राष्ट्रीय प्रौद्योगिकी संस्थान वारंगल National Institute of Technology Warangal

PROJECT REPORT

ON

PREDICTION OF GLASS TRANSITION TEMPERATURE FROM CHEMICAL STRUCTURE

SESSION 2018-2022

UNDER THE GUIDANCE OF

Prof. Anand Kishore Kola

SUBMITTED BY: -

G.Sreesai Vaishnavi
Grandhe Sreesai Vaishnavi (186119)

Lokesh
Mareddy Sri Lokesh reddy (186229)

Soumyajit Das
Soumyajit Das (186246)

R. Ravindhar
Renu Ravindhar (186143)

CERTIFICATE

This is to certify that the project report entitled "PREDICTION OF GLASS TRANSITION TEMPERATURE FROM CHEMICAL STRUCTURE" is a bonafide record of work carried out by "Grandhe Sreesai Vaishnavi (186119), Mareddy Sri Lokesh reddy (186229), Soumyajit Das (186246), Renu Ravindhar (186143) " submitted to the **"Department of Chemical Engineering"**, in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in "Chemical Engineering" at National Institute of Technology, Warangal during the academic year 2021-2022.

Dr. Surananai Srinath
Head of the Department
Department of Chemical Engineering
NIT Warangal

Dr. Anand Kishore Kola
Professor
Department of Chemical Engineering
NIT Warangal

Dr. ANAND KISHORE KOLA
M.Tech, Ph.D, AMIIChE,MBA,M.Sc,MA,BCJ
PROFESSOR
Department of Chemical Engineering
NATIONAL INSTITUTE OF TECHNOLOGY
Warangal-506 004. TS, India.

ACKNOWLEDGEMENT

Firstly, we would like to express our deep and sincere gratitude to our guide, Prof. ANAND KISHORE KOLA, Department of Chemical Engineering, National Institute of Technology, Warangal, for being our supervisor for this project. Thank you, sir for your support, guidance and suggestions during this project. We are very grateful to have the chance to learn from you to be rigorous, serious and reliable for research.

We are grateful to Dr. SURANANAI SRINATH, Head of Department of Chemical Engineering, National Institute of Technology, Warangal, for his moral support to carry out this project. Our deep gratitude to all the professors and scholars who guided, inspired and helped us in the successful completion of the final year project.

We are very thankful to the Project Evaluation Committee, for their strenuous efforts to evaluate our projects.

Last but not the least, our heartfelt gratitude is extended to our parents and friends for their ongoing understanding and unconditional support throughout the semester for this project.

G.Sreesai Vaishnavi

Grandhe Sreesai Vaishnavi (186119)

Mareddy Sri Lokesh reddy (186229)

Soumyajit Das

Soumyajit Das (186246)

R.Ravindhar

Renu Ravindhar (186143)

Acknowledgement

First of all, we would like to express our deep and sincere gratitude to our project guide, Dr Anand Kishore Kola, for being our supervisor for this minor research project. Thank you, sir, for your guidance and support which helped us understand about a research project and for completion of our project successfully. We are very grateful to have the chance to learn from you to be rigorous, serious and reliable for research.

We would also like to give our special thanks to Dr Manohar Kakunuri. We are very grateful for your general support and cooperation.

Deep appreciation to the Dept. of Chemical Engineering, NITW for providing us with this opportunity in this semester.

Last but not the least, our heartfelt gratitude is extended to our parents and friends for their ongoing understanding and unconditional support throughout the semester for this project.

ABSTRACT

Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks thus form a subset of machine learning and are at the heart of deep learning algorithms. In this study we will design a neural network and train it to predict the glass transition temperature of polymers based only on their chemical structure. The possibility of predicting the properties of polymers not even synthesized will save time and resources for industrial development as well as accelerate the scientific understanding of structure-properties relationships in polymer science.

Contents

List of Figures

List of Tables

Introduction

Glasses are non-equilibrium, non-crystalline materials that spontaneously relax to the supercooled liquid state. Unlike crystals, glasses do not need to satisfy rigid stoichiometry rules and can be thought of as continuous solutions of chemical elements. There is a huge number of possible compositions for forming glassy materials. Indeed, 80 chemical elements combined in discrete quantities of 1 mol% would produce 10^52 possible glass compositions. Nevertheless, the number of inorganic glasses reported is only around 10^6.

All polymers exhibit glass transition temperatures. The glass transition temperature (Tg) is defined as the temperature at or above which the molecular structure exhibits macromolecular mobility. Typically, this is when fifty carbons along the molecular chain can move in concert. More practically, it is defined as the temperature range where the molecular structure is transformed from being a brittle solid to being a ductile or rubbery solid. Thermoplastic polymers are generally of two morphological types. Amorphous polymers, such as PVC, ABS, and polycarbonate, are characterized as having no crystalline structure or crystalline order. Amorphous thermoplastic polymers and essentially all thermosetting polymers have only one thermodynamic transition, the glass transition. Thermoplastic polymers simply get softer and softer as the temperature is raised above Tg. Crystalline polymers, on the other hand, have ordered molecular structure above Tg. Crystalline levels vary from about 20% for polyethylene terephthalate, to 70% for polypropylene, to as high as 98% for polytetrafluoroethylene (PTFE) fluoropolymer. The molecular structure of a crystalline polymer is for the most part, dictated by its crystalline structure or morphology. As an example, polyethylene has a glass transition temperature of about -100~ and a melting temperature or Tm of about 135~ The crystalline structure of polyethylene allows parts to retain their shapes at boiling water temperatures or more than 200~ above its Tg.

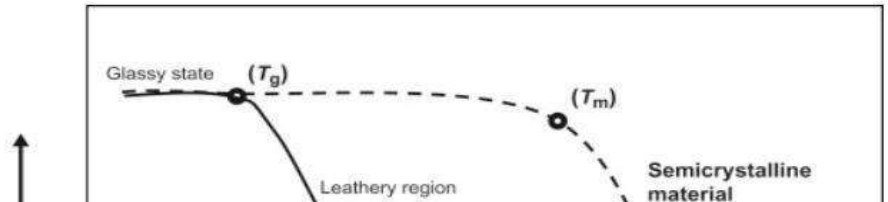


Fig.1. Modulus Vs Temperature graph

Literature Review

Extensive research has been conducted on polymer properties and utilities. Nuha Salih Mustafa et al consider thermal properties to be the primary driver when it comes to polymer characterization. Changes in the compositional and structural parameters can be linked to many performance parameters. For semi crystalline polymers it is an important method to measure crystallinity.

Advancements in neural machinery have led to a wide range of algorithmic solutions for molecular property prediction. Further research has also been conducted to study the main frameworks generally used: networks applied to computed molecular fingerprints or expert-crafted descriptors and graph convolutional neural networks that construct a learned molecular representation by operating on the graph structure of the molecule. Wu et al show CNN models typically outperform fingerprint-based models, while experiments reported in Mayr et al. report the opposite.

Liu and Cao proposed a model to correlate the Tg of different polymers with four physical quantities (polarizability, orbital energy, thermal energy and total entropy) obtained from density functional theory (DFT) calculations. Cassar et al. have recently published a different approach where they use an ANN to predict the Tg of oxide glasses based on their chemical composition (i.e. the relative amount of each atom in the glass).

Dataset

Sourcing:

Data used in this project has been sourced from Open Access Database from CROW polymer property database in the form of a Comma Separated File(csv).

About:

Our dataset for this study comprises 351 polymers along with their SMILES codes, molecular names as input attributes and glass transition temperatures as the output variable. Subsets of 300 polymers and their Tg values will be used for training validating the dataset, whereas the rest 51 unseen polymers will be used to test the results for models developed. We removed the empty columns and rows. Molecular structures are encoded by a string of text known as the SMILES notation which is an acronym for Simplified Molecular-Input Line-Entry System. Then we use the rdkit Python library to decode the SMILES data.

	Nomenclature Name	Molecular Structure	Tg
0	Poly(4-biphenyl acrylate)	C=CC(=O)Oc2ccc(c1ccccc1)cc2	383.0
1	Poly(butyl acrylate)	CCCCOC(=O)C=C	219.0
2	Poly(sec-butyl acrylate)	CC(OC(=O)C=C)CC	250.0
3	Poly(2-tertbutylphenyl acrylate)	C=CC(=O)Oc1ccccc1C(C)(C)C	345.0
4	Poly(4-tertbutylphenyl acrylate)	C=CC(=O)Oc1ccc(C(C)(C)C)cc1	344.0

Fig. 2. Top 5 rows of the dataset.

SMILES (Simplified Molecular Input Line Entry System) is a chemical notation system designed for modern chemical information processing. Based on principles of molecular graph theory, SMILES allows rigorous structure specification by use of a very small and natural grammar. The SMILES notation system is also well suited for high-speed machine processing. The resulting ease of usage by the chemist and machine compatibility allow many highly efficient chemical computer applications to be designed including generation of a unique notation, constant-speed (zeroth order) database retrieval, flexible substructure searching, and property prediction models. SMILES is simple to write because rules and hierarchical procedures, which are inherently difficult for the chemist, are relegated to computer algorithms.

The SMILES molecular structure can be visualized using chem module in rdkit Package. A package is a collection of Python modules: while a module is a single Python file, a package is a directory of Python modules containing an additional_init_.py file, to distinguish a package from a directory that just happens to contain a bunch of Python scripts. RDkit is an open-source toolkit for cheminformatics. It handles both 2D and 3D molecular operations and performs descriptor generation for machine learning.

```
print(df.Molecular Structure[0])
Chem.MolFromSmiles(df.Molecular Structure[0])
```

```
C=CC(=O)Oc2ccc(c1ccccc1)cc2
```

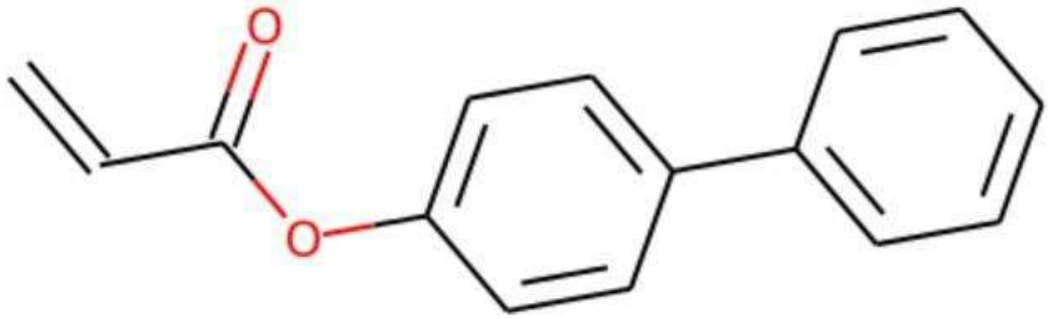


Fig. 3. SMILES Structure and Original Molecule of Poly(4- biphenyl acrylate)

Exploratory Data Analysis

This part refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

Using Pandas library in python we classified the dataset into eight different classes of polymers - acrylates, styrenes, amides, alkenes, ether, amides, carbonates, and others.

```
Acrylates      149
Others         101
Styrenes        69
Amides          15
Ether            9
Carbonates       8
Name: Polymer_Class, dtype: int64
```

Fig. 4. Classification of polymers

- Pie plot shows the exact composition of the dataset, with acrylates and styrenes being the highest contributors.
- Histogram represents the distribution of glass transition temperature values. 210-240 , 325-375 are the ranges that have large counts.
- The box plot is used to show the scatter plot of the underlying Tg distributions for each polymer class. From Fig. 4. We can infer that styrenes tend to have higher Tg whereas acrylates have a fairly mixed distribution.

