends to be more accurate. the predicted curve closely follows the clean curve particularly compared with the CNN model.

Among the three, the LSTM model demonstrates the best results. It highly follows the original curve, with minimum noise and fluctuations. It is more suitable to deal with sequential data and eliminates high frequency noise hence more accurate than CNN.

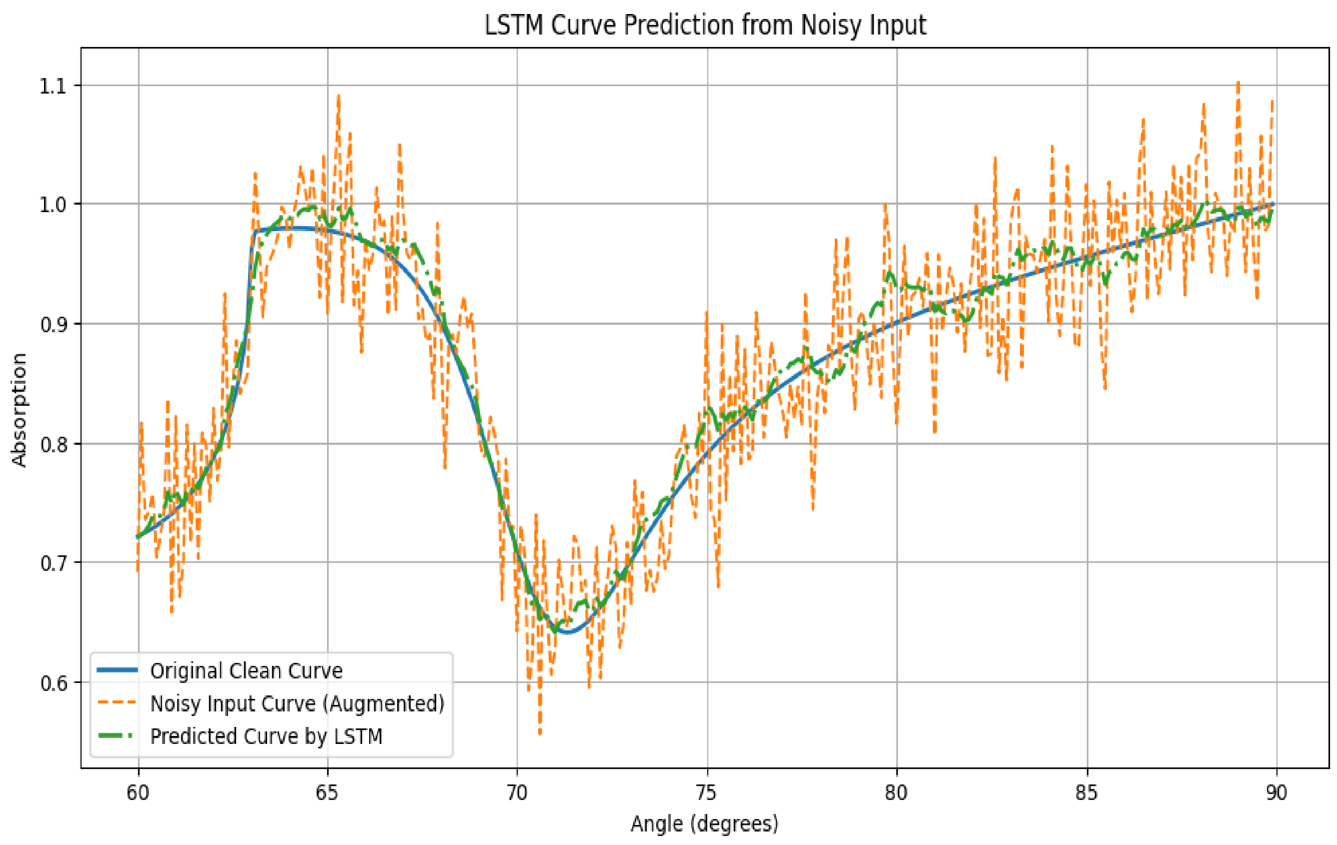


Figure 19: Curve Prediction using LSTM (Composition 1)

6.1.1.3 Curve Prediction Using RNN:

The prediction of the LSTM model is almost similar to the RNN model. But it has greater variation in the predicted curve than the LSTM model, particularly around the high range that is around 300 degrees. The RNN exhibits a certain number of oscillations and increased variance on the predicted curve, particularly at the larger angles. Although it provides a smoothing effect on the noisy data as compared to the simple CNN, it is not as good as the LSTM model. The model might struggle with long term dependencies which causes a slight overfitting on the high ends. The RNN does a reasonable job, however, predictions are not as clean as LSTM predictions, and the higher angle range, which lies above 150 degrees, is not well predicted as compared to the LSTM predictions.

The RNN model provides a decent forecast but is not as smooth as the LSTM model and particularly not in the latter parts of the curve. The curve has some unwarranted variations and the model is not ideal in dealing with the noise compared to the LSTM.

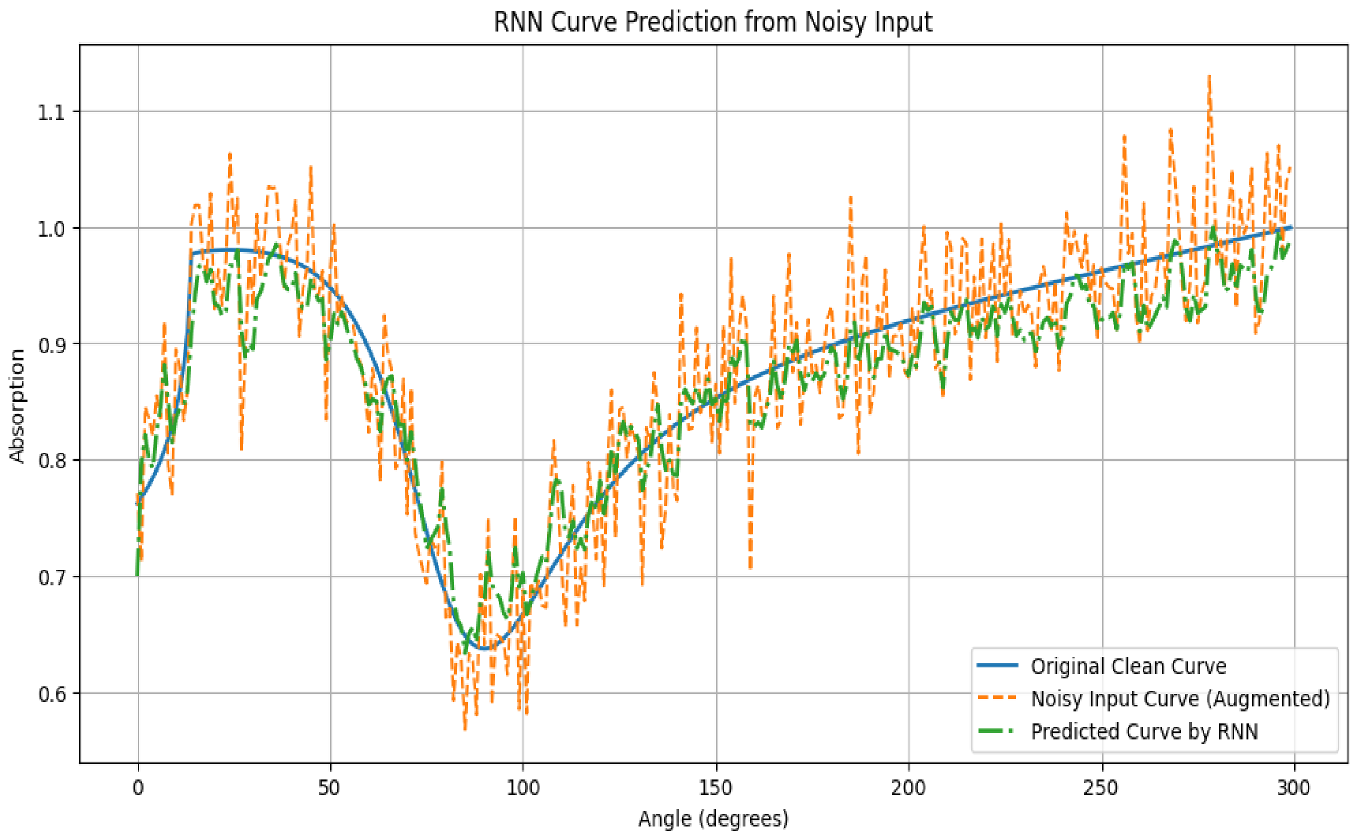


Figure 20: Curve prediction using RNN for noisy input (Composition 1)

According to the discussion on the prediction curve models, the LSTM model is the most suitable model to use in predicting the material behavior with noisy data. The fact that it can deal with sequential data and smooth noisy data points makes it very well applicable in modeling complicated and noisy material behaviors. The CNN and RNN models are also reasonable alternatives, yet they lose in long-term accuracy and noise processing. Hence, LSTM is the preferable method when high-accuracy results in predicting material properties, especially in noisy environments, are desired.

6.1.2 Material Evaluation – Composition 2 (a0_b30_c2):

Prediction Accuracy: The general prediction accuracy of the models differs as some of the models closely track the original clean curve whereas others show a significant deviation. The models that predict the curves that match the clean curve closely are the best performing ones and the differences between them are small. Such models are good at catching the overall trend of the data, and despite the best fitting models, there remains some errors, particularly in areas where noise is stronger. Although high prediction accuracy can be attained there still exists a small margin of error especially when highly varying input data is involved.

Noise Impact: An essential element of effectiveness of these models is noise suppression and the difference between the models in their ability to deal with noisy input data is significant. Certain models will greatly succeed in noise reduction, leaving smooth steady predicted curves that will closely follow the initial clean curve. Others, though, have a harder time when it comes to suppressing noise which causes the predicted output to fluctuate and become unstable. In the models which perform well in noisy environments, the latter prediction curve is not very sensitive to the high frequency variations whereas the others are more vulnerable to noisy environments which interfere with the overall smoothness and stability of the prediction.

Long-Term Behavior: A related concept to long-term consistency is important when the task involves making predictions over a long time, in which case it is also relevant to determine model performance. Other models exhibit resilience to longer sustained noise by having a stable predicted curve over longer sequences. Other models, by contrast, start deviating (as the sequence length grows) on the clean curve. The deviations observed in these models are more prominent after the course of time, meaning that these models cannot demonstrate the same accuracy and consistency in the long-term. If the long-term behavior of a model is good, then it can capture the original trend even in the presence of noise or variable data, whereas models with poor long-term stability become more unstable as the sequence length grows longer.

