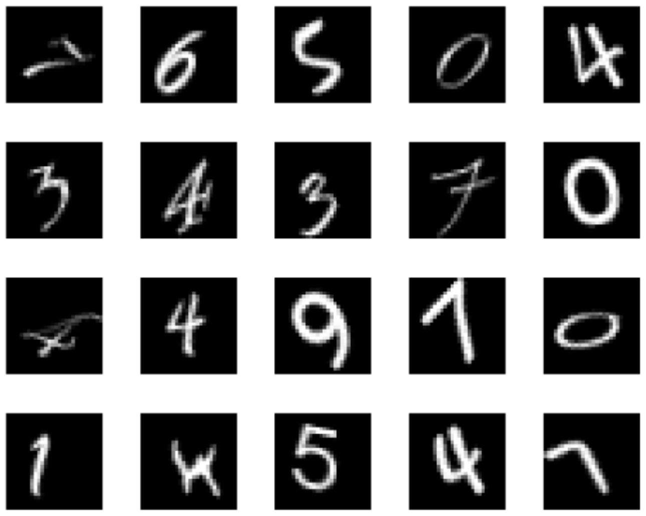
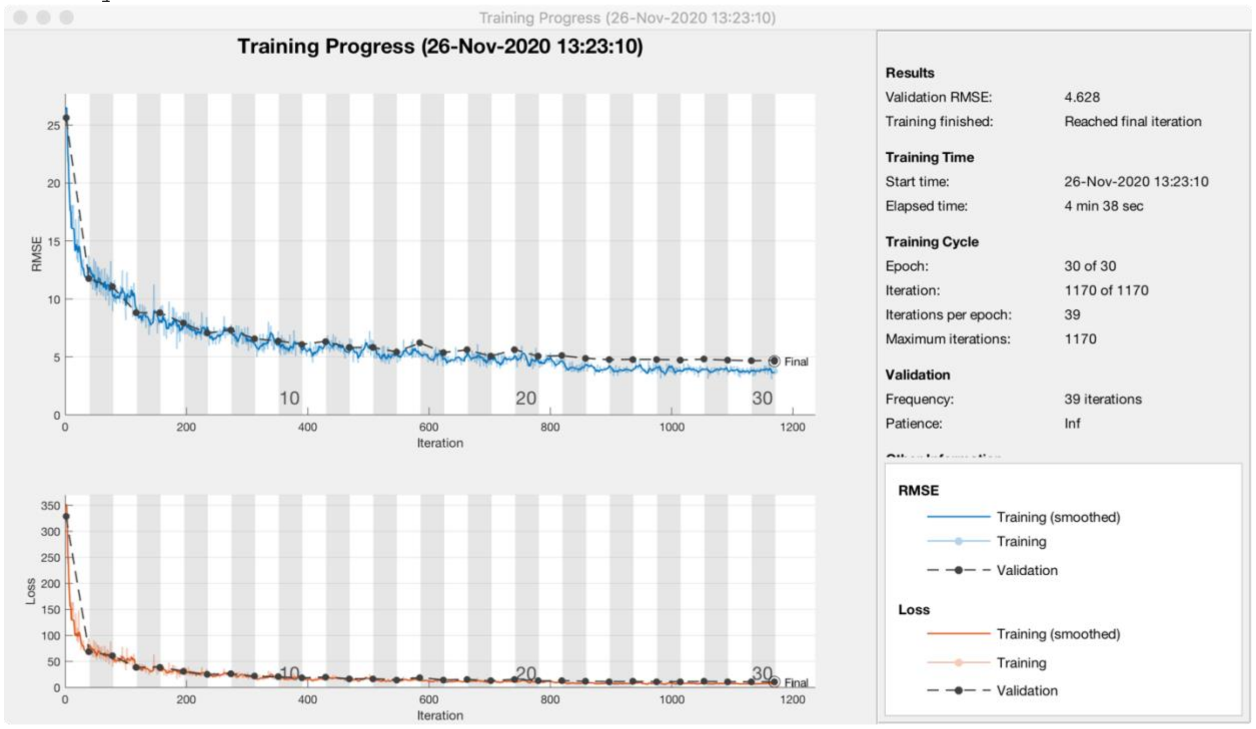
Build a Convolutional Neural Networks