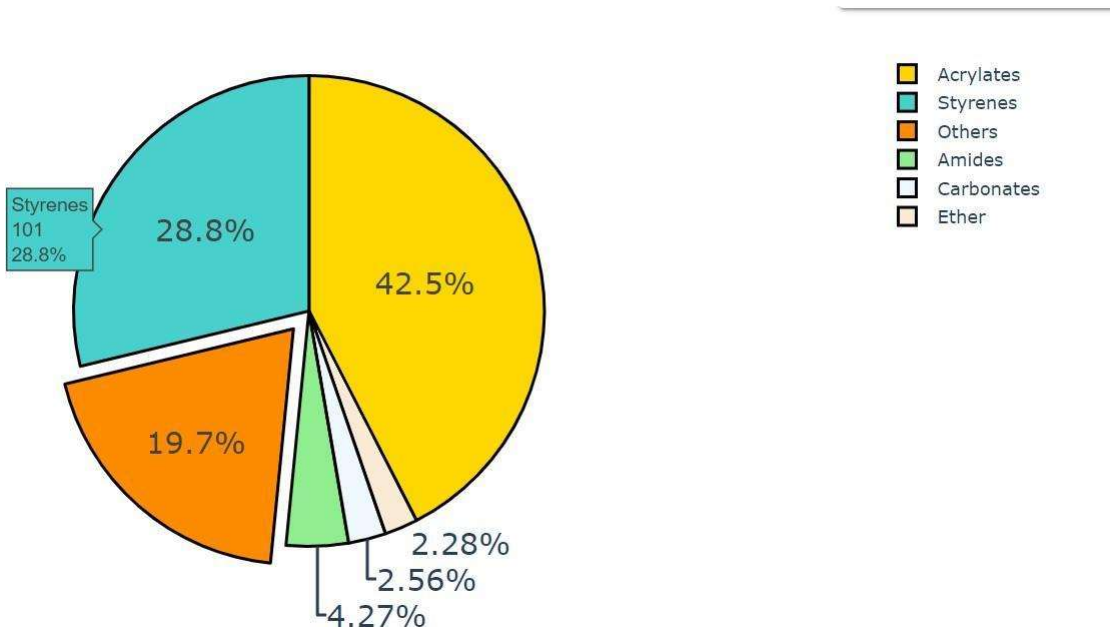


Fig.5. Pie plot

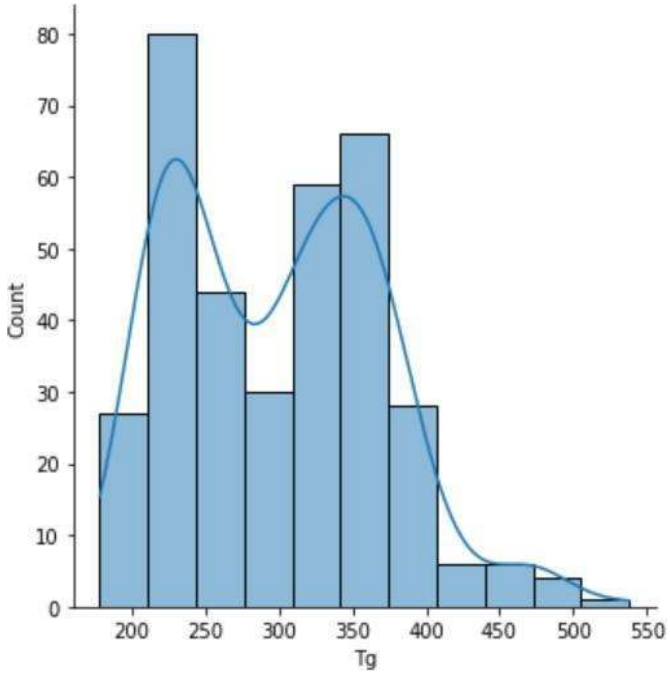


Fig. 6. Distribution of glass transition temperature

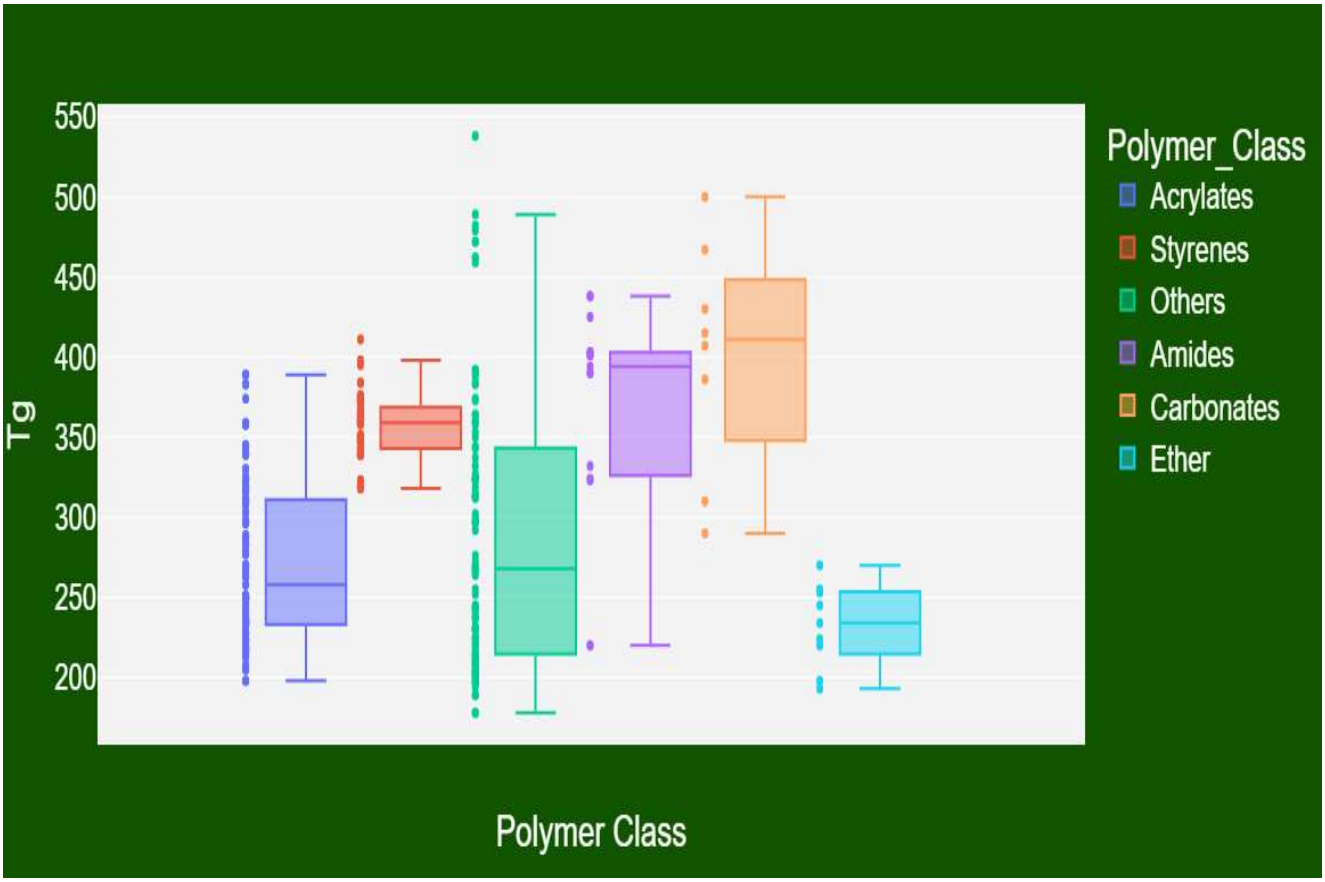


Fig. 7. Box plot

Neural Networks

Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are composed of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Each individual node is its own linear regression model, composed of input data, weights, a bias (or threshold), and an output.
$\sum wixi + bias = w1x1 + w2x2 + w3x3 + bias$ And,
output = $f(x) = 1$ if $\sum w1x1 + b >= 0$; 0 if $\sum w1x1 + b < 0$

Weights are assigned once an input layer has been defined. These weights are used to determine the importance of each variable, with larger ones contributing more to the output than smaller ones. After that, all of the inputs are multiplied by their respective weights and then added together. The input is then run through an activation function to determine the output. If the output reaches a certain threshold, the node "fires" (or activates), sending data to the network's next tier. The output of one node becomes the input of the following node as a result of this. This neural network is regarded as a feedforward network since data is passed from one layer to the next.

The ultimate goal is to reduce our cost function in order to ensure that each given observation is correctly suited. The model employs the cost function and reinforcement learning to approach the point of convergence, or the local minimum, as it adjusts its weights and bias. Gradient descent is the method by which the algorithm modifies its weights, allowing the model to discover the best

path to minimize mistakes (or minimize the cost function). The model's parameters adjust with each training case, gradually converging at the minimum, as shown below[9]

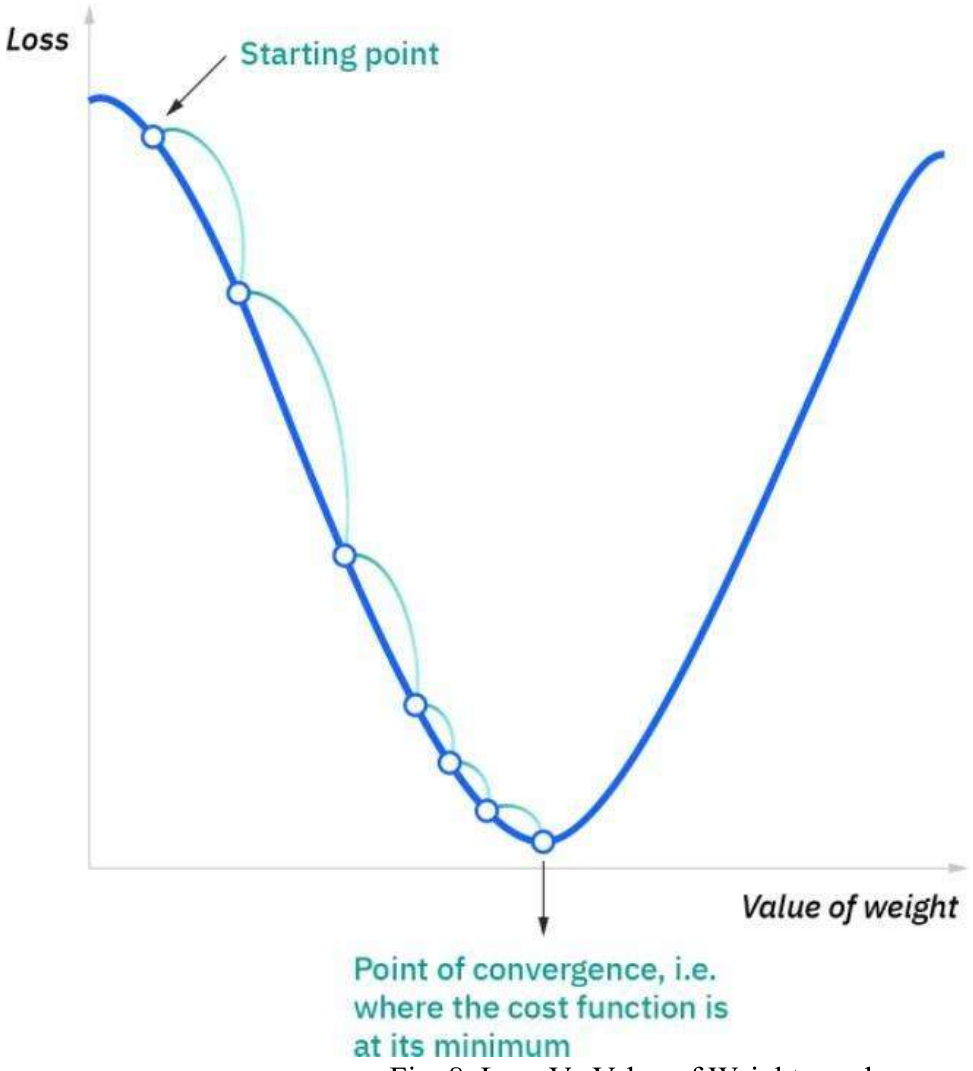


Fig. 8. Loss Vs Value of Weight graph

The learning rate is a hyper-parameter that governs how our neural network's weights react to the loss gradient. It specifies the rate at which the neural network updates the concepts it has learned. An ideal learning rate is one that is low enough for the network to converge on something useful while still being high enough for it to be trained in a reasonable length of time. Due to the smaller changes made to the weights in each update, smaller learning rates require more training epochs (more time to train), whereas bigger learning rates result in quicker changes and require fewer

training epochs. Larger learning rates, on the other hand, frequently result in a sub-optimal final set of weights.

