

# Curtin University – Department of Computing

## Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:		Student ID:	
Other name(s):			
Unit name:		Unit ID:	
Lecturer / unit coordinator:		Tutor:	
Date of submission:		Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: \_\_\_\_\_ Date of signature: \_\_\_\_\_

*(By submitting this form, you indicate that you agree with all the above text.)*

Curtin University

## COMP3010 Machine Learning Assignment Report

**Student:** Izan Muhammad

**Student ID:** 19831134

**Unit ID:** COMP3010

**Unit Coordinator:** Senjian An

## Contents

1. Task 1.....	pg. 3
2. Task 2.....	pg. 11
3. References.....	pg. 18
4. Source Code.....	pg. 19

# 1.Task 1: Building signage detection and digit extraction

## 1.1 Overview

The task of finding digits on the Curtin campus consisted of reliably identifying the consistent features associated with the digits. The following will explain which algorithms were used to be most suitable for the extraction of the digits.

## 1.2 Pre-processing

Pre-processing was performed to minimise noise and allow for easier detection of the digits within the image.

Figure 1.1: Pre-processing

```
def preprocessing(image):  
    #convert to hsv  
    hsv = cv.cvtColor(image.copy(), cv.COLOR_BGR2HSV)  
  
    #create upper and lower mask  
    hsv_lower = np.array([0,0,163])  
    hsv_upper = np.array([179,45,255])  
    mask = cv.inRange(hsv, hsv_lower, hsv_upper)  
  
    #remove mask from original image  
    image = cv.bitwise_and(image.copy(), image.copy(), mask=mask)  
  
    #dilate image  
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (3,3))  
    image = cv.dilate(image, kernel, iterations=1)  
  
    #convert to grayscale  
    image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)  
  
    #smooth  
    image = cv.GaussianBlur(image,(5,5),0)  
  
    return image
```

The image was resized to have a consistent width of 1000 pixels, with the height ratio being kept consistent. Afterwards, the image was converted to HSV, and a mask was applied to remove majority of the background (as shown in the figure).

Figure 1.2: Resizing

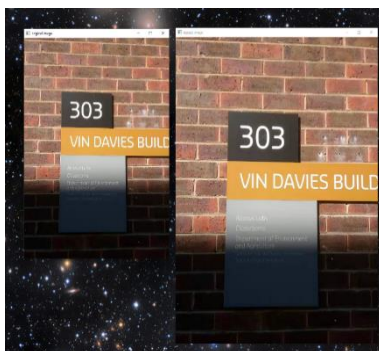
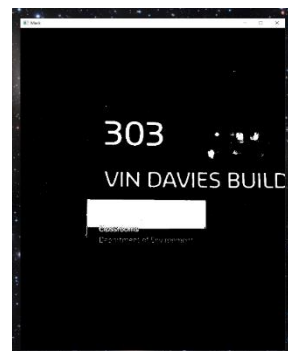
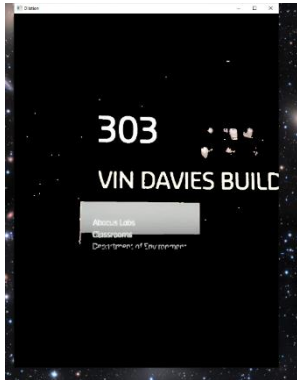


Figure 1.3: HSV Mask



Dilation was applied to the image to ensure all contours were connected, as there was an issue of certain digits showing as separate contours.

Figure 1.4: Dilation



The image was then converted to grayscale and smoothed for preparation to detect edge detection.

Figure 1.5: Grayscale



### 1.3 Edge Detection

I found canny edge detection to most suitable for the task as the white pixel of the digits had high contrast between black pixels of the sign which allowed for efficient edge detection of the digits.

