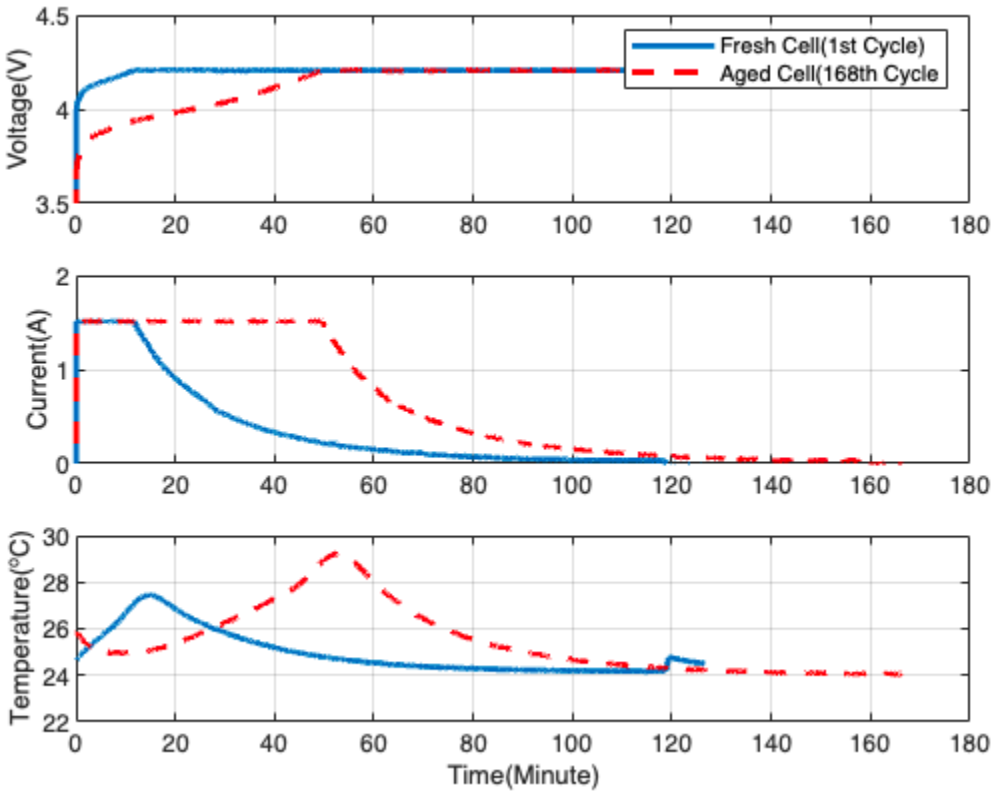
CNN_final.mlx file:

Multi-Channel charging profiles of V, I, T data

To observe if charging profiles shows battery aging, we need to explore Voltage(V), Current(I) and Temperature(T) path through battery aging.