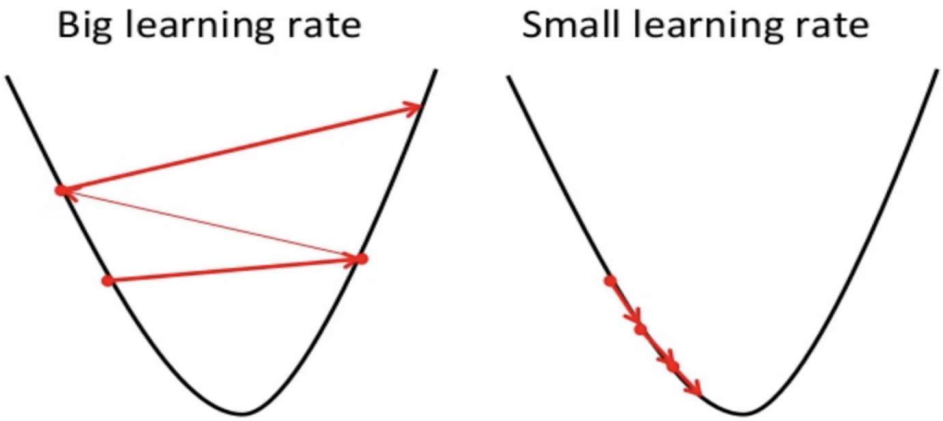


Fig. 9. Difference between Big learning rate and Small learning rate

An epoch is a complete transit of the training data through the algorithm in machine learning. The algorithm's epoch number is a significant hyperparameter. It defines the number of epochs or full passes through the algorithm's training or learning phase for the entire training dataset. With each epoch, the dataset's internal model parameters are changed. It can also be thought of as an epoch-numbered for-loop, with each loop path traversing the whole training dataset.

Convolutional Neural Network (CNN)

One of the most common ANNs is the convolutional neural network. In the disciplines of image and video recognition, it is commonly employed. It is founded on the mathematical concept of convolution. It's almost identical to a multi-layer perceptron, with the exception that it has a sequence of convolution and pooling layers before the fully connected hidden neuron layer. It has three important layers.

The basic construction piece is the convolution layer, which performs computational tasks using the convolution function. The pooling layer is located next to the convolution layer and is used to minimize the size of inputs by removing unnecessary data, allowing computation to be conducted more quickly. Fully connected layer is situated adjacent to a sequence of convolution and pooling layers, and it classifies input into several categories.

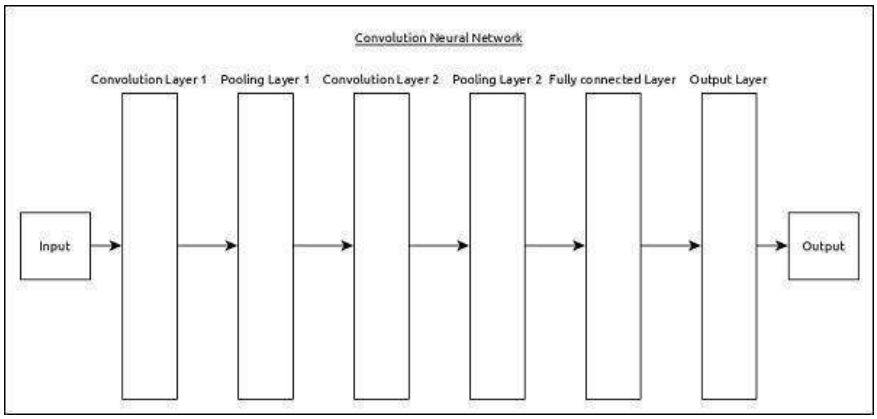


Fig. 10. Representation of a simple CNN

Here,

- 2 series of Convolution and pooling layer is used and it receives and process the input (example: image).
- A single fully connected layer is used and it is used to output the data (example: classification of image)

Artificial Neural Network (ANN)

One of the most well-known advantages of an ANN is that it can learn from watching data sets. In this technique, ANN is used to approximate random functions. While defining computing functions or distributions, these tools assist in estimating the most cost-effective and optimum techniques for arriving at solutions.

To arrive at solutions, ANN uses data samples rather than whole data sets, which saves time and money. ANNs are relatively simple mathematical models that can be used to improve existing data analysis technology. Three or more layers are interconnected in an artificial neural network. The input neurons make up the first layer. These neurons send data to the deeper layers, which deliver the final output data to the final output layer.

All of the inner layers are concealed and are made up of units that use a series of transformations to modify the information received from layer to layer. Each layer serves as both an input and an output layer, allowing the ANN to comprehend more complicated things. The neural layer is the collective name for these inner layers.

The neural layer's units strive to learn about the data collected by weighing it according to the ANN's internal framework. These principles enable units to provide a changed result, which is subsequently sent to the following layer as an output. Backpropagation, a mechanism by which the ANN can alter its output results by taking errors into account, is used in another set of learning rules. Each time the output is tagged as an error during the supervised training phase, the information is relayed backward using backpropagation. Each weight is adjusted in accordance to how much they contributed to the inaccuracy. As a result, the error is utilized to reweight the ANN's unit connections to account for the discrepancy between the desired and actual outcomes. The ANN will eventually "learn" how to reduce the likelihood of errors and undesirable

outcomes. When training an artificial neural network, you must choose from a set of authorized models and related techniques.

One of the most well-known advantages of an ANN is that it can learn from watching data sets. In this technique, ANN is used to approximate random functions. While defining computing functions or distributions, these tools assist in estimating the most cost-effective and optimum techniques for arriving at solutions.

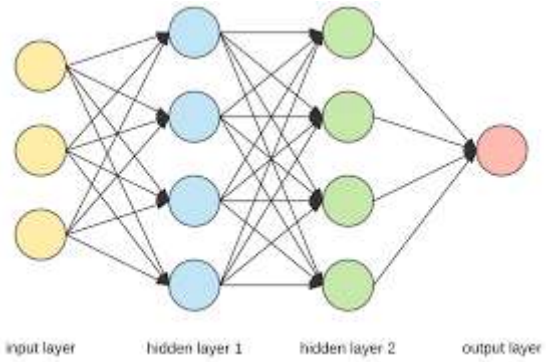


Fig. 11. Artificial Neural Network

Keras

Keras is a python-based deep learning framework that is open source. Francois Chollet, a Google artificial intelligence researcher, came up with the idea. Keras is a machine learning framework built on top of open source libraries such as TensorFlow, Theano, and Cognitive Toolkit (CNTK). Theano is a Python library for doing quick numerical computations. The most well-known symbolic math toolkit for constructing neural networks and deep learning models is TensorFlow.

TensorFlow is extremely adaptable, and its main advantage is distributed computing. Microsoft developed the CNTK deep learning framework. It makes use of libraries like Python, C#, and C++, as well as standalone machine learning toolkits. Theano and TensorFlow are great libraries for building neural networks, but they are challenging to grasp.

Keras is based on a simple framework that makes it simple to build deep learning models using TensorFlow or Theano. Keras is a deep learning framework that allows you to quickly define models. Keras, on the other hand, is an excellent choice for deep learning applications. Keras makes high-level neural network API easier and more performant by utilising various optimization techniques. It has the following capabilities:

- API that is consistent, simple, and extensible.
- Minimal structure - result can be achieved easily without any frills. It works with a variety of platforms and backends.
- It's an easy-to-use framework that works on both the CPU and the GPU. Computational scalability is really high.

Keras APIs

In Python, Keras and TensorFlow 2.0 provide you with three methods or Application programming interfaces (APIs) to implement your own neural network architectures: Sequential, Functional and Model API.

Sequential- The Sequential model API is a way of creating deep learning models where an instance of the Sequential class is created and model layers are created and added to it.

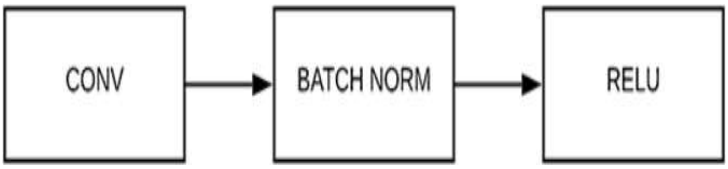


Fig. 12. Sequential API

Functional- The Keras functional API provides a more flexible way for defining models.
It specifically allows you to define multiple input or output models as well as models that share layers. More than that, it allows you to define ad hoc acyclic network graphs.
Models are defined by creating instances of layers and connecting them directly to each other in pairs, then defining a Model that specifies the layers to act as the input and output to the model.

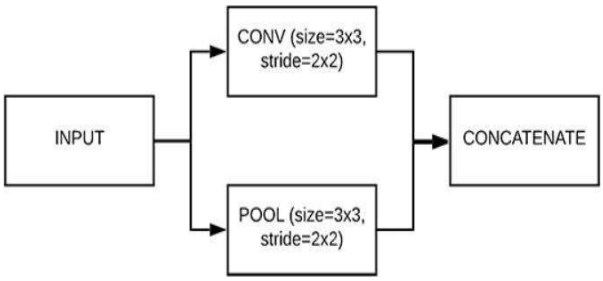


Fig. 13. Functional API

Model- Inside of Keras the Model class is the root class used to define a model architecture. Since Keras utilizes object-oriented programming, we can actually subclass the Model class and then insert our architecture definition. Model subclassing is fully-customizable and enables you to implement your own custom forward-pass of the model. However, this flexibility and customization comes at a cost — model subclassing is way harder to utilize than the Sequential API or Functional API.

Scikit-learn

Scikit-learn is undoubtedly Python's most helpful machine learning library. Classification, regression, clustering, and dimensionality reduction are some of the useful tools in the Scikit-learn toolkit for machine learning and statistical modeling. Scikit-learn is not used for reading the data, manipulating and summarizing it.

Almost all common supervised learning algorithms, such as Linear Regression, Support Vector Machine (SVM), Decision Tree, and others, are included in scikit-learn. On the other hand, it includes all of the popular unsupervised learning algorithms, including as clustering, factor analysis, PCA (Principal Component Analysis), and unsupervised neural networks.

- Ensemble methods combine the predictions of multiple supervised models, as the name implies.
- Feature extraction is a technique for extracting data features and defining attributes in image and text data.
- Feature selection is a technique for identifying interesting attributes that can be used to build supervised models.
- It is an open-source library that can also be used commercially under the BSD license.
- Clustering is a model for grouping data that hasn't been labelled.
- Cross Validation is a technique for testing the accuracy of supervised models with previously unknown data.
- Dimensionality reduction is an approach for lowering the number of attributes in data which can be further used for summarization, visualization and feature selection

Layers

In deep learning, the highest-level building block is called a layer. A layer is a container that accepts weighted input, changes it using a set of primarily non-linear functions, and then delivers the transformed values to the next layer as output.

