# ME5411 Robot Vision & AI

# CA – AY22/23 Semester 2


**Computing Project**

**Made By Group 02:**

**Dong Haoyu (A0268359L)**

**Kent Alvin Nugroho (A0268307Y)**

**Lu Xianglong (A0268486J)**

**Department of Mechanical Engineering**

**National University of Singapore**

**Academic Year 2022/2023 Semester 2**

## Content

# Chapter 1
# Introduction

## I. Introduction to the Problem

Nowadays, we can see that a lot of people like to have a digital image rather than physical film or print image because they can store this digital image into their storage, they can share this digital image via online and they can easily edit the digital image such as adjusting the contrast of image, rotating the image, sizing the image, etc. As we know, we probably can get several problems when we retrieve a digital image. The most common issues are damaged files and poor quality of the digital image.

The damaged file may be occurred because there is an error when transferring or storing this image file and the quality of the picture may be poor due to factors such as low resolution, imbalance of brightness, noise, and blur [1]. These factors sometimes make people get difficulty to understand what useful information come from the image. Because of that, we need to do some image processing which is used to enhance and repair the image quality, remove of noise, sharp the edge of the object so that we can extract useful information from that image.

After we can get a proper quality of the image, we can get useful information from identify every object in the image. To identify this object, we can use segmentation techniques start from detecting the outline/edge of the character in the image and then we can determine the connectivity of these character whether the object have a connectivity with their neighbourhood pixel or not. The last thing we need to do is give some label to each of segmented object so we can clearly identify the difference between every object in the image.

After this image processing, we can apply a deep learning algorithm such as CNN (Convolutional Neural Networks) which are used for analysing the object/character in the image. So, the final objective of this deep learning algorithm is it can give output from the last layer to classify/recognize each character in the image. This method needs to have train using large datasets and during this training, is required to adjust the value of weight and bias to minimize the error between the predicted output and actual output [2]. After completing the training, this algorithm can be used to analyse new image by extracting features in the image and providing accurate analysis from that.

From this computing project, we have an original image with name of file “charact2.bmp” which contain a BMP image of a label on a microchip as shown in the picture 1.


Picture 1. Original Image which Shows a Label on a Microchip

From this original image, we can see clearly that there is three mains degradation/defect in this image. First, we can see there is a lot of random speckles or white dots in the image which contribute as noise in the image. Second, the image has a contrast/brightness which is not spread equally from low to high gray scale value. Third, it is difficult to determine the edge in some characters due to not sharp or blurry in the outline of the character. This edge should be determined clearly to prevent the object connected to each other so that each object can be segmented properly. Some of degradation in the image can be solved through image processing techniques which will be follow from the list of performing task in this project.

## II. List of Performing Task

In this computing project, there is a list of performing task which should be followed it, namely:

1. Display the original image on screen.
2. Implement and apply a 3x3 averaging mask and a 3x3 rotating mask on the image. Compare and comment on the results of the respective image smoothing methods.
3. Create an image which is a sub-image of the original image comprising the line: HD44780A00.
4. Create a binary image from Step 2 using thresholding.
5. Determine the outline(s) of characters of the image.
6. Segment the image to separate and label the different characters.
7. Divide the dataset which contained in the file *p_dataset_26.zip* into two portions: (a) a 75% portion that will be used as the training set, and (b) the remaining 25% portion as the validation (testing) set. Use these two sub-datasets to complete the following three sub-tasks:

   *Sub-task 1:* Create a CNN to classify each character in the second line in Image 1.

*Sub-task 2*: Create a classification system, using a non-CNN based method to classify each character in the second line in Image 1.

*Sub-task 3*: Report the results obtained from Sub-task 1 and Sub-task 2 and compare the effectiveness and efficiency of the two approaches. Provide your own explanation on any differences in the results between these two approaches.

8. Throughout step 7, also experiment with pre-processing of the data (e.g., padding/resizing input images) as well as with hyperparameter tuning. In your report, discuss how sensitive your approach is to these changes.

## III. Objective of Computing Project

The goal and objective for this computing project:

1. Applying image processing techniques to improve image quality such as reducing the noise, smoothing the image, and creating a sub-image so the object in the image can be seen clearly.
2. To identify the objects (Extracting feature) using segmentation techniques such as converting to binary, creating the outline of characters, and labelling each object so that can be useful for applying deep learning algorithm (CNN and non-CNN method) for further analysis.
3. To create a classification system using CNN and non-CNN method to classify each character in the original image.
4. To compare the effectiveness and efficiency of the two approaches in terms of classifying each character in the image.

# Chapter 2
# Description of Algorithm and Flow Chart

In this chapter, we will describe the algorithm and flow chart that we use in this computing project. The algorithm that we use to do image processing computation is shown below.

Step 1 : Load the image into variable (in this project the first variable is called “inputim”) as matrix form using function ‘imread’.

Step 2 : Convert the original image to grayscale image using function ‘rgb2gray’. Converting a colour image to grayscale can simplify the algorithm used for image processing on the image. Then, display the image on screen using function ‘imshow’.

Step 3 : Apply 3x3 averaging mask and 3x3 rotating mask on the image to smooth the image and help to reduce noise. To get the best result, we try to use different size of mask and find the minimum dispersion. This filter mask can be implemented in the image using ‘convn’ function and the mask as parameter input to this function. In this step, we also compare the result after applying the mask with other pre-processing technique to reduce random speckles noise using morphological filter. This image filtering use structural element to reduce noise. This filter can be made from function ‘strel’ to create structural element and ‘imopen’ as function to apply this filter into image.

Step 4 : Create a new image from sub-image of the original image that showing the line HD44780A00.

Step 5 : Before doing segmentation part, convert the sub-image into binary image using threshold value. To get the correct threshold value, apply trial and error to this value to get the perfect binary image. The binary image can be made from a function ‘imbinarize’.

Step 6 : Find the outline/edge of the character in binary image using ‘edge’ function with ‘log’ method.

Step 9 : Do segmentation technique to the binary image to separate each character and give label to different characters. Separation of character can be done using function ‘bwconncomp’ to count the number of separated objects. Each separated object will be given a different label using function ‘labelmatrix’ and convert it to color label using ‘label2rgb’ function.

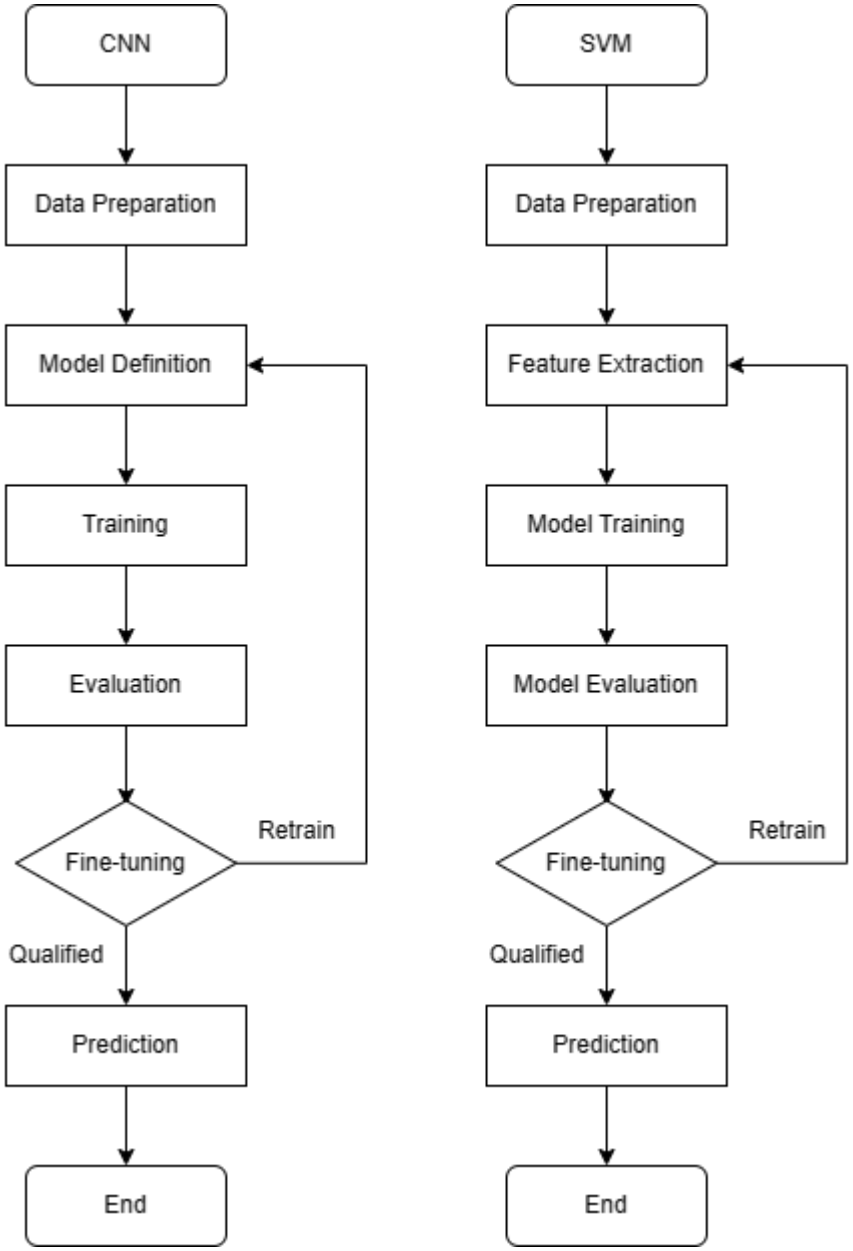
Step 10 : Create a classification system for CNN and non-CNN based method (for non-CNN, we use SVM method) from dataset. This system will be trained by 75% portion of dataset and 25% portion of dataset will be used to do validation test.

Step 11 : Classify the region of interest (the segmented object in step 9) using a classification system that have been made from step 10.

The flow chart of these algorithms can be shown below in picture 2.


Picture 2. Flow Chart of Image Processing Task

The flow chart which shows the step or process of CNN and non-CNN based implementation is shown in the picture 3.



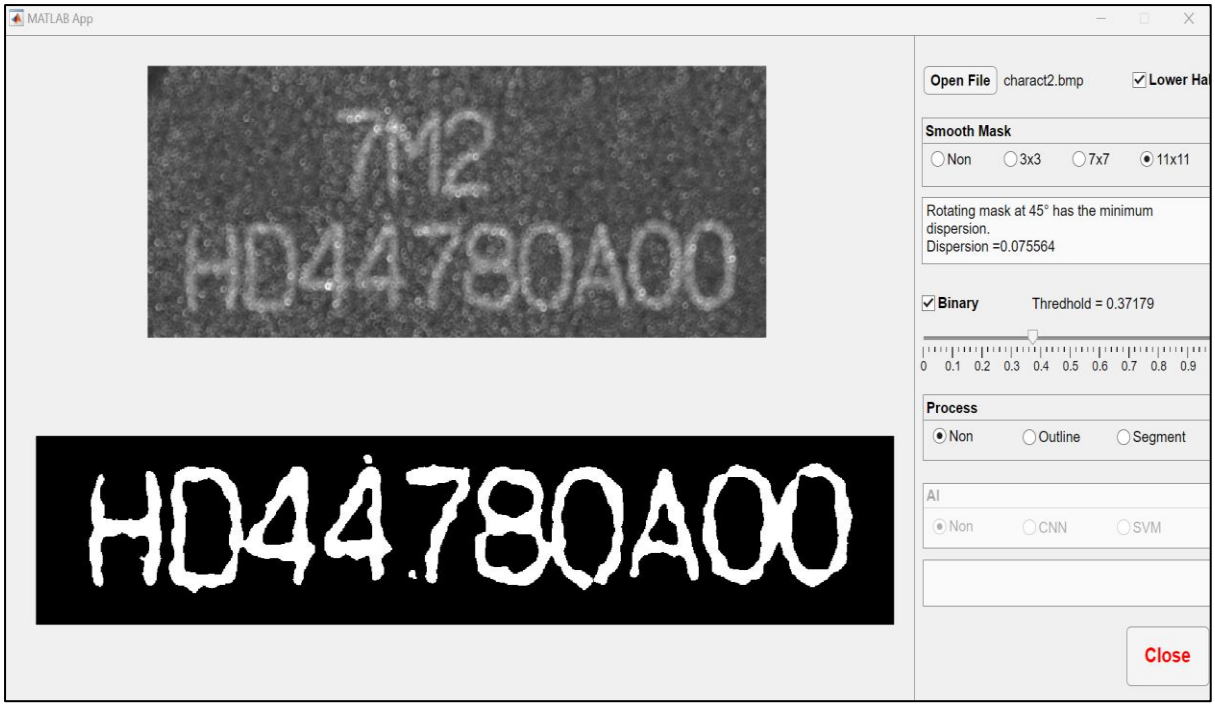

Picture 3. Flow chart of CNN and SVM

# Chapter 3
# Explanation and Analysis of the Image Processing Stage

According to the algorithm and the flow chart that have been made in previous chapter, in this chapter we will explain and describe the method we use for every stage in image processing task. First, we will introduce how to use User Graphic Interface which have been made from programming code in MATLAB. This GUI can be used to see every result from image processing task. After running the code, the GUI will appear as shown in picture 4.