6.1.2.1 Curve Prediction Using CNN:

The CNN model is seen to be very capable of learning the clean curve based on the noisy data, where the predicted curve closely resembles the original clean curve. It works really well removing most of the noise, but there are some small irregularities, especially in the areas with stronger noise. The overall trend of the clean curve is fairly represented, noisy as it is. This model however decreases in performance somewhat in longer term prediction, where it is challenged to continue being consistent in the presence of long duration noise. The CNN model works significantly in the identification of local patterns and reducing noise in the short sequences. It suppresses high frequency noises but some noises are still observed in the forecasted curve. The model is good at noise suppression; however, as the sequence length enlarges, the model accuracy is reduced, and the long-term behavior is not so stable. All in all, the CNN would be a robust contender in denoising applications where the short-term trends and patterns are of greater importance.

The CNN model generates a forecasted curve that is fairly near to the actual clean curve. In the predicted curve, small variations can be seen especially at points of high noise but the overall shape and direction have been maintained. The curve predicted by the model however contains some differences in long term behavior, suggesting that the CNN is not as good at dealing with long term noise, or consistent in longer sequences.

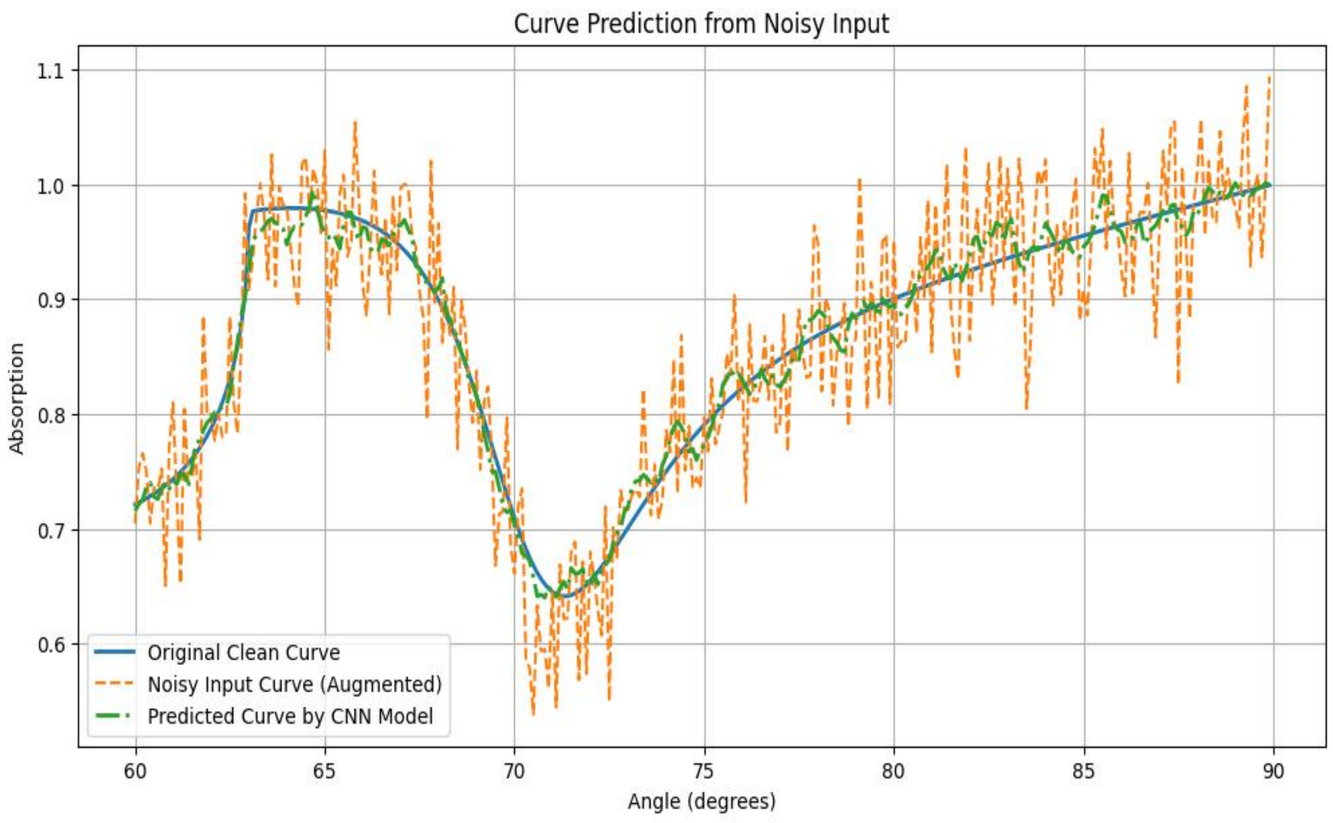


Figure 21: Curve Prediction using CNN (Composition - 2)

6.1.2.2 Curve Prediction Using LSTM:

The LSTM model gives an estimated curve which is practically the same as the original clean curve. The estimated curve is smooth, with the minimal deviation even in the areas with the significant noise. The ability of LSTM to learn long-term dependencies in a sequence data has also empowered it to be very consistent in the long-term. Even with noisy input, the model output closely follows the original clean curve. When comparing LSTM to the other models, it excelled in every field of prediction such as noise processing, prediction accuracy and long-term behavior. Noise aside, it is smooth and stable in the curve, and capable of filtering out the fluctuations without Significant loss of important detail of the original curve. The fact that it can capture long-term trends and yet deals with noise is a major plus and therefore LSTM is the best at denoising noisy sequential data.

The LSTM model's predicted curve is almost similar to the original clean curve. Deviations are few, especially in comparison with the CNN and RNN models. The LSTM model captures the shape and the long-term trend of the clean curve and thus it is the best model in this assessment. It can deal with long- and short-term variations very accurately.

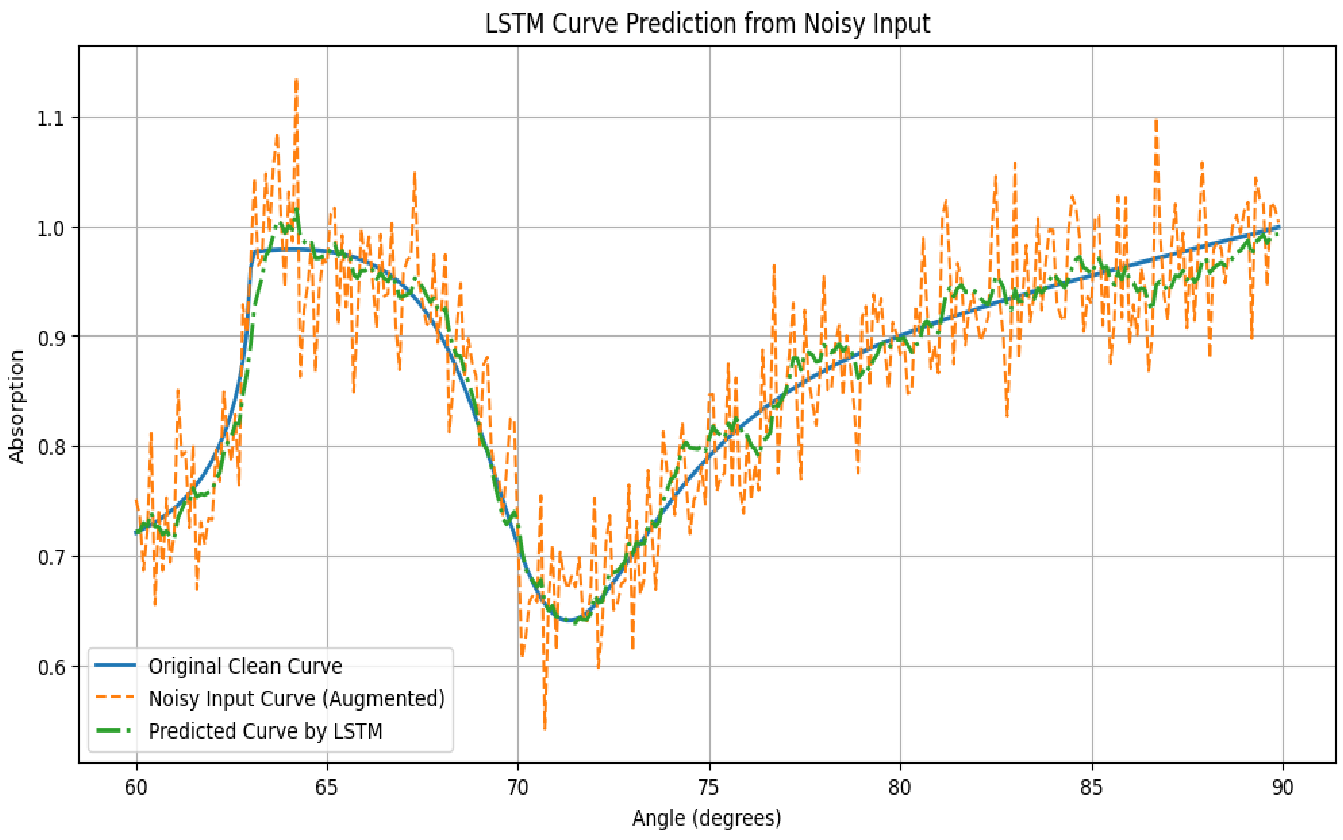


Figure 22: Curve Prediction using LSTM (Composition - 2)

6.1.2.3 Curve Prediction Using RNN:

The RNN model gives the predicted curve which follows the overall trend of the original clean curve although the deviations are more prominent than in CNN and LSTM models. The predicted curve has bigger oscillations in high-noise areas, demonstrating that the RNN model is not good at suppressing noise. The RNN also suffers the problem of finding it harder to sustain consistent predictions on longer sequences and the curve gets progressively more unstable the longer the sequence. Although the RNN model continues to follow the overall trend of the clean curve, it is more vulnerable to noise, and its forecast is not as accurate and stable as the CNN model and LSTM. This is because the simple architecture of the RNN does not allow it to deal well with long-term dependencies thus the more the distance between the original curve and the one obtained increase with time. This model is not very good at short-term noise suppression and long-term prediction of behavior, and is, therefore, the least reliable among the three models.

The RNN model is more unstable and has bigger deviations in comparison to the original clean curve. The predicted curve has some obvious noises, especially in high-noise areas, and the model cannot be accurate in a long period. Generally, the RNN model is the weakest of all three models especially in assignments that deal with noisy and long-term data.