A variety of levels perform different functions, and each layer has its own utility and value.

Input Layer:

The input layer is the most basic of all the layers, as a neural network cannot produce results without it. Any algorithm that does not accept input is useless.
They come in a number of different forms, depending on the type of input:

- Two-dimensional image
- Three-dimensional image
- Input of sequenced data, etc.

Convolutional Layers:

Convolutional Neural Networks are made up of these components, which is the application of a filter to an input.
The same filter, when applied to the input again, can provide a map of activations known as a feature map, which shows the positions and strengths of detected features in the input, such as an image.

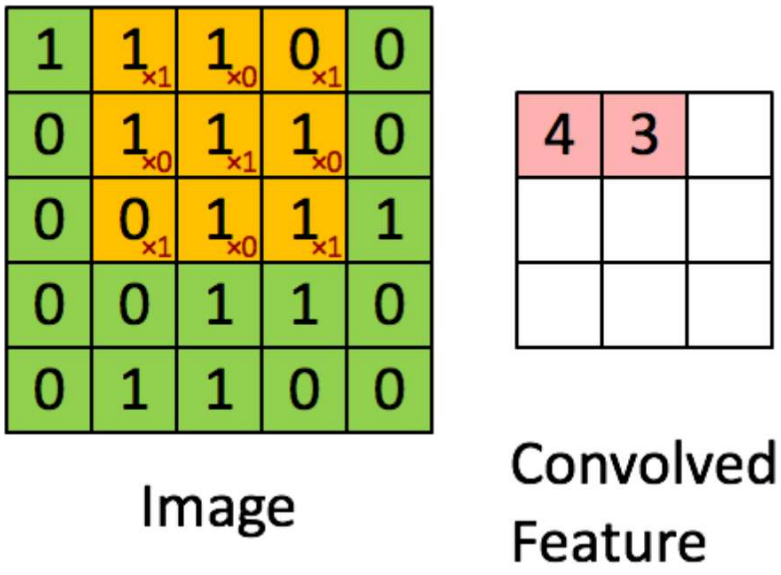


Fig. 14. Representation of Convolution process

These filters can be customised to identify a variety of features, like as

- Detecting the edges
- Object detection, for example.

Fundamentally, these layers could be of numerous types.

- Convolution 2D/3D layer - It applies sliding convolutional filters to the input.
- Grouped Convolution 2D layer - It applies sliding cuboidal convolution filters to three-dimensional input.
- Transposed Convolution 2D/3D layer - It separates the input channels into groups and applies sliding convolutional filters on them.

Activation Layer:

We use the activation function f(x) to strengthen the network and give it the ability to learn complex and difficult tasks, as well as to represent non-linear complex arbitrary function mappings between input and output.

The layer's activation functions can be used to uniquely identify it. Activation functions are mentioned below.

Crop layer:

The volume is cropped to the size of the supplied feature map for example: 2DCroplayer, 3DCroplayer, et cetera. The resize layer, on the other hand, resizes the shape of the input based on the input feature map. The image is cropped.

Dropout layer:

It nullifies input elements at random. If elements indicate probability, for example, it must set input elements to zero at random. Dropout increases generalization by preventing overfitting caused by a layer's "over-reliance" on a handful of its inputs.

Normalisation layer:

It is a mechanism to scale data into appropriate intervals, as stated earlier while addressing input layers. It generally accelerates learning and results in faster convergence, and is thus preferred when taking input. They can also be introduced in the middle, according to the programmer's approach.

Pooling layers:

Convolutional layers reveal the "presence" of features, and as a result, they are sensitive to the placement of features, leading to bias. Pooling layers lessen sensitivity to feature location.
Pooling can be accomplished by a variety of layers, including the following: Max Pooling, Average Pooling, et cetera

Max Pooling

A max pooling layer downsamples the input by dividing it into pooling zones and determining the maximum for each of them.

Output Layer:

This is the layer that produces results, devised by our neural network.

Optimizers

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

1. Gradient Descent- it is one of the most popular processes used[10]. It's an optimization process that iteratively modifies the parameters of a convex function to minimize a function to its local minimum. It starts with various coefficients, calculates their cost, and looks for a cost value that is lower than the current one. It shifts to the lesser weight and updates the coefficients' values. The process continues until the local minimum is found. A local minimum is a point beyond which it is unable to continue. It is easy to implement, however, because this method calculates the gradient for the entire data set in one update, the calculation is very slow. Also, It requires large memory and it is computationally expensive.
2. Stochastic Gradient Descent Instead of taking the entire dataset for each iteration, stochastic gradient descent selects batches of data at random. That means we only sample a small portion of the dataset. In comparison to the gradient descent approach, the path taken by the algorithm is full of noise because we are not using the entire dataset but only batches of it for each iteration. As a result, SGD requires more iterations to reach the local minima. The overall computing time increases as the number of iterations increases. However, even when the number of iterations is increased, the computation cost remains lower than that of the gradient descent optimizer.
3. Mini-Batch Gradient Descent- It simply divides the training dataset into small batches and updates each batch separately. This strikes a balance between stochastic gradient descent's robustness and batch gradient descent's efficiency. When the parameters are adjusted, the variance is reduced, and the convergence is more stable. It divides the data collection into batches of 50 to 256 randomly selected examples. It results in more consistent convergence and faster gradient calculations. Mini-batch gradient descent, on the other hand, does not guarantee good convergence. The convergence rate will be slow if the learning rate is too low. The loss function will oscillate if it is too large.
4. Momentum- It's a stochastic optimization method that combines standard stochastic gradient descent[11] with a momentum term. During the update, the direction of the previous update is kept

to some extent, while the current update gradient is used to fine-tune the final update direction. This allows you to boost the stability to a certain amount, allowing you to learn faster while also eliminating local optimization.

5. AdaGrad- Other gradient descent algorithms differ slightly from the adaptive gradient descent process. Adagrad may also be considered an optimiser for stochastic gradient descent.[13]This is due to the fact that each iteration employs different learning rates. The difference in the parameters during training determines the change in learning rate. The learning rate changes become more small as the parameters alter. Because real-world datasets contain both sparse and dense features, this change is quite advantageous.
6. RMS-Prop (Root Mean Square Propagation)- The learning rate of RMS-Prop is an exponential average of the gradients rather than the cumulative sum of squared gradients, as in Adagrad. RMS-Prop is a combination of momentum and AdaGrad. Because it reduces the monotonically falling learning rate, RMS prop can be regarded an innovation in the AdaGrad optimizer. The main goal of the approach is to speed up the optimization process by reducing the number of function evaluations required to attain the local minima. The algorithm divides the gradient by the square root of the mean square and preserves the moving average of squared gradients for each weight.
7. Adam(Adaptive Moment Estimation)- To update network weights during training, this optimization approach is a further extension of stochastic gradient descent. Unlike SGD, Adam optimizer modifies the learning rate for each network weight independently, rather than keeping a single learning rate entire training. It stores the decaying average of prior gradients, like momentum, as well as the decaying average of past squared gradients, like RMS-Prop and Adadelta. As a result, it incorporates the benefits of both strategies. It's simple to implement, computationally efficient, and memory-friendly.

Regularization Techniques

What is the need for Regularization?

A successful machine learning model has the capacity to generalise well from training data to any data from the problem domain, allowing it to make accurate predictions on data it has never seen before. To define generalization, it refers to the model's ability to apply concepts to any data rather than only the data it was trained on throughout the training process.

On the other hand, if the model is not generalized, an overfitting problem arises. Overfitting occurs when a machine learning model performs well on training data but fails when applied to testing data. Even the noise and fluctuations in the training data are picked up and learned as a concept. This is where Regularization comes in, making minor adjustments to the learning procedure to improve model generalization. The following are some examples of Regularization techniques:

L2 and L1 Regularization

The most popular methods of Regularization are L2 and L1. Regularization is based on the idea that lesser weights result in simpler models, which helps to avoid overfitting. To create a reduced weight matrix, these strategies combine the loss with a 'regularization term' to obtain the cost function.

Cost function = Loss + Regularization term

The nature of this Regularization term is the distinction between L1 and L2 Regularization approaches. In general, adding this Regularization component reduces the weight matrices' values, resulting in simpler models.

In L2, the cost function is represented as

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||^2$$

The Regularization parameter lambda is equal to the sum of squares of all feature weights. The L2 approach causes the weight to decrease but never to zero. This technique, also known as ridge Regularization, works best when all of the input characteristics influence the output and all of the weights are almost equal in size. The L2 penalty is used for decaying weights in neural networks, which is why it is the most commonly used strategy for Regularization or weight size reduction.

In L1, the cost function is represented as

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||$$

Unlike L2 Regularization, where weights are never decreased to zero, L1 Regularization penalises the absolute value of the weights. When the goal is to condense the model, this technique comes in handy. This technique, also known as Lasso Regularization, assigns zero weight to insignificant input characteristics and non-zero weight to valuable ones.

Dropout

Dropout is another commonly used Regularization approach. It basically means that throughout the training, neurons are turned off or 'dropped' at random. In a forward pass, they are briefly prevented from influencing or activating the descending neuron, and no weight updates are applied on the backward pass.

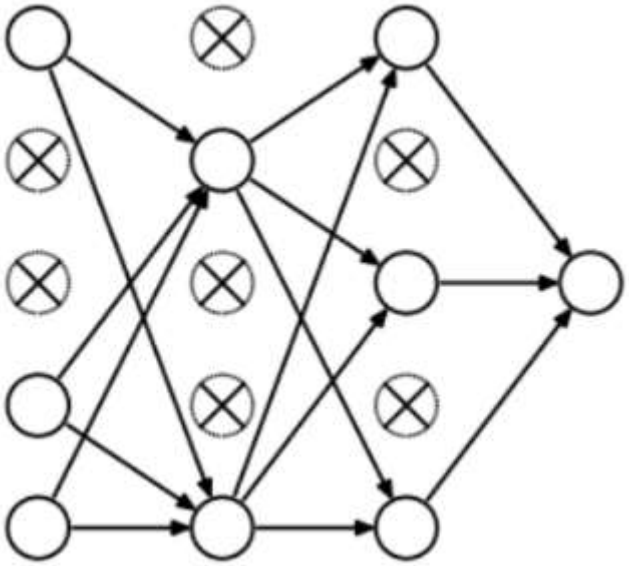


Fig. 15. Dropout Approach

If neurons are dropped out of the network at random during training, the remaining neurons step in to provide predictions for the missing neurons.
As a result, the network learns separate internal representations, making it less sensitive to the exact weight of the neurons. This type of network is more generalised and less likely to produce overfitting.

Early Stopping

It's a cross-validation approach in which a portion of the training set is utilised as a validation set, and the model's performance is measured against it. As a result, if the validation set's performance deteriorates, the model's training is halted immediately.

The key principle behind this technique is that after each iteration of fitting a neural network on training data, the model is tested on unseen data or the validation set. If the performance on this validation set is declining or staying the same for a particular number of iterations, the model training process is stopped.