Picture 4. GUI that will show every result from image processing task.

In this GUI, there are two panel which upper panel show the input image as original image before applying image processing task and lower panel show the result after applying image processing task. On the right side, there are several tools that can be used to do image processing. ‘Open File’ tool is used to import the original image into upper panel. ‘Smoothing mask’, ‘Lower Half’, ‘Binary’, ‘Outline’, and ‘Segment’ tools are used to apply image processing task into input image on the upper panel. The ‘Threshold’ drag button is used to provide threshold value into binary image so that we can try to adjust this threshold to find a good binary image. After applying each processing task, the result will be shown in the lower panel. The rest tools which are CNN and SVM are used to apply deep learning algorithm into input segmented objects in order to analysis, recognize, and predict the output class of each object. The example of input and result of image processing task is shown in the picture 5 below. After we get the result of the binary image, the next of image processing task can be done by pressing another tool such as ‘outline’ tool.




Picture 5. GUI show input image (sub-image) on upper panel and result image (binary image) on lower panel.

Now, we will explain every step of image processing task in this project. First, we need to load the original image into MATLAB workspace using ‘imread’ function. This function is used to read an image file and create a matrix which contain the pixel values of the image. After we create this matrix, we can display the original image on screen using ‘imshow’ function and this matrix as data input. The code and original image on screen shown in the picture 6.

```
% Button pushed function: OpenFileButton
function OpenFileButtonPushed(app, event)
    [filename,pathname] = uigetfile(["*.bmp";"*.jpg";"*.jpeg";"*.png"],"Choose an image");
    inputim = imread(fullfile(pathname,filename));
```


Picture 6. Code and the result of the original image display on screen

Next, we convert the original image into grayscale image using ‘rgb2gray’ function. The reason why we convert this original image to grayscale image because it is easier to apply image processing function which can simplify the image data from colour image into grayscale image [3]. If the image data become simple, the filtering and segmentation will be performed more quickly in grayscale image. We also need to convert type of image into double using ‘im2double’ function because image processing function like filtering require the image data to be presented as floating-point numbers in the range of 0 to 1. The code and the result grayscale image are shown in the picture 7.

```
inputim = im2double(rgb2gray(inputim));
app.img_processing = inputim;
app.img_Original = inputim;
imshow(inputim,"Parent",app.UIAxes)
```


Picture 7. The code and the result show grayscale image

From grayscale image, we can do smoothing method using 3x3 averaging mask and 3x3 rotating mask. The 3x3 averaging mask can be used to smooth out the image by helping to reduce noise but the sharp edge of character become blur. The 3x3 rotating mask also can be used to smooth out the image without giving a blur in the edge of the object. This is happened because this filter can adjust its orientation to match the orientation of features in the image so that it can keep more detail and edge information [4]. To select the better mask between averaging and rotating mask, it can be chosen from the calculation of the minimum dispersion. Dispersion is the variability of the pixel value within a certain region of the image and by minimizing the dispersion, image can become smoother and more homogenous appearance [4]. Therefore, applying this mask can help to reduce the variance and make the image smoother. The result of applying 3x3 averaging and rotating mask are shown in the picture 8. From this picture, we can see that the rotating mask has smaller minimum dispersion than averaging mask so that the rotating mask is selected to smooth the image. It can be seen that the minimum dispersion is 0.089 and the angle that provide the minimum dispersion is $0^{o}$.

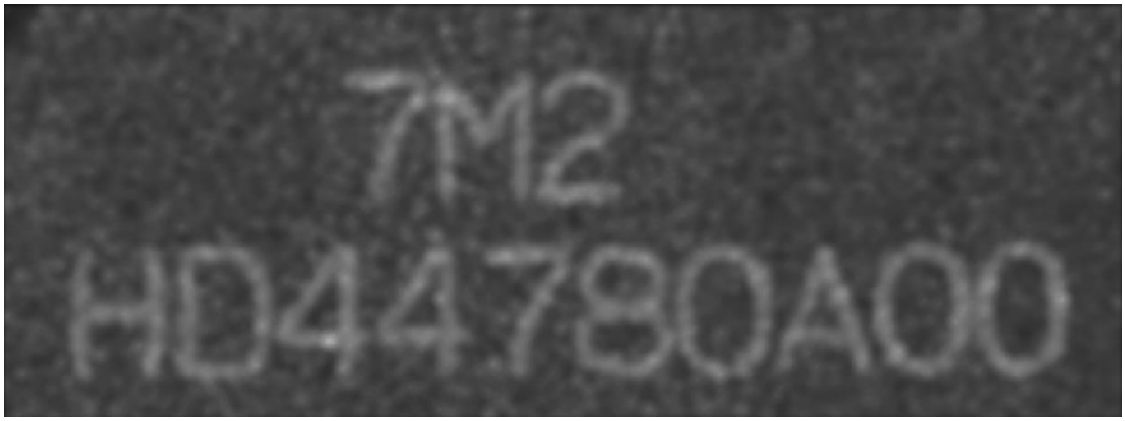

**Smooth Mask**

○ Non ◉ 3x3 ○ 7x7 ○ 11x11

Rotating mask at 0° has the minimum dispersion
Dispersion =0.089992

Picture 8. The result image after applying 3x3 averaging and rotating mask

Furthermore, we can see that there is no improvement at all in the image when using 3x3 rotating mask. There are still a lot of noise in the background of the image. Because of that, we try to use another size of mask which may give more improvement. We choose the 11x11 size of mask as this size give a significant improvement to the image and has the smallest value of minimum dispersion than other. The result of applying 11x11 averaging and rotating mask are shown in the picture 9.




**Smooth Mask**

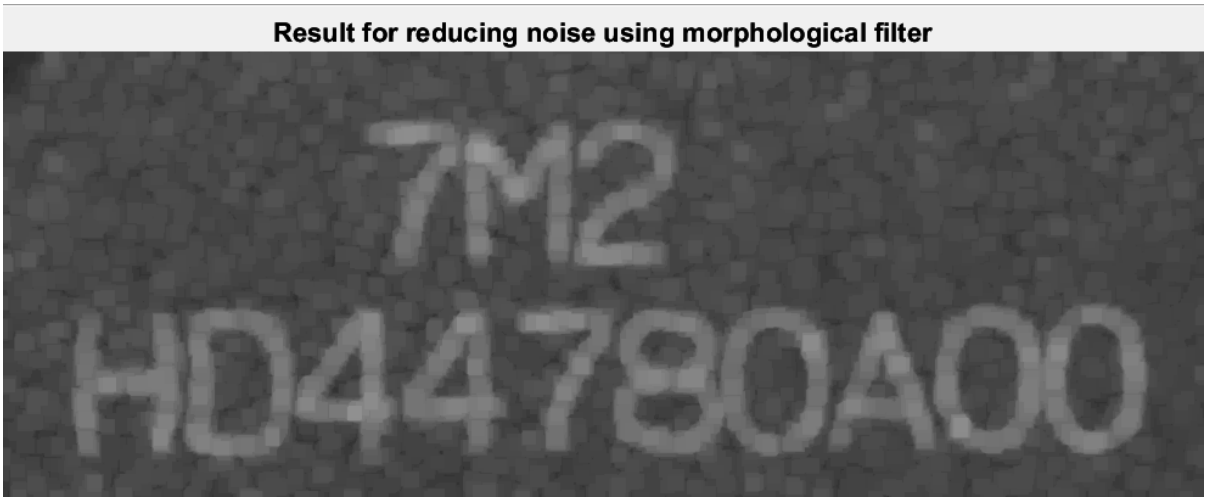
○ Non ○ 3x3 ○ 7x7 ◉ 11x11

Rotating mask at 45° has the minimum dispersion.
Dispersion =0.065752

Picture 9. The result image after applying 11x11 averaging and rotating mask

From the picture 9, we can see that there is an improvement in the picture where the noise is reduced in the background. However, applying larger size of mask tend to blur the image rather than using smaller size of mask. Even though the image gets blurred, the effect of applying averaging mask can sharper the edge of each object. After smoothing the image, we try to use another method to suppress the rest of noise using morphological filter.

The morphological filter is one of the pre-processing techniques that use the shape of structural element to modify or extract information from the image [5]. This filter commonly used to suppress noise in image so that it can be easier to extract features from region of the interest (object in the image). Before using this filter, first we need to define the shape and size of structural element (small shape of pixel used to interact with another pixel in image) that can be used with this filter. In this project, we choose to use structural element with 'square' shape and size of 11 which will give the best result to supress noise. To apply the filter, we use the 'imopen' function which is used to remove small noise while preserving the overall shape of the object. The code and result of applying this morphological filter shown in the picture 10.




```
% use structure element to reduce noise in background before convert it to
% binary image
SE = strel('square',11);
image6 = imopen(image1,SE);
figure; imshow(image6); title('Result for reducing noise using morphological filter');
```

Picture 10. The code and result for reducing noise using morphological filter

From this result, we can see that the noise on the background have been suppressed and the edges of object can be seen clearly because there is no blurring effect on the image. The next step is to create a sub-image from original image which show the middle line of the object. This sub-image can be created from geometric transformation technique and in the MATLAB, we can use 'imcrop' function to crop the original image or can be done by inserting the size of sub-image into the matrix of input image. The result of sub-image is shown in the picture 11.


```
app.img_processing = app.img_Original;
if app.LowerHalfCheckBox.Value == 0
    app.img_processing = app.img_processing;
else
    imheight = size(app.img_processing,1);
    app.img_processing = app.img_processing(round(imheight/2):imheight,:);
end
```

Picture 11. The code and the sub-image showing middle line of the object

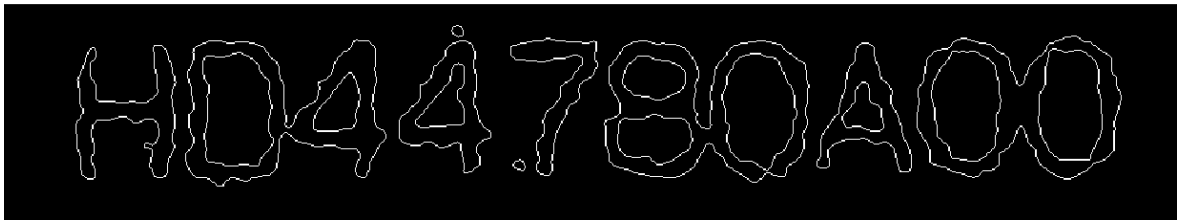
After we have this sub-image, we can start the segmentation part by converting this sub-image into binary image. The reason to have a binary image is that easier for our eyes to extract the features of the image especially the region of the interest (object in the image) [6]. To convert sub-image into binary image can be used 'imbinarize' function and then determine the proper value for threshold which will result in clearly seen of the object. In this project, we have tried to adjust the threshold value and we find that threshold with 0.37 has the best result. The result image after converting into binary image are shown in the picture 12.


```
if app.BinaryCheckBox.Value ~= 0
    threshold = app.Slider.Value;
    app.ThredholdLabel.Text = ['Thredhold = ',num2str(threshold)];
    app.img_processing = imbinarize(app.img_processing,threshold);
end
```