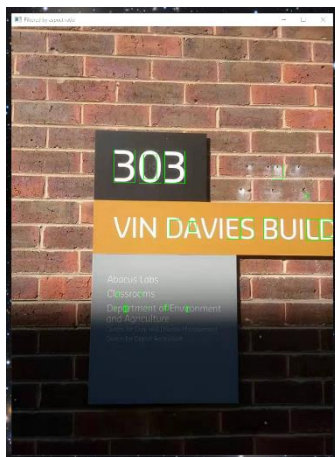
Figure 1.6: Canny edge detection



### 1.4 Filtering

Firstly, the detected contours were filtered by aspect ratio, as digit contours were observed to have a larger width to height ratio  $> 1$ .

Figure 1.7: Filtered by aspect ratio



Afterwards, the contours were filtered by Area, to ensure that the contour size was appropriate relative to the image area, as contours less than minimum threshold and greater than maximum threshold were assumed to not be a contour of a digit.

Figure 1.8: Filtered by area



Afterwards, contours which were located inside another contour were filtered out, this was performed as zero-digit inner edge was considered another contour.

Figure 1.9: Filter contour inside contour



Afterwards, contours were filtered by proximity to other contours, if a contour was not next to another contour it was removed. A contour is considered a neighbour if the contour's width is the radius of the contour and if there is a contour within that radius.

Figure 1.10: filter by neighbour contours



Finally, a while loop was used to filter all the remaining contours until 3 contours were remaining or if the contour count remained the same as the last while loop. Contours were filtered by relative maximum area (if any contour was less than the average of the 3 largest contours it was removed), relative minimum area (if any contour was greater than the average of the 3 largest contours it was removed), if contours overlapped, if contours were directly on the border of the image, if the contours height was larger than the average height (based on the height of the 3 largest contours).

Figure 1.11: Filter by maximum relative area



Figure 1.12: Filter by minimum relative area





Figure 1.13: Filter contours which are next to border



Figure 1.14: Filter by relative height

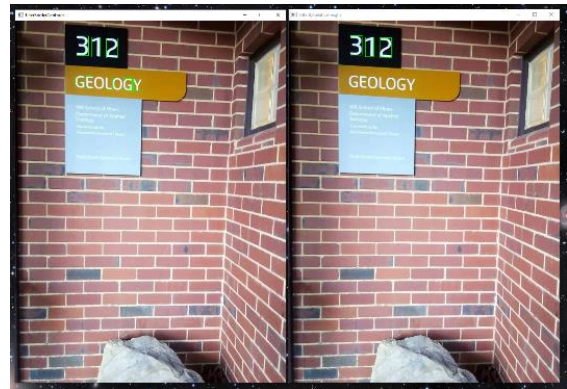
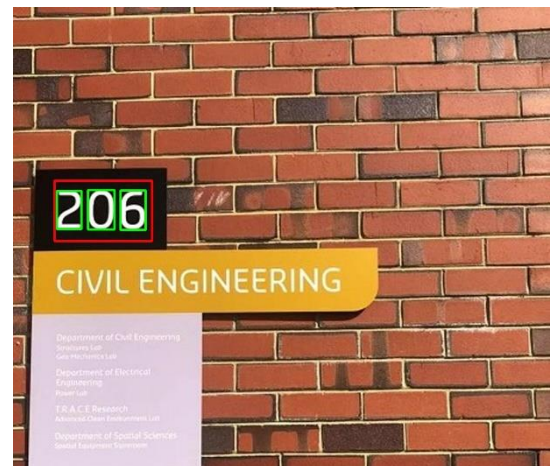
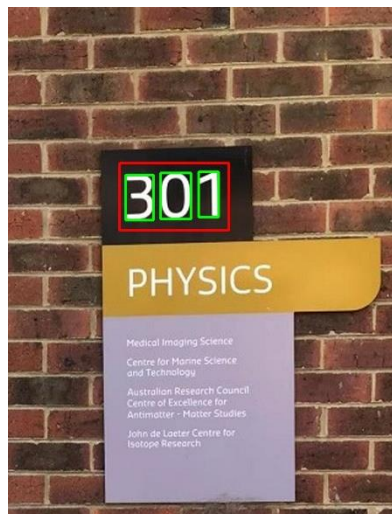


Figure 1.15: Final Digits and Signage



## 1.5 Testing dataset results







## 2. Task 2: Coral Image Classification

### 2.1 Overview

The chosen image classification algorithm used are Support vector Classifier and SoftMax regression. The features are extracted used the convolutional neural network ResNet50.