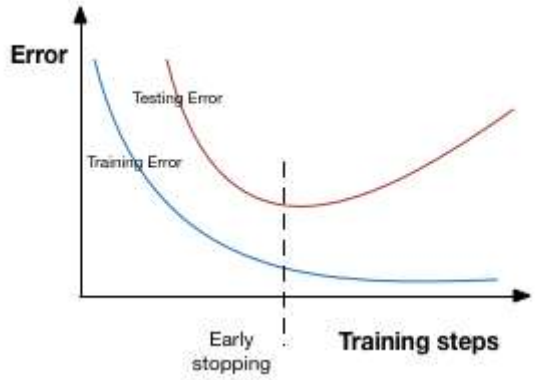


Fig. 16. Early Stopping Appproach

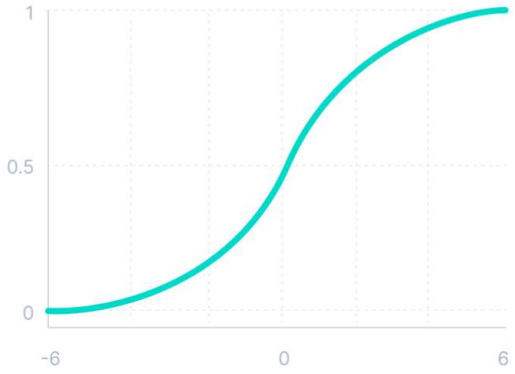
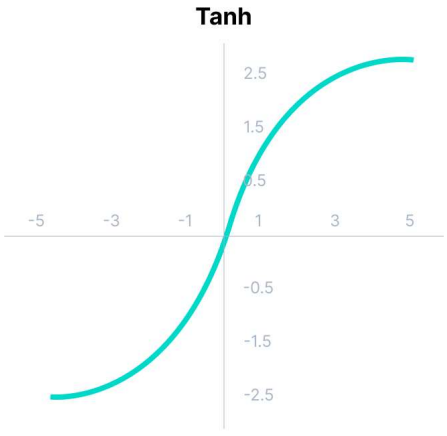
Data Augmentation

The most straightforward strategy to reduce overfitting is to expand the data, which this technique does. Data augmentation is a Regularization approach that is commonly employed when dealing with image data sets. It artificially generates more data from current training data by rotating, flipping, cropping, or blurring a few pixels in the image, and this process generates an increasing amount of data. The model variance is minimised with this Regularization strategy, which lowers the Regularization error.

ACTIVATION FUNCTIONS

Activation functions are a critical part of the design of a neural network.

The choice of activation function in the hidden layer will control how well the network model learns the training dataset. The choice of activation function in the output layer will define the type of predictions the model can make.

Name of function	formula	Graph[13]
Sigmoid	$f(x)=\frac{1}{1+e^{-x}}$	
Hyperbolic Tangent	tanh(x)	

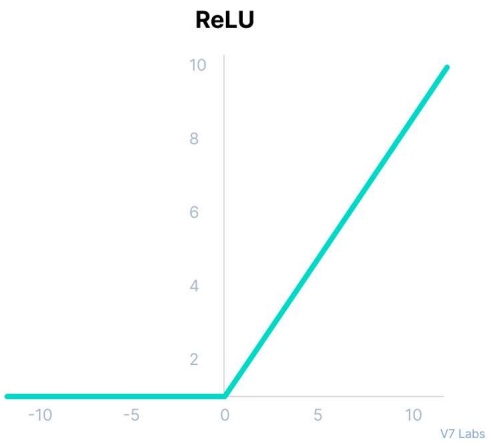
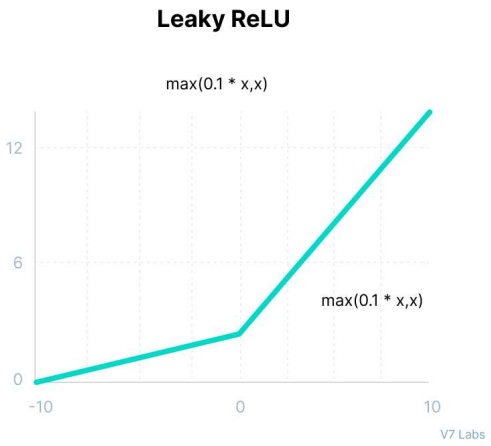
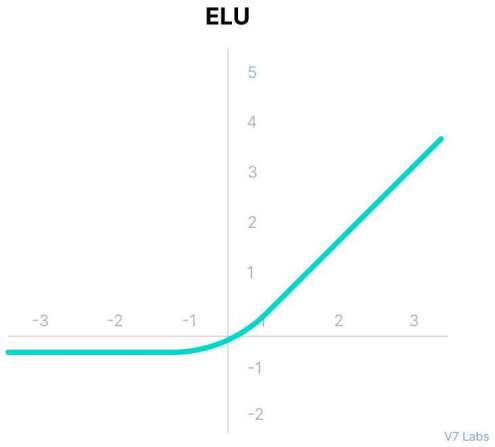
ReLU	max(0,x)	
Leaky ReLU	max(0.1x,x)	
ELU	=x , x>=0 =α(e^x-1)	

Table. 1.

Why we are using ReLU Activation:

RELU- The rectified linear activation function, or ReLU for short, is a piecewise linear function that, if the input is positive, outputs the input directly; else, it outputs zero. Because a model that utilises it is quicker to train and generally produces higher performance, it has become the default activation function for many types of neural networks. The solution had been bouncing around in the field for some time, with a lot of work being done to further analyse and improve it[14].

ReLU helps a model account for interactions. For example, if our function is a+2b, then the output from the node would be f(a+2b). Now if we use f(1,1) we get f=3, so according to ReLU we would get an output of 3. On the other hand, if b=-50 then the output is 0, and if a increases moderately, the output remains 0. So a might increase our output, or it might not. It just depends what the value of b is. This relation is captured by the activation function.

For each node, most models incorporate a bias term. The bias term is just a fixed number determined during model training. Consider a node with a single input called A and a bias for simplicity. When the bias term is set to 5, the node output is f(5+A). If A is less than -5, the output is 0 and the slope is 0 in this case. If A is greater than -5, the output of the node is 5+A, with a slope of 1. As a result of the bias term, we can relocate where the slope changes.

Real models, have a lot of nodes. Because each node (even within a single layer) might have a distinct bias value, each node can vary slope at different input values. We get a combined function that changes slopes in numerous places when we add the resultant functions back up. These models may generate non-linear functions and account for interactions well.

Implementation of CNN

One-Hot Encoding

Firstly, we defined a list named elements with all the unique characters that are present in all the SMILES strings of different polymers.

```
elements=[['c'], ['n'], ['o'], ['C'], ['N'], ['F'], ['='], ['O'],
          ['('], [')'], ['1'],['2'],['#'],['Cl'],['/']]
```

Fig. 17. Elements list

Next One-Hot Encoding is performed on the linear string of polymers in the line notation form. Here an array will be created for each molecular structure with binary inputs 0,1. Array will be of dimension m × n where m is the number of characters in the polymer chosen and n is the number of characters in the unique SMILES list.

The **imshow() function** in pyplot module of matplotlib library is used to display data as an image; i.e. on a 2D regular raster.

Strings are stored as Unicode, i.e. each character in the string is represented by a code point. So, each string is just a sequence of Unicode code points.

For efficient storage of these strings, the sequence of code points is converted into a set of bytes. The process is known as encoding.

There are various encodings present which treat a string differently. The popular encodings being utf-8, ascii, etc.Using the string encode() method, you can convert unicode strings into any encodings supported by Python. By default, Python uses utf-8 encoding.