Picture 12. The code and result after converting into binary image

From picture 12, it can be seen that the first segmentation part is not fully successful because there is still an object which connect/overlap with another object (object D & 4, 8 & 0 and 0 & 0). Due to this, further processing technique need to be done to separate each object into single character. Moreover, there is still have two dots of white pixel as noise in the binary image because not all noise can be removed when using a filter mask to smooth the image. The next segmentation process is to create the outline (showing only the edge of each object) using gradient operator function 'edge' in MATLAB. To detect the edge of the object, we use Laplacian of Gaussian (LoG) method for edge detection. We choose this method rather than

using Sobel and Canny operator because this operator can provide excellent localization of edges as it uses the second derivative of the Gaussian function to detect zero crossing in the image [7]. This operator also has better performance in terms of edge detection accuracy, noise robustness, and computational efficiency. The picture 13 is shown about the outline of each character after using gradient operator function.



```
if app.Process == 1
    app.img_processing = edge(app.img_processing,"log");
elseif app.Process == 2
    delete('segmented_chars/*.png')
```

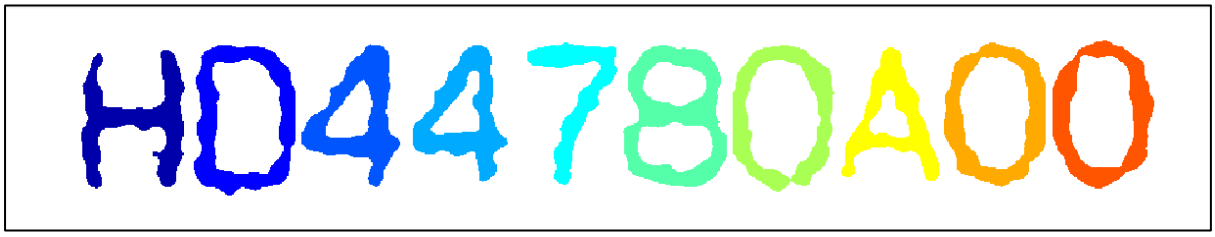
Picture 13. The code and the image showing outline of each characters

Finally, we can start to label each character so that it can be distinguish clearly the difference between each object from different label. Before labelling each character, we need to do some processing to cut half the remaining connecting characters. We need to calculate the cluster width and then determine the threshold of cluster width to set the value of 0 pixel in the middle column (This step is similar to cut in half horizontally of the connected character). In this project we find that the threshold is 130 can set up 0 pixel in the middle column and cluster width below 30 is used to remove small object (remaining noise in binary image). The code after doing this process is shown in the picture 14.

```
elseif app.Process == 2
    delete('segmented_chars/*.png')
    imageSeperated = app.img_processing;
    CC = bwconncomp(app.img_processing);
    raws = size(app.img_processing,1);
    number = CC.NumObjects;
    for i=1:number
        cluster = cell2mat(CC.PixelIdxList(i));
        startPoint = cluster(1);
        endPoint = cluster(end);
        startColumn = round(startPoint/raws)-1;
        endColumn = round(endPoint/raws)-1;
        clusterWidth = endColumn-startColumn;
        middleColumn = startColumn+round(clusterWidth/2);
        if clusterWidth>130
            imageSeperated(:,middleColumn:middleColumn+4) = 0;
        elseif clusterWidth<30
            imageSeperated(CC.PixelIdxList{i}) = 0;
        end
    end
```

Picture 14. The code used to separate each character

After each character successfully separated, we can give a label into each character with 'labelmatrix' function to create a matrix which assign a unique label to each component. Before that we can use 'bwconncomp' function to find the number of the object in the foreground image. Then we can add a different colour for each character using function 'label2rgb'. Finally, the last segmentation process is shown in the picture 15.




```
CC = bwconncomp(imageSeperated);
L = labelmatrix(CC);
app.img_processing = label2rgb(L);
```

Picture 15. The code and the image show the final result of segmentation process

After segmentation process, the next step is build an algorithm for deep learning using CNN and non-CNN based method which in this project we use SVM. Before doing that, we need to convert each character into different file of picture so that after training and validation process of deep learning algorithm, this method can be used to classify and recognize each character as input new data set. The code in the picture 16 shows about how to assign every character into different file for further analysis.

```
% create a binary image containing only the i-th connected component
binaryImage = false(size(imageSeperated));
binaryImage(CC.PixelIdxList{i}) = true;
% use the connected component bounding box to crop the image
stats = regionprops(binaryImage, 'BoundingBox');
bbox = stats.BoundingBox;
croppedImage = imcrop(binaryImage, bbox);
% pad the cropped image with black border to make it 128x128
[height, width] = size(croppedImage);
if height > width
    border = floor((height-width)/2);
    resizedImage = imresize(croppedImage, [128 NaN]);
    paddedImage = padarray(resizedImage, [0 border], 0, 'both');
    paddedImage = padarray(paddedImage, [0 128-min(size(paddedImage,2),128)], 0, 'post');
else
    border = floor((width-height)/2);
    resizedImage = imresize(croppedImage, [NaN 128]);
    paddedImage = padarray(resizedImage, [border 0], 0, 'both');
    paddedImage = padarray(paddedImage, [128-min(size(paddedImage,1),128) 0], 0, 'post');
end
% save the padded image as a separate file in the new directory
imwrite(paddedImage, fullfile('segmented_chars', ['char_', num2str(i), '.png']));
```

Picture 16. The code showing how to assign every character into different file

The first step to build the CNN algorithm is loading the dataset. The dataset which has been stored in a folder newdataset can be loaded into MATLAB. This data set will be read by the 'ImageDatastore' function which read images from the folder and stores them into image datastore. These images will be organized in subfolder based on each label. Next, we need to separate this dataset into training and validation part using 'splitEachLabel' function. This function will separate each label into two parts with a 75 % for training part and 25 % for validation part. The number of classes in training dataset can be determined using 'numel' function and 'categories' function which will give result 7 number classes. We can count the number of pictures in training and validation dataset under each label using 'countEachLabel' function. The process to load dataset into MATLAB can be shown in the picture 17.

```
%% load data
digitDatasetPath = fullfile('./', '/newdataset/');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');

% Seprate trainning and validation dataset
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.75);

% Label Number
numClasses = numel(categories(imdsTrain.Labels))

% Count picture number of Trainning Dataset under each lable
TrainLabelCounter = countEachLabel(imdsTrain)
ValiLabelCounter = countEachLabel(imdsValidation);

% Size of img
img=readimage(imds,1);
ImgSize = size(img)
```

Picture 17. Code to load dataset into MATLAB

Next, we need to do pre-processing the dataset Data Augmented technique. This technique used to artificially increase the size of a training dataset by creating a new training example from the existing one. Data augmented process can be done by applying transformation to the image so that the network can get more variations with the same image. The reason we apply this technique is to improve the generalization ability of a deep learning model and reduce overfitting when the available training data is limited [8]. To implement this technique, first we apply data augmentation to the training dataset using the 'imageDataAugmenter' function. This function will perform random reflection, rotation, and scaling on the images to increase the size of dataset. Then we can create augmented image datastore to resize all image from augmented data into the same size with original image so that it can provide into the network layer. The pre-processing of the training and validation dataset using augmented data are shown in the picture 18.

```
%% preprocessing
% corrode

% rotation and scaling
%pixelRange = [-64 64]; %[-30,30]
scaleRange = [0.8 1.2];
rotateRange = [-40,40];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandYReflection',true, ...
    'RandRotation',rotateRange, ...
    'RandXScale',scaleRange, ...
    'RandYScale',scaleRange, ...
    'RandXTranslation',[-5 5],...
    'RandYTranslation',[-5 5] ...
    );

%     'RandXTranslation',pixelRange, ...
%     'RandYTranslation',pixelRange, ...

inputSize = [128 128 1];
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter);

augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

Picture 18. The code show pre-processing using augmented data technique.

The next process is building the network layers for CNN model. First, we define the image input layer which specifies the size of input images (128 x 128 pixels). Next, we build the convolution layer using ‘convolution2dlayer’ function where first layer using 32 filters of 32 x 32 pixels and subsequent layer using 64 and 128 filters of the same size. These layers are useful to perform convolution into input image. In this convolution layer, the ‘Padding’ set into ‘same’ which give the output size same as input size. After convolution, we set the batch normalization layer to normalize the output by subtracting the batch mean and dividing by batch standard deviation to speed up training and improve generalization [9]. Then, we can build rectified linear unit activation layer (reluLayer) which will give negative values to zero and leave positive values unchanged. The maximum pooling layer can be set to reduce spatial dimension of the input by taking the maximum value within a set of rectangular regions. We set the ’stride’ parameter into 2 which mean the regions are shifted by 2 pixels. In the last of sublayer, we can set drop out layer to drop out a fraction of input unit during training which help to prevent overfitting [10].

In the last of CNN layers, we can define additional three sublayers which are fully connected layer and soft max layer, and classification layer. Fully connected layer is used to connects all input units to all output units with weights that have been determined. Softmax layer is used to convert output of network into probability distribution over the classes. And the last is classification layer which used to calculate the loss between predicted and actual class probabilities. The code to set up each layer of CNN part is shown in the picture 19.

```
layers = [

    imageInputLayer([128 128 1])

    convolution2dLayer(5,6,'Padding',2)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'stride',2)
    convolution2dLayer(5, 16)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'stride',2)
    convolution2dLayer(5, 128)
    batchNormalizationLayer
    reluLayer