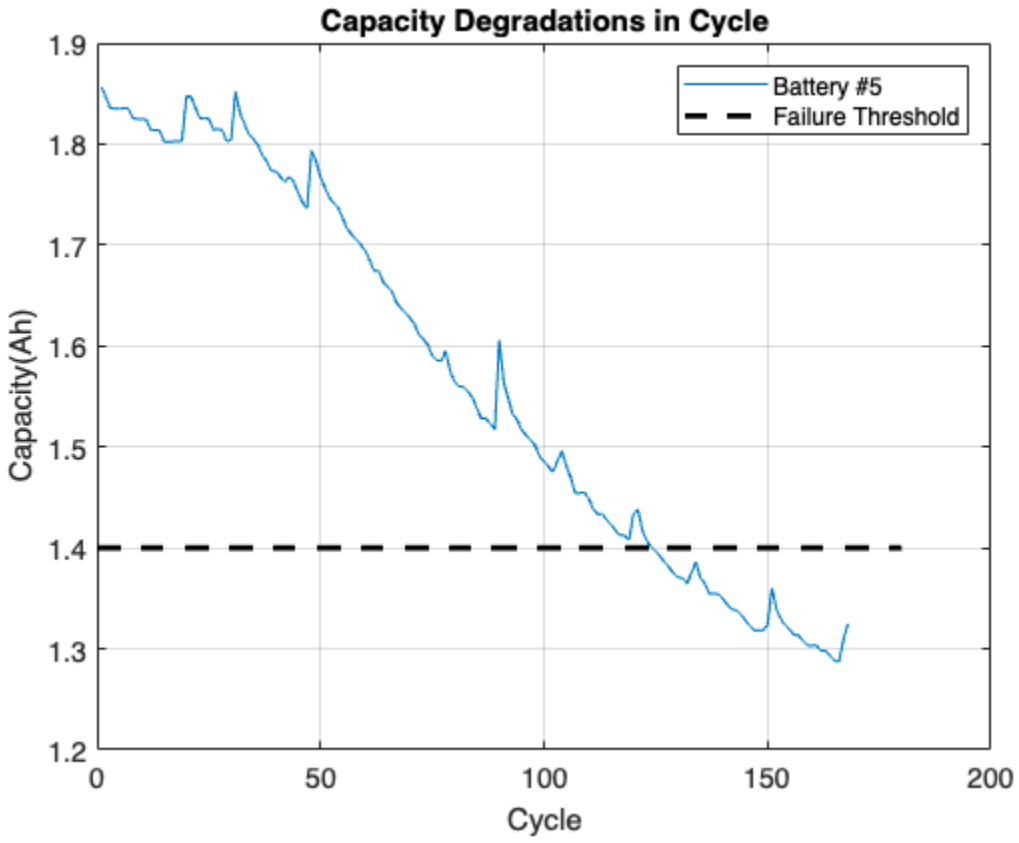
In below figure, we can observe that there are significant changes in charging profiles of V, I and T as a battery ages.

```
clear;
load B0005.mat
FTime = B0005.cycle(1).data.Time/60;
FreshCell_V = B0005.cycle(1).data.Voltage_measured;
FreshCell_I = B0005.cycle(1).data.Current_measured;
FreshCell_T = B0005.cycle(1).data.Temperature_measured;

ATime = B0005.cycle(168).data.Time/60;
AgedCell_V = B0005.cycle(168).data.Voltage_measured;
AgedCell_I = B0005.cycle(168).data.Current_measured;
AgedCell_T = B0005.cycle(168).data.Temperature_measured;

figure(1)
subplot(311)
plot(FTime, FreshCell_V, 'linewidth', 2), hold on, plot(ATime, AgedCell_V, 'r--','linewidth', 2)
hold off, legend('Fresh Cell(1st Cycle)', 'Aged Cell(168th Cycle'), ylabel('Voltage(V)')
ylim([3.5 4.5]), grid on
subplot(312)
plot(FTime, FreshCell_I, 'linewidth', 2), hold on, plot(ATime, AgedCell_I, 'r--', 'linewidth', 2)
hold off, ylabel('Current(A)'), ylim([0 2]), grid on
subplot(313)
plot(FTime, FreshCell_T, 'linewidth', 2), hold on, plot(ATime, AgedCell_T, 'r--', 'linewidth', 2)
hold off, ylabel('Temperature(^oC)'), ylim([22 30]), grid on, xlabel Time(Minute)
```



Eventually, we observe the degradation characteristics per cycle as below figure.

```
cap = extract_discharge(B0005);
figure
plot(cap), hold on
plot(0:180, 1.4*ones(1, 181),'k--','LineWidth', 2)
hold off, grid on
xlabel Cycle, ylabel Capacity(Ah)
legend('Battery #5', 'Failure Threshold')
title('Capacity Degradations in Cycle')
```



Data Preprocessing

Get ready for the training. The inputs of the proposed models in are the extracted features, which are obtained by the uniform sampling of the raw battery data. Specifically, they configure the input matrix as 30-dimensional vectors by concatenating the V, I, T charging profiles, each with 10 samples. The number of samples is chosen to consider the distinct changes in time and the model complexity. In addition, we average the data over sampling interval to prevent oscillation in short time interval.

```
charInput = extract_charge_preprocessing(B0005);
```