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
```

Fig. 18. One-Hot Encoded array

Input layer for the Convolutional Neural Network should contain image data. Image data is represented by a three dimensional matrix.

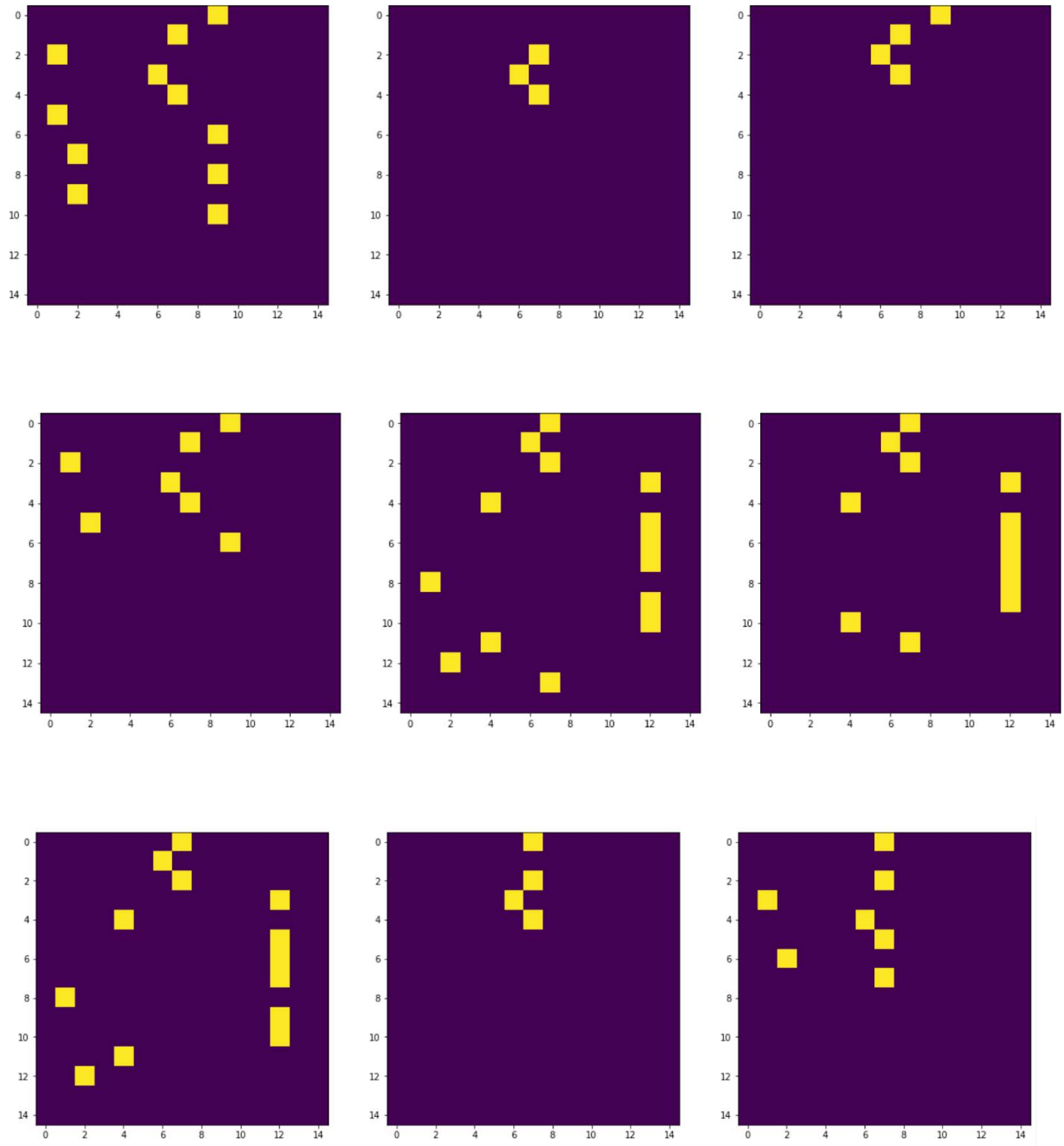


Fig. 19. Encoded Images of last 9 molecular structures.

Deploying Model

The structure of the proposed model, which accepts image inputs, has been well researched in the literature. A CNN must initially assess the influence of nearby pixels by sliding a filter across the binary image to learn the different features. This method is used to determine how many weights the neural network should learn and how essential portions of an image should be prioritised (pixels). To ensure that the pixels in this study weigh equally, a frame of 0's (or zero padding) has been added to each image. Finally, layers are given batch normalisation to change the weights and normalise the outputs while keeping the filters and window widths vary.

This model was created with the Keras library, which is a Tensorflow Application Programming Interface (API). The final hyper-parameters were chosen by combining different combinations of all the other hyper- parameters. In the first layer, the best observed configuration uses filter size 64 with a window size of (5,5), and in the second layer, size (3,3) with 32 filters. Then there's a maximum pooling layer with a window size of (3,3). We have three dense layers after the max pooling layer, each having 32, 10 or 1 neurons, with the final dense layer being the output of our suggested ANN model. All layers with l2 Regularization employed the ReLu activation function. The model achieved its best generalization by training upto 180 epochs with a batch size of 64 and learning rate of 0.03.

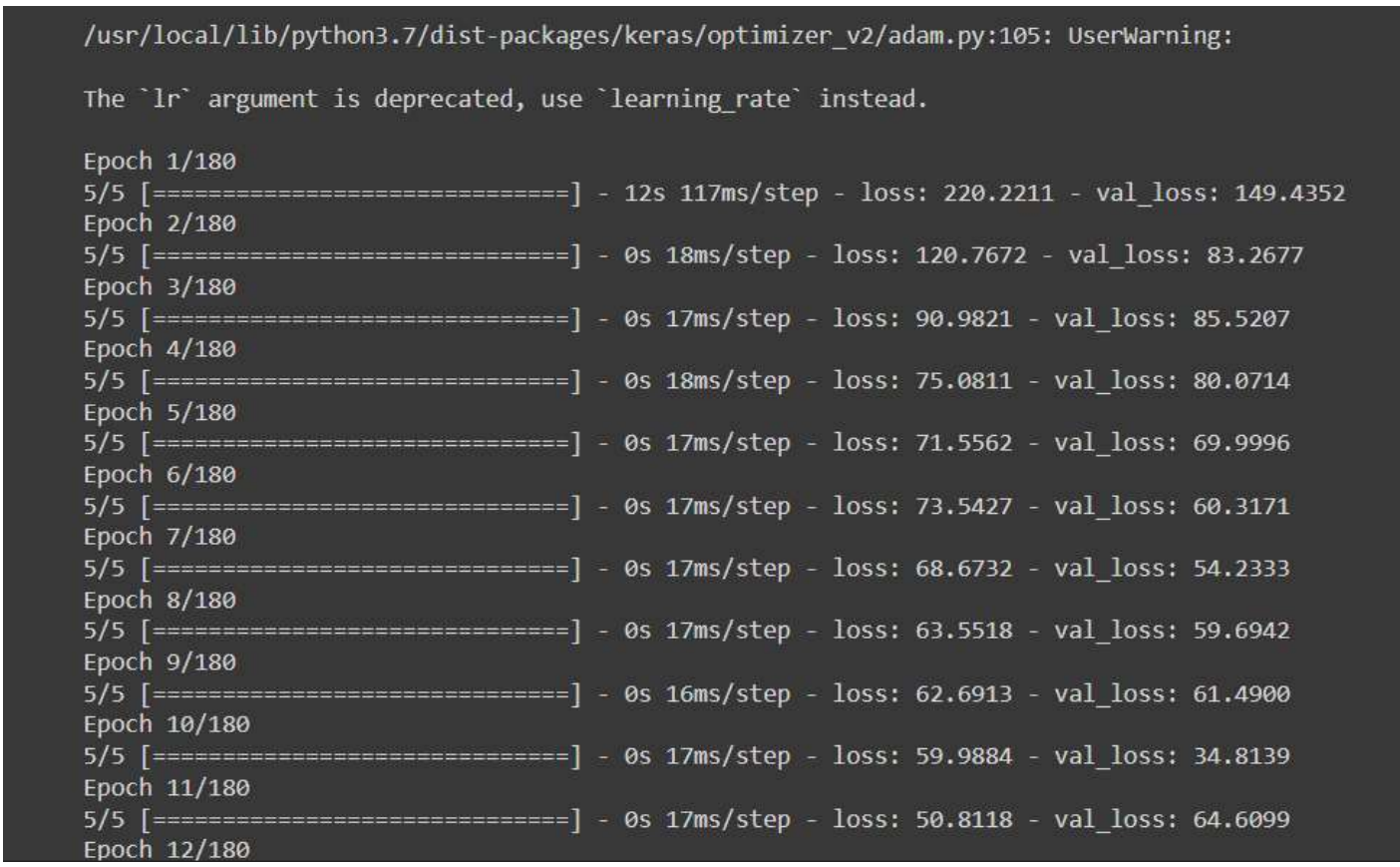


Fig. 20.

We also plot the loss vs epoch graph for both training and testing datasets. During an epoch, the loss function is calculated across every data items and it is guaranteed to give the quantitative loss measure at the given epoch.The graph obtained is indicative of a good learning rate.

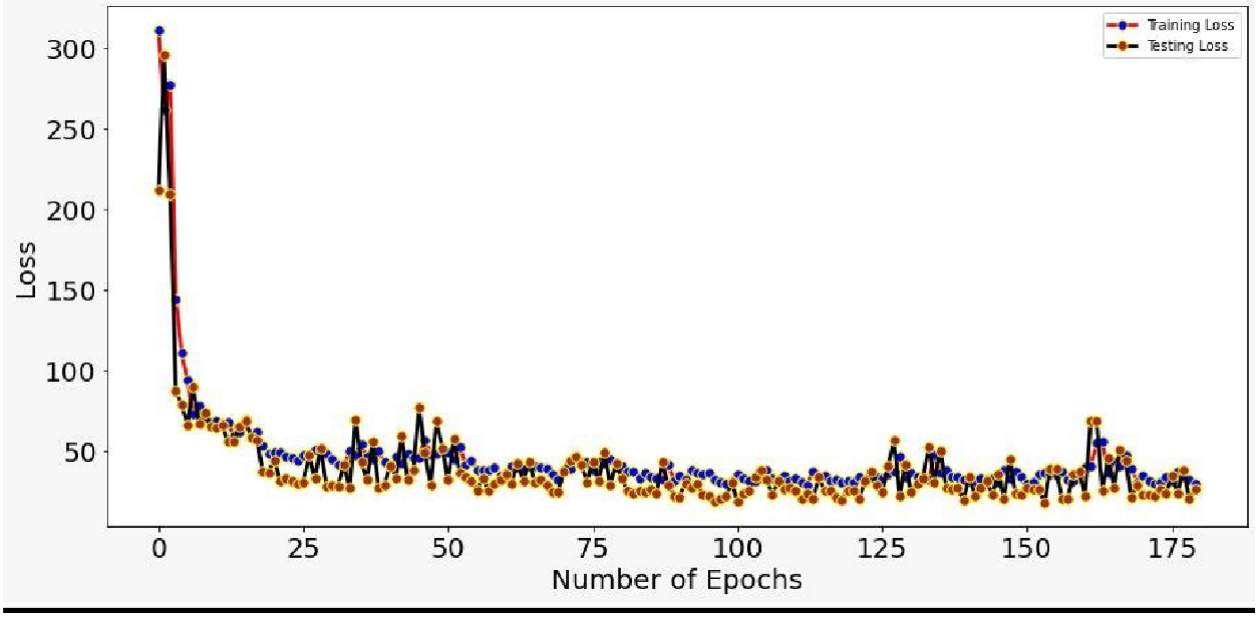


Fig. 21. Loss vs Number of Epochs graph

Implementation of ANN

Feature Engineering

For this, features were extracted using every polymer's SMILES code. Each polymer's SMILES code was analyzed at character level to understand the coding terminology. The properties extracted and the code used is shown in figures below:

Features	Extraction
Number of C Atoms in Ring	count('c')
Number of C Atoms in Chain	count('C')
Number of O Atoms in Ring	count('o')
Number of O Atoms in Chain	count('O')
Number of Double Bonds in Ring	count('c')/2
Number of Double Bonds in Chain	count('=')
Number of N Atoms	count('N')
Number of F Atoms	count('F')
Number of Cl Atoms	count('Cl')
Number of Br Atoms	count('Br')
Number of S Atoms	count('S')
Molecular Weight of the Monomer	MW Function
Degree of Branching	count('(')+count(')')/2

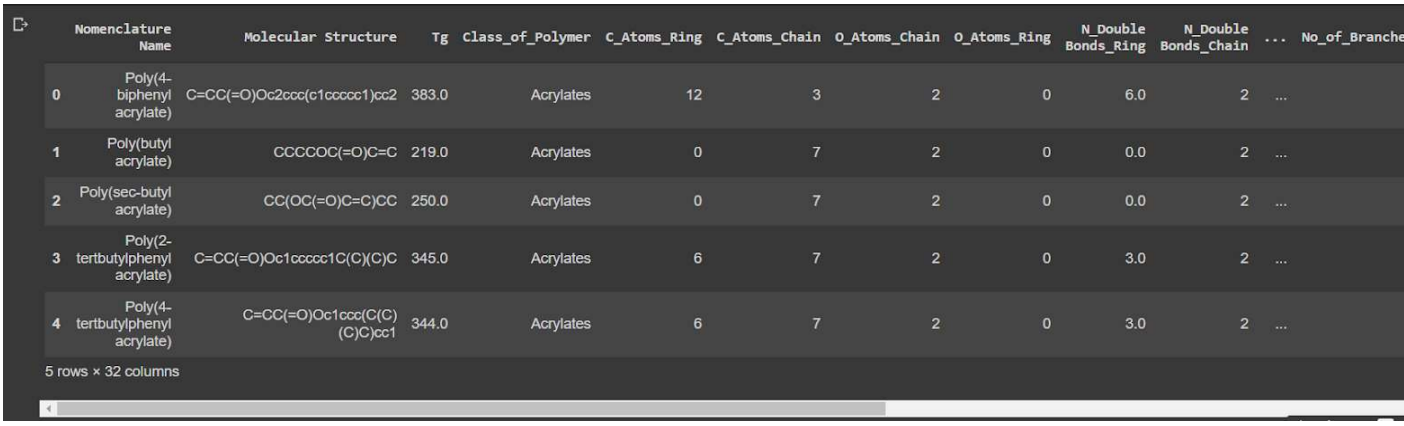
Number of Single Bonds	len(Smiles)
Number of Double Bonds	len(Smiles)-count('=')
Number of Triple Bonds	count(' ')
Number of branched C Atoms	count('(C)')
Number of branched F Atoms	count('(F)')

Table. 2.

This feature extraction is then stored as a dataframe that is shown in following figures.