    dropoutLayer(0.5)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer
    ];
```

Picture 19. The code used to build the CNN layers (16 layers)

After we build the layers, we can define the training options. This option is important to be set because it determines how the network will be trained and optimized. These options specify the optimization algorithm to determine the network's weight during training to minimize the loss function, learning rate to determine how quickly the weights can be adjusted, and number of epochs used to specify how many times the entire dataset will be used to update the weights. In this training options, we set several parameters which will be explained below.

A. Sgdm: we use this option because it can specify the stochastic gradient descent with momentum optimizer so that it can minimize the loss function during training.
B. Initial learning rate: we set the initial learning rate to 0.001 for the optimizer.
C. Max Epochs: we set the maximum number of epochs to 40 which the network should be trained.
D. Shuffle: This specify the training data need to be shuffle before each epoch to help from memorizing the order of the training example and overfitting to the training data.
E. Validation data: this parameter used to evaluate the model's performance on a separate dataset during training.
F. Validation Frequency: this parameter used to determine how often to evaluate the model on the validation data. We set this parameter to 5 so that the model will evaluate every 5 epochs.

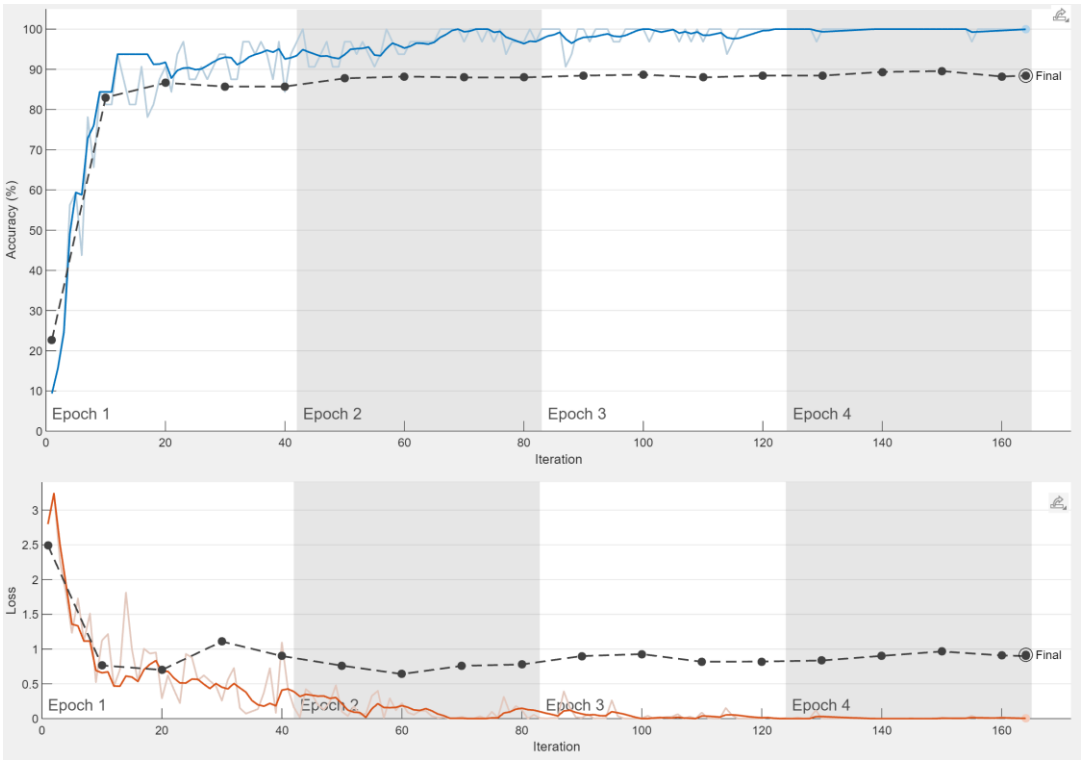
G. Plots: This parameter can plot of the training progress and help to visualize the model's performance over time.
H. Validation Patience: the number of consecutive epochs that validation accuracy can remain unchanged before training is stopped.

After we set the layers and training options, we can start the training process of the model using 'trainNetwork' function. This function has input parameter from augmented image datastore, layers, and training options. After running this function, we can show the progress of the training network through plot and determine whether the model has already good performance. The code to set the training options is shown in the picture 20 and the result of training and validation model without pre-processing is shown in the picture 21.

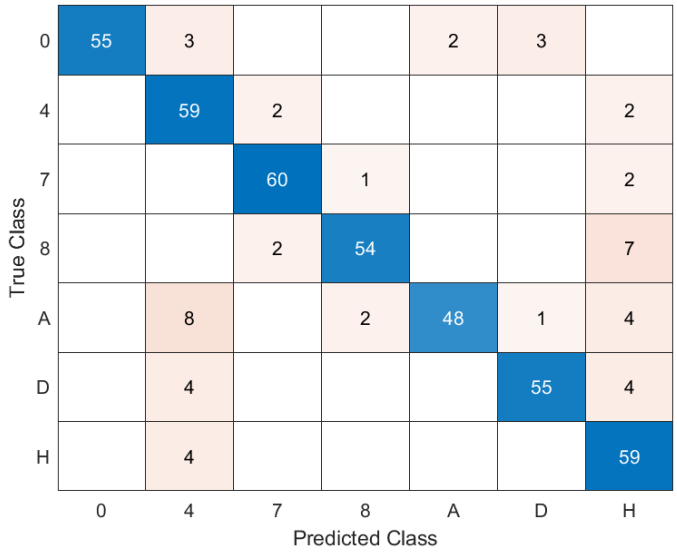
```
%% Training Network
% training options
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.001, ...
    'MaxEpochs',40, ...
    'MiniBatchSize',32, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',20, ...
    'Verbose',false, ...
    'Plots','training-progress', ...
    'ValidationPatience',4 ...
    );

% training
net = trainNetwork(augimdsTrain,layers,options);
save 'CNN.mat' net

YPred = classify(net,augimdsValidation);
YValidation = imdsValidation.Labels;

figure
confusionchart(YValidation,YPred)

