# Data preprocessing

Data preprocessing is crucial in data analysis as it involves cleaning the data, removing redundant information, and estimating missing values. This process ensures that the data is accurate and complete, enabling more reliable and meaningful analysis. Without proper preprocessing, the quality of the analysis could be compromised.

## Extraction and Plotting of Raw Data

In the initial step of any data analysis, it's essential to collect the necessary data and store it in a dataset. Since the required data was previously gathered by NASA and stored in the file *algae.mat*, this report focuses on extracting the relevant data. It's important to note that the *DICVsChloride*, *NitrateVsChloride*, *PhosphateVsChloride* datasets, and comments were not used in this report, so they were not extracted. To load and extract the data, the following code needs to be executed.

***CodeBlock 1***

```
%% Clear and laod data set************************************************
clc;
close all;
clear;
load ('algae.mat','algae');
% ************************************************Clear and laod data set
%% Extract data*************************************************************
Dataset = struct;
notUsedData = ["DICVsChloride",...
        "NitrateVsChloride","PhosphateVsChloride","comments"];
for i=1:3
    fieldNames = setdiff(string(fieldnames(algae(i))),notUsedData);
    for j=1:numel(fieldNames)
        dataName = fieldNames(j);
        Time = datetime(string(algae(i).(dataName).time));
        Data = algae(i).(dataName).data;
        Dataset(i).(dataName).TimeTable = timetable(Time,Data,...
            'VariableNames',"raw");
    end
    Dataset(i).Names = fieldNames;
end
clearvars -except Dataset
%*************************************************************Extract data
```

After executing **CodeBlock1**, the data along with the sampled times will be extracted from the dataset. To visualize the charts for each of the three raceways, the following code can be used. Additionally, the *Nsample* value can be adjusted to plot a desired number of data points.

***CodeBlock 2***

```
%% Plot raw data************************************************************
nSample = 3000;
for i=1:3
    figure
    t=tiledlayout("flow");
    for j=1:numel(Dataset(i).Names)
        nexttile
```

```
        dataName = Dataset(i).Names(j);
        Time = Dataset(i).(dataName).TimeTable.Time;
        if numel(Time)>nSample
            data = Dataset(i).(dataName).TimeTable.raw(1:nSample);
            plot(Time(1:nSample),data)
        else
            data = Dataset(i).(dataName).TimeTable.raw;
            plot(Time,data)
        end
        xlabel('Time')
        ylabel(dataName)
     end
    title(t,"Raw data of the "+i+"'th raceway");
end
clearvars -except Dataset
% ************************************************************Plot raw data
```

For example, the figure below shows one of the outputs of the above code, corresponding to the first raceway.

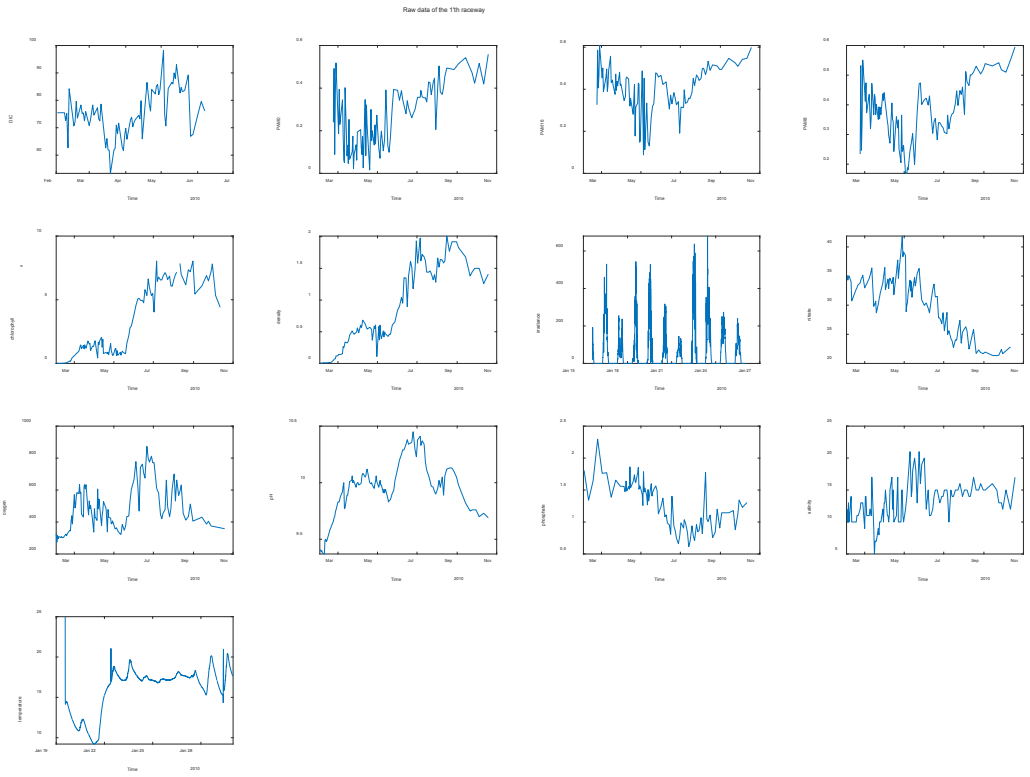



*Figure 1: Raw data for the first raceway*

## Identification and Replacement of Outlier Data

In identifying outlier data, various parameters play a significant role. Key factors include the method used and the number of samples in each dataset. In this report, the methods listed in the

table below were employed to identify abnormal data. The descriptions provided in Table 1 can be found in the documentation of the ***isoutlier*** function in MATLAB.

***Table 1:Outlier detection methods***

| Method | Description |
|---|---|
| **"median"** | Outliers are defined as elements more than three scaled MAD from the median. The scaled MAD is defined as c*median(abs(A-median(A))), where c=-1/(sqrt(2)*erfcinv(3/2)). |
| **"mean"** | Outliers are defined as elements more than three standard deviations from the mean. This method is faster but less robust than "median". |
| **"quartiles"** | Outliers are defined as elements more than 1.5 interquartile ranges above the upper quartile (75 percent) or below the lower quartile (25 percent). This method is useful when the data in A is not normally distributed. |
| **"grubbs"** | Outliers are detected using Grubbs' test for outliers, which removes one outlier per iteration based on hypothesis testing. This method assumes that the data in A is normally distributed. |
| **"gesd"** | Outliers are detected using the generalized extreme Studentized deviate test for outliers. This iterative method is similar to "grubbs", but can perform better when there are multiple outliers masking each other. |

Given that the number of samples in the *temperature* and *irradiance* datasets is higher than in the others, the impact of sample size on outlier detection can be analyzed using one of these datasets, specifically *irradiance*. Initially, 2,000 samples are considered, and outliers are identified using the five methods listed in the table below. Then, 5,000 samples are considered, and outlier detection is performed again. The results of this process, which are the output of the code provided below, can be seen in Figure 2.