```
[ ]  df["C_Atoms_Ring"]=df["Molecular Structure"].apply(lambda x: x.count('c'))
     df["C_Atoms_Chain"]=df["Molecular Structure"].apply(lambda x: x.count('C'))
     df["O_Atoms_Chain"]=df["Molecular Structure"].apply(lambda x: x.count('O'))
     df["O_Atoms_Ring"]=df["Molecular Structure"].apply(lambda x: x.count('o'))
     df["N_Double Bonds_Ring"]=df["Molecular Structure"].apply(lambda x: x.count('c')/2)
     df["N_Double Bonds_Chain"]=df["Molecular Structure"].apply(lambda x: x.count('='))
     df["Nitrogen_Atoms"]=df["Molecular Structure"].apply(lambda x: x.count('N'))
     df["F_Atoms"]=df["Molecular Structure"].apply(lambda x: x.count('F'))
     df["MW"]=df["Molecular Structure"].apply(lambda x: Molecular_Weight(x))
     df["Total Number of Atoms"]=df["Molecular Structure"].apply(lambda x: len(x))
     df["Number of Double Bonds"]=df["Molecular Structure"].apply(lambda x:len(x)-x.count('=')-1)
     df["Number of Triple Bonds"]=df["Molecular Structure"].apply(lambda x: x.count("#"))
     df["Branching"]=df["Molecular Structure"].apply(lambda x: (x.count("(")+x.count(")"))/2)
     df["Chlorine_Atoms"]=df["Molecular Structure"].apply(lambda x: x.count('Cl'))
     df["Bromine_Atoms"]=df["Molecular Structure"].apply(lambda x: x.count('Br'))
     df["Sulphur_Atoms"]=df["Molecular Structure"].apply(lambda x: x.count('S'))
     df["Two_Structures"]=df["Molecular Structure"].apply(lambda x: x.count('.'))
     df["If_Aromatic"]=df["Molecular Structure"].apply(lambda x: 1 if 'c' in x else 0)
     df["No_of_Branched_C"]=df["Molecular Structure"].apply(lambda x: x.count('(C)'))
     df["No_of_Branched_F"]=df["Molecular Structure"].apply(lambda x: x.count('(F)'))
     df["No_of_Branched_Cl"]=df["Molecular Structure"].apply(lambda x: x.count('(Cl)'))
     df["No_of_Branched_Br"]=df["Molecular Structure"].apply(lambda x: x.count('(Br)'))
     df["No_of_Branched_=O"]=df["Molecular Structure"].apply(lambda x: x.count('(=O)'))
     df["Number_of_C=C"]=df["Molecular Structure"].apply(lambda x: x.count('C=C')+x.count('cc'))
     df["Number_of_C-C"]=df["Molecular Structure"].apply(lambda x: x.count('CC'))
     df["Benze_Ring"]=df["Molecular Structure"].apply(lambda x: x.count('c1ccccc1'))
     df["Benze_Ring_var"]=df["Molecular Structure"].apply(lambda x: x.count('c1cc'))
     df["Benze_Ring_varr"]=df["Molecular Structure"].apply(lambda x: x.count('cc1'))
```

Fig. 22.



	Nomenclature Name	Molecular Structure	Tg	Class_of_Polymer	C_Atoms_Ring	C_Atoms_Chain	O_Atoms_Chain	O_Atoms_Ring	N_Double Bonds_Ring	N_Double Bonds_Chain	...	No_of_Branche
0	Poly(4-biphenyl acrylate)	C=CC(=O)Oc2ccc(c1ccccc1)cc2	383.0	Acrylates	12	3	2	0	6.0	2	...	
1	Poly(butyl acrylate)	CCCCOC(=O)C=C	219.0	Acrylates	0	7	2	0	0.0	2	...	
2	Poly(sec-butyl acrylate)	CC(OC(=O)C=C)CC	250.0	Acrylates	0	7	2	0	0.0	2	...	
3	Poly(2-tertbutylphenyl acrylate)	C=CC(=O)Oc1ccccc1C(C)(C)C	345.0	Acrylates	6	7	2	0	3.0	2	...	
4	Poly(4-tertbutylphenyl acrylate)	C=CC(=O)Oc1ccc(C(C)(C)C)cc1	344.0	Acrylates	6	7	2	0	3.0	2	...	

5 rows × 32 columns

Fig. 23.

Deploying model

This model took the newly designed features as inputs. Three dense hidden layers of 30,10, and 30 neurons are the ideal setup for this neural network. The neural network's output is the final layer of one neuron. With a batch size of 100 and a learning rate of 0.05, this model was trained for 180 epochs. On the training set, cross validation was used with a validation split of 0.1. As explained in literature, ANN layers extract the hidden features from the data using a ReLu activation function and ADAM optimizer.

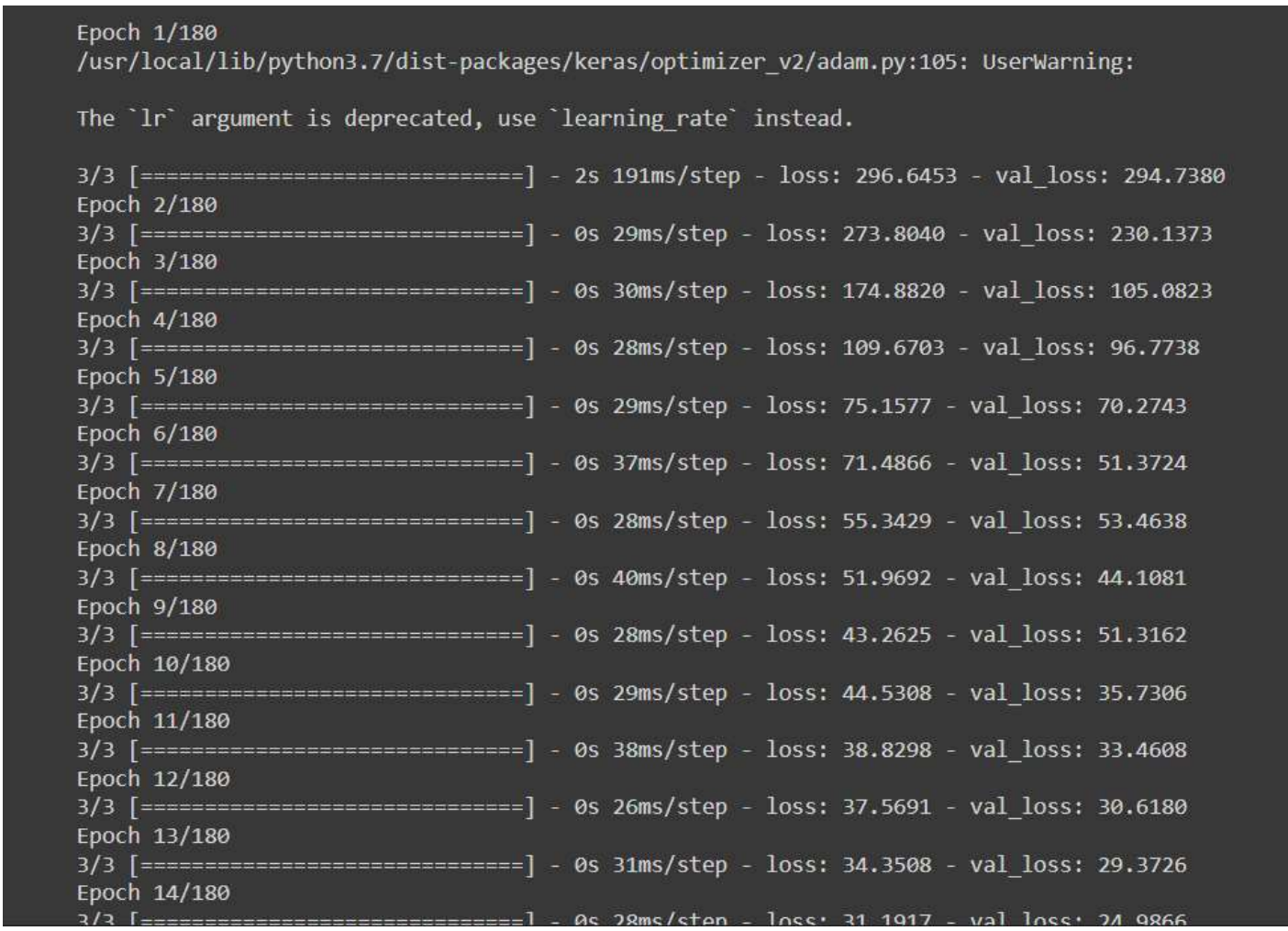


Fig. 24.

Once model is implemented we derive the loss vs epoch graph to see the working of model over time.

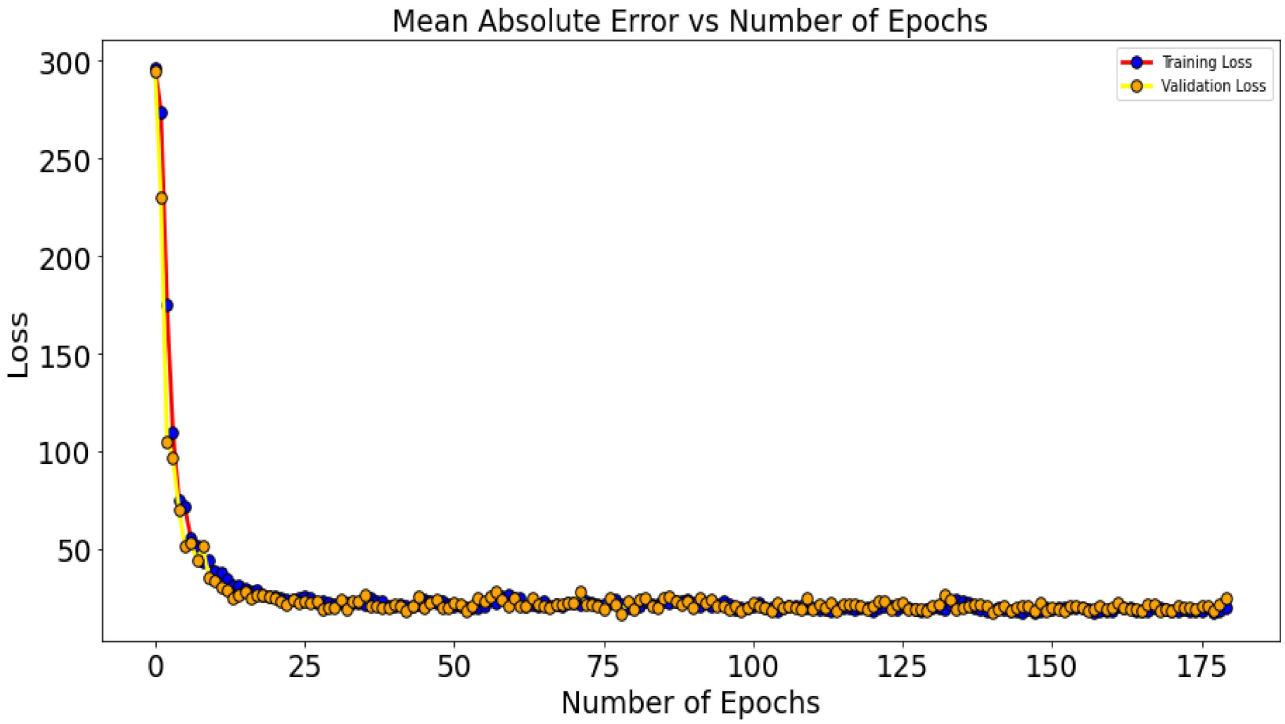


Fig. 25. Loss vs Number of Epochs graph

Results and Graphs

1. CNN

Our CNN model is best illustrated from the graphs obtained for the respective datasets.

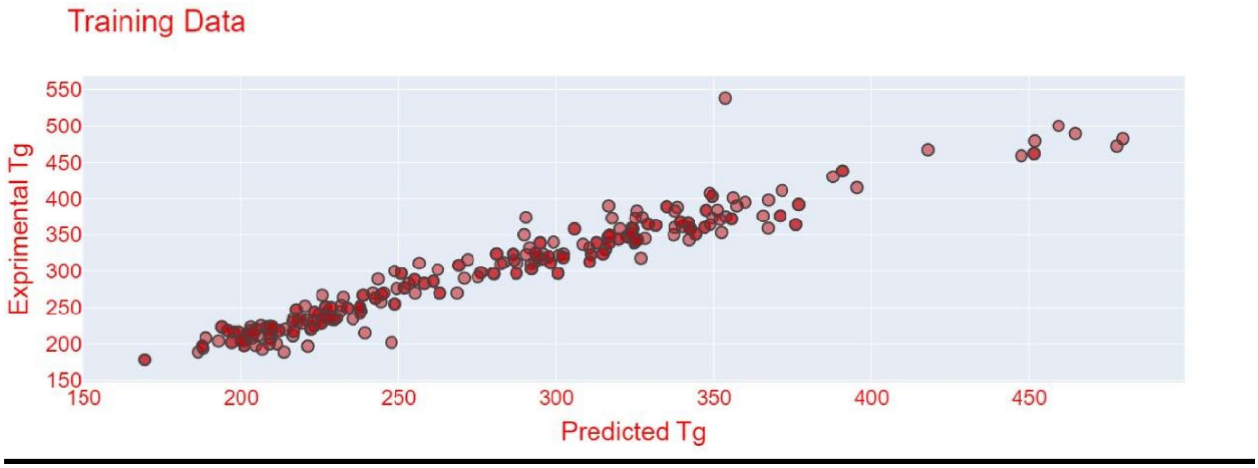


Fig. 26. Experimental Tg vs Predicted Tg of Training data

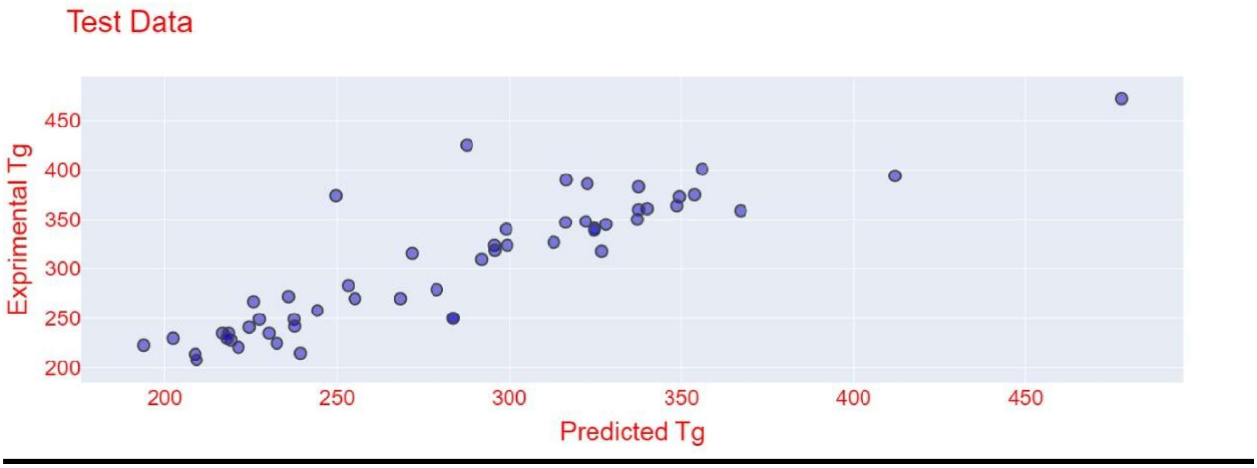


Fig. 27. Experimental Tg vs Predicted Tg of Test data for CNN Model