fprintf('finish training')
```

Picture 20. The code show the training options and training network function

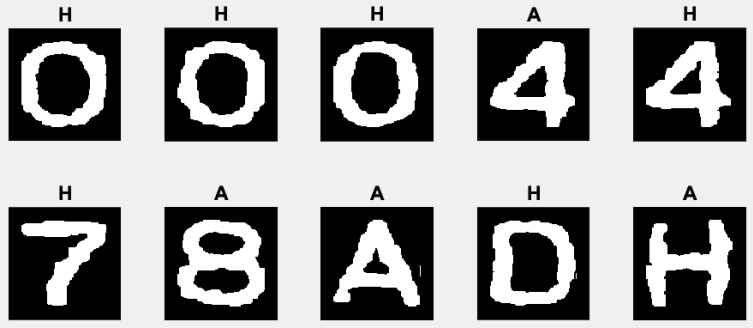
From the plot in picture 21, we can see that the accuracy of the training set can reach approximately 99 % after 205 iterations. However, the accuracy of validation on the final iterations is around 85 %. There is one possibility that make the accuracy of validation is lower than the accuracy of training which the model is overfitting to the training data. Overfitting may occur when the model learns to recognize specific patterns in the training data too well and make it become too specialized and will fail to recognize on new data set [11]. This overfitting issue also can be seen in the picture 22 which show confusion matrix. There are some misclassifications as shown by the off-diagonal elements. This issue may happen because the model is too complex and has many parameters relative to the amount of training data. To address this issue, we try to increase the amount of training data set using pre-processing augmented data technique and add more layers to change the architectures of this network.




Picture 21. The plot showing the training and validation result without pre-processing

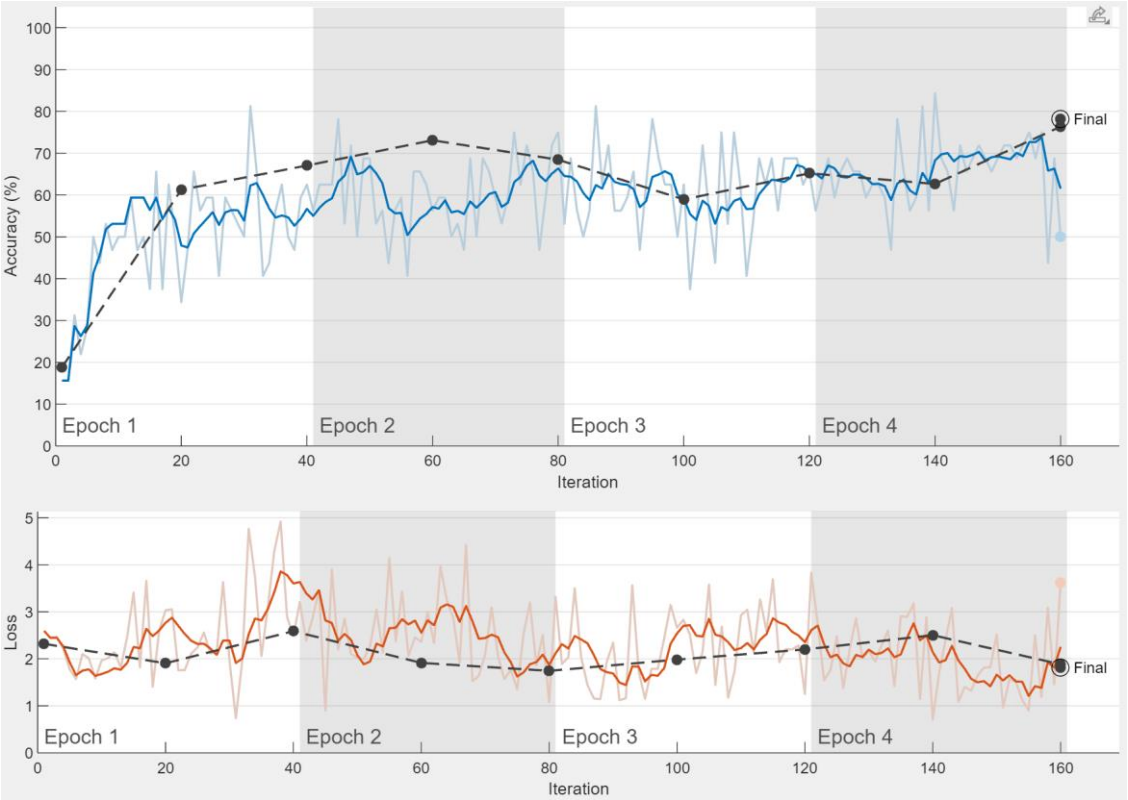



Picture 22. The confusion matrix showing overfitting issue

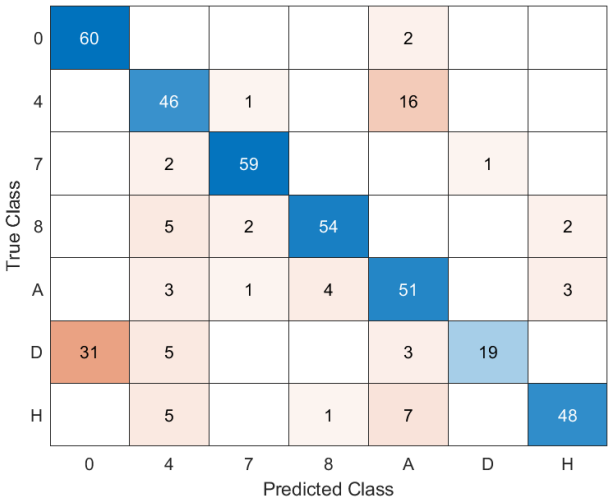



Picture 23. The visualisation of predicting object without pre-processing

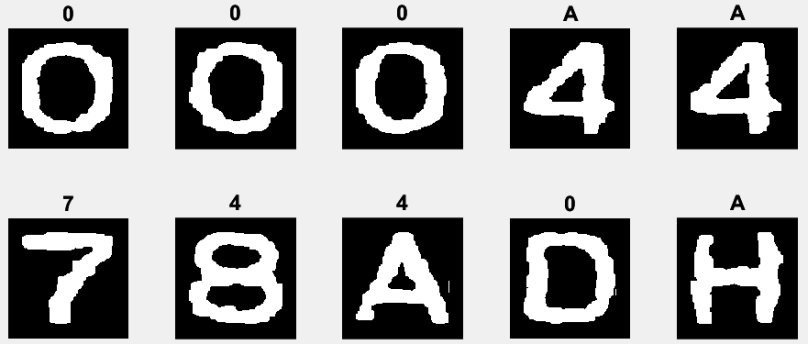
In the first design of CNN, we set the number of layers into 16 layers and for the improvement, we add pre-processing of augmented data technique. The graph plot after adding pre-processing is shown in picture 24 and the confusion matrix is shown in picture 25. From the graph, we can see that the final validation accuracy only reaches around 80 % after 160 iterations which means adding pre-processing doesn't provide a good result than the first design of CNN. From the confusion matrix in picture 25, most of the images were not correctly classified because they still have a misclassification in each class. So overall it can be concluded that this new model still has bad performance.




Picture 24. The graph plot after adding pre-processing in CNN with 16 layers



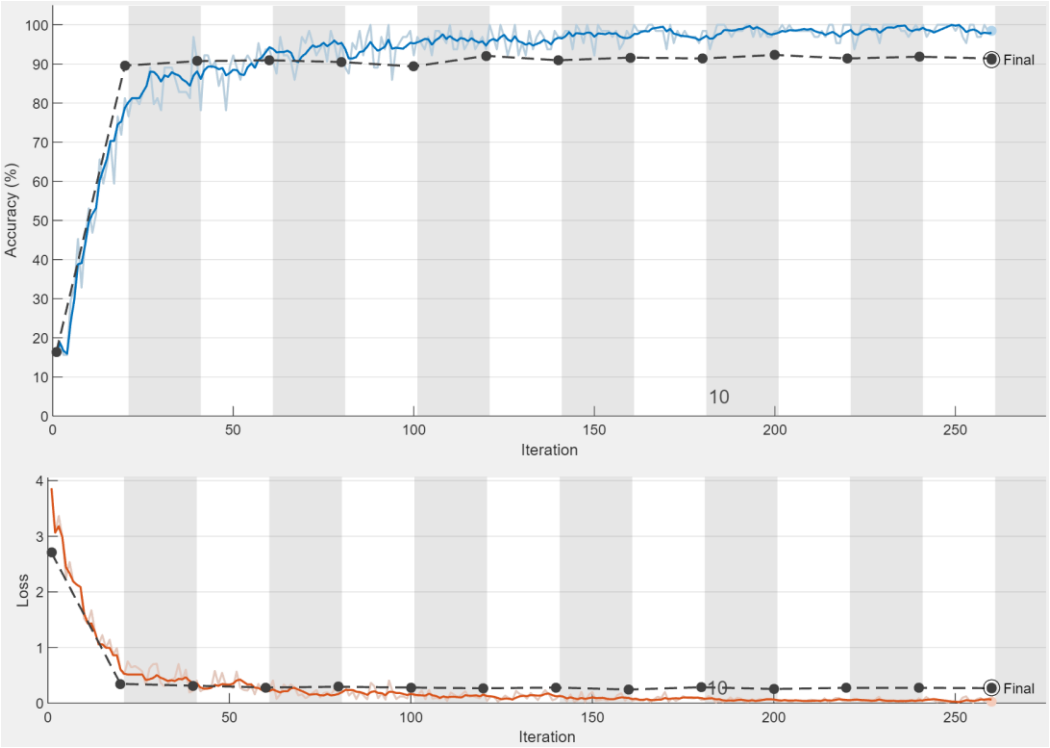

Picture 25. The confusion matrix showing underfitting issue



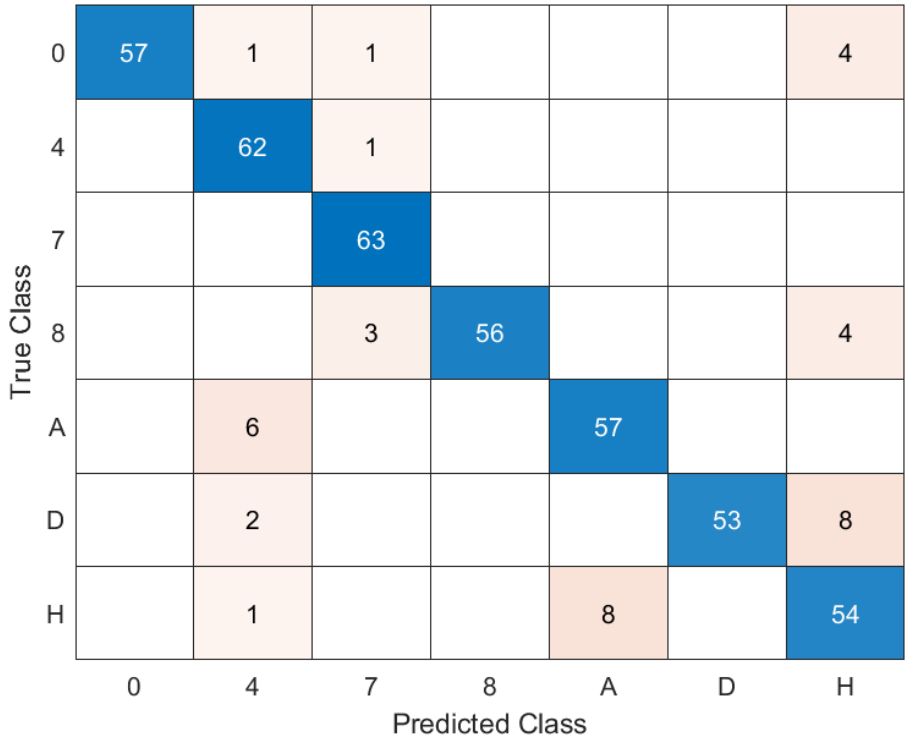

Picture 26. The visualisation of predicting class after pre-processing with 16 layers

One of the reasons that this model has a bad performance is the complexity of neural networks may not be sufficient to capture the complex relationships in the data, leading to the neural network's inability to correctly classify the training data, resulting in poor performance and low accuracy on both training and test data, which is called underfitting [12].

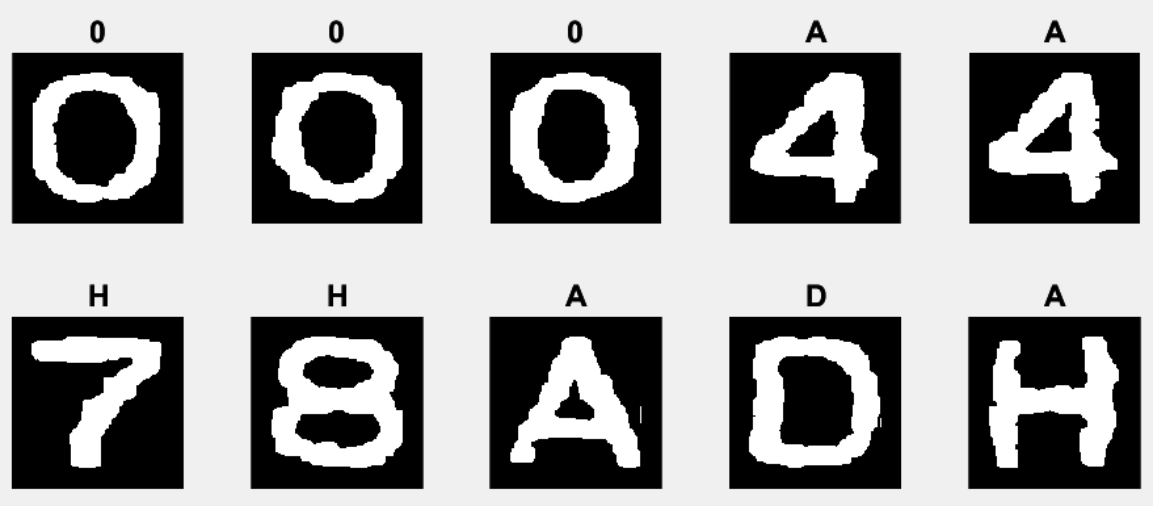
Then, we try to add more layers into original CNN from 16 layers into 29 layers without using pre-processing technique. The graph plot after adding these layers is shown in picture 27 and the confusion matrix is shown in picture 28. From the graph, we can see that the accuracy of validation can reach around 90 % which mean that this model has better performance than the original 16 layers. From confusion matrix, there is still several misclassifications in predicting the class however there are 3 classes (O, 8, D) that completely classify. From this model, we can conclude that adding more layers can provide better performance in analysing character.




Picture 27. The graph plot after adding more layers into 29 layers

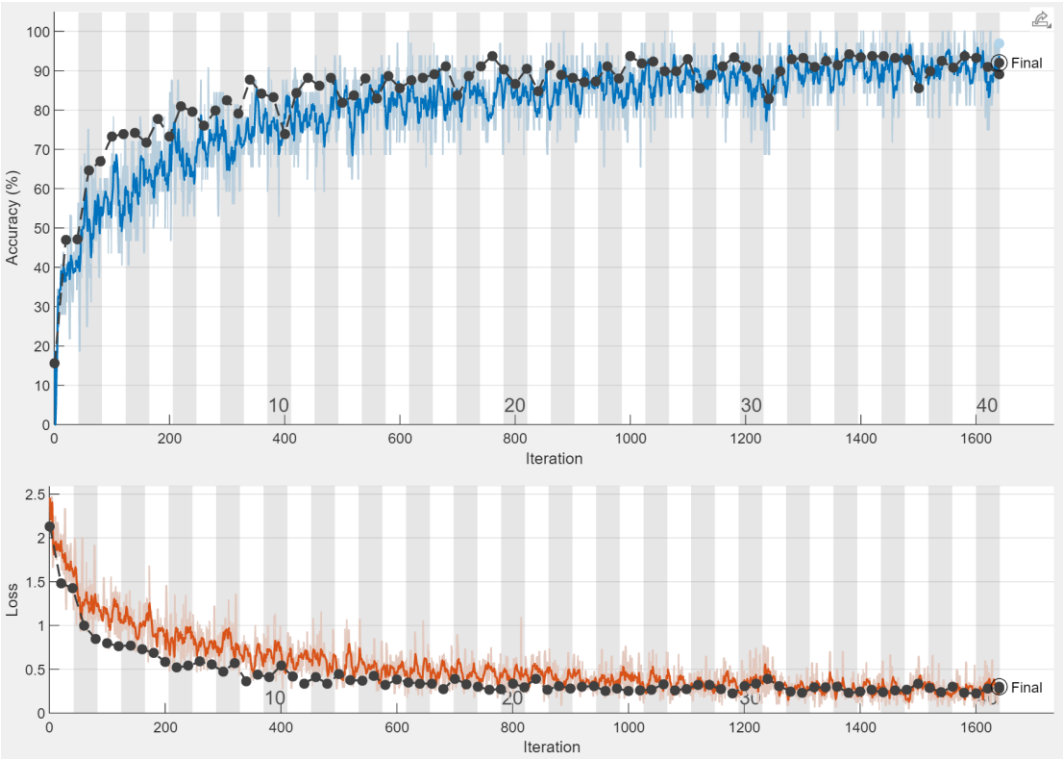



Picture 28. The confusion matrix after adding more layers into 29 layers

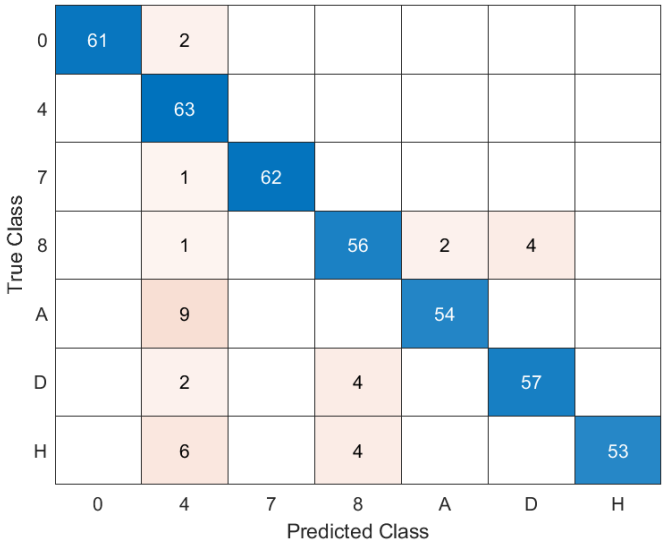



Picture 29. The visualisation of predicting class after adding more layers into 29 layers

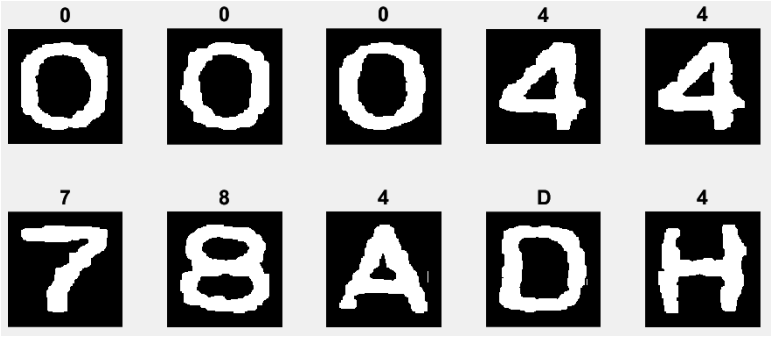
Finally, we try to add more layers into 29 layers and add pre-processing augmented data technique with dataset reflection, rotation, scaling, and translation to see whether this model has the best performance than others. The graph plot after adding pre-processing and more layers is shown in picture 30 and the confusion matrix is shown in picture 31. From the graph, we can see that the final validation accuracy can reach around 91 % which gives the best performance and accuracy than other models. The model seems to have achieved a reasonable accuracy on both training and validation data. From the confusion matrix, the majority of the images were correctly classified but there is still have a misclassification in class 4 and 8 where there are relatively high numbers in the off-diagonal positions.




Picture 30. The graph plot after adding pre-processing and more layers into 29 layers




Picture 31. The confusion matrix shows less misclassification than other models




Picture 32. The visualisation of predicting class after adding pre-processing and layers

From this result, we can conclude that pre-processing and increasing the depth of the neural network have been shown to improve the accuracy of CNN for image recognition tasks. Pre-processing, including data augmentation and normalization, helps to improve the quality and quantity of data, which enhances the CNN's ability to adapt to the data and increases its robustness. By generating more training data through data augmentation techniques, the CNN can better capture the variability of the input images and learn to recognize the same object from different perspectives [8]. Furthermore, increasing the depth of the neural network can improve the CNN's representation and abstraction capabilities, as it enables the model to learn hierarchical features and more complex patterns [13]. Deeper CNN can use multiple convolutional and pooling layers to extract features and down sample the input data, which helps to capture visual patterns and relationships in the images more effectively. By increasing the number of layers, the model can also learn more discriminative features, which can improve its classification performance. In summary, pre-processing and increasing the depth of the neural network are effective strategies to improve the accuracy of CNN, as they help to enhance the quality of the data and increase the model's ability to capture complex patterns and relationships in the images.

After we build the CNN model, the next step is building another model which is not CNN-based model. In this project we choose the SVM model to analysis the features of the image (segmented object). SVM is a popular deep learning algorithm used for classification and regression analysis. It is a supervised learning method that learns a decision boundary by mapping input data into a high-dimensional feature space, then finding the optimal hyperplane that separates the classes with maximum margin. SVM has several advantages, such as being effective in high-dimensional spaces, having a solid theoretical foundation, and being able to handle both linear and non-linear classification tasks. SVM can also be extended to handle multi-class classification problems and regression analysis. SVM has found applications in a wide range of fields, including computer vision, bioinformatics, text classification, and finance [14].

To build this model, first we build a code for load training and test data set using ‘imageDataStore’ function in similar way for loading data in CNN model. Next, we can extract the HOG (Histogram of Oriented Gradient) and visualize this feature for every specific picture in training data set. HOG is one of feature that is widely used in image processing for object detection and recognition. This feature represents an image by counting occurrences of gradient orientation in localized portion of the image [15]. In the MATLAB, we can extract this feature using ‘extractHOGFeatures’ function and pre-process the features which may include normalization and dimensionality reduction. The code used to extract the HOG features in every image inside training data set is shown in the picture 33.

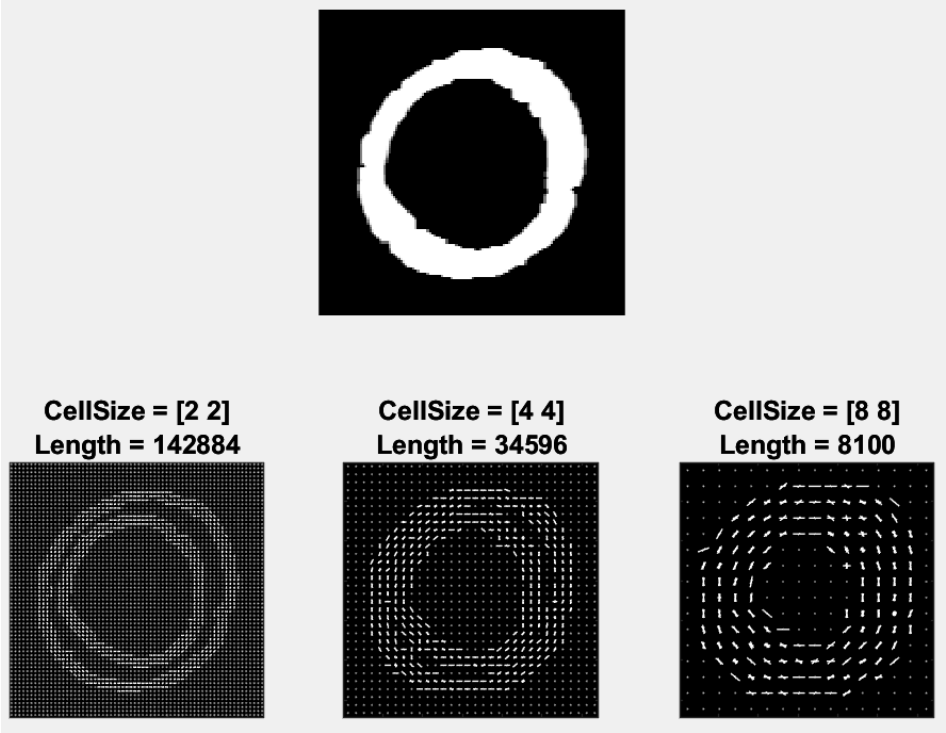
```
%% Train a Digit Classifier