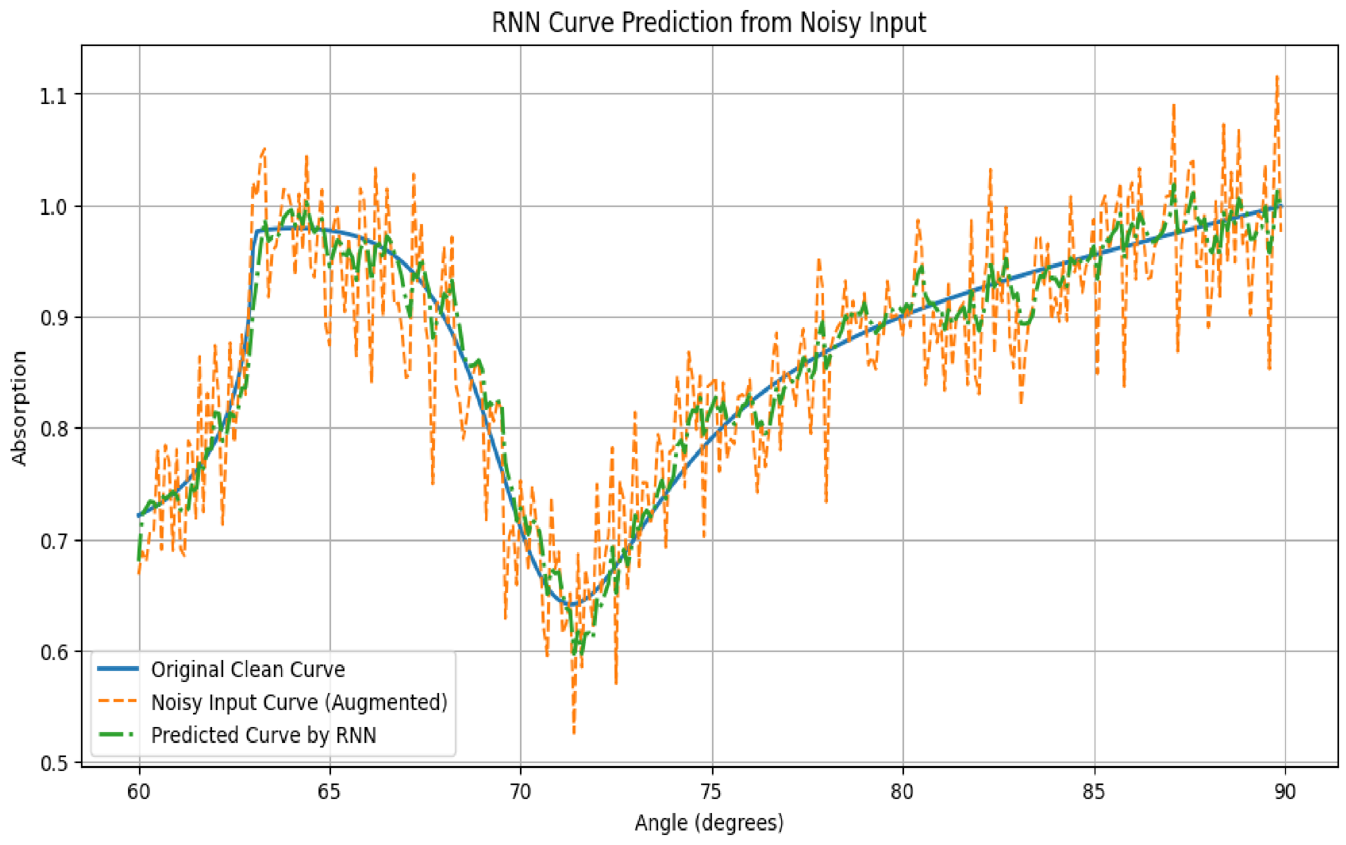


Figure 23: Curve Prediction using RNN (Composition - 2)

LSTM offers the best accuracy and noise processing capabilities and is superior in modelling long-term dependencies and is stable in predictions even in noisy environments. Nonetheless, it is at the expense of computational efficiency since LSTM is more demanding in terms of resources to train and run inference. CNNs are competitive in terms of accuracy and noise processing (specially on shorter sequences) and are the least computationally demanding model, which makes them suitable in tasks where real-time prediction or reduced resource consumption is a concern. RNN is also outperformed by LSTM and CNN in accuracy and noise processing, as it tends to further differ from clean curves, particularly in noisy areas. It is, however computationally more efficient than LSTM, but more demanding than CNN on large-scale tasks.

6.1.3 Material Evaluation - Composition 3 (a0_b30_c3):

Prediction Accuracy: In all three models, CNN, LSTM, and RNN, the prediction accuracy is overall high, and the predicted curves (green lines) are close to the original clean curve (blue). The level of accuracy is however slightly different among the models. The LSTM model has superior accuracy especially on complicated noisy input. It is highly consistent with the clean curve across the range with minor differences in areas of high fluctuations. The CNN model is also quite good as it predicts a correct value and catches the majority of the significant trends of the clean curve. The RNN model, although correct, has some visible discrepancies, particularly in noisy, or highly variable parts of the data, meaning that it predicts the trends of the data slightly less accurately than the other two models.

Impact of Noise: When noise is added to the data set, all the three prediction measures are highly affected. The models however differ in the sensitivity to the noise. The CNN model is not sensitive to noise, and the predicted curve is usually smooth and close to the clean curve. It might however have a bit of difficulty in areas that have a high intensity of noise. The LSTM model is the most noise robust, where its capture of sequential dependencies enables it to smooth noise more efficiently and still preserve a high prediction accuracy. The RNN model is more susceptible to noise where in most cases the predicted curve will have larger fluctuations and will go out of shape compared to the clean curve at noisy regions. This demonstrates how the RNN is vulnerable to noise, a fact that affects its prediction outcomes throughout the models.

Long-Term Behavior: Regarding the long-term behavior, in particular, how well the models maintain the predictive power throughout the entire range of angles (60 90), the LSTM model is the best. The LSTM model, because of how it is designed to capture long-term dependencies, remains to make consistent and smooth predictions even as the data shifts to higher angles. The CNN model is a strong model that might have some slight drift away from the clean curve in the areas of long-term fluctuations, but the overall performance is dependable. This is a weakness of the RNN model, as it can be seen to be rather unstable and sensitive to noise in the longer parts of the data, and thus demonstrates greater variation of the clean curve in some areas. It indicates that RNNs, which lack the capacity to efficiently model long-term dependencies, have trouble in ensuring a steady prediction accuracy as time progresses.

6.1.3.1 Curve Prediction Using CNN:

The model is quite smooth in its predictions, at least it catches both major trends of the curve and minor details as well. But it could have slight inconsistencies in the noisier areas, as CNN models typically have with time-series data. The CNN model has rather good results in predicting the clean curve. The model is not overfitting to the noise but generalizes quite well and gives accurate predictions throughout the angle range (60-90). One of the selling points of CNNs is their capability to capture spatial patterns, and in this situation, it appears to be dealing with the pattern embodied in the time-series data fairly effectively.
The predicted curve closely estimates the original clean curve, and remains very close within the range of angles. The predicted curve closely resembles the original clean curve in the areas with small peaks and troughs, which implies that the CNN model is useful in capturing the underlying trends of the data. The minute anomalies particularly in noisy sections indicate that the model can probably be enhanced by changing the hyperparameters of the model or training the model longer.

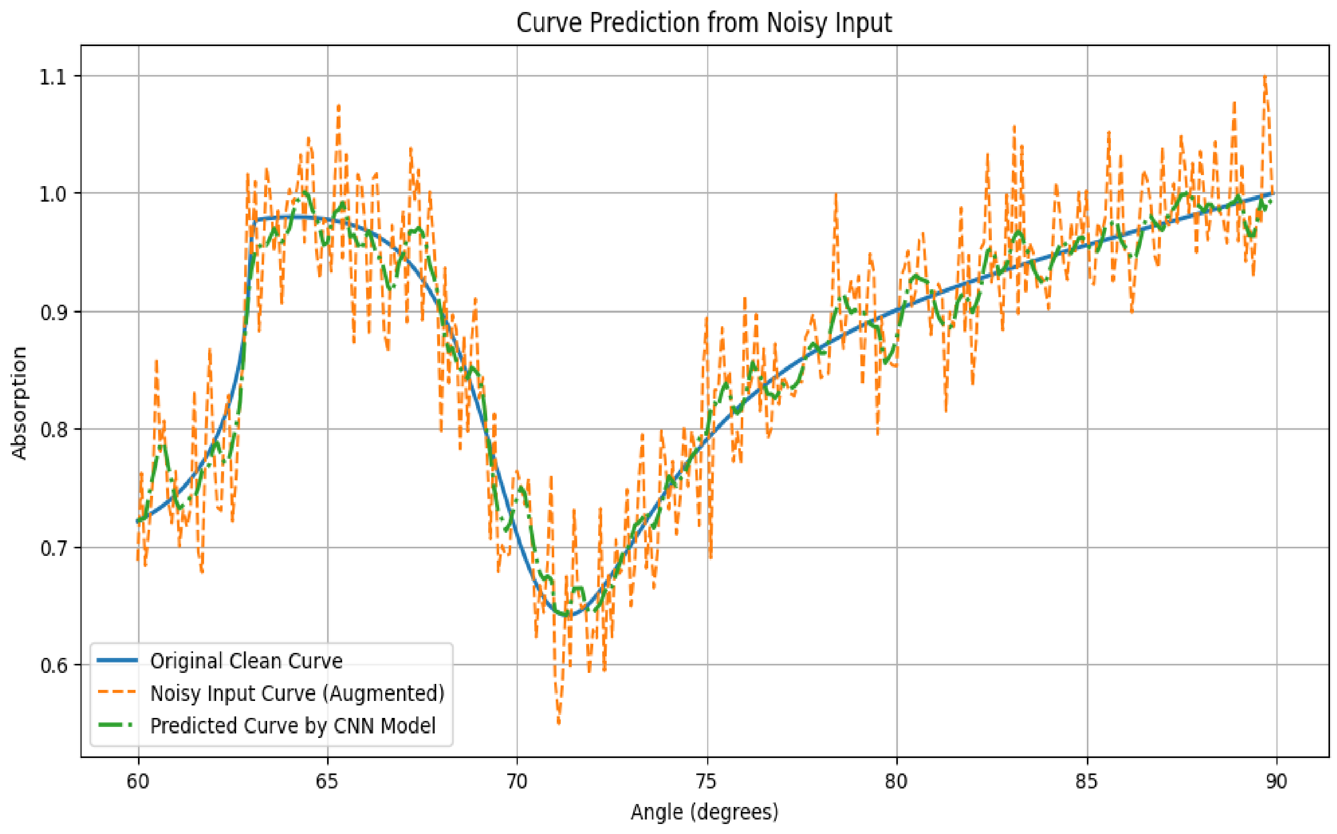


Figure 24: Curve prediction for noisy input (Composition 3)

6.1.3.2 Curve Prediction Using LSTM:

In contrast to CNNs, LSTMs are sequential models and long-term dependency networks. As can be anticipated, the LSTM is unfazed by the variation in the noisy data and retains a consistent trend of prediction, which corresponds to the clean curve. The LSTM model does almost the same as the CNN model in this case. The model copes with the noise and is generalizing to predict the clean data. The LSTM however has a benefit over CNNs in time-series data since it intrinsically learns sequential patterns and dependencies. The model could further perform better with larger dataset or more intricate time-series data because LSTMs are superb at learning long-term sequences.