Initial capacity for each battery data is provided as belows in the dataset:

```
InitC = 1.86;
```

For better training since it retains the original distribution of data except for a scaling factor and transforms all the data into the range of [0,1]:

```
[xB, yB, ym, yr] = minmax_norm(charInput, InitC, cap);
Train_Input = xB;
Train_Output = yB;
```

- CNN1 : 2 Convolution Layer with filter size [1, 2] and number of filter 10, 5.

```
networkNeeded = feedforwardnet(10);
networkNeeded.trainParam.epochs = 100;
[networkNeeded, tr] = train(networkNeeded, Train_Input', Train_Output', 'useparallel',  'yes');
```

```
layerCNN1 = [
    imageInputLayer([1, 30]);
    convolution2dLayer([1, 2], 10, 'Stride', 1);
    leakyReluLayer
    convolution2dLayer([1, 2], 5, 'Stride', 1);
    leakyReluLayer
    fullyConnectedLayer(1)
    regressionLayer();
    ];
cellx = num2cell(Train_Input', 1)';
cellx = cellfun(@transpose, cellx, 'UniformOutput', false);
cellyB = num2cell(Train_Output);
tbl = table(cellx);
tbl.cellyB = cellyB;

Traintbl = tbl(tr.trainInd, :);
valtbl = tbl(tr.valInd, :);
testtbl = tbl(tr.testInd, :);

options = trainingOptions('adam', ...
    'InitialLearnRate', 0.001, ...
    'MaxEpochs',500, ...
    'MiniBatchSize',50, ...
    'Plots','training-progress', 'ValidationData', valtbl);

netCNN1 = trainNetwork(Traintbl, layerCNN1, options);
```

```
Training on single CPU.
Initializing input data normalization.
```

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch RMSE	Validation RMSE	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:08	0.62	0.64	0.1917	0.2071	0.0010
25	50	00:00:09	0.23	0.27	0.0258	0.0374	0.0010
50	100	00:00:10	0.17	0.19	0.0141	0.0187	0.0010
75	150	00:00:10	0.12	0.14	0.0069	0.0097	0.0010
100	200	00:00:11	0.09	0.12	0.0041	0.0073	0.0010
125	250	00:00:11	0.08	0.12	0.0033	0.0073	0.0010
150	300	00:00:12	0.07	0.12	0.0028	0.0076	0.0010
175	350	00:00:13	0.07	0.13	0.0024	0.0078	0.0010
200	400	00:00:13	0.07	0.13	0.0021	0.0080	0.0010
225	450	00:00:14	0.06	0.13	0.0019	0.0081	0.0010
250	500	00:00:14	0.06	0.13	0.0017	0.0081	0.0010
275	550	00:00:15	0.05	0.13	0.0015	0.0080	0.0010
300	600	00:00:15	0.05	0.13	0.0013	0.0079	0.0010
325	650	00:00:16	0.05	0.13	0.0012	0.0078	0.0010
350	700	00:00:17	0.05	0.12	0.0011	0.0077	0.0010
375	750	00:00:17	0.05	0.12	0.0010	0.0075	0.0010
400	800	00:00:18	0.04	0.12	0.0009	0.0073	0.0010
425	850	00:00:18	0.04	0.12	0.0009	0.0071	0.0010
450	900	00:00:19	0.04	0.12	0.0008	0.0069	0.0010
475	950	00:00:19	0.04	0.12	0.0008	0.0067	0.0010
500	1000	00:00:20	0.04	0.11	0.0007	0.0065	0.0010

```
Training finished: Max epochs completed.
```