% Loop over the trainingSet and extract HOG features from each image. A
% similar procedure will be used to extract features from the testSet.

numImages = numel(trainingSet.Files);
trainingFeatures = zeros(numImages,hogFeatureSize,'single');

for i = 1:numImages
    img = readimage(trainingSet,i);

    % img = im2gray(img);

    % Apply pre-processing steps
    % img = imbinarize(img);

    trainingFeatures(i, :) = extractHOGFeatures(img,'CellSize',cellSize);
end

% Get labels for each image.
trainingLabels = trainingSet.Labels;

% fitcecoc uses SVM learners and a 'One-vs-One' encoding scheme.
classifier = fitcecoc(trainingFeatures, trainingLabels)
save 'SVM1.mat' classifier
```

Picture 33. The code showing the process of extracting HOG features and build classifier

Next, we create a multiclass SVM classifier using extracted features and the training label as input argument for ‘fitcecoc’ function. This function uses a ‘one-vs-one’ encoding scheme for multiclass classification which train binary classifier for each pair of classes in the training set. Next, we build a function called ‘helperExtractHOGFeaturesFromImageSet’ to extract HOG features from a test data set which store in image datastore. The function takes three input arguments: an ‘imageDatastore’ object ‘imds’, which represents a collection of images; ‘hogFeatureSize’, which specifies the size of the HOG feature vector; and ‘cellSize’, which specifies the size of the HOG cells. It first extracts the labels of each image in the ‘imageDatastore’ and stores them in the variable ‘setLabels’. Next the function determines the number of images in the ‘imageDatastore’ using the ‘numel’ function. After that, it creates a features matrix of size (numImages, hogFeatureSize), where each row represents the HOG feature vector of an image. It also processes each image in the ‘imageDatastore’ by reading it using the ‘readimage’ function, converting it to grayscale using the ‘im2gray’ function, and binarizing it is using the ‘imbinarize’ function. The function then extracts the HOG features of each processed image using the ‘extractHOGFeatures’ function with the specified ‘cellSize’ parameter. The extracted features are stored in the corresponding row of the features matrix. And finally, it returns the features matrix and the setLabels variable. The code shows this support function is shown in picture 34 and the visualization of HOG features is shown in picture 35.

```
function [features, setLabels] = helperExtractHOGFeaturesFromImageSet( ...
    imds, hogFeatureSize, cellSize)
% Extract HOG features from an imageDatastore.

setLabels = imds.Labels;
numImages = numel(imds.Files);
features  = zeros(numImages,hogFeatureSize,'single');

% Process each image and extract features
for j = 1:numImages
    img = readimage(imds,j);
    img = im2gray(img);

    % Apply pre-processing steps
    img = imbinarize(img);

    features(j, :) = extractHOGFeatures(img,'CellSize',cellSize);
end
end
```

Picture 34. The code show support function of 'helperExtractHOGFeaturesFromImageSet'




Picture 35. The result of visualisation of HOG features for object O in training data set

Before we make class prediction, we need to build another support function which is called 'helperDisplayConfusionMatrix(confMat)' to take a confusion matrix as input and display it in a formattable table. Overall, this function will take the confusion matrix and normalize it. Then formatting it into a table for easy visualization and interpretation of classification result. The code that shows this support function is shown in picture 36.

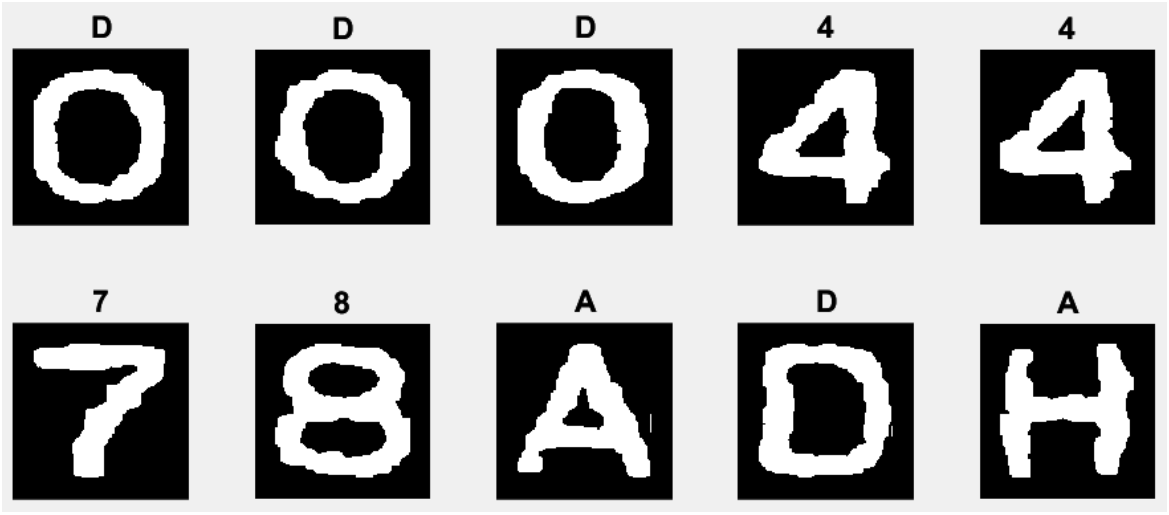
```
function helperDisplayConfusionMatrix(confMat)
% Display the confusion matrix in a formatted table.

% Convert confusion matrix into percentage form
confMat = bsxfun(@rdivide,confMat,sum(confMat,2));

digits = ['0','4','7','8','A','D','H'];
colHeadings = arrayfun(@(x)sprintf('%c',x),digits,'UniformOutput',false);
format = repmat('%-9s',1,11);
header = sprintf(format,'ConfusionMatrix  |',colHeadings{:});
fprintf('\n%s\n%s\n',header,repmat('-',size(header)));
for idx = 1:numel(digits)
    fprintf('%-9s',   [digits(idx) '                |']);
    fprintf('%-9.2f', confMat(idx,:));
    fprintf('\n')
end
end
```

Picture 36. The code show support function of 'helperDisplayConfusionMatrix'

The final process to build SVM model is to make class prediction using the test data set which have been extracted it features using extract HOG Features support function. The class prediction can be made from 'predict' function which required two input arguments, one from classifier and the test features. Then we can tabulate the results using a confusion matrix function and display it into table form using 'helperDisplayConfusionMatrix' support function. Overall, this SVM model is defined using 'fitcsvm' which specifying the type of kernel to be used, and the pre-processed training data is used to train the model. The model is fine-tuned by adjusting the SVM parameters and re-training if necessary. Finally, the trained model is evaluated on the validation set using 'predict' function and used to make predictions on new data. The result of the confusion matrix in table form without pre-processing data set is shown in picture 37 and the result visualization of the predicted character is shown in picture 38.