### 2.1 Dataset

The dataset is constructed using a custom Dataset class using the Dataset interface from torch.

Figure 2.1 : Coral Dataset

```
class CoralDataset(Dataset):
    def __init__(self, data, root_dir, transform=None):
        self.data = data
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        img_path = self.data[index, 0]
        image = cv.imread(img_path)
        #print(self.data[index, 1])

        y_label = torch.tensor(int(self.data[index, 1]))

        if self.transform:
            image = self.transform(image)

        return image, y_label
```

### 2.2 CNN with SoftMax

ResNet50 is used due its deep layers and use of residual blocks which allows for efficient training. Res-Net is known to work well for image classification. SoftMax regression is used for classification. Pre-trained model used and is then fine-tuned on coral dataset.

### 2.3 CNN with SoftMax Hyper-parameters

Testing and fine tuning of the hyper parameters was conducted on the pretrained model. Fine tuning was conducted by adjusting the hyperparameters of the model and recording the results.

## 2.4 Learning Rate

Learning rate	Training loss	Training accuracy	Validation Accuracy
0.001	0.026	0.994	0.998
0.0005	0.042	0.989	0.994
0.00025	0.116	0.961	0.988

Figure 2.2 : learning rate = 0.001

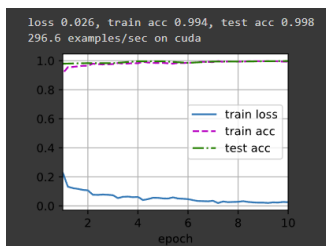


Figure 2.3 : learning rate = 0.0005

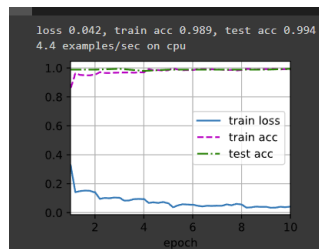
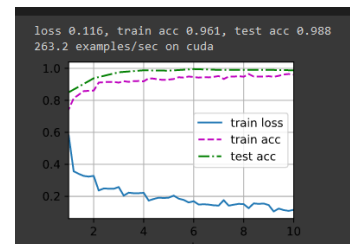


Figure 2.4 : learning rate = 0.00025



## 2.5 Batch Size

Batch size	Training loss	Training accuracy	Validation Accuracy
8	0.032	0.992	1.000
16	0.026	0.994	0.998
64	0.059	0.987	0.980

Figure 2.5 : batch size = 8

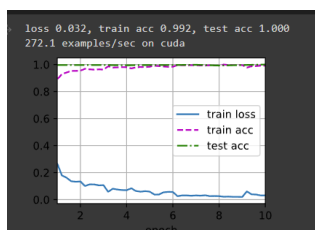


Figure 2.6 : batch size = 64

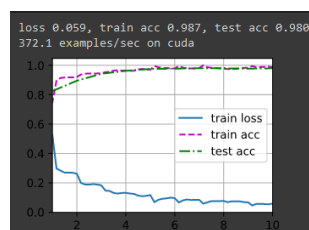
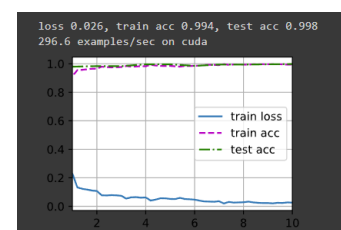


Figure 2.7 : batch size = 16



## 2.6 Epochs

Epochs	Training loss	Training accuracy	Validation Accuracy
5	0.067	0.979	0.991
10	0.026	0.994	0.998
20	0.040	0.987	0.998