The forecasted curve almost coincides with the initial clean curve with some deviation being noticed in some areas. Such deviations are rather minor, and the LSTM model approximates the original curve quite well, as was the case with CNN. The LSTM performs a little better than the CNN in areas where the data varies quickly (sharp peaks and valleys), which demonstrates its strength when dealing with time-series data modeling.

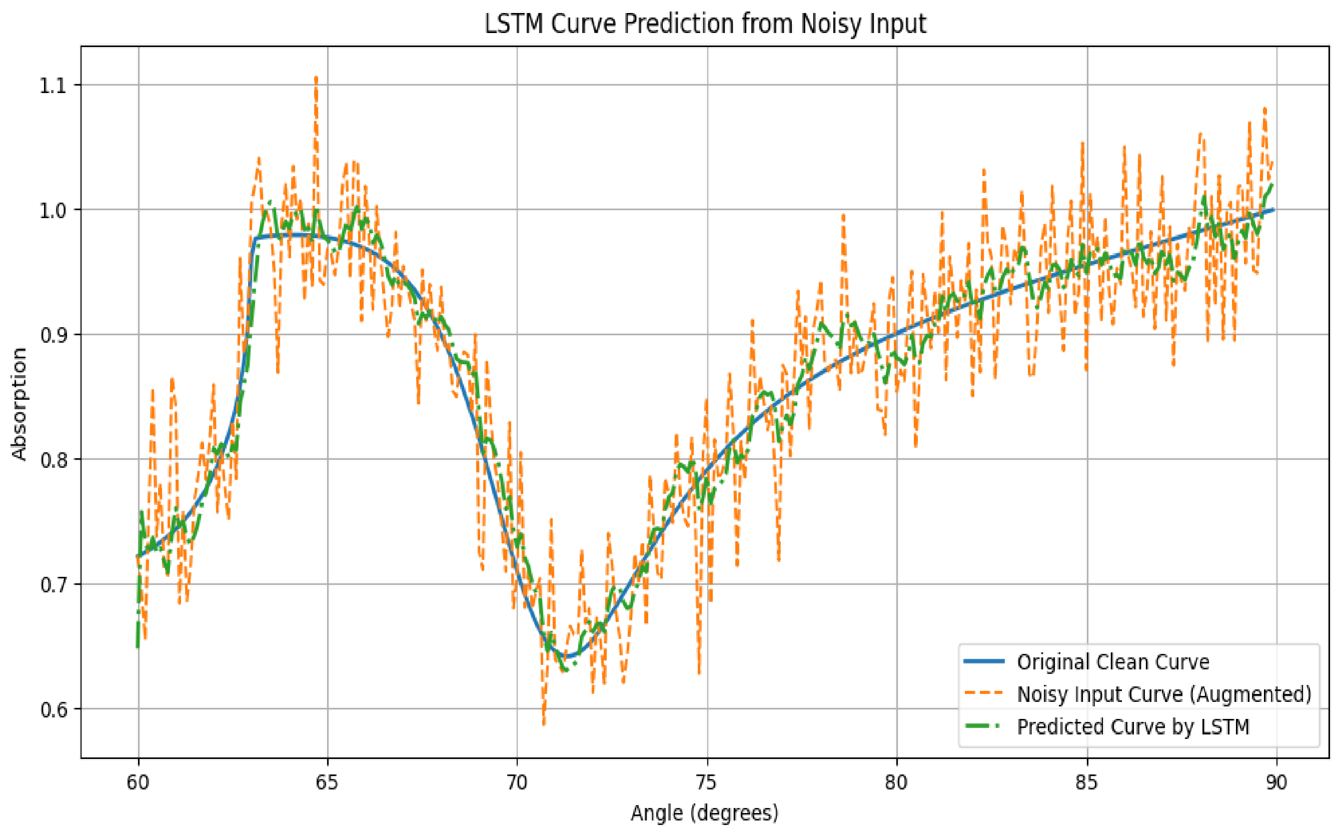


Figure 25: Curve Prediction using LSTM (Composition - 3)

6.1.3.3 Curve Prediction Using RNN:

Although the RNN model closely tracks the clean curve, it is seen to be more erratic in the prediction. The model appears to be more sensitive to the noise in the data so it causes larger differences to the clean curve in some areas. The RNN model performance is a little bit weaker than the CNN and LSTM models, particularly in those areas where the data is noisy or has high variation. RNNs tend not to perform as well as LSTMs on long-term dependencies in sequential data, and here that distinction is seen. Compared to the CNN and LSTM models, the predicted curve by the RNN has a greater variation indicating that the RNN model might be sensitive to noise and long-range dependencies in the sequences.

The RNN model predicted curve is a little more erratic and does not closely follow the original clean curve as the CNN or LSTM models. The RNN is over-sensitive to the noisy input variability and this causes the predictions to be further away (in time) with respect to the clean curve. Compared with the clean curve, the predicted curve by RNN is not as accurate, particularly, where the data shows sharp variations or where noise is high.

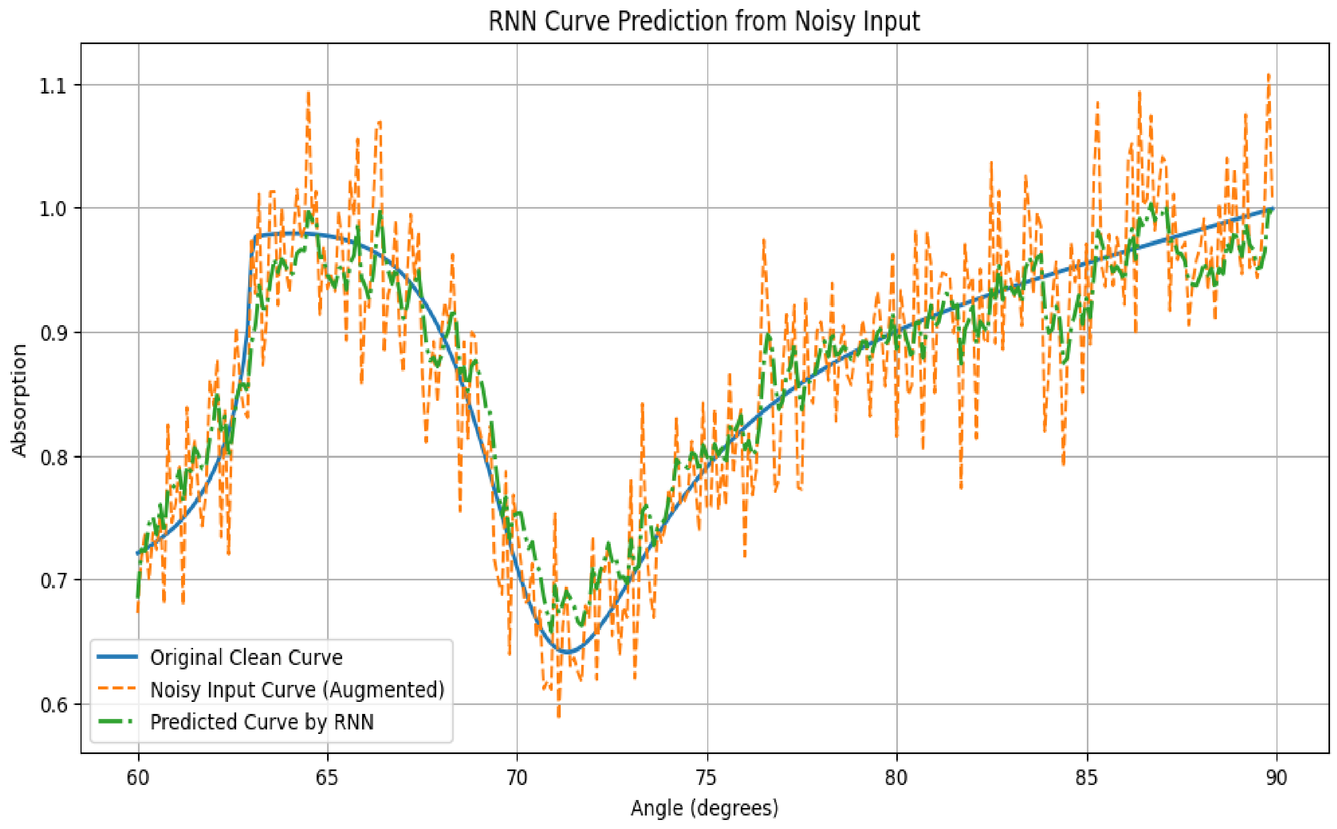


Figure 26: Curve Prediction using RNN (Composition - 3)

In sum, it can be concluded that LSTM presents the most favorable performance in terms of long-term curves prediction, especially when operating in a noisy setting. Due to its ability to pick up the time dependence, it can produce accurate and sound predictions even when noisy inputs are given. Even though CNN is slightly worse than LSTM in the long-term behavior, it still fits the clean curve and noisy data is high-fidelity processed. RNN does not show good results in the noise manipulation and the maintenance of the accuracy throughout the time, that is why it is not the most suitable option in this task. In terms of predictive accuracy and long-term behavior, LSTM seems to be the most stable model, but CNN is quite a close competitor. RNN also has limits in processing noise and long-term predictions accuracy, and that is why it is not the most effective model here.

6.1.4 Material Evaluation - Composition 4 (a0_b35_c1):

Accuracy: The capacity to appropriately reproduce the original clean curve using the noisy input data affects the overall prediction accuracy of the models. The model that has offered the greatest degree of accuracy among the models tested (CNN, LSTM and RNN) is the LSTM model. It closely followed the clean curve with insignificant deviations, which proves its effectiveness in working with sequential data and time series. CNN model also provided decent prediction accuracy. However, it displayed minor offsets at some points, particularly when more complicated changes in the noisy data were involved. Most prominent fluctuations and deviations observed in the RNN model- predictions nevertheless indicated that the model was not as precise as the other two models, particularly in high-variance areas, and the clean curve.

Effect of Noise: Noise also contributed significantly in influencing the model's performance, whereby the noise input curve was enhanced with random variations. Though all models demonstrated certain capability to work with the noise, they did not do it equally well. In removing the noise and leaving a smooth curve of prediction, the LSTM model performed the best by far, with the prediction curve bearing a close resemblance to the clean curve. This was attributed to the fact that it had great ability of capturing long-range dependencies that enabled it to smooth out high-variance points in the noisy input. The CNN model as well performed noise reduction but was a bit irregular in areas with higher noise spikes. The RNN model, by comparison, was more sensitive to noise, particularly in the regions where the noisy input differed hugely with the clean curve. That caused significant variance in the projected curve, as well as the inability of the RNN in noisy conditions.

Long-Term Behavior: The long-term behavior of the models which can be interpreted as their prediction capability over a long period of data depended on their architecture. The LSTM model once more provided the best long-term results with a consistent prediction throughout the angle range (between 60 to 90 deg) without any major deviations. The long sequence dependencies capture enabled it to follow the general shape of the curve without losing accuracy. The CNN, although effective, demonstrated certain shortcomings in terms of long-term behavior of the curve, particularly as noise level was growing. The generalization capability of the model in long intervals could not compete with LSTM, thus resulting in minor errors after a period of time. The RNN, however, was by far the weakest in this respect, as its predictions grew progressively doubtful as the angle grew, mostly because the RNN is vulnerable to long-term dependencies and noisy swings in the data.