```
ConfusionMatrix  |0        4        7        8        A        D        H
-------------------------------------------------------------------------------
0                |0.00     0.00     0.00     0.00     0.00     1.00     0.00
4                |0.00     1.00     0.00     0.00     0.00     0.00     0.00
7                |0.00     0.00     1.00     0.00     0.00     0.00     0.00
8                |0.00     0.00     0.00     1.00     0.00     0.00     0.00
A                |0.00     0.00     0.00     0.00     1.00     0.00     0.00
D                |0.00     0.00     0.00     0.00     0.00     1.00     0.00
H                |0.00     0.00     0.00     0.00     1.00     0.00     0.00

accuracy =

    0.6000
```

Picture 37. The confusion matrix in table form without pre-processing the data set

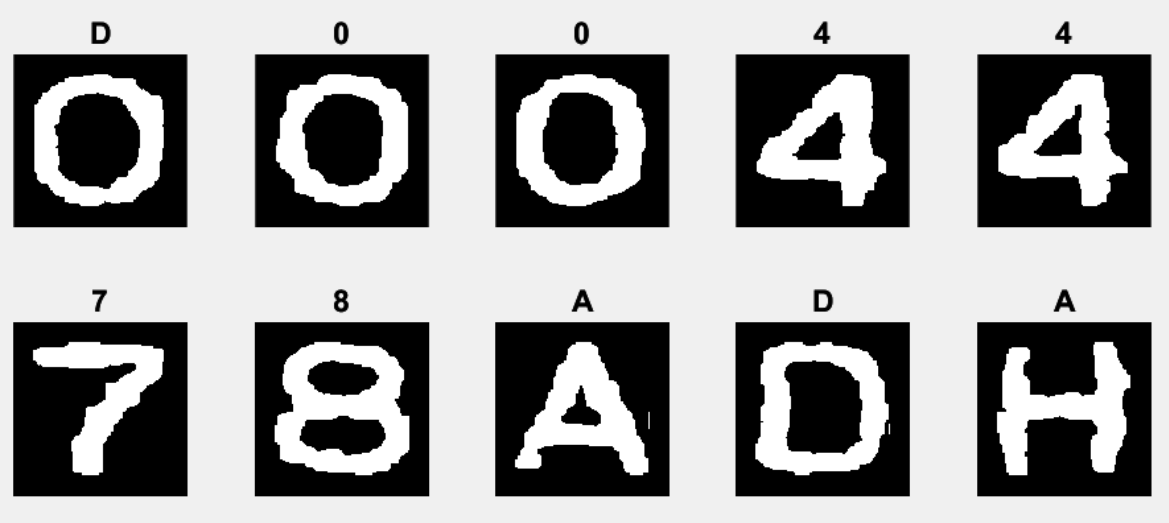



Picture 38. The result of visualization of the predicted character

From this result without pre-processing, we can see that there are several misclassifications when predict a class O, D, A and H. This model makes wrong prediction when recognizing between O & D object and A & H object. The accuracy of this model only reaches 0.6 (60 %) when analysing the HOG features on each object. To get the best performance of the SVM model, we try adding some pre-processing that can help to increase the accuracy of recognizing characters. In this project, we performed a commonly used pre-processing step of eliminating specific handwritten fonts, which contained inappropriate data, such as erroneous, exceptional, or duplicated values. This step improved the data quality and allowed the SVM to produce higher quality classification results. The elimination of inappropriate data is a useful pre-processing step as it helps to eliminate noise and unnecessary variability and allows the models to capture patterns and relationships more accurately in the data [16]. The result of the confusion matrix after pre-processing is shown in the picture 39 and the visualization of predicted character after pre-processing is shown in the picture 40.

```
ConfusionMatrix  |0        4        7        8        A        D        H
-------------------------------------------------------------------------------
0                |0.67     0.00     0.00     0.00     0.00     0.33     0.00
4                |0.00     1.00     0.00     0.00     0.00     0.00     0.00
7                |0.00     0.00     1.00     0.00     0.00     0.00     0.00
8                |0.00     0.00     0.00     1.00     0.00     0.00     0.00
A                |0.00     0.00     0.00     0.00     1.00     0.00     0.00
D                |0.00     0.00     0.00     0.00     0.00     1.00     0.00
H                |0.00     0.00     0.00     0.00     1.00     0.00     0.00

accuracy =

    0.8000
```

Picture 39. The confusion matrix after pre-processing




Picture 40. The visualization of predicted character after pre-processing

It can be seen clearly from this result which the SVM model after pre-processing has better performance than without pre-processing. The accuracy of this model is 0.8 (80 %) which the model now can recognize the object between object O and object D.

In comparison of CNN and SVM for image classification, CNNs are well-suited for large datasets because they require a large amount of data to train the system effectively. CNNs are particularly good at learning the spatial relationships between pixels in an image and can extract a hierarchy of features that are progressively more abstract. In contrast, SVMs can be trained with smaller datasets, making them a good choice for smaller datasets or when computational resources are limited [17]. One advantage of SVMs is that they require a separate feature extraction step, which allows for more control over the types of features used for classification. Choosing a suitable feature extraction method can help to achieve maximum accuracy. For example, selecting appropriate feature descriptors such as Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT) can improve the performance of SVM classifiers. Therefore, the choice of classifier depends on the specific characteristics of the dataset and the computational resources available.

# Chapter 4
# Discussion and Conclusion

In this section, we will discuss and conclude about the method which we choose in image processing task. First, after smoothing the image to reduce the noise, we found out that a morphological filter provides better results in reducing the noise on the background of the image than using averaging and rotating mask as filter. Morphological filters can remove small, isolated pixels or noise component while preserving the structure of the object meanwhile averaging and rotating mask are more focused on smoothing or blurring an image to reduce noise which resulting in a loss of important detail in the image [18].

Second, in the segmentation part, we find out that there is still several object/character that is connected to each other after getting the binary image. The best method to separate this connected object is using processing technique to cut half the remaining connected object by calculating the cluster width to set the value of 0 pixel in the middle of column. By using this technique, each character can be separated completely.

Third, in the deep learning part, we find out that the best result can be achieved when we apply some pre-processing (using augmented data technique for CNN and elimination of inappropriate data for SVM). This pre-processing help to improve the accuracy of recognizing and analysing the features of segmented characters as new data input. Furthermore, for CNN model, increasing the depth of the neural network can improve abstraction capabilities that enables the model to learn hierarchical features and more complex patterns.

Finally, in conclusion, what lessons we learned from this computing project is we can fix the degradation image using image pre-processing task such as smoothing image and reducing noise in the background image. From every digital image, we can extract the important features from the image using segmentation techniques such as converting to a binary image, finding the outline of each character, and labelling each object so that it can be easily recognized in the image processing task. And lastly, to build deep learning algorithms that can analyse and recognize the feature of the object, we can implement CNN and SVM algorithms to analyse these by adding some pre-processing techniques to increase the accuracy of recognition and prediction of the class objects.

# Reference


[1] Chan, T.F., & Shen, J. 2005. *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods.* Society for Industrial and Applied Mathematics.

[2] LeCun, Yann, Yoshua Bengio, & Geoffrey Hinton. 2015. *Deep Learning.* MIT Press. Chapter 5: Machine Learning Basics.

[3] Klette,R., & Zamperoni, P. 2012. *Image Processing – Dealing with Color.* In R. Klette & P. Zamperoni (Eds.), Computer Vision: Three-Dimensional Data from Images (pp. 83-91). Springer.

[4] Gonzalez, R.C. & Woods, R.E. 2018. *Digital Image Processing (4th Edition).* Pearson Education. Chapter 3: Spatial Domain Image Processing.

[5] Gonzalez, R.C. & Woods, R.E. 2018. *Digital Image Processing (4th Edition).* Pearson Education. Chapter 9: Morphological Image Processing.

[6] Gonzalez, R.C. & Woods, R.E. 2018. *Digital Image Processing (4th Edition).* Pearson Education. Chapter 10: Image Segmentation. Section 10.1.1: Thresholding.

[7] Vijayakumaran, S., Devi, M. N., & Santhanakrishnan, T. 2012. *Edge Detection Techniques – An Overview.* International Journal of Computer Science & Information Technology (IJCSIT), 4(1), 243-258.

[8] Shorten, N., & Khoshgoftaar, T. M. 2019. *A Comprehensive Survey on Data Augmentation for Deep Learning.* Big Data and Cognitive Computing, 3(1), 1-23.

[9] Ioffe, S. & Szegedy, C. 2015. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* In Proceedings of the 32nd International Conference on Machine Learning. pp. 448-456.

[10] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research, 15(1), 1929-1958.

[11] Jha, R. K., Singh, R. S., Ekbal, A., & Bhattacharyya, P. 2020. *A Comprehensive Survey on Convolutional Neural Network Overfitting and Its Solutions.* Neural Computing and Applications, 32(16), 12379 – 12405.

[12] Shah, R. 2019. *Understanding Underfitting and Overfitting in Deep Learning.* Towards Data Science (Online Publication).

[13] He, K., Zhang, X., Ren, S., & Sun, J. 2016. *Deep Residual Learning for Image Recognition.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770-778.

[14] Burges, C. J. C. 1998. *A Tutorial on Support Vector Machines for Pattern Recognition.* Data Mining and Knowledge Discovery, 2(2), 121-167.

[15] Dalal, N., & Triggs, B. 2005. *Histograms of Oriented Gradients for Human Detection.* Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 886-893.

[16] Priya, S. S., Jayashri, S., & Geetha, T. V. 2018. *A Comprehensive Study on Pre-processing Techniques for SVM Based Classification of Lung Cancer Data.* Journal of Medical Systems, 42(5), 84.

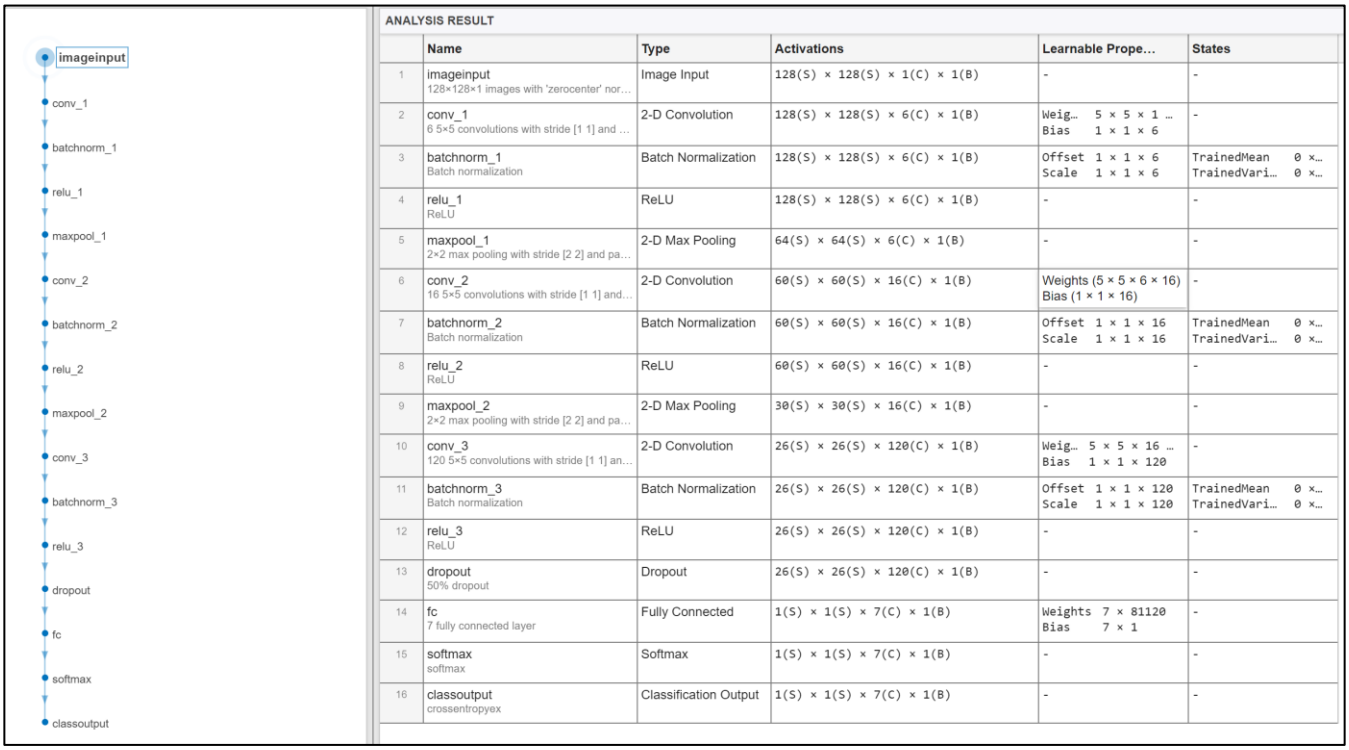
[17] Lefakis, T., Bampis, A., & Galatsanos, N. 2019. *Comparison of SVM and CNN for Image Classification on Small Datasets.* In Proceedings of SPIE – The International Society for Optical Engineering.

[18] Dougherty, E. R., & Astola, J. 1994. *Image Processing Using Morphological Filters.* Proceedings of the IEEE, 82(6), 804-821.

# Appendices