***CodeBlock 3***

```
%% Effect of number of samples in outlier detection**********************
outlierDetectionMethods = ["median","mean","quartiles","grubbs","gesd"];
nSample = [2000,5000];
nout = zeros(2,6);
for i=1:numel(nSample)
    figure
    t2 = tiledlayout("flow");
    Y = Dataset(1).irradiance.TimeTable.raw(1:nSample(i));
    Time = Dataset(1).irradiance.TimeTable.Time(1:nSample(i));
    for j=1:numel(outlierDetectionMethods)
        TF = isoutlier(Y,outlierDetectionMethods(j));
        nout(i,j) = sum(TF);
        nexttile
        plot(Time,Y,Time(TF),Y(TF),'o',"MarkerFaceColor",'r');
        title(outlierDetectionMethods(j))
        ylabel("Irradiance")
        xlabel("Time")
    end
    title(t2,nSample(i)+" samples ")
end
clearvars -except Dataset outlierDetectionMethods
%%************************Effect of number of samples in outlier detection
```

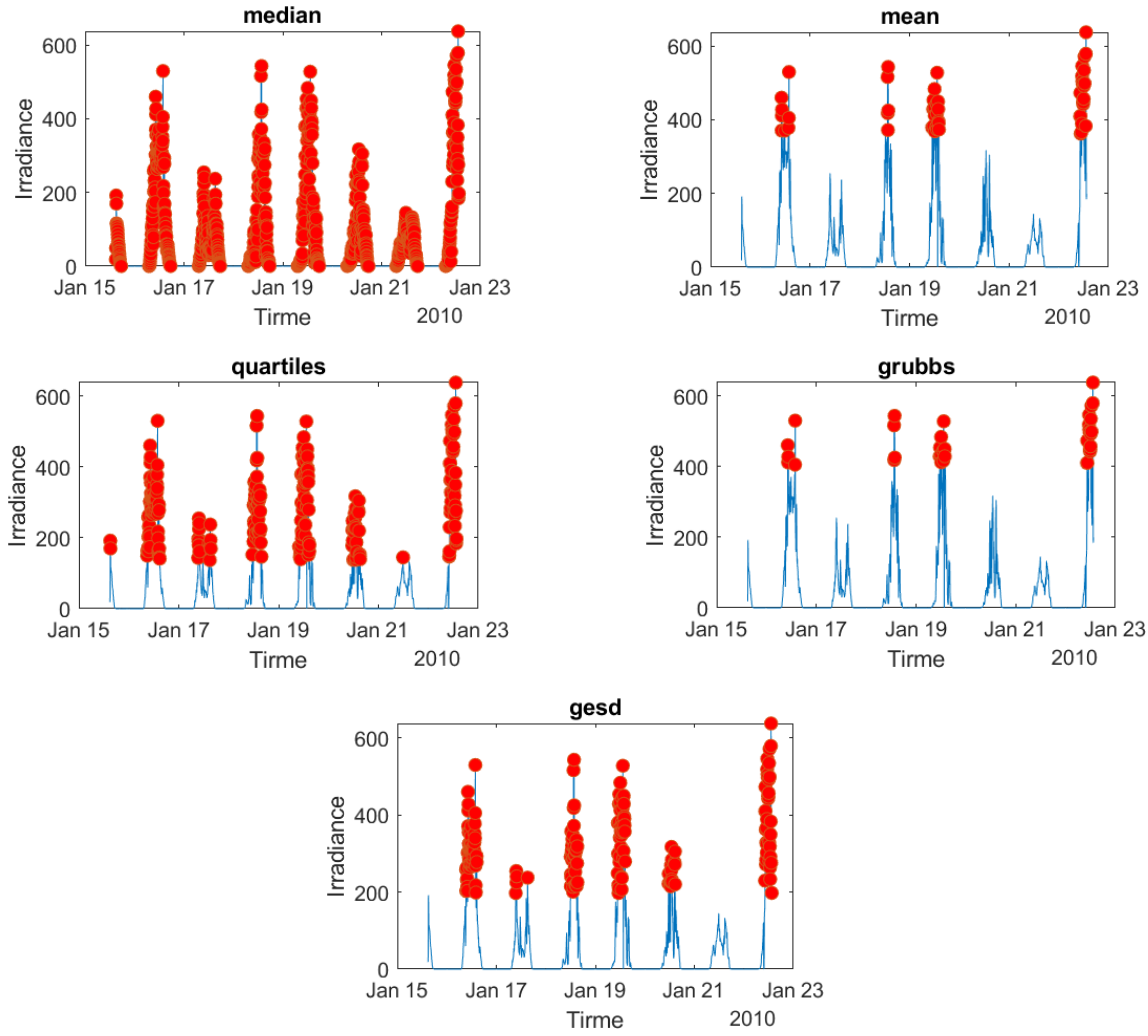
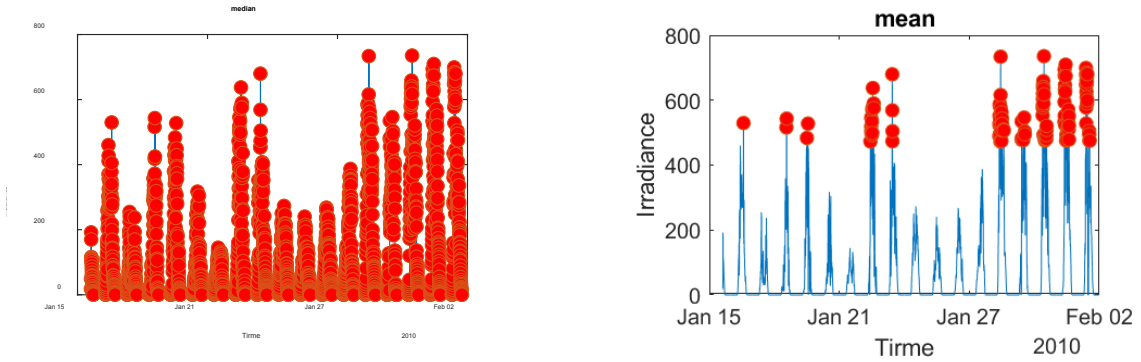



***Figure 2: Output of CodeBlock3 for 2000 sample***

As shown in Figure 2, the choice of method significantly impacts outlier detection. It can also be concluded that the *median* and *quartiles* methods are not suitable because they identify a relatively large number of outliers. To further examine the effect of sample size and to compare the results, Figure 3 has been plotted.