Figure 2.8 : epochs = 10

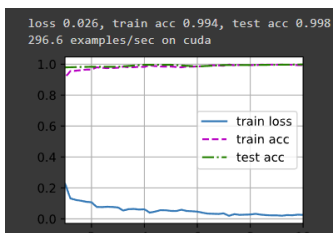


Figure 2.9 : epochs = 20

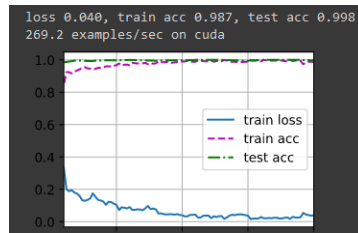
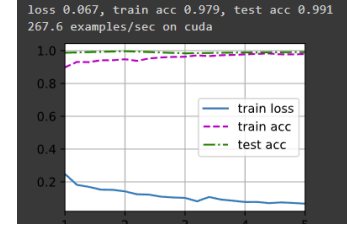


Figure 2.10 : epochs = 5



## 2.7 Final Hyper-parameters

Hyper parameters chosen are shown below:

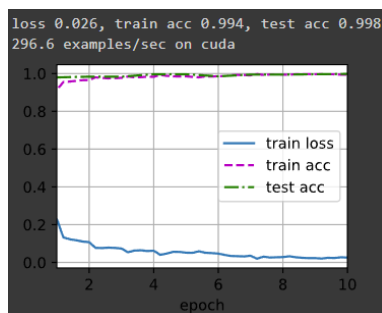
Learning Rate	0.001
Batch Size	16
Epochs	10

## 2.3 Training CNN with SoftMax

The following table shows the result of the training on the pre-trained model architecture using SoftMax

Training loss	Training accuracy	Validation Accuracy
0.026	0.994	0.998

Figure 2.11 : Final Results

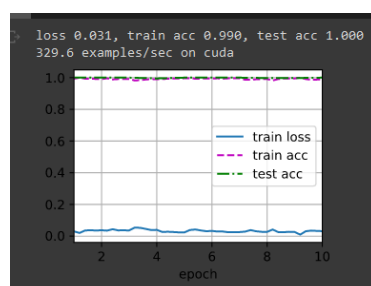


## 2.4 Training CNN with SoftMax on test dataset

The following table shows the result of the training on the pre-trained model architecture using SoftMax

Training loss	Training accuracy	Test Accuracy
0.031	0.990	1.00

Figure 2.12 : Final Results



## 2.9 Conclusion

As can be seen in the results, ResNet50 performed well in classifying the dataset.

## 2.3 HOG with SVM

HOG is used due to its invariance properties; Support vector classifier is used due to its efficiency in training. HOG feature descriptors are extracted using sklearn library.

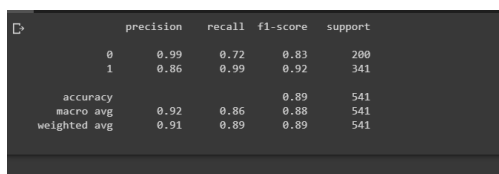
## 2.3 CNN with SVM Hyper-parameters

There did not seem to be a need to fine-tune the hyper-parameters from the default parameters already used as the accuracy on the validation dataset was 99%. The below table shows the accuracy with different hyper-parameters.

Gamma

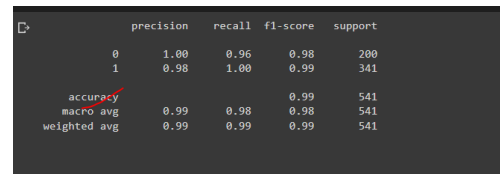
Gamma	Validation Accuracy
scale	0.99
auto	0.89

Figure 2.13 : Gamma: auto



	precision	recall	f1-score	support
0	0.99	0.72	0.83	200
1	0.86	0.99	0.92	341
accuracy			0.89	541
macro avg	0.92	0.86	0.88	541
weighted avg	0.91	0.89	0.89	541

Figure 2.14 : Gamma: scale



	precision	recall	f1-score	support
0	1.00	0.96	0.98	200
1	0.98	1.00	0.99	341
accuracy			0.99	541
macro avg	0.99	0.98	0.98	541
weighted avg	0.99	0.99	0.99	541

Kernel

Gamma	Validation Accuracy
Linear	0.99
RBF	0.97
Poly	0.99
sigmoid	0.16



Figure 2.15 : kernel: linear

	precision	recall	f1-score	support
0	1.00	0.96	0.98	200
1	0.98	1.00	0.99	341
accuracy			0.99	541
macro avg	0.99	0.98	0.98	541
weighted avg	0.99	0.99	0.99	541