```
layers = [

    imageInputLayer([128 128 1])

    convolution2dLayer(5,6,'Padding',2)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'stride',2)
    convolution2dLayer(5, 16)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'stride',2)
    convolution2dLayer(5, 128)
    batchNormalizationLayer
    reluLayer

    dropoutLayer(0.5)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer
    ];
```

Picture A. The code show function to build 16 layers of CNN model



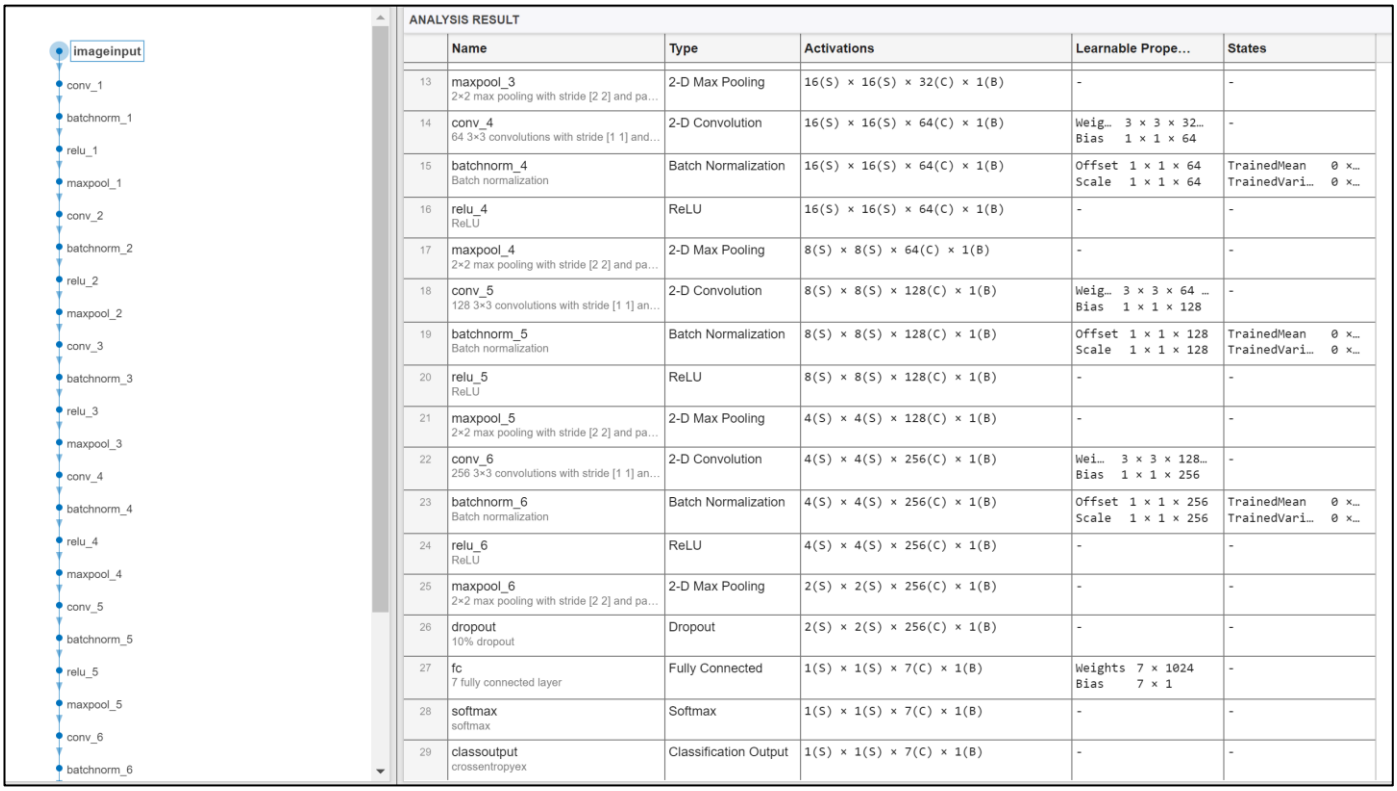

ANALYSIS RESULT

| | Name | Type | Activations | Learnable Prope… | States |
|---|---|---|---|---|---|
| 1 | imageinput<br>128×128×1 images with 'zerocenter' nor… | Image Input | 128(S) × 128(S) × 1(C) × 1(B) | - | - |
| 2 | conv_1<br>6 5×5 convolutions with stride [1 1] and … | 2-D Convolution | 128(S) × 128(S) × 6(C) × 1(B) | Weig… 5 × 5 × 1 …<br>Bias 1 × 1 × 6 | - |
| 3 | batchnorm_1<br>Batch normalization | Batch Normalization | 128(S) × 128(S) × 6(C) × 1(B) | Offset 1 × 1 × 6<br>Scale 1 × 1 × 6 | TrainedMean 0 ×…<br>TrainedVari… 0 ×… |
| 4 | relu_1<br>ReLU | ReLU | 128(S) × 128(S) × 6(C) × 1(B) | - | - |
| 5 | maxpool_1<br>2×2 max pooling with stride [2 2] and pa… | 2-D Max Pooling | 64(S) × 64(S) × 6(C) × 1(B) | - | - |
| 6 | conv_2<br>16 5×5 convolutions with stride [1 1] and… | 2-D Convolution | 60(S) × 60(S) × 16(C) × 1(B) | Weights (5 × 5 × 6 × 16)<br>Bias (1 × 1 × 16) | - |
| 7 | batchnorm_2<br>Batch normalization | Batch Normalization | 60(S) × 60(S) × 16(C) × 1(B) | Offset 1 × 1 × 16<br>Scale 1 × 1 × 16 | TrainedMean 0 ×…<br>TrainedVari… 0 ×… |
| 8 | relu_2<br>ReLU | ReLU | 60(S) × 60(S) × 16(C) × 1(B) | - | - |
| 9 | maxpool_2<br>2×2 max pooling with stride [2 2] and pa… | 2-D Max Pooling | 30(S) × 30(S) × 16(C) × 1(B) | - | - |
| 10 | conv_3<br>120 5×5 convolutions with stride [1 1] an… | 2-D Convolution | 26(S) × 26(S) × 120(C) × 1(B) | Weig… 5 × 5 × 16 …<br>Bias 1 × 1 × 120 | - |
| 11 | batchnorm_3<br>Batch normalization | Batch Normalization | 26(S) × 26(S) × 120(C) × 1(B) | Offset 1 × 1 × 120<br>Scale 1 × 1 × 120 | TrainedMean 0 ×…<br>TrainedVari… 0 ×… |
| 12 | relu_3<br>ReLU | ReLU | 26(S) × 26(S) × 120(C) × 1(B) | - | - |
| 13 | dropout<br>50% dropout | Dropout | 26(S) × 26(S) × 120(C) × 1(B) | - | - |
| 14 | fc<br>7 fully connected layer | Fully Connected | 1(S) × 1(S) × 7(C) × 1(B) | Weights 7 × 81120<br>Bias 7 × 1 | - |
| 15 | softmax<br>softmax | Softmax | 1(S) × 1(S) × 7(C) × 1(B) | - | - |
| 16 | classoutput<br>crossentropyex | Classification Output | 1(S) × 1(S) × 7(C) × 1(B) | - | - |

Picture B. The structure of CNN model for 16 layers

```
layers = [
    imageInputLayer([128 128 1])
    convolution2dLayer(3, 8, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
    convolution2dLayer(3, 16, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
    convolution2dLayer(3, 32, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
    convolution2dLayer(3, 64, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
    convolution2dLayer(3, 128, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
    convolution2dLayer(3, 256, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
    dropoutLayer(0.5)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer
];
```

Picture C. The code show function to build 29 layers of CNN model




ANALYSIS RESULT

| | Name | Type | Activations | Learnable Prope… | States |
|---|---|---|---|---|---|
| 13 | maxpool_3<br>2×2 max pooling with stride [2 2] and pa… | 2-D Max Pooling | 16(S) × 16(S) × 32(C) × 1(B) | - | - |
| 14 | conv_4<br>64 3×3 convolutions with stride [1 1] and… | 2-D Convolution | 16(S) × 16(S) × 64(C) × 1(B) | Weig… 3 × 3 × 32…<br>Bias 1 × 1 × 64 | - |
| 15 | batchnorm_4<br>Batch normalization | Batch Normalization | 16(S) × 16(S) × 64(C) × 1(B) | Offset 1 × 1 × 64<br>Scale 1 × 1 × 64 | TrainedMean 0 ×…<br>TrainedVari… 0 ×… |
| 16 | relu_4<br>ReLU | ReLU | 16(S) × 16(S) × 64(C) × 1(B) | - | - |
| 17 | maxpool_4<br>2×2 max pooling with stride [2 2] and pa… | 2-D Max Pooling | 8(S) × 8(S) × 64(C) × 1(B) | - | - |
| 18 | conv_5<br>128 3×3 convolutions with stride [1 1] an… | 2-D Convolution | 8(S) × 8(S) × 128(C) × 1(B) | Weig… 3 × 3 × 64 …<br>Bias 1 × 1 × 128 | - |
| 19 | batchnorm_5<br>Batch normalization | Batch Normalization | 8(S) × 8(S) × 128(C) × 1(B) | Offset 1 × 1 × 128<br>Scale 1 × 1 × 128 | TrainedMean 0 ×…<br>TrainedVari… 0 ×… |
| 20 | relu_5<br>ReLU | ReLU | 8(S) × 8(S) × 128(C) × 1(B) | - | - |
| 21 | maxpool_5<br>2×2 max pooling with stride [2 2] and pa… | 2-D Max Pooling | 4(S) × 4(S) × 128(C) × 1(B) | - | - |
| 22 | conv_6<br>256 3×3 convolutions with stride [1 1] an… | 2-D Convolution | 4(S) × 4(S) × 256(C) × 1(B) | Wei… 3 × 3 × 128…<br>Bias 1 × 1 × 256 | - |
| 23 | batchnorm_6<br>Batch normalization | Batch Normalization | 4(S) × 4(S) × 256(C) × 1(B) | Offset 1 × 1 × 256<br>Scale 1 × 1 × 256 | TrainedMean 0 ×…<br>TrainedVari… 0 ×… |
| 24 | relu_6<br>ReLU | ReLU | 4(S) × 4(S) × 256(C) × 1(B) | - | - |
| 25 | maxpool_6<br>2×2 max pooling with stride [2 2] and pa… | 2-D Max Pooling | 2(S) × 2(S) × 256(C) × 1(B) | - | - |
| 26 | dropout<br>10% dropout | Dropout | 2(S) × 2(S) × 256(C) × 1(B) | - | - |
| 27 | fc<br>7 fully connected layer | Fully Connected | 1(S) × 1(S) × 7(C) × 1(B) | Weights 7 × 1024<br>Bias 7 × 1 | - |
| 28 | softmax<br>softmax | Softmax | 1(S) × 1(S) × 7(C) × 1(B) | - | - |
| 29 | classoutput<br>crossentropyex | Classification Output | 1(S) × 1(S) × 7(C) × 1(B) | - | - |

Picture D. The structure of CNN model for 29 layers