- CNN2 :2 Convolution Layer with filter size [1, 2] and number of filter 30, 15.

```
layerCNN2 = [
    imageInputLayer([1, 30]);
    convolution2dLayer([1, 2], 30, 'Stride', 1);
    leakyReluLayer
    convolution2dLayer([1, 2], 15, 'Stride', 1);
    leakyReluLayer
    fullyConnectedLayer(1)
    regressionLayer();
    ];
netCNN2 = trainNetwork(Traintbl, layerCNN2, options);
```

```
Training on single CPU.
Initializing input data normalization.
```

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch RMSE	Validation RMSE	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:06	0.55	0.46	0.1507	0.1063	0.0010
25	50	00:00:07	0.14	0.14	0.0101	0.0102	0.0010
50	100	00:00:07	0.10	0.14	0.0046	0.0093	0.0010
75	150	00:00:08	0.08	0.17	0.0031	0.0148	0.0010
100	200	00:00:09	0.07	0.20	0.0024	0.0193	0.0010
125	250	00:00:09	0.06	0.21	0.0018	0.0216	0.0010
150	300	00:00:10	0.05	0.21	0.0014	0.0226	0.0010
175	350	00:00:10	0.05	0.22	0.0011	0.0232	0.0010
200	400	00:00:11	0.04	0.22	0.0008	0.0237	0.0010
225	450	00:00:11	0.04	0.22	0.0007	0.0241	0.0010
250	500	00:00:12	0.03	0.22	0.0005	0.0244	0.0010
275	550	00:00:12	0.03	0.22	0.0004	0.0248	0.0010
300	600	00:00:13	0.03	0.22	0.0004	0.0253	0.0010
325	650	00:00:13	0.03	0.23	0.0003	0.0257	0.0010
350	700	00:00:14	0.02	0.23	0.0003	0.0258	0.0010
375	750	00:00:14	0.02	0.23	0.0003	0.0257	0.0010
400	800	00:00:15	0.02	0.23	0.0003	0.0256	0.0010
419	838	00:00:15	0.02		0.0003		0.0010

```
Training finished: Stopped manually.
```

Prediciton using each trained model

Make a prediction using trained models

- CNN1 : 2 Convolution Layer with filter size [1, 2] and number of filter 10, 5.

```
cellx = num2cell(Train_Input(tr.testInd, :)', 1)';
cellx = cellfun(@transpose, cellx, 'UniformOutput', false);
tbl = table(cellx);
x_4d = zeros(1, 30, 1, height(tbl));
for i = 1:height(tbl)
    x_4d(:,:,:,i) = tbl.cellx{i};
end
pCNN1 = predict(netCNN1, x_4d);
```

- CNN2 : 2 Convolution Layer with filter size [1, 2] and number of filter 30, 15.

```
pCNN2 = predict(netCNN2, x_4d);
```