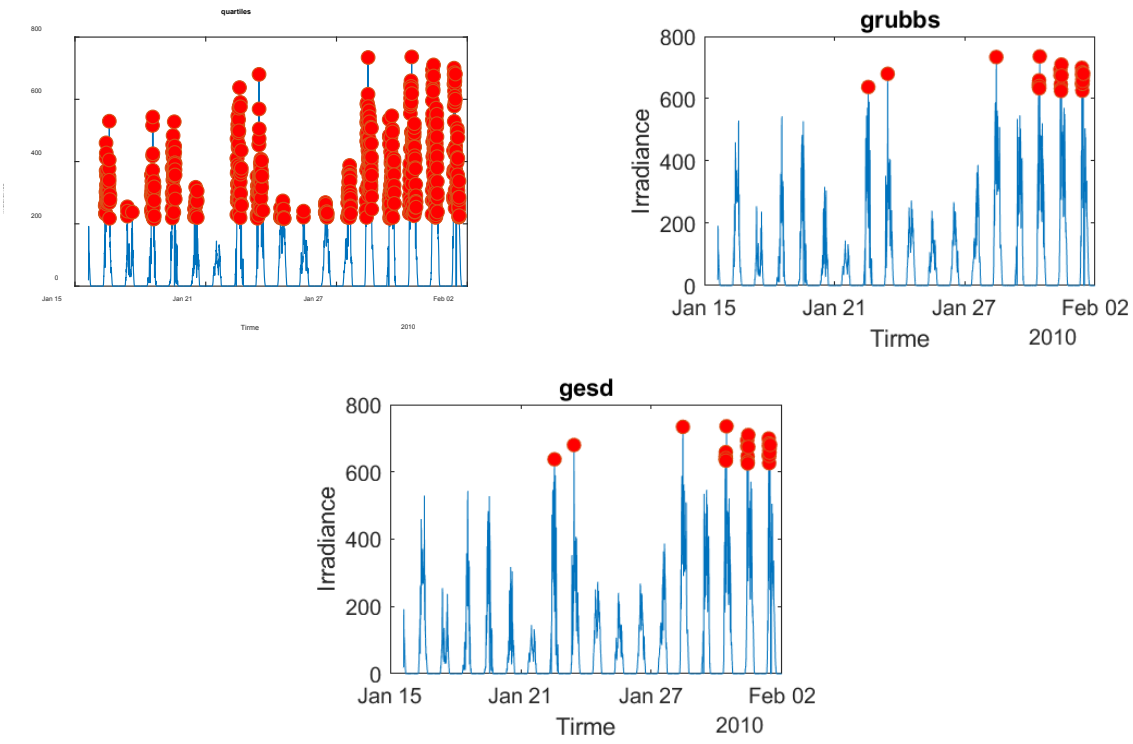



***Figure 3:Output of CodeBlock3 for 5000 sample***

A side-by-side comparison of the results from each method in Figures 2 and 3 reveals the impact of sample size on outlier detection. The numerical results of this comparison, specifically the number of outliers detected in the 2,000 and 5,000 samples, are presented in Table 2.

***Table2 : Number of identified outlier for each method***

| Method<br>Number of samples | median | mean | quartiles | grubbs | gesd |
|---|---|---|---|---|---|
| **2000** | 859 | 57 | 254 | 40 | 197 |
| **5000** | 2200 | 129 | 671 | 30 | 30 |

Based on Table 2, it can be inferred that as the number of samples increases, models such as median, mean, and quartiles identify more external data. In contrast, the number of identified outliers in models such as Grubbs and GESD decreases. Therefore, if we want to be more cautious when removing external data, we can consider data points that are identified as outliers in at least 5 of the methods. As a result, the approach is to use all available methods in Table 2 to detect anomalies and then store the data points that are identified as outliers in at least 5 methods as NaN. Finally, the following commands are executed to implement these steps.

***CodeBlock 4***

```
%% Outlier replacement with NAN*****************************************
for i=1:3
     nr = numel(Dataset(i).Names); nc = numel(outlierDetectionMethods)+1;
     Data = zeros(nr,nc);
    for j=1:nr
```

```
        dataName = Dataset(i).Names(j);
        data = Dataset(i).(dataName).TimeTable.raw;
        TF = zeros(numel(data),nc);
        for k=1:nc-1
            TF(:,k) = isoutlier(data,outlierDetectionMethods(k));
            Data(j,k)=sum(TF(:,k));
        end
         intersected = all(TF(:,1:end-1),2);
         data(intersected) = nan;
         Dataset(i).(dataName).TimeTable.filled_Nan = data;
         Data(j,end) = sum(intersected);
    end
    Dataset(i).outlierNumbers=array2table(Data);
    Dataset(i).outlierNumbers.Properties.RowNames=Dataset(i).Names;
    Dataset(i).outlierNumbers.Properties.VariableNames=...
         [outlierDetectionMethods,"intersected"];
end
clearvars -except Dataset
%*****************************************Outlier replacement with NAN
```

To view the detailed results, you can use the outlierNumbers table, which is stored for each raceway. For example, the outlierNumbers table for the third raceway looks like this:

***Table 3: OutlierNumbers table for the third raceway***

| Method<br>Dataset | median | mean | quartiles | grubbs | gesd | intersected |
|---|---|---|---|---|---|---|
| **DIC** | 0 | 0 | 0 | 0 | 0 | 0 |
| **PAM0** | 0 | 0 | 0 | 0 | 0 | 0 |
| **PAM16** | 5 | 1 | 4 | 2 | 2 | 1 |
| **PAM8** | 3 | 1 | 1 | 1 | 1 | 1 |
| **chlorophyll_a** | 0 | 0 | 0 | 0 | 0 | 0 |
| **density** | 0 | 0 | 0 | 0 | 0 | 0 |
| **irradiance** | 29644 | 1213 | 4903 | 0 | 0 | 0 |
| **nitrate** | 0 | 0 | 0 | 0 | 0 | 0 |
| **oxygen** | 0 | 0 | 0 | 0 | 0 | 0 |
| **pH** | 9 | 0 | 2 | 0 | 0 | 0 |
| **phosphate** | 0 | 0 | 0 | 0 | 0 | 0 |
| **salinity** | 0 | 0 | 0 | 0 | 0 | 0 |
| **temperature** | 4723 | 180 | 1345 | 0 | 0 | 0 |

Table 3 shows the number of identified outliers for each data set and for each method, as well as the "intersected" column, which indicates the number of outliers that are common to all methods.