Conclusion: The failed prediction samples are shown above in the "`%list the five worst predicted samples`" part. As we can see, there are three '7', a '2' and a '8'. I think the most important reason is that it looks so much like other numbers. The '7' looks like '1', so the model may predicted its rotation as '1', that is definitely a bad prediction, As well as '2'. '2' looks seem like a rotating '5'. And for the sample '8', since it is a symmetric number, it is difficult to recognize which end supposes to be above and which end supposes to be below.

```
%%
%Load Data
[XTrain,~,YTrain] = digitTrain4DArrayData;
[XValidation,~,YValidation] = digitTest4DArrayData;
numTrainImages = numel(YTrain);
figure
for i = 1:numel(idx)
    subplot(4,5,i)
    imshow(XTrain(:,:,:,idx(i)))
end
```



```
%%
%Check data normalization
figure
histogram(YTrain)
axis tight
ylabel('Counts')
xlabel('Rotation Angle')
%%
%Create newwork layers
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer
    averagePooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer
    averagePooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer
    dropoutLayer(0.2)
    fullyConnectedLayer(1)
    regressionLayer];
%%
%Train network
miniBatchSize  = 128;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',30, ...
    'InitialLearnRate',1e-3, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',20, ...
    'Shuffle','every-epoch', ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',validationFrequency, ...
    'Plots','training-progress', ...
    'Verbose',false);
net = trainNetwork(XTrain,YTrain,layers,options);
net.Layers
```


Training Progress (26-Nov-2020 13:23:10)
Training Progress (26-Nov-2020 13:23:10)
RMSE
Iteration
Loss
Iteration
Final
Final

Results

Validation RMSE:	4.628
Training finished:	Reached final iteration

Training Time

Start time:	26-Nov-2020 13:23:10
Elapsed time:	4 min 38 sec

Training Cycle

Epoch:	30 of 30
Iteration:	1170 of 1170
Iterations per epoch:	39
Maximum iterations:	1170

Validation

Frequency:	39 iterations
Patience:	Inf

RMSE
- Training (smoothed)
- Training
- Validation

Loss
- Training (smoothed)
- Training
- Validation

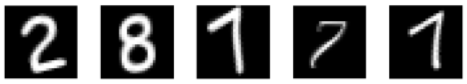
```
%%
%Test network
YPredicted = predict(net,XValidation);
```

```
predictionError = YValidation - YPredicted;

thr = 10;
numCorrect = sum(abs(predictionError) < thr);
numValidationImages = numel(YValidation);

accuracy = numCorrect/numValidationImages
squares = predictionError.^2;
rmse = sqrt(mean(squares))
```