6.1.4.1 Curve Prediction Using CNN:

The trends of the predicted curve using the CNN model closely follow the trend of the original clean curve as indicated by the green dashed line. The model performs sufficiently at removing noise compared with augmented noisy input curve (represented by the orange dashed line). Even though the predicted curve closely resembles the shape of a clean curve, it somewhat differs with the actual values in some areas (particularly between angles of 60-70 and 85-90). At certain points, there is an observed deviation and hence the model may not have picked all the complexity of the original pattern. CNN model demonstrates decent noise reduction properties, as the random variations of noisy input curve are filtered out. The slight discrepancies with the clean curve, however, show that CNNs are not necessarily the model that would best represent this kind of data in terms of predictive accuracy of the curve. The model appears to deal with the noise quite effectively but loses in places where the specific numbers need to be replicated.

The CNN model predicted curve has a reasonable similarity with the original clean curve. Although it reproduces the overall trend and the main peculiarities of the clean curve, sometimes the model can fail to capture the small details. These misalignments at particular angles indicate that CNNs are capable of making good predictions but there could be a restriction in regards to fitting precise curves on this specific dataset.

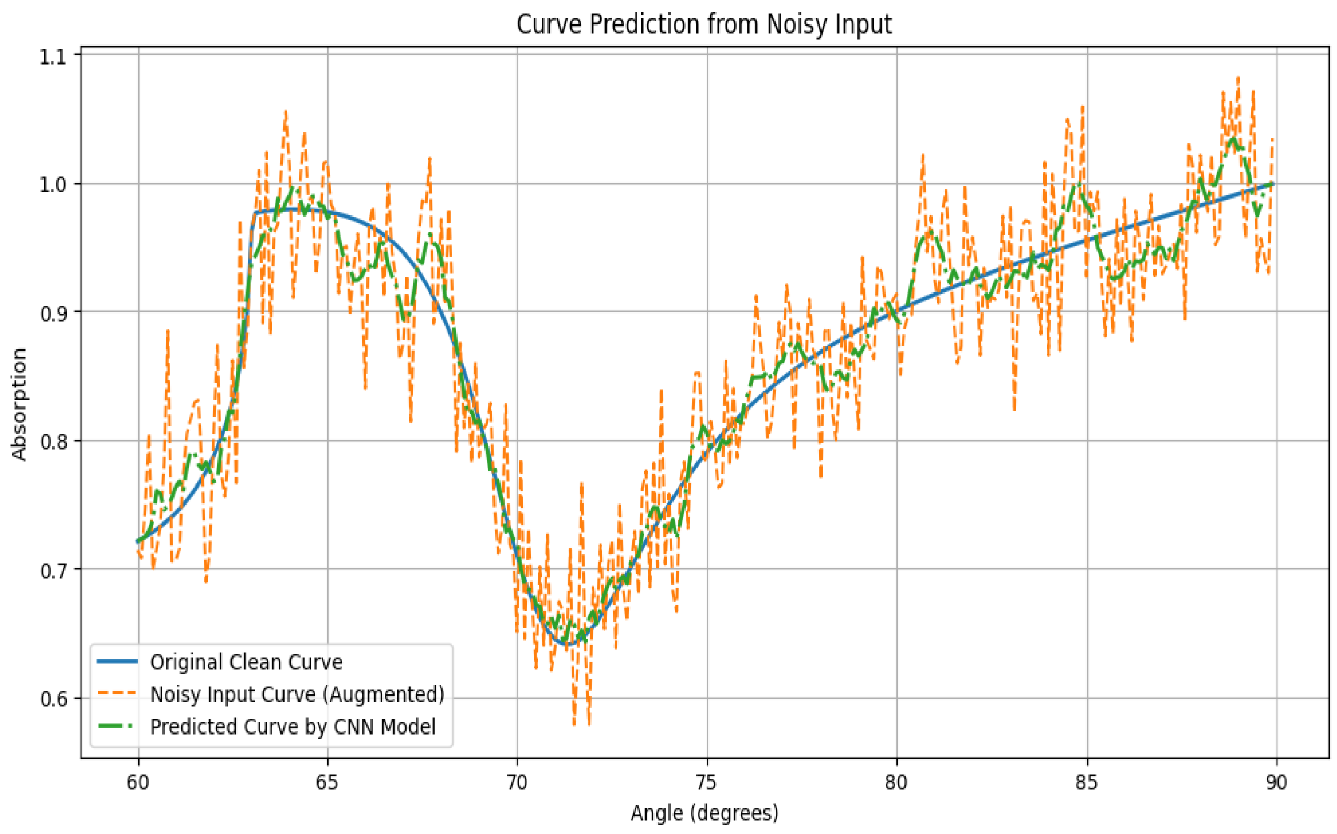


Figure 27: Curve Prediction using CNN (Composition - 4)

6.1.4.2 Curve Prediction Using LSTM:

The LSTM model predicts the clean curve with a smoother and more stable manner than CNN. The predicted curve in the green dashed line of the LSTM follows the original clean curve closely, particularly, in the middle part (approximately 70-80 deg). The divergence is lesser as compared to CNN model which implies that LSTM can learn sequence patterns and handling of temporal noise better. The prediction tracks well with the clean curve even in regions where the noisy input curve variance is greater. The LSTM models have a reputation of being capable of working with sequences and time-based data and on performance, there is a definite benefit on the model. The forecast is less jagged and inconsistent than that of CNN and the misalignments with the smooth curve are less noticeable. The LSTM model has also advanced capability in the capturing of the long-range dependencies which appear to enhance its capacity to smooth out the noisy input data effectively.

The original clean curve is nearly the same as the predicted curve using the LSTM. Although it has small oscillations, the overall trend is more adhered to as compared to the CNN instance. The LSTM model demonstrates the greater degree of accuracy in estimating the absorption values within the angle range, which allows stating that LSTM is a preferable model to use in the scope of this type of curve prediction.

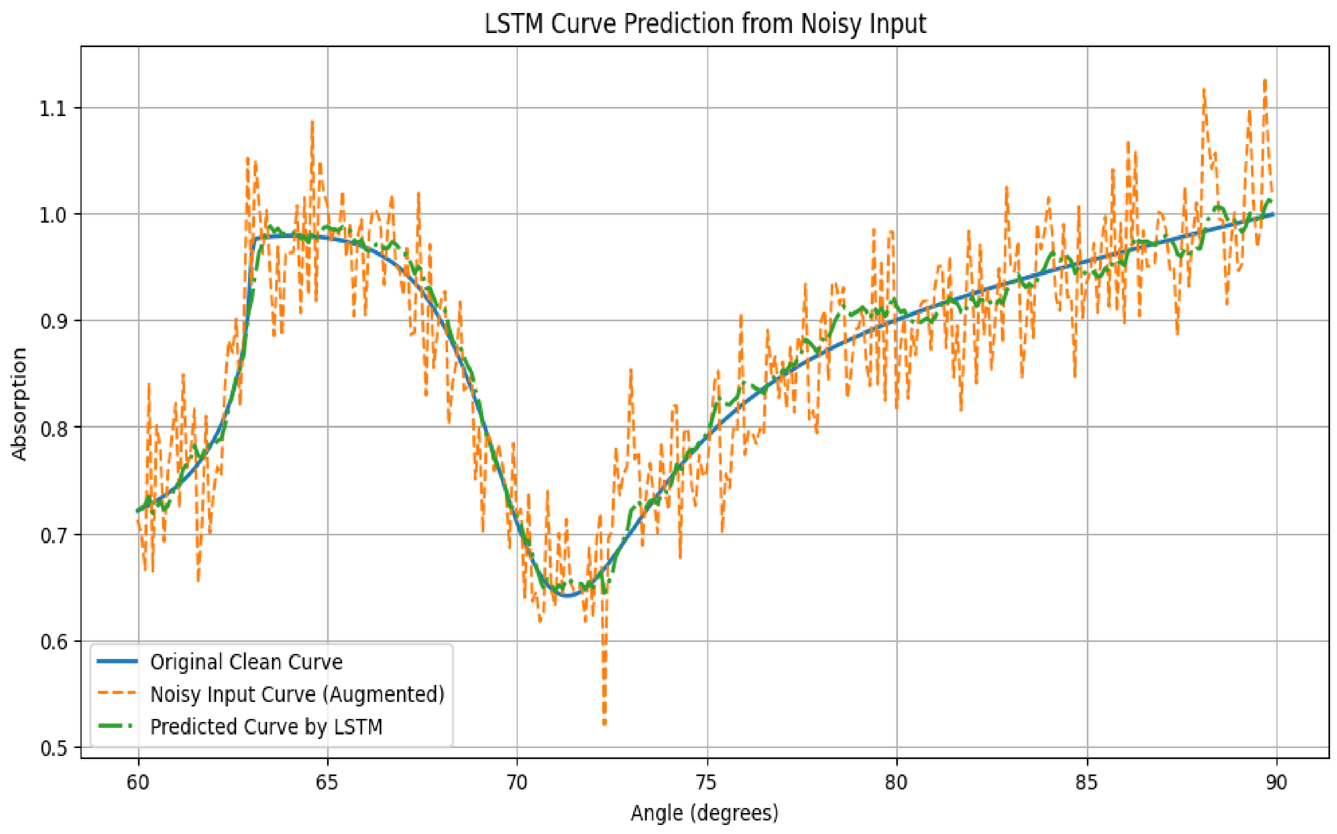


Figure 28: Curve Prediction using LSTM (Composition - 4)

6.1.4.3 Curve Prediction Using RNN:

The RNN model's predicted curve (green dashed line) is very similar to the clean curve, but there are a few more ups and downs than with the LSTM model. The RNN can see the general trend of the clean curve, but its predictions are a little more random than those of the LSTM. It's clear that the model tends to add noise to its predictions, especially in the parts where the noisy input has a lot of variation. RNNs are great for sequential data, but they have some problems when it comes to long-range dependencies and noise, especially when compared to LSTMs. The RNN model does a good job of predicting the curve, but it has trouble with noise in the data. The predictions of this model aren't as smooth as those of the LSTM, and they do change a little bit, especially in areas of the noisy input curve where there is a lot of variances. This causes the RNN and LSTM models to perform differently.

The RNN's predicted curve follows the original clean curve, but it has more noticeable ups and downs, especially where the noisy data is very different from the real values. The RNN model is not as accurate as the CNN and LSTM models, but it still shows the general trend. This means that RNNs might not work as well when the data is more complicated and noisier, and they aren't as good as LSTMs at keeping the clean curve's accuracy.