## Removing Duplicates and Filling Missing Values

In this stage, the missing and duplicate values are removed from the time series data for each dataset, and the time series tables are updated accordingly. It's worth noting that for measurements taken at the same time, their average will be used as a replacement. After removing duplicates in

the dataset, the missing values that are identified as NaN in the dataset are approximated using a linear interpolation method. To perform these steps, the following code must be executed.

***CodeBlock 5***

```
%% Remove missing time data and fill nan values**************************
for i=1:3
    for j=1:numel(Dataset(i).Names)
        dataName = Dataset(i).Names(j);
        Tbl = Dataset(i).(dataName).TimeTable;
        uniqueTime = unique(Tbl.Time);
        uniqueTime(ismissing(uniqueTime)) = [];
        Dataset(i).(dataName).TimeTable = retime(Tbl,uniqueTime,'mean');
        Tbl = Dataset(i).(dataName).TimeTable;
        Dataset(i).(dataName).TimeTable = sortrows(Tbl);
        data = Dataset(i).(dataName).TimeTable.filled_Nan;
        data = fillmissing(data,"linear");
        Dataset(i).(dataName).TimeTable.replaced_Nan = data;
        Time = Dataset(i).(dataName).TimeTable.Time;
        if numel(unique(Time))==numel(Time)
           msg1 = sprintf("%s Data set for %d'th raceway is unique",...
               Dataset(i).Names(j),i);
        else
           msg1 = sprintf("%s Data set for %d'th raceway is not " + ...
               "unique",Dataset(i).Names(j),i);
        end
        if isempty(find(ismissing(data), 1))
           msg2 = sprintf("%s Data set for %d'th raceway has no" + ...
               " missing value",Dataset(i).Names(j),i);
        else
           msg2 = sprintf("%s Data set for %d'th raceway has " + ...
               "missing value",Dataset(i).Names(j),i);
        end
        disp(msg1)
        disp(msg2)
    end
end
clearvars -except Dataset
%*************************** Remove missing time data and fill nan values
```

If the output of the code above is as follows for each dataset (e.g. DIC), it indicates that the removal of duplicates and filling of missing values has been successfully performed for that dataset.

*DIC Data set for 1'th raceway is unique*

*DIC Data set for 1'th raceway has no missing value*

## Smoothing

Given that the data collection process in a specific date can be considered an event, and repeated events can lead to different readings due to human errors, device malfunctions, etc., we can use the central limit theorem to state that the distribution of the collected values will be normal. With this assumption, we need to find a function that passes through the points of interest (the collected readings) and is smooth. In other words, we can use the Gaussian process theory. Readers can refer to **https://arxiv.org/html/2009.10862v5** for more information on this topic.

It is worth noting that in Gaussian process theory, the dimension of the solution depends on the number of training data, and as the number of training data increases, the time it takes to solve the problem also increases. Considering this point, we can say that using Gaussian process theory for datasets such as temperature and irradiance takes a long time. Therefore, for simplicity, we can use moving averages for these two cases. After several trials and errors, it seemed that the 48-hour moving average (or 4-hour average) was suitable. To execute these steps, the following commands must be executed.

***CodeBlock 6***

```
%% Data smoothing************************************************************
for i=1:3
    for j=1:numel(Dataset(i).Names)
        dataName = Dataset(i).Names(j);
        Tbl = Dataset(i).(dataName).TimeTable;
        Y = Tbl.replaced_Nan;
        if ismember(dataName,["temperature","irradiance"])
            smoothed_version = movavg(Y,"simple",48);
        else
            X = daysdif(Tbl.Time(1),Tbl.Time);
            Mdl = fitrgp(X,Y,"FitMethod","fic",...
                "KernelFunction","matern52");
            smoothed_version = predict(Mdl,X);
            Dataset(i).MDModels.(dataName)=Mdl;
        end
            Dataset(i).(dataName).TimeTable.smoothed = smoothed_version;
     end
end
clearvars -except Dataset
%************************************************************Data smoothing
```

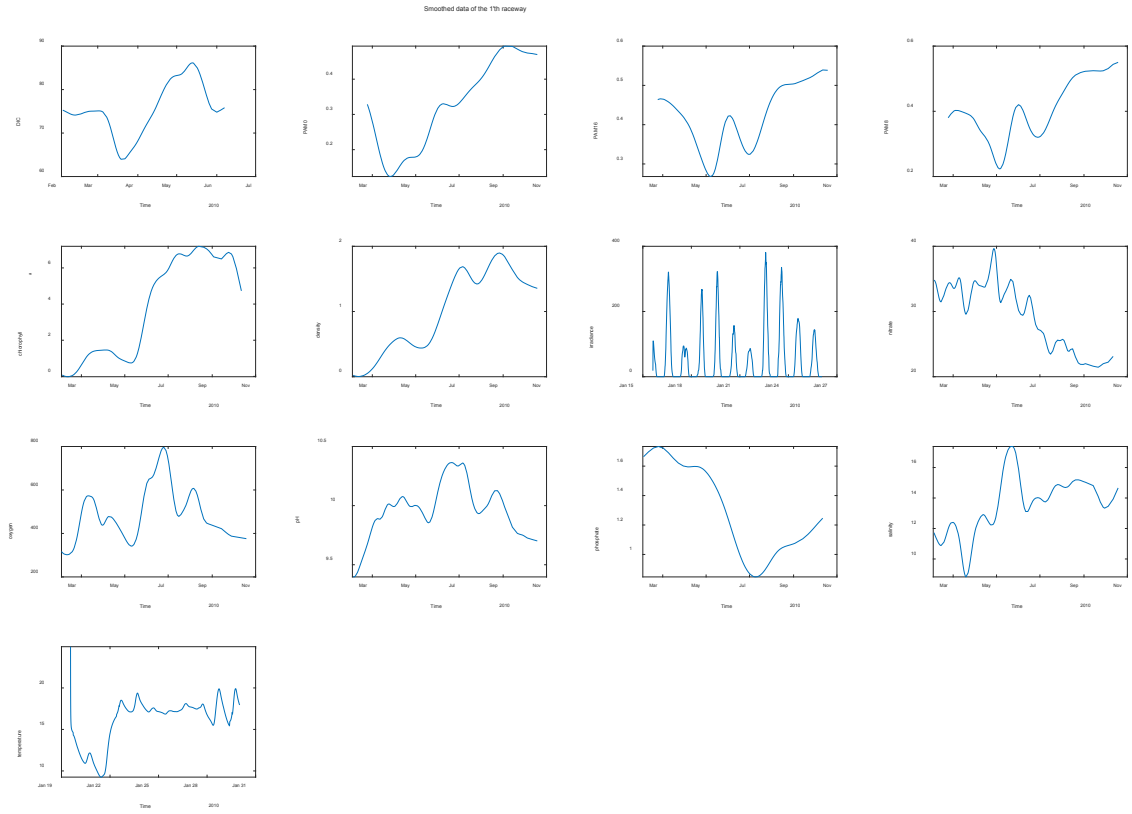
If the user wants to view the smoothed plots, they can use the following commands:

***CodeBlock 7***

```
%% Plot smoothed data********************************************************
nSample = 3000;
for i=1:3
    figure
    t=tiledlayout("flow");
    for j=1:numel(Dataset(i).Names)
        nexttile
        dataName = Dataset(i).Names(j);
        Time = Dataset(i).(dataName).TimeTable.Time;
        if numel(Time)>nSample
            data = Dataset(i).(dataName).TimeTable.smoothed(1:nSample);
            plot(Time(1:nSample),data)
        else
            data = Dataset(i).(dataName).TimeTable.smoothed;
            plot(Time,data)
        end
        xlabel('Time')
        ylabel(dataName)
```

```
    end
    title(t,"Smoothed data of the "+i+"'th raceway");
end
clearvars -except Dataset
%*********************************************************Plot smoothed data
```

For example, the figure below is one of the output plots of the code above, which corresponds to the first raceway.




# Data processing

Considering that the goal of analyzing data is to gain knowledge that can be used to make decisions, predictions, and maintenance, it is necessary to discover the relationships between different data sets and variables controlled by humans. Therefore, in this report, based on the fact that is mentioned in the attached document of the dataset, only the temperature variable is controllable, and it is necessary to extract and interpret the relationship between this data and other data sets. Therefore, it is necessary to first synchronize all datasets with temperature to initially draw the changes in each dataset based on temperature. By drawing these plots, we can initially predict how to handle the data and make decisions accordingly.

## Synchronization tables

Considering the points mentioned, in the context of synchronizing the data, it is necessary to take into account that the temperature and irradiance data are extracted every 5 minutes, while the other data is extracted daily. Therefore, for all data except irradiance, we need to select one data point

from all the 5-minute readings taken in a day. For this purpose, we can use a function that takes as input the 5-minute readings in a day and produces a single output value. For example, we can use the mean, median, maximum, and minimum daily values. To perform these steps, we can execute the following commands:

***CodeBlock 8***

```
%% Data resampling************************************************************
timeframe = "daily";
funcNames = ["max","min","mode","mean"];
newColNames = funcNames+"-temp";
for i=1:3
    temperature = Dataset(i).temperature.TimeTable(:,"smoothed");
    irradiance = Dataset(i).irradiance.TimeTable(:,"smoothed");
    Tbl = innerjoin(temperature,irradiance);
    Tbl.Properties.VariableNames = "smoothed-"+["temperature","irradiance"];
    Dataset(i).irradiance.TTsynchTemp = Tbl;
    for j=1:numel(funcNames)
        data = retime(temperature,timeframe,funcNames(j));
        if j==1
            dailyTemp = timetable(data.Time,data.smoothed,...
                'VariableNames',funcNames(j));
        else
            dailyTemp.(funcNames(j)) = data.smoothed;
        end
    end
    dailyTemp.Properties.VariableNames=newColNames;
    Names = setdiff(Dataset(i).Names,["temperature","irradiance"]);
    Dataset(i).phoDetails = Names;
    for j=1:numel(Names)
        data = Dataset(i).(Names(j)).TimeTable(:,"smoothed");
        data.Properties.VariableNames = "smoothed-"+Names(j);
        Tbl = innerjoin(dailyTemp,data);
        Dataset(i).(Names(j)).TTsynchTemp = Tbl;
        pho = corrcoef(table2array(Tbl));
        Dataset(i).(Names(j)).phoTable = array2table(pho,"RowNames",...
            [newColNames,"smoothed-"+Names(j)],"VariableNames",...
            [newColNames,"smoothed-"+Names(j)]);
        diagPho = diag(pho);
        pho = pho-diag(diagPho);
        [pmax,pmaxInd] = max(abs(pho(end,:)));
        Dataset(i).phoDetails(j,2)=newColNames(pmaxInd);
    end
end
clearvars -except Dataset
%************************************************************Data resampling
```

In CodeBlock8, the correlation between the mean, median, maximum, and minimum daily temperature and each data set is calculated and stored. Additionally, the data set that has the highest correlation with the target data set is stored in the phoDetails table. For example, the table for the second raceway is shown below:

| Name | Maximum corrilation |
|---|---|

| DIC | min-temp |
|---|---|
| PAM0 | mean-temp |
| PAM16 | mean-temp |
| PAM8 | mean-temp |
| chlorophyll_a | max-temp |
| density | mean-temp |
| nitrate | min-temp |
| oxygen | max-temp |
| pH | mean-temp |
| phosphate | max-temp |
| salinity | min-temp |

At the end, by executing the code below, you can create plots for the table by executing the following commands for each raceway:

***CodeBlock 9***

```
%% Plot synched data*****************************************************
for i=1:3
    figure
    t=tiledlayout("flow");
    axLabel = Dataset(i).phoDetails;
    for j=1:size(axLabel,1)
        nexttile
        dataName = axLabel(j,1);
        colName = "smoothed-"+axLabel(j,1);
        varName = axLabel(j,2);
        X = Dataset(i).(dataName).TTsynchTemp.(varName);
        Y = Dataset(i).(dataName).TTsynchTemp.(colName);
        scatter(X,Y)
        xlabel(varName)
        ylabel(colName)
    end
    title(t,"Data of the "+i+"'th raceway vs temperature");
end
clearvars -except Dataset
%*********************************************************Plot synched data
```


***Figure 4: Relation between each dataset and temperature***

The plots provided do not reveal a clear relationship between temperature and other variables, and the current presentation of the data does not provide sufficient insight. To gain a deeper understanding, we will employ a probabilistic approach and regression analysis to examine the problem.