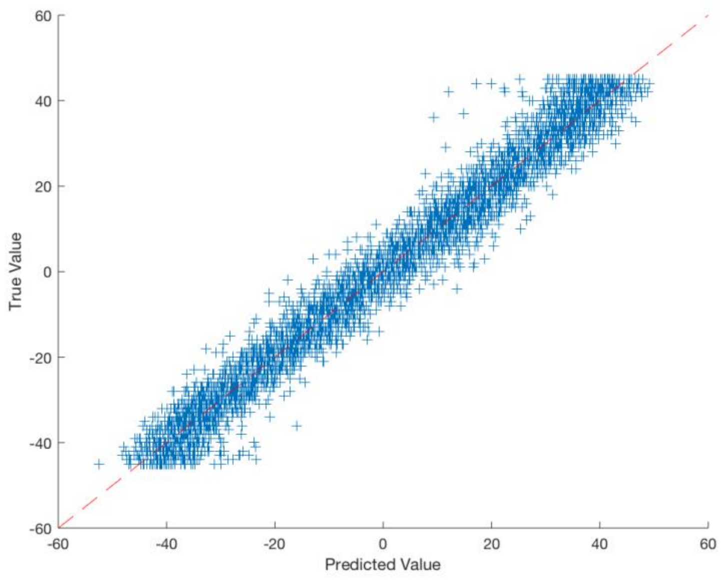
```
accuracy =

    0.9674


rmse =

  single

    4.6280
```

```
%%
%list the five worst predicted samples
[B,I] = maxk(predictionError,5)
figure
for i = 1:numel(I)
    subplot(1,5,i)
    imshow(XValidation(:,:,:,idx(i)))
end
```

```
B =

  5×1 single column vector

   25.7697
   23.7743
   23.2417
   21.9588
   20.8117
```



```
%%
%Visualize predictions
figure
scatter(YPredicted,YValidation,'+')
xlabel("Predicted Value")
ylabel("True Value")

hold on
plot([-60 60], [-60 60],'r--')
```



Comparison with Random Learning

Conclusion: The accuracy of random learning is lower than the gradient descent, and the time is much more. Since the loss decreasing of random learning don't have a direction, it is totally random. Thus, the gradient descent is much better than random learning.

```
%%
%Load Data
[XTrain,~,YTrain] = digitTrain4DArrayData;
[XValidation,~,YValidation] = digitTest4DArrayData;
dsTX = arrayDatastore(XTrain,'IterationDimension',4);
dsTY = arrayDatastore(YTrain);
dsTrain = combine(dsTX,dsTY);
numTrainImages = numel(YTrain);
figure
idx = randperm(numTrainImages,20);
for i = 1:numel(idx)
    subplot(4,5,i)
    imshow(XTrain(:,:,:,idx(i)))
end
classes = categories(YTrain);
numClasses = numel(classes);
%%

%Check data normalization
figure
histogram(YTrain)
axis tight
ylabel('Counts')
xlabel('Rotation Angle')
%%
%Create newwork layers
layers = [
    imageInputLayer([28 28 1],'Normalization','none','Name','input')
    convolution2dLayer(3,8,'Padding','same','Name','c1')
    batchNormalizationLayer('Name','bn1')
    reluLayer('Name','rl1')
    averagePooling2dLayer(2,'Stride',2,'Name','p1')
    convolution2dLayer(3,16,'Padding','same','Name','c2')
    batchNormalizationLayer('Name','bn2')
    reluLayer('Name','rl2')
    averagePooling2dLayer(2,'Stride',2,'Name','p2')
    convolution2dLayer(3,32,'Padding','same','Name','c3')
    batchNormalizationLayer('Name','bn3')
    reluLayer('Name','rl3')
    convolution2dLayer(3,32,'Padding','same','Name','c4')
    batchNormalizationLayer('Name','bn4')
    reluLayer('Name','rl4')
    dropoutLayer(0.2,'Name','dl1')
    fullyConnectedLayer(1,'Name','cc1')];
lgraph = layerGraph(layers);

dlnet = dlnetwork(lgraph)
%%
%Train network
numEpochs = 30;
miniBatchSize = 128;
LearnRate = 1e-3;
learnRateDropPeriod = 20;
learnRateDropFactor = 0.1;

mbq = minibatchqueue(dsTrain,...
    'MiniBatchSize',miniBatchSize,...
    'MiniBatchFcn',@preprocessMiniBatch,...
    'MiniBatchFormat',{'SSCB',''});

figure
lineLossTrain = animatedline('Color',[0.85 0.325 0.098]);
ylim([0 inf])
xlabel("Iteration")
ylabel("Loss")
grid on

velocity = [];

iteration = 0;
start = tic;

% Loop over epochs.
for epoch = 1:numEpochs
    % Shuffle data.
    shuffle(mbq);

    % Loop over mini-batches.
    while hasdata(mbq)
        iteration = iteration + 1;

        % Read mini-batch of data.
        [dlX, dlY] = next(mbq);
        % Evaluate the model gradients, state, and loss using dlfeval and the
        % modelGradients function and update the network state.
        [gradients,state,loss] = dlfeval(@modelGradients,dlnet,dlX,dlY);
        dlnet.State = state;

        % Determine learning rate for time-based decay learning rate
schedule.
        if mod(epoch,learnRateDropPeriod) == 0
            learnRate = learnRate * learnRateDropFactor;
        end

        % Update the network parameters using the SGDM optimizer.
        dlnet = dlupdate(randomlearningf,dlnet)

        % Display the training progress.
        D = duration(0,0,toc(start),'Format','hh:mm:ss');
        addpoints(lineLossTrain,iteration,loss)
        title("Epoch: " + epoch + ", Elapsed: " + string(D))
        drawnow
    end
end

net.Layers
%%
%Test network
YPredicted = predict(net,XValidation);
predictionError = YValidation - YPredicted;

thr = 10;
numCorrect = sum(abs(predictionError) < thr);
numValidationImages = numel(YValidation);

accuracy = numCorrect/numValidationImages
squares = predictionError.^2;
rmse = sqrt(mean(squares))
%%
%list the five worst predicted samples
[B,I] = maxk(predictionError,5)
figure
for i = 1:numel(I)
    subplot(1,5,i)
    imshow(XValidation(:,:,:,idx(i)))
end
```

```
%%
%Visualize predictions
figure
scatter(YPredicted,YValidation,'+')
xlabel("Predicted Value")
ylabel("True Value")

hold on
plot([-60 60], [-60 60],'r--')

randomlearningf.m
function weight = randomlearningf(weight,loss)
    weightnew = -5 + (5+5)*rand;
    prevalue = weight * dlX
    losstry = crossentropy(prevalue, dlY)
    if losstry < loss
        loss = losstry
        weight = weightnew
    end
end
```

```
dlnet =

  dlnetwork with properties:

         Layers: [17×1 nnet.cnn.layer.Layer]
    Connections: [16×2 table]
     Learnables: [18×3 table]
          State: [8×3 table]
```