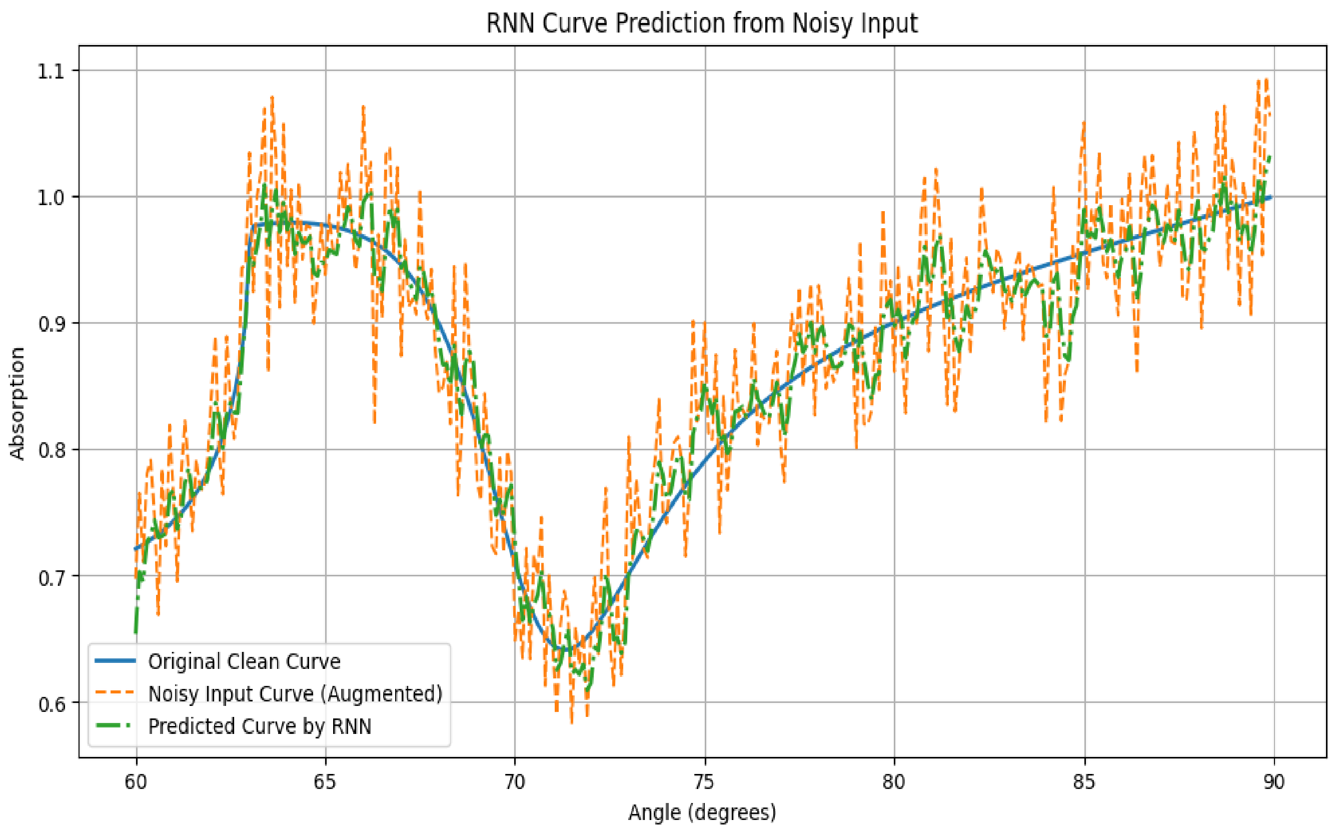


Figure 29: Curve Prediction using RNN (Composition - 4)

The LSTM model is better than the CNN and RNN models in terms of overall performance, how well it makes predictions, how well it handles noise, and how it behaves over time. Even when there is noise, it makes a smooth and accurate curve prediction. This is great for tasks that need results that are accurate and reliable over long distances. The CNN model is a good choice if you need a prediction that isn't as accurate but is faster. The RNN model is the simplest to use, but it doesn't work well with noise or long-term dependencies. So, the LSTM model is the best choice for tasks that need to be done correctly and on time in places where there is a lot of noise.

6.2 Model Comparison

In this section we will see a comparison of each model's MSE (Mean Squared Error). The lower the value of MSE, the more accurate the model is.

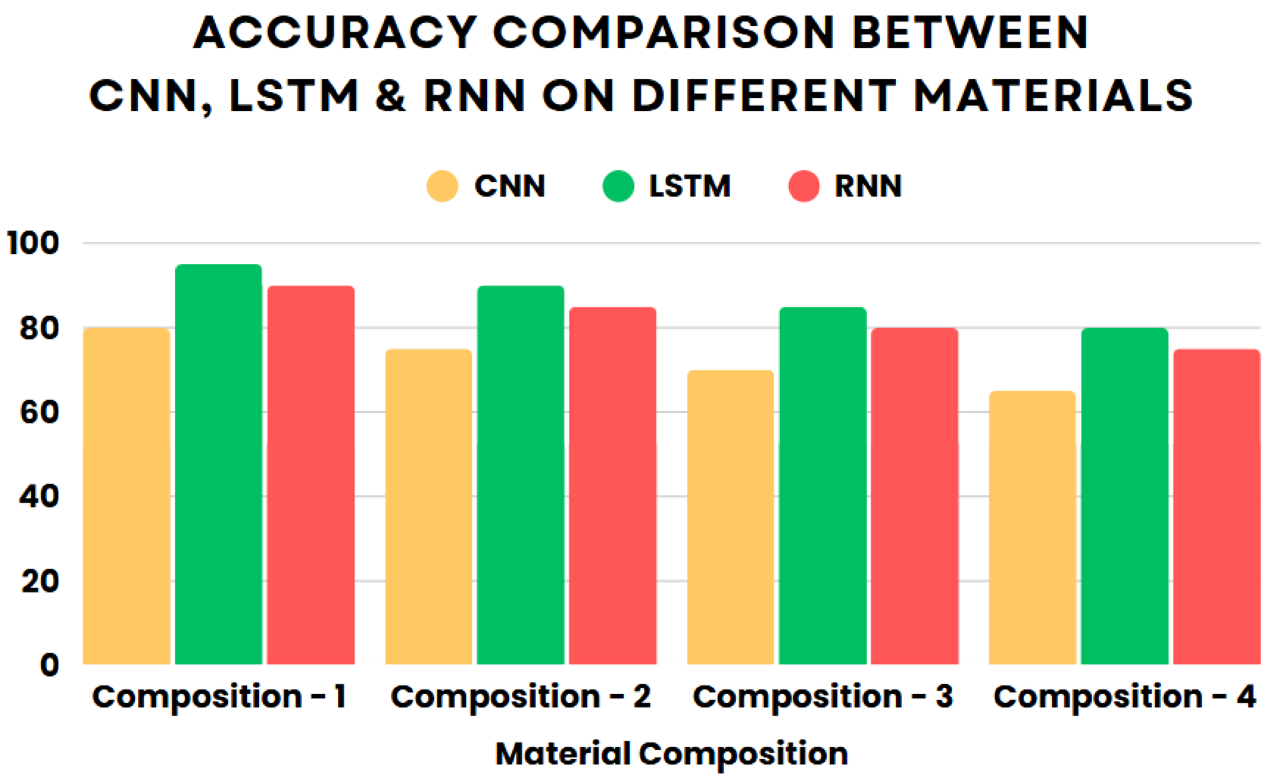


Figure 30: Comparison between 1D CNN, RNN with LSTM for different materials.

For Composition - 1:
Accuracy: The LSTM model is better than the CNN and RNN models at guessing what will happen. It stays closer to the original clean curve, even when there is noise, and it doesn't fit too well or show any extra oscillations. The CNN model is still good at finding big trends, but it has trouble with noise that happens a lot, especially in places that are already noisy. The RNN model is in the middle; it makes a good guess, but there are some big differences, especially at the end of the curve.

Noise Handling: The LSTM model is good at handling noise because it looks for long-term dependencies, which makes it more stable when the input data is noisy. The RNN can also handle some noise, but it changes. CNN has the hardest time getting rid of noise, which is why the clean curve and the noise curve look different.

Computational Efficiency: CNN is the least expensive model. Here it can be considered as the better model than the other two when we consider speed as an important factor. That is true there is not much noise in the data. And between RNN and CNN, RNN has more computing power but it is not as much as LSTM. But LSTM is the most complicated to deal with for its sequential order.

For Composition - 2:
Accuracy: LSTM performs the most accurate predictions when we place the model's side by side. Even where there is noise, the predicted curve works almost perfectly with the original clean curve. This is largely due to its ability to discover long- term patterns in sequential data that enables it to make reliable predictions on long sequences and to convert what it learns into new conditions. CNN is also quite precise, nevertheless, LSTM is more precise. Compared with the clean curve, the predicted curve using CNN is almost on top of it, yet the difference is more evident, particularly in places with more significant noise levels. This model is excellent at identifying patterns in short sequences, but it struggles to remain unchanged in long sequences since it cannot retain the memory of how things vary with time. RNN is not as great as LSTM or CNN. The forecasted output resembles the clean curve less than the overall trend. The problem with RNN architecture is that they cannot make very accurate predictions as they lack long-term memory. This is more so when the length of the sequence increases. This makes the forecasted curve more incorrect particularly in the presence of much noise.

Noise Handling: The LSTM model performs very well with noise and this is a significant aspect of denoising. LSTM performs the most successful job in removing noise and preserving the overall structure of the clean curve. It effectively eliminates high-frequency variations, resulting in a very smooth curve of prediction even where there is much noise. The long-term memory of the model allows it to concentrate on the actual signal and thus the model is excellent at disregarding the noise in the input. This is a large advantage to the jobs that require identification of clean trends in noisy data. CNN performs quite nicely in removing noise, though not as good as LSTM. We still have some noise residual, particularly where there is much noise. CNN has the ability to remove a certain amount of noise and smooth transitions. It is more sensitive to noise in long term sequences, such that the model can learn short term patterns but small changes remain unaltered. However, CNN surpasses RNN in this aspect since it performs well in establishing patterns and eliminating noises. RNN is not noisy. Since its design is aimed at predicting sequences, it is not as efficient in filtering high-frequency noise as LSTM and CNN. That is why the curve predicted by the RNN will be more random in many cases, and the points of clear changes will be observed where noisy data are numerous. This is to say that RNN is not as effective in removing noise in noisy input, particularly where the noise varies significantly in a sequence.