## Probability function calculation

In this approach, the probability of occurrence of different values of a variable in various temperature ranges (daily mean temperature) is calculated discretely and then interpolated using a probability function. To achieve this, data sets corresponding to all raceway are collected. Then, the discrete probability function is calculated and interpolated using a thin plate interpolation model to obtain the probability function. The following commands need to be executed to perform the above steps.

***CodeBlock 10***

```
%% Probaility model*******************************************************
Names = setdiff(Dataset(1).Names,["temperature","irradiance"]);
fitType = "thinplateinterp";
```

```
t=tiledlayout("flow");
for i=1:numel(Names)
    data = [];
    for j=1:3
        data = [data;Dataset(j).(Names(i)).TTsynchTemp.Variables];
    end
    data(:,1:3)=[];
    [N,c] =hist3(data,"nbins",[15,15],"FaceColor","interp",...
        "CDataMode","auto");
    Pdf = N/sum(N,"all");
    Pdf = Pdf(:);
    [temp,var] = meshgrid(c{1},c{2});
    temp = temp(:);var = var(:);
    fitobject = fit([temp,var],Pdf,fitType);
    Dataset(j).(Names(i)).fitObject = fitobject;
    nexttile
    plot(fitobject)
    xlabel("temperature")
    ylabel(Names(i))
    zlabel("PDF")
end
clearvars -except Dataset
%*************************************************************Probaility model
```

The output of the code above will be a probability function of temperature and the corresponding dataset, which will be stored in the fitObject variable for each dataset. Additionally, executing this code will generate Figure 5, which displays the PDF function for each dataset.

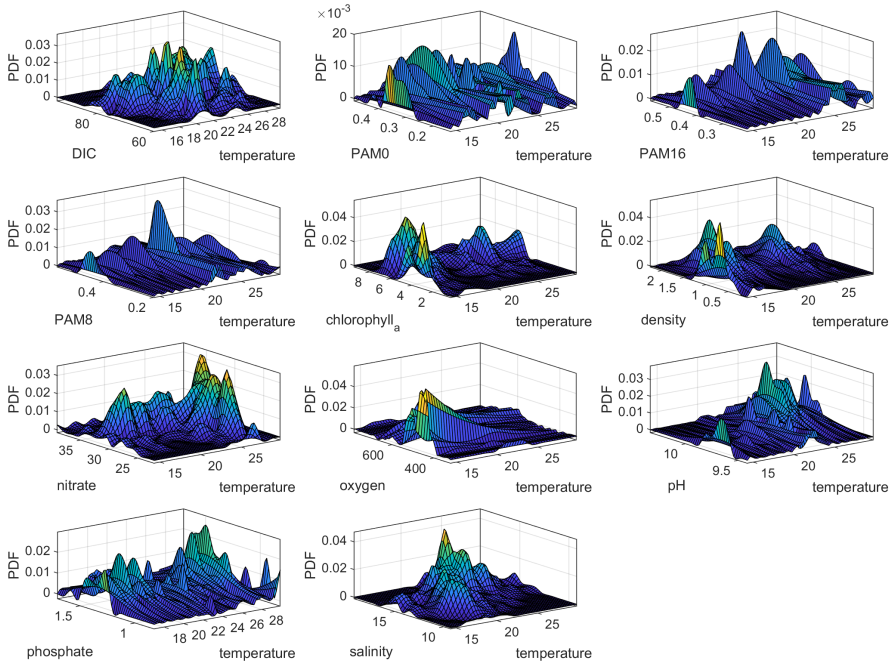



*Figure 5: PDF functions for each dataset and temperature*

**Regression analysis by using deep learning models**

Given that the value of each variable in the new measurement may be dependent on the current status and temperature changes from the current time to the new measurement, it is possible to formulate its mathematical form as follows:

$$y_{t+1} = f(y_t, y_{t-1}, ..., y_{t-d_y}, x_t, x_{t-1}, ..., x_{t-d_x})$$

where y(t+1) is the value of the variable at time t+1, dy is the delay required for the variable y, x is the temperature, and dx is the delay required for the temperature. In this report, we assume that the delay values dy and dx are 1 and 287, respectively, where the latter represents a 5-minute lag of the previous day's temperature. As a result, the input to the network consists of 288 features, while the output is the predicted y value in the new measurement. Each input is transformed into a 12x24 matrix, which is then treated as an image in the deep learning network. To generate the required dataset for training and testing, we execute the following code commands. Please note that you must enter the desired variable name in the first line of the code below as dataName.

```
%% Feature extraction*****************************************************
dataName = "salinity";
Hx = 24;Mx=0;Sx=0;
dy = 1;
allData = {};
allTarget = [];
for i=1:3
    temp = Dataset(i).temperature.TimeTable(:,"smoothed");
    T2 = Dataset(i).(dataName).TimeTable.Time(dy+1:end);
    T1 = T2-duration(Hx,Mx,Sx);
    data = Dataset(i).(dataName).TimeTable.("smoothed");
    for k=1:numel(T1)
        tdate = find(temp.Time>=T1(k) & temp.Time<T2(k));
        allData{end+1} = [temp(tdate,:).smoothed;data(k)];
        allTarget = [allTarget,data(k+1)];
    end
end
% Remove smaples with unsufficient features
a = cell2mat(cellfun(@(x)size(x,1),allData,'UniformOutput',false));
[b,c] = groupcounts(a');
[~,ind] = max(b);
badData = find(a~=c(ind));
allData(badData)=[];
allTarget(badData)=[];
% Split train and test data
allData = cell2mat(allData);
[nfeature,nSample] = size(allData);
trSamples = randi([1,nSample],floor(0.7*nSample),1);
teSamples = setdiff(1:nSample,trSamples);
allData = reshape(allData,24,12,1,nSample);
Xtrain = arrayDatastore(allData(:,:,:,trSamples),"IterationDimension",4);
Xtest = arrayDatastore(allData(:,:,:,teSamples),"IterationDimension",4);
Ytrain = arrayDatastore(allTarget(:,trSamples),"IterationDimension",2);
Ytest = arrayDatastore(allTarget(:,teSamples),"IterationDimension",2);
% Create Data sotres
Dataset(4).(dataName).TrDs = combine(Xtrain,Ytrain);
```

```
Dataset(4).(dataName).TeDs = combine(Xtest,Ytest);
clearvars -except Dataset dataName
%******************************************************Feature extraction
```

Following a series of iterative refinements and improvements, a proposed architecture has been established as a comprehensive model for deep learning, applicable to all variables.