Recurrent Neural Networks

$$h(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

$$U = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad W = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad b_h = \begin{pmatrix} -0.5 \\ -1.5 \\ -2.5 \end{pmatrix} \qquad v = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \qquad b_y = -0.5$$

LSTM Gradient

a/ $\overline{h^{(t)}} = \overline{g^{(t+1)}} (1 - \tanh^2 (W_{gx} X^{(t+1)} + W_{gh} h^{(t)})) \cdot W_{gh}$

$+ \overline{i^{(t+1)}} \sigma' (W_{ix} X^{(t+1)} + W_{ih} h^{(t)}) \cdot W_{ih}$

$+ \overline{f^{(t+1)}} \sigma' (W_{fx} X^{(t+1)} + W_{fh} h^{(t)}) \cdot W_{fh}$

$+ \overline{O^{(t+1)}} \sigma' (W_{ox} X^{(t+1)} + W_{ox} h^{(t)}) \cdot W_{oh}$

$\overline{C^{(t)}} = \overline{C^{(t+1)}} f^{(t+1)} + \overline{h^{(t)}} O^{(t)} (1 - \tanh^2 (C^{(t)}))$

$\overline{g^{(t)}} = \overline{C^{(t)}} i^{(t)}$ $\qquad$ $\overline{O^{(t)}} = \overline{h^{(t)}} \tanh(C^{(t)})$

$\overline{f^{(t)}} = \overline{C^{(t)}} C^{(t-1)}$ $\qquad$ $\overline{i^{(t)}} = \overline{C^{(t)}} g^{(t)}$

b/ $\overline{W_{ix}} = \sum_t \overline{f^{(t)}} \sigma' (W_{ix} X^{(t)} + W_{ih} h^{(t-1)}) \cdot X^{(t)}$