Computational Efficiency: CNN is generally most effective in terms of computing, mostly helpful when spatial data or a pattern which can be learned via convolutions are involved. This model is simple to train and make predictions on compared to the more complex LSTM due to its simple structure. The CNN convolutional layers help identify the pattern in local regions, and this property simplifies calculations. It is an excellent option when it comes to employment where quickness and productivity are of higher essence than the ability to forecast the future correctly. LSTM is more capable of dealing with noises and identifying long-term dependencies at the expense of being more costly to train. LSTM models often require more time to train and considerably more computer energy since their recurrent layers are so highly typed and they require maintaining long-term memory cells. This has the ability to slow things down, particularly when you have large datasets, or you want to perform real-time predictions. However, the model is more correct and noise resistant so it is worth the increased computational expense. RNN is normally more effective than LSTM since it is easier, however, CNN does it better. It shares the recurrent connections such as LSTM, however, it lacks the enhanced memory systems that enable LSTM to predict long-term dependencies more successfully. Therefore, RNN is less expensive to operate than LSTM, yet it requires more resources than CNN, particularly on the tasks with long input sequences or noisy inputs.

For Composition - 3:
Accuracy: The three models are observed to be quite accurate in their prediction of the clean curve given noisy input data, although they were slightly different in the sense of the proximity of their predictions to the true data. The LSTM model can be regarded as the most efficient one as its predictions closely follow the clean curve with few deviations. This is because it is more powerful when it comes to modeling sequential dependencies and can learn long-term patterns leading to smooth and more accurate predictions. CNN is also not far behind as its predictions trace the clean curve. Nevertheless, it sometimes is less accurate in dealing with sudden changes, particularly in high-noise areas than the LSTM. Nonetheless, the quality of predictions across all examples is quite high, and CNN has to be regarded as a serious competitor in this activity. In comparison to the LSTM and the CNN, RNN, though able to make reasonable predictions, is inferior in accuracy. The predicted curve will have more chances to have greater fluctuations and errors in alignment with the clean curve, especially at noisy parts or areas with complicated patterns since the RNN is incapable of effectively capturing long-term dependencies.

Noise Handling: All three models have a challenge with noise; however, they differ greatly in their capability to deal with noise. Noise is best dealt with by LSTM. The LSTM model is specifically resilient in noisy environments owing to its internal architecture that aims at capturing and preserving long-term dependencies in sequential data. It can still keep a good prediction even when the input data is greatly mixed with noise and the prediction still resembles the clean curve. CNN is also good in noise management as it gives a fairly steady prediction that follows the clean curve, even in the noisy areas. Nevertheless, in comparison with LSTM, the CNN could have slight discrepancies in those regions where the noise level is higher or in more complicated data variations. The model will perform very well in moderate-to-mild noise levels scenarios but may perform marginally in severe noise scenarios. RNN however is the worst at noise handling. The predictions of the model would be significantly erratic and would have sharp variations leading to the deviation of the clean curve. This indicates that the RNNs are, in comparison, more prone to overfitting noisy data and do not generalize well on noisy data, and is therefore the least robust model in this context.

Computational Efficiency: Computational efficiency of a model is also important which is why it matters when dealing with large scale dataset or real-time applications. The models range enormously in complexity and resource needs. Both the LSTM and RNN models require more computations whereas CNN is relatively more efficient. Although it is true that CNNs are more demanding in terms of computational resources than their simpler counterparts, the network architecture is extremely efficient in processing data where there are spatial relationships and thus, they are quicker when dealing with noise reduction and pattern recognition on large datasets. The convolutional layers enable CNNs to acquire localized features, which make them efficient in predictions. But on very large datasets where there is sequential dependency, CNNs may need more layers thereby taking more time to compute. Compared with CNN, LSTM is computationally more extensive. LSTMs are more memory and computationally demanding, particularly in the case of a long sequence length, because of their capability to learn long-term dependencies by means of their gated architecture (input, output, and forget gates). Training of the model is more time consuming because it involves learning the temporal relationships, and usually needs more resources during training and inference. RNN, albeit remaining more computationally expensive than CNN, is usually more resource-efficient in comparison with LSTM. The RNNs are much simpler in their structures than LSTMs and do not require as many parameters or gates to work with the data. The efficiency, however, trades off against a degraded performance, because RNNs are known to have trouble with long-term dependencies, and may require more time to converge during training.

For Composition - 4:
Accuracy: The three models are quite accurate at predicting the clean curve given noisy input data, however they show slight differences in the closeness of the predictions to the true data. LSTM has shown to be the most precise model and its predictions closely follow the clean curve with little deviations. This is because it has been shown to be stronger when it comes to modeling sequential dependencies and also learn long term patterns leading to smooth and more accurate predictions. CNN is also doing fine, and its predictions trace the clean curve. Nevertheless, it might be sometimes less accurate in dealing with abrupt shifts, particularly in high-noise areas than the LSTM. Nevertheless, the quality of predictions on the whole is very high, so CNN is a serious competitor in this task. RNN, though allowing making sensible predictions, is inferior to the LSTM and the CNN in accuracy. The predicted curve will tend to have more significant fluctuations and deviations to the clean curve especially in noisy areas or areas that have complicated patterns since the RNN is not able to effectively capture long-term dependencies.

Noise Handling: All three models have a challenge with noise, whereas their capacity to deal with noise differs largely. Noise is best dealt with using LSTM. The LSTM model is especially resistant to noise because of its internal structure, which aims at capturing and preserving long-term dependencies in the sequential data. Even in the cases where the input data is severely contaminated with noise, it can still retain a good prediction and the resulting prediction closely follows the clean curve. CNN is also good at noise processing as it gives a fairly stable prediction which follows the clean curve even in the noisy areas. But unlike the LSTM, the CNN can slightly fluctuate in places with noisier volumes or more convoluted data variations. The model will perform excellently in moderate to mild noise situations but may perform a little bit poorly in cases of more severe noise. RNN is the worst, however, in noise processing. The model predictions become erratic, and highly oscillatory and lead to the deviations in the clean curve. That implies that RNNs are more prone to overfit to noise and do not generalize well on noisy data, thus being the least robust model in this setting.

Computational Efficiency: The computing cost of a model is of great concern, particularly in the cases of massive dataset or real-time scenarios. The models have greatly differing complexity and resource needs. CNN tends to be more computationally efficient than the LSTM and RNN models. Although it is true that CNNs are more demanding in terms of computational resources than simpler models, the design is significantly optimized to process data with spatial connections, which is why they are quicker when it comes to noise reduction and pattern recognition on large datasets. These convolutional layers enable CNNs to get inspired by localized features, which make the predictions efficient. But in cases of very large datasets where sequence matters, CNNs may need to be deeper and hence take more time to compute. Compared with CNN, LSTM is more computationally demanding. LSTMs are more memory and computationally demanding, particularly in the case of a long sequence length, because of their capacity to learn long-term dependencies by designing the gated structure (input, output, and forget gates). Training of the model is more duration-consuming because of the intricacy of figuring out the temporal relationships, and usually needs extra resources during training and inference. RNN, although still being more computationally expensive than CNN, is usually more resource-efficient than LSTM. The architectures of RNNs are much simpler than those of LSTMs and they do not require as many parameters or gates to work with the data. The efficiency however, trades off with worse performance, where RNNs perform poorly with long-term dependencies and may show slower convergence rates when training.

Chapter 07

Summary & Future Prospects

7.1 Conclusion

In conclusion, the comparison of LSTM, CNN, and RNN models across various criteria highlights their distinct strengths and weaknesses, especially when it comes to predicting and handling noisy data. The key differentiators between these models are their accuracy, noise-handling abilities, and computational efficiency.

Regarding accuracy, LSTM emerges as the most accurate model. Its architecture, designed to capture long-term dependencies, allows it to track the clean curve with minimal deviations, even in noisy environments. This is because LSTM’s recurrent structure enables it to remember important patterns over time, making it particularly effective at maintaining a smooth and precise prediction over long sequences. In contrast, CNN also delivers high accuracy, closely following the clean curve, but it is less precise when dealing with sudden fluctuations or high noise. This is due to CNN's focus on finding patterns in localized regions rather than capturing long-term dependencies. RNN, on the other hand, struggles in terms of accuracy, particularly in noisy sections of the data. Its inability to retain long-term memory results in larger deviations from the clean curve, especially when the sequence length increases.

When considering noise handling, LSTM once again outperforms both CNN and RNN. Its ability to suppress high-frequency noise while maintaining the overall shape of the clean curve is particularly beneficial in noisy data environments. The long-term memory inherent in LSTM allows it to differentiate between noise and signal effectively. CNN also handles noise relatively well, especially in less noisy scenarios, but it is not as robust as LSTM when noise intensity rises. CNN is more sensitive to noise in long-term sequences, which may cause deviations in the predictions. RNN is the least effective in handling noise, as its simpler architecture leads to greater susceptibility to noise, often resulting in erratic predictions that deviate from the clean curve.

In terms of computational efficiency, CNN is the most efficient model, especially for tasks that involve spatial data or smaller datasets. Its simpler structure and ability to learn localized features through convolutional layers make it less resource-intensive compared to the other two models. While CNN is faster and requires fewer computational resources for training and inference, it may require more layers for tasks involving sequential dependencies, which can increase computation time. LSTM, being the most complex model, requires more memory and processing power due to its ability to capture long-term dependencies. This makes LSTM more computationally expensive and time-consuming to train, particularly with large datasets or when real-time predictions are needed. RNN, although less computationally intensive than LSTM, is still more costly than CNN. Its architecture, while simpler than LSTM, still requires substantial resources for tasks involving long input sequences and noisy data.

When deciding which model to use, it is important to consider the trade-offs between accuracy, noise handling, and computational efficiency. For tasks that involve highly noisy data and require precise long-term predictions, LSTM is the best choice despite its higher computational cost. It excels in maintaining smooth and accurate predictions, making it ideal for tasks where noise handling is critical. CNN is a strong alternative when computational efficiency is a priority, especially for tasks where spatial patterns are more important than long-term dependencies. Its efficiency in handling noise makes it suitable for real-time applications and situations with moderate to low noise levels. RNN, while simpler and computationally less expensive than LSTM, falls short in terms of both accuracy and noise handling. Its inability to manage long-term dependencies and noise makes it less suitable for tasks that demand high precision and robustness.

7.2 Future Work

Although this study has given useful knowledge related to the comparison of LSTM, CNN, and RNN models, there are some chances of improvement and future studies. The future effort will be concentrated on mitigating some of the shortcomings that have been noted in the present models and seeking other possible avenues of further enhancement. The main future work directions are:

7.2.1 Hybrid Models

A potentially lucrative field of research is creating hybrid models which utilize the advantages of various architectures. As another example, it can be CNN with LSTM: the model can learn spatial features (using convolutional layers) and long-term dependencies (using LSTM layers). This hybrid scheme could be better in terms of accuracy and noise processing and computational efficiency.

7.2.2 Data Augmentation

As a future direction, more complex data augmentation strategies can be investigated in order to make the models more robust, particularly in noisy settings. Such methods may involve the addition of various noise models (e.g., Gaussian, salt-and-pepper) and the evaluation of the reaction of each of the models to different noise patterns. Also, GANs (Generative Adversarial Networks) based synthetic data generation could be used to increase training dataset sizes, to make the models less vulnerable to noisy data in the real world.