2. ANN

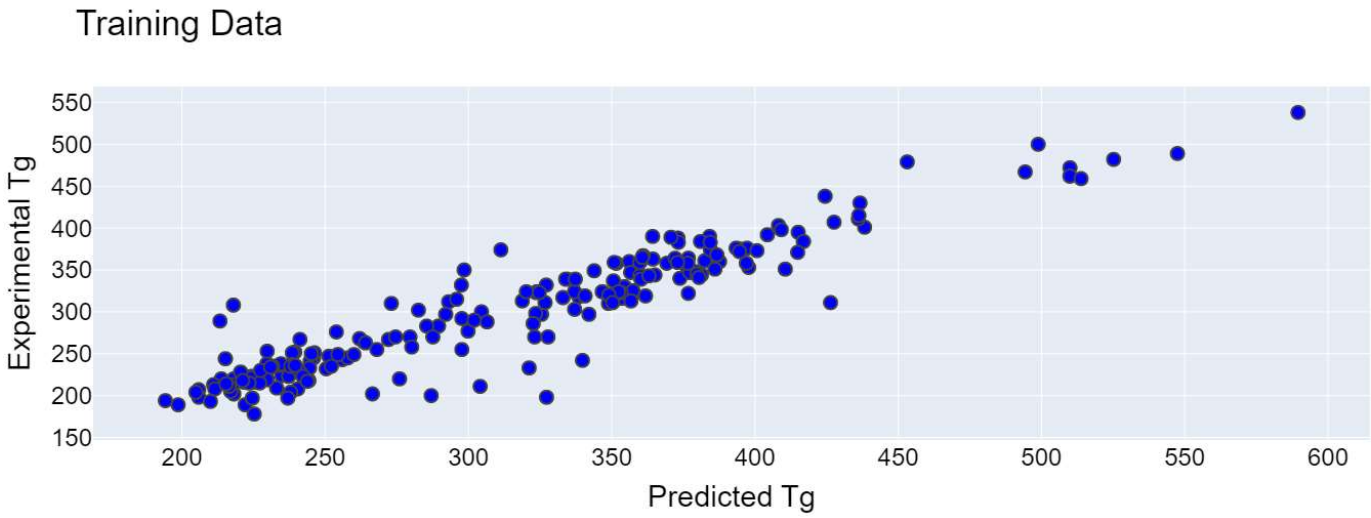


Fig. 28. Experimental Tg vs Predicted Tg of Training data for ANN Model

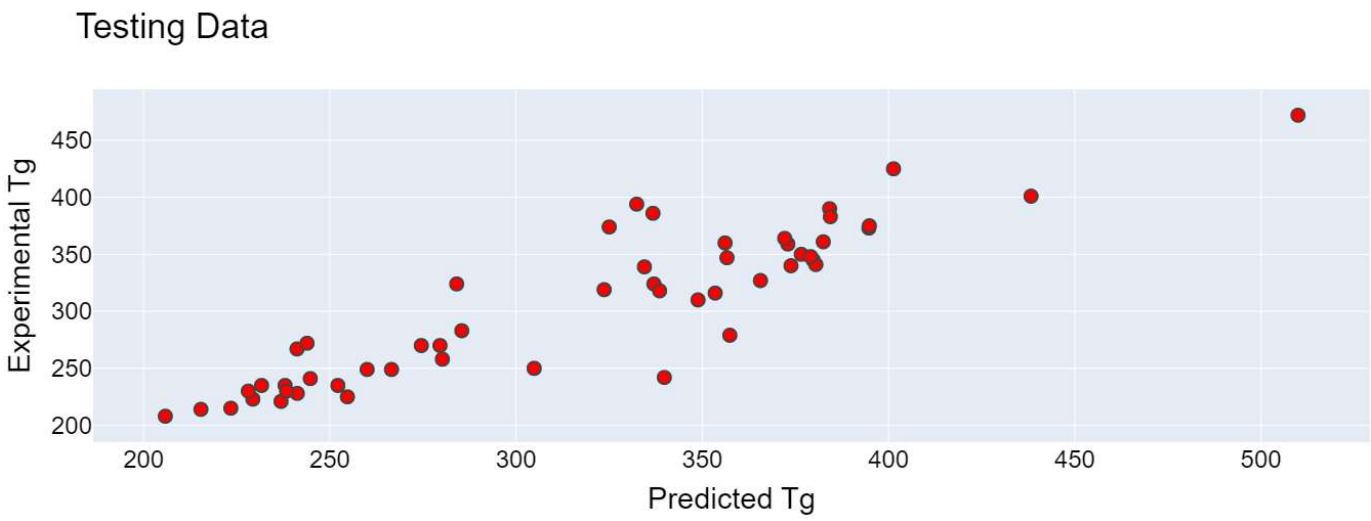


Fig. 29. Experimental Tg vs Predicted Tg of Test data for ANN Model

COMPARISON

For mathematical calculation of error we use the following formula:

$$err(\%) = \frac{1}{m} \Sigma_{i=1}^{m} 100 \frac{A_i - P_i}{A_i}$$

where A_i is the actual Tg value and P_i is the predicted Tg value. The average of this relative % error was taken over the full dataset of m polymers. The following values were obtained

	CNN Model	ANN Model
Relative Train error(%)	7.1051	7.6399
Avg unseen Test error(%)	9.7755	7.7787

Table. 3.

In general in the the graphs of both both models, when compared to genuine Tg values, it can be shown that the majority of the samples exhibit fairly accurate prediction. However, due to their

lack of representation in the training set, a few polymers contribute to a high level of uncertainty in prediction. These polymers belong to the minority classes of esters and ethers, and their Tg is not learned successfully due to insufficient training for either.
After implementing both the models we find that CNN performs marginally better on the training set. However most importantly, we find significant improvement by ANN when deployed on the testing set, with an error of 7.78% compared to the CNN model's 9.78%.

In the end we must remember that the proposed models relies only on the chemical structure of the monomer unit of the corresponding polymer. Hence, this model can become a beneficial tool for materials scientists and professionals in the polymer sphere to discover new polymeric structures at a faster pace. However, there is always a scope of further research and looking into how we can make the process even more accurate.

References

[1] Mustafa, N.S., Omer, M.A., Garlnabi, M.E., & Ismail, H. Reviewing of General Polymer Types, Properties and Application in Medical Field(2016)

[2] YunZhang∗, XiaojieXu. Machine learning glass transition temperature of polymers. North Carolina State University, Raleigh, NC 27695, USA

[3] Luis A. Miccio , Gustavo A. Schwartz. From chemical structure to quantitative polymer properties prediction through convolutional neural networks

[4] Yang, Kevin & Swanson, Kyle & Jin, Wengong & Coley, Connor & Eiden, Philipp & Gao, Hua & Guzman-Perez, Angel & Hopper, Tim & Kelley, Brian & Mathea, Miriam & Palmer, Andrew & Settels, Volker & Jaakkola, Tommi & Jensen, Klavs & Barzilay, Regina.Analyzing Learned Molecular Representations for Property Prediction. Journal of Chemical Information and Modeling.(2019)

[5] Edesio Alcoba¸caa,1, Saulo Martiello Mastelinia,1, Tiago Botaria,1, Bruno Almeida Pimentela, Daniel Roberto Cassarb,*, Andre Carlos Ponce de Leon Ferreira de Carvalhoa,Edgar Dutra Zanottob (2020) Explainable Machine Learning Algorithms For Predicting Glass Transition Temparatures.

[6] Deokar, Swapnil & Visal, Saleel. A Review Paper on Properties of Carbon Fiber Reinforced Polymers. 2. 238-243. (2016)

[7] E. Zanotto and J. Mauro, “The glassy state of matter: its definition and ultimate fate,” Journal of Non-Crystalline Solids, vol. 471, 2017.

[8] Z. Zhang and K. Friedrich, “Artificial neural networks applied topolymer composites: a review,” Composites Science and technology,2003.

[9] Andrew Blais, David Mertz, IBM- An introduction to neural networks

[10] Sebastian Ruder . An overview of gradient descent optimization algorithms, July 2017. Insight Centre for Data Analytics, NUI Galway

[11] Yanli Liu, Yuan Gao, Wotao Yin. An Improved Analysis of Stochastic Gradient Descent with Momentum. 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada

[12] A. Agnes Lydia, F. Sagayaraj Francis, Department of Computer Science and Engineering, Pondicherry Engineering College. Adagrad - An Optimizer for Stochastic Gradient Descent. INTERNATIONAL JOURNAL OF INFORMATION AND COMPUTING SCIENCE

[13] Pragati Baheti. 12 Types of Neural Network Activation Functions: How to Choose? V7 Labs

[14] Abien Fred M. Agarap. Deep Learning using Rectified Linear Units. February 20