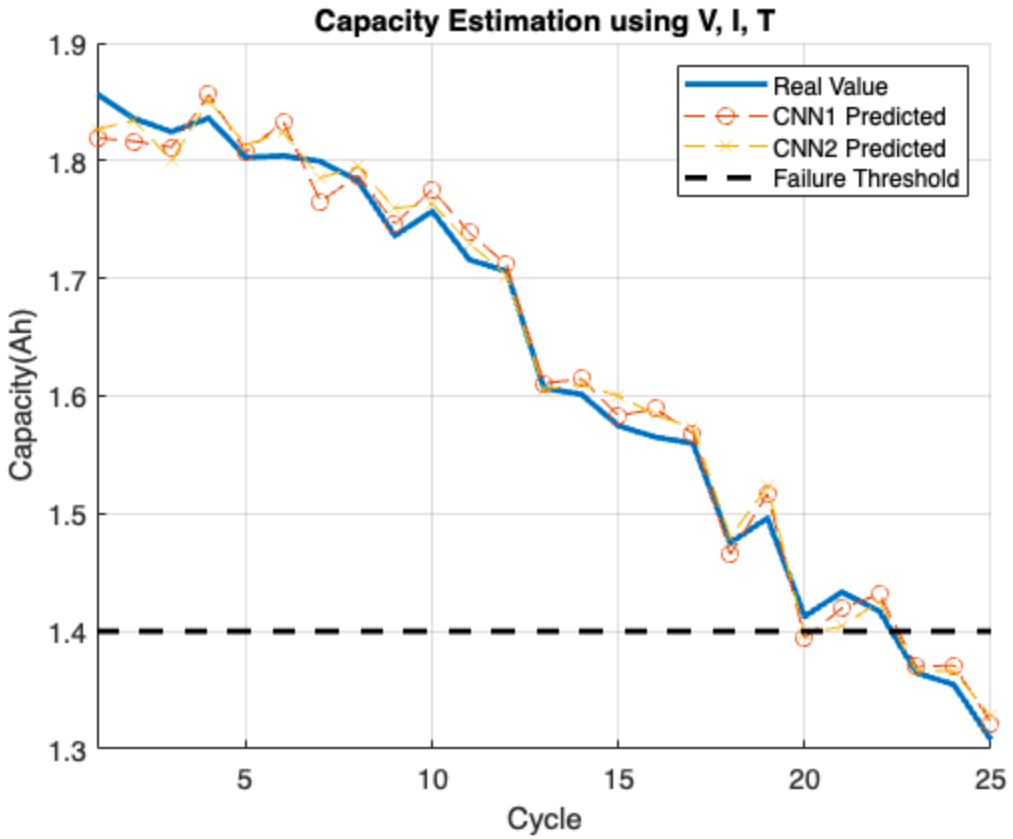
Denormalization for graphical ouput. Multiplying the range of original and add minimum value.

```
Train_Output = Train_Output(tr.testInd, :)*yr + ym;
pCNN1 = pCNN1*yr + ym;
pCNN2 = pCNN2*yr + ym;
```

Result visualization

Visualize the prediction result. Battery capacity estimation based multi-channel charing profiles, using Voltage, Current and temperature.

```
figure, hold on, grid on,
plot(Train_Output, 'linewidth', 2), plot(pCNN1, 'o--'), plot(pCNN2, 'x--')
plot(1:25, 1.4*ones(1, 25),'k--','LineWidth', 2), xlim([1 25])
title(['Capacity Estimation using V, I, T'])
xlabel Cycle, ylabel Capacity(Ah)
legend('Real Value', 'CNN1 Predicted', 'CNN2 Predicted', 'Failure Threshold')
```



extract_charge_preprocessing.m file:

```
function charInput = extract_charge_preprocessing(B)
bcycle = B.cycle;
for i = 1:length(bcycle)-1
    if isequal(bcycle(i).type, 'charge')
        le = mod(length(bcycle(i).data.Voltage_measured), 10);
        vTemp = bcycle(i).data.Voltage_measured(:, 1:end-le);
        vTemp = reshape(vTemp, length(vTemp)/10, []);
        vTemp = mean(vTemp);

        iTemp = bcycle(i).data.Current_measured(:, 1:end-le);
        iTemp = reshape(iTemp, length(iTemp)/10, []);
        iTemp = mean(iTemp);

        tTemp = bcycle(i).data.Temperature_measured(:, 1:end-le);
        tTemp = reshape(tTemp, length(tTemp)/10, []);
        tTemp = mean(tTemp);
        charInput(i, :) = [vTemp, iTemp, tTemp];
    end
end
charInput(~any(charInput, 2), :) = [];
```

extract_discharge.m:

```
function cap = extract_discharge(B)
bcycle = B.cycle;
for i = 1:length(bcycle)
    if isequal(bcycle(i).type, 'discharge')
        cap(i) = bcycle(i).data.Capacity;
    end
end
cap(cap==0) = [];
```

minmax_norm.m:

```
function [xData, yData, ym, yr] = minmax_norm(charInput, InitC, cap)
r = max(charInput) - min(charInput);
xData = (charInput - min(charInput))./r;
comp = length(charInput) - length(cap);
yData = [InitC*ones(comp, 1); cap'];
ym = min(yData);
yr = max(yData) - min(yData);
yData = (yData - ym)/yr;
```