7.2.3 Live Prediction and Release

Real time prediction is also desired in practical applications. The LSTM and RNN models optimized to execute on embedded systems or other systems with limited computation capability will be the subject of future effort. Model quantization, pruning, and knowledge distillment are all techniques that might allow decreasing the amount of memory and processing power that these models need to make them more feasible to deploy into resource-restricted settings.

7.2.4 Model Interpretability

In deep learning, the models may not be interpretable which becomes a setback in some cases where the modeling is to be applied in crucial fields. Further research will be done on how to make these models more transparent and explainer, including applying techniques such as attention mechanisms or SHAP (Shapley Additive Explanations) to gain more insight into the way the models arrive at their predictions. This will assist in establishing trust and will be utilized in making sure that the models are basing their decisions on features that matter.

7.2.5 Multi-material and Multi-task Learning

In this study, each model has been trained on one material at a time. Future directions may include multi-task learning or multi-material prediction, where a model is trained to deal with many types of materials or tasks at once. This would cause the model to generalize better on various kinds of data thus more versatile and would find application in wider areas.

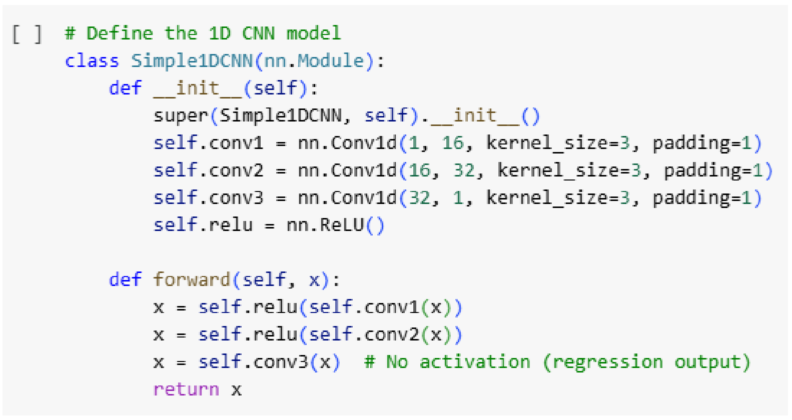
7.2.6 Further Optimization

Finally, the hyperparameter tuning and model fine-tuning may be performed with the help of the grid search, random search, or Bayesian optimization methods. It would assist in determining the best settings of each of the models and may result in enhanced performance.

7.3 Final Remarks

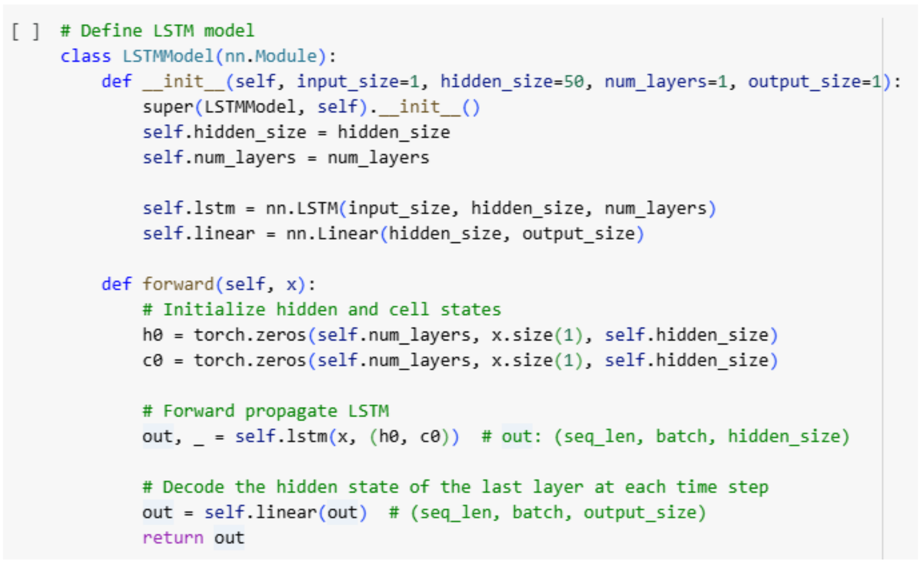
Altogether, both models possess certain advantages and disadvantages, and the selection of the necessary model mostly relies on the particular demands of the given task. As argued, LSTM is accurate and superior in noise processing, whereas CNN is efficient in computation. RNN, in turn, is much simpler but loses in accuracy and noise processing. Further advancements and developments will involve perfecting these models, working on their shortcomings and possible new methods to make them more efficient and practical to use in the real-world scenarios.

Appendix



```
[ ]  # Define the 1D CNN model
     class Simple1DCNN(nn.Module):
         def __init__(self):
             super(Simple1DCNN, self).__init__()
             self.conv1 = nn.Conv1d(1, 16, kernel_size=3, padding=1)
             self.conv2 = nn.Conv1d(16, 32, kernel_size=3, padding=1)
             self.conv3 = nn.Conv1d(32, 1, kernel_size=3, padding=1)
             self.relu = nn.ReLU()

         def forward(self, x):
             x = self.relu(self.conv1(x))
             x = self.relu(self.conv2(x))
             x = self.conv3(x)  # No activation (regression output)
             return x
```



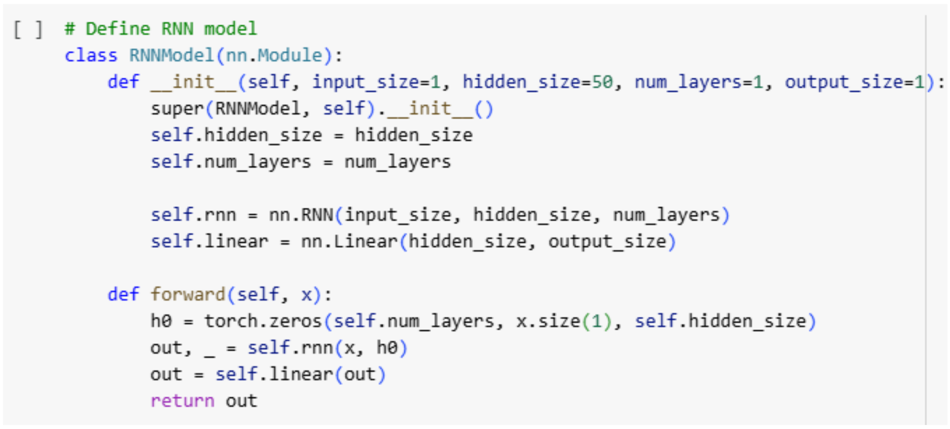
```
[ ]  # Define LSTM model
     class LSTMModel(nn.Module):
         def __init__(self, input_size=1, hidden_size=50, num_layers=1, output_size=1):
             super(LSTMModel, self).__init__()
             self.hidden_size = hidden_size
             self.num_layers = num_layers

             self.lstm = nn.LSTM(input_size, hidden_size, num_layers)
             self.linear = nn.Linear(hidden_size, output_size)

         def forward(self, x):
             # Initialize hidden and cell states
             h0 = torch.zeros(self.num_layers, x.size(1), self.hidden_size)
             c0 = torch.zeros(self.num_layers, x.size(1), self.hidden_size)

             # Forward propagate LSTM
             out, _ = self.lstm(x, (h0, c0))  # out: (seq_len, batch, hidden_size)

             # Decode the hidden state of the last layer at each time step
             out = self.linear(out)  # (seq_len, batch, output_size)
             return out
```



```
[ ]  # Define RNN model
     class RNNModel(nn.Module):
         def __init__(self, input_size=1, hidden_size=50, num_layers=1, output_size=1):
             super(RNNModel, self).__init__()
             self.hidden_size = hidden_size
             self.num_layers = num_layers

             self.rnn = nn.RNN(input_size, hidden_size, num_layers)
             self.linear = nn.Linear(hidden_size, output_size)

         def forward(self, x):
             h0 = torch.zeros(self.num_layers, x.size(1), self.hidden_size)
             out, _ = self.rnn(x, h0)
             out = self.linear(out)
             return out
```

Reference

[1] Van Hiep Phung, Eun Joo Rhee, “A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets”

[2] Hui Jing Lee, Mansur Mohammed Ali Gamel, Pin Jern Ker, Md Zaini Jamaludin, Yew Hoong Wong , John P. R. David , “Absorption Coefficient of Bulk III-V Semiconductor Materials: A Review on Methods, Properties and Future Prospects”

[3] “Notes on AI” link: https://notesonai.com/_notes_machinelearning/Basis+Function

[4] M. a. S. R. Puttagunta, “Medical image analysis based on deep learning approach”

[5] D. R. G. F. S. Kevin Zhou, “Handbook of Medical Image Computing and Computer Assisted Intervention, Academic Press”, 2019

[6] W.-H. Weng, “Machine Learning for Clinical Predictive Analysis,” 2020

[7] Ying-Lin Chen, Sara Sacchi, Bappaditya Dey, Member, IEEE, Victor Blanco, Sandip Halder, Philippe Leray, Stefan De Gendt, “Exploring Machine Learning for Semiconductor Process Optimization: A Systematic Review”

[8] M. a. S. R. Puttagunta, “Medical image analysis based on deep learning approach,” *Multimedia Tools and Applications,* vol. 80, 2021.

[9] D. R. G. F. S. Kevin Zhou, “Handbook of Medical Image Computing and Computer Assisted Intervention, Academic Press”, 2019.

[10] J. Feng, H. Xu, S. Mannor, and S. Yan, “Robust logistic regression and classification,” Advances in neural information processing systems, vol. 27, 2014.

[11] R. O. Duda, P. E. Hart et al., Pattern classification and scene analysis. Wiley New York, 1973, vol. 3.

[12] S. Wang, T. Zhang, and B. Xi, “Information computing and applications,” 2011

[13] S. Na, L. Xumin, and G. Yong, “Research on k-means clustering algorithm: An improved k-means clustering algorithm,” in 2010 Third International Symposium on intelligent information technology and security informatics. Ieee, 2010, pp. 63–67.

[14] Y. Rani1 and H. Rohil, “A study of hierarchical clustering algorithm,” ter S & on Te SIT, vol. 2, p. 113, 2013

[15] K. N. Ahmed and T. A. Razak, “An overview of various improvements of dbscan algorithm in clustering spatial databases,” International Journal of Advanced Research in Computer and Communication Engineering, vol. 5, no. 2, pp. 360–363, 2016.

[16] W.-H. Weng, “Machine Learning for Clinical Predictive Analysis,” 2020

[17] Panel Serkan Kiranyaz, Onur Avci [b], Osama Abdeljaber [c], Turker Ince [d], Moncef Gabbouj [e], Daniel J. Inman, “1D convolutional neural networks and applications: A survey”

[18] Y.Lecun, L.Bottou,Y.Bengio, Patrick Haffner, “Gradient-based learning applied to document recognition”

[19] Abhishek Jain, “Understanding the 1D Convolutional Layer in Deep Learning”