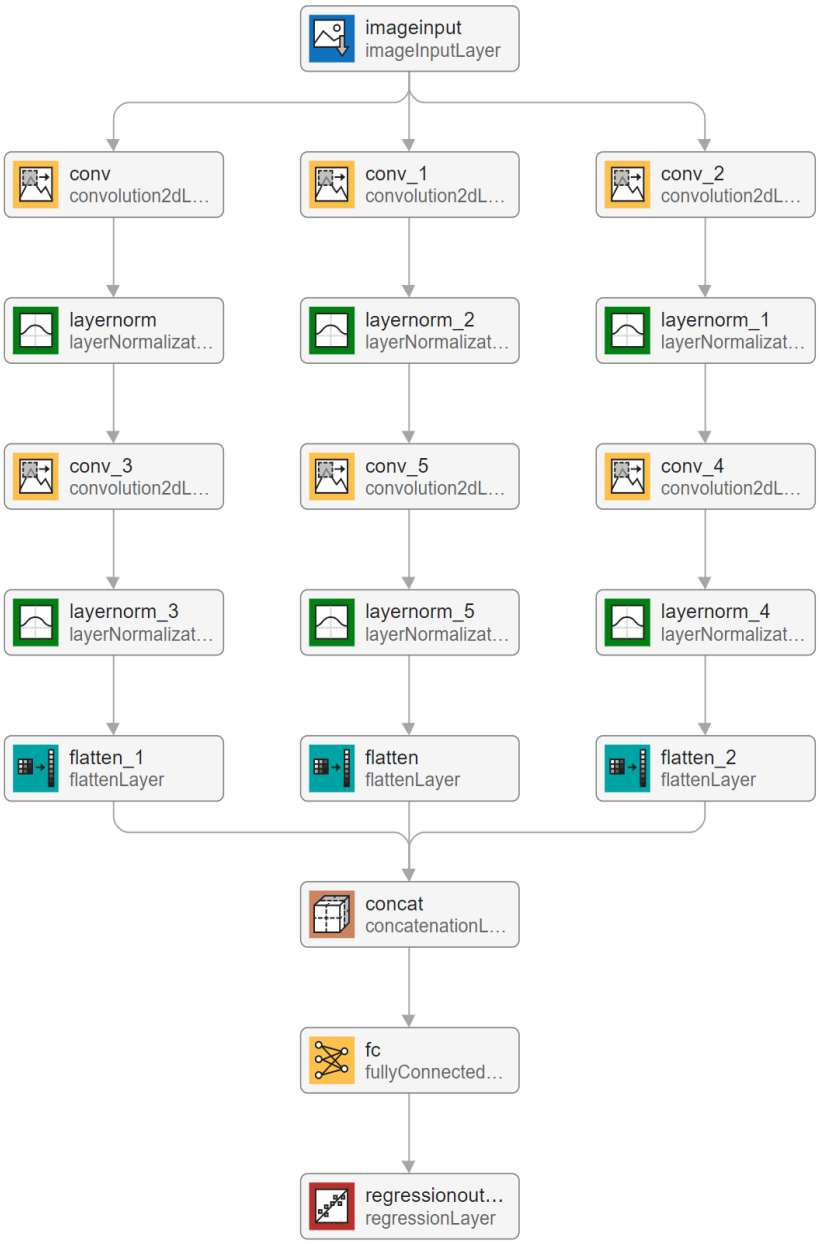



***Figure 6: Deep learning architecture used for finding relation between temperature and other datasets***

To create the architecture, you can store the following commands in a script named LG and then execute it.

```
%% Layer generatation*****************************************************
lgraph = layerGraph();
```

```
% Add Layer Branches
% Add the branches of the network to the layer graph. Each branch is a
% linear array of layers.
tempLayers = imageInputLayer([24 12 1],"Name","imageinput");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([4 4],32,"Name","conv","Padding","same")
    layerNormalizationLayer("Name","layernorm")
    convolution2dLayer([4 4],32,"Name","conv_3","Padding","same")
    layerNormalizationLayer("Name","layernorm_3")
    flattenLayer("Name","flatten_1")];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([3 3],32,"Name","conv_1","Padding","same")
    layerNormalizationLayer("Name","layernorm_2")
    convolution2dLayer([3 3],32,"Name","conv_5","Padding","same")
    layerNormalizationLayer("Name","layernorm_5")
    flattenLayer("Name","flatten")];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([5 5],32,"Name","conv_2","Padding","same")
    layerNormalizationLayer("Name","layernorm_1")
    convolution2dLayer([5 5],32,"Name","conv_4","Padding","same")
    layerNormalizationLayer("Name","layernorm_4")
    flattenLayer("Name","flatten_2")];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    concatenationLayer(1,3,"Name","concat")
    fullyConnectedLayer(1,"Name","fc")
    regressionLayer("Name","regressionoutput")];
lgraph = addLayers(lgraph,tempLayers);

% clean up helper variable
clear tempLayers;

% Connect Layer Branches
% Connect all the branches of the network to create the network graph.
lgraph = connectLayers(lgraph,"imageinput","conv");
lgraph = connectLayers(lgraph,"imageinput","conv_1");
lgraph = connectLayers(lgraph,"imageinput","conv_2");
lgraph = connectLayers(lgraph,"flatten_1","concat/in1");
lgraph = connectLayers(lgraph,"flatten_2","concat/in2");
lgraph = connectLayers(lgraph,"flatten","concat/in3");

plot(lgraph);
%**************************************************Layer generatation
```

To conclude, the following commands can be used for training the network.

```
%% Layer generatation**************************************************
LG
```

```
%*********************************************************Layer generatation
%% Train Network*************************************************************
Options = trainingOptions("adam",...
                          "MiniBatchSize",20,...
                          "InitialLearnRate",0.001,...
                          "ValidationData",Dataset(4).(dataName).TeDs,...
                          "MaxEpochs",300,...
                          "Plots","training-progress",...
                          "OutputNetwork","best-validation-loss");

net = trainNetwork(Dataset(4).(dataName).TrDs,lgraph,Options);
Dataset(4).(dataName).network = net;
%*************************************************************Train Network
```

For example, by selecting dataName="DIC", after executing the code above, the training history, changes in the cost function, and the RMSE value will be displayed as shown in the figure below. If the user has another variable in mind, they should modify the dataName accordingly.

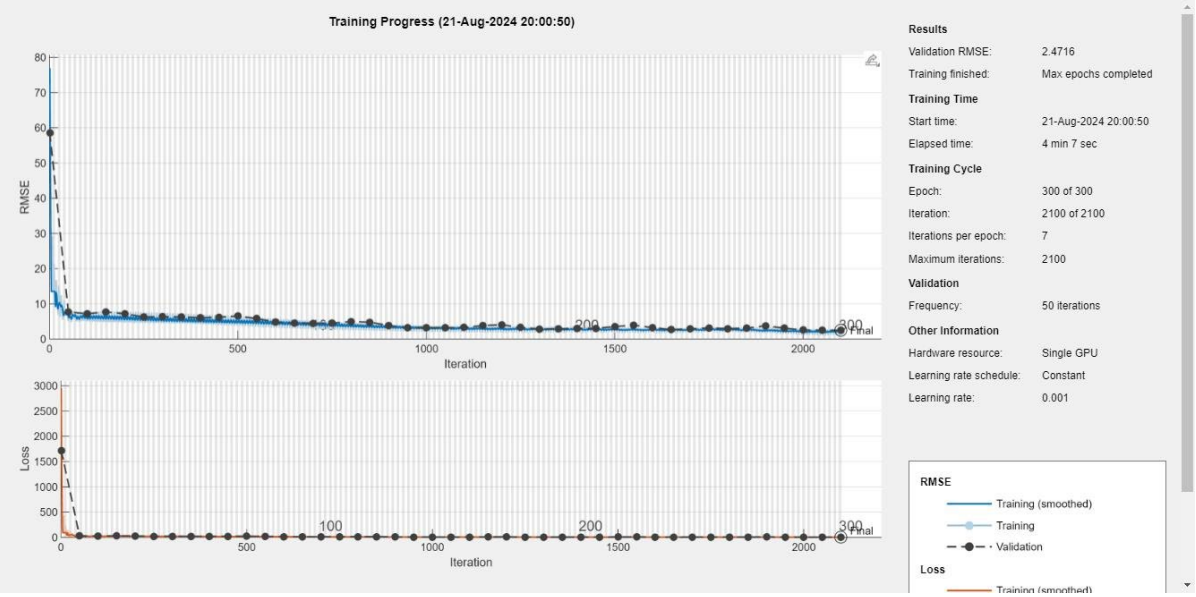



***Figure 7: Iteration history of deep learning training***

After training the network for all variables, the results for different datasets are summarized in the table below.

***Table4 : Validation RMSE for each dataset***

| Variable | Validation RMSE |
|---|---|
| DIC | **2.47** |
| PAM0 | **0.08** |
| PAM16 | **0.09** |
| PAM8 | **0.09** |
| chlorophyll_a | **0.29** |
| density | **0.33** |
| nitrate | **0.56** |
| oxygen | **19.97** |
| pH | **0.20** |

| | |
|---|---|
| phosphate | **0.15** |
| salinity | **0.31** |

To investigate the relationship between irradiance and temperature at a future time step (t+1), the network is fed with temperature values at both time steps t and t+1, as well as the irradiance value at time t. To create the training and testing datasets in accordance with the established protocol, the following commands should be executed.

***CodeBlock 11***

```
%% Feature extraction for Temp and Irr************************************
Tbl = Dataset(1).irradiance.TTsynchTemp;
nSample = size(Tbl,1)-2;
allData = [Tbl.("smoothed-temperature")(1:end-2)';...
            Tbl.("smoothed-temperature")(2:end-1)';...
            Tbl.("smoothed-irradiance")(2:end-1)'];
allTarget = Tbl.("smoothed-irradiance")(3:end)';
allData = reshape(allData,3,1,nSample);
% Split train and test data
trSamples = randi([1,nSample],floor(0.7*nSample),1);
teSamples = setdiff(1:nSample,trSamples);
Xtrain = arrayDatastore(allData(:,:,trSamples),"IterationDimension",3);
Xtest = arrayDatastore(allData(:,:,teSamples),"IterationDimension",3);
Ytrain = arrayDatastore(allTarget(:,trSamples),"IterationDimension",2);
Ytest = arrayDatastore(allTarget(:,teSamples),"IterationDimension",2);
% Create Data sotres
Dataset(4).irradiance.TrDs = combine(Xtrain,Ytrain);
Dataset(4).irradiance.TeDs = combine(Xtest,Ytest);
clearvars -except Dataset dataName
%************************************ Feature extraction for Temp and Irr
```

To create the layers of the deep learning network and train the network, the following commands can be used.

***CodeBlock 12***

```
%% Layer generatation*****************************************************
LG2
%*****************************************************Layer generatation
%% Train Network**********************************************************
Options = trainingOptions("adam",...
                          "MiniBatchSize",500,...
                          "InitialLearnRate",0.001,...
                          "ValidationData",Dataset(4).irradiance.TeDs,...
                          "MaxEpochs",300,...
                          "Plots","training-progress",...
                          "OutputNetwork","best-validation-loss");

net = trainNetwork(Dataset(4).irradiance.TrDs,lgraph,Options);
Dataset(4).irradiance.network = net;
%*********************************************************Train Network
```

The network used was created using the LG2 script and has the architecture depicted in the following diagram. The network's RMSE value is 9.7.

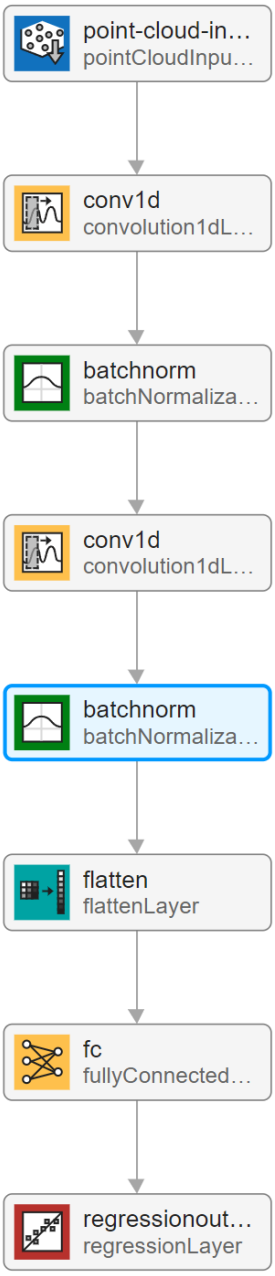



***Figure 8: Deep learning used for finding relation between temperature and irradiance***