Figure 2.16 : kernel: rbf

	precision	recall	f1-score	support
0	1.00	0.95	0.97	200
1	0.95	1.00	0.98	200
accuracy			0.97	400
macro avg	0.98	0.97	0.97	400
weighted avg	0.98	0.97	0.97	400

Figure 2.17 : kernel: poly

	precision	recall	f1-score	support
0	1.00	0.97	0.98	200
1	0.98	1.00	0.99	341
accuracy			0.99	541
macro avg	0.99	0.98	0.99	541
weighted avg	0.99	0.99	0.99	541

Figure 2.18 : kernel: sigmoid

	precision	recall	f1-score	support
0	0.00	0.00	0.00	200
1	0.30	0.25	0.27	341
accuracy			0.16	541
macro avg	0.15	0.13	0.14	541
weighted avg	0.19	0.16	0.17	541

## C (Regularization Parameter)

C	Validation Accuracy
0.1	0.96
1	0.99
2	0.98

Figure 2.19 : C: 0.1

	precision	recall	f1-score	support
0	0.98	0.94	0.96	200
1	0.94	0.98	0.96	200
accuracy			0.96	400
macro avg	0.96	0.96	0.96	400
weighted avg	0.96	0.96	0.96	400

Figure 2.20 : C: 1

	precision	recall	f1-score	support
0	1.00	0.96	0.98	200
1	0.98	1.00	0.99	341
accuracy			0.99	541
macro avg	0.99	0.98	0.98	541
weighted avg	0.99	0.99	0.99	541

Figure 2.21 : C: 2

	precision	recall	f1-score	support
0	1.00	0.96	0.98	200
1	0.96	1.00	0.98	200
accuracy			0.98	400
macro avg	0.98	0.98	0.98	400
weighted avg	0.98	0.98	0.98	400

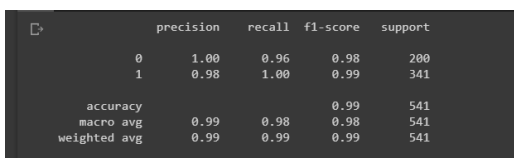
## 2.4 Training HOG with SVM

Final Parameters

Gamma	Scale
Kernel	RBF
C	1

Below figure shows the accuracy of the model on the validation dataset:

Figure 2.22 : Accuracy on validation dataset



```

precision    recall  f1-score   support

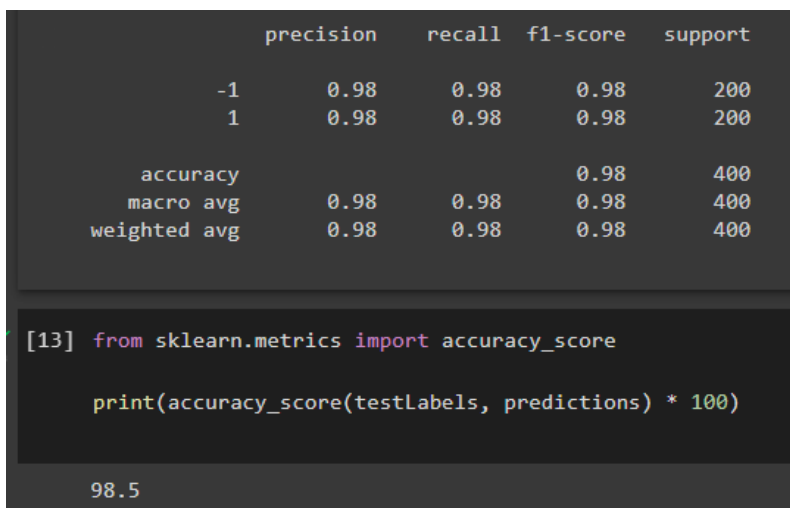
      0       1.00      0.96      0.98        200
      1       0.98      1.00      0.99        341

 accuracy          0.99
 macro avg          0.99
 weighted avg       0.99
```

## 2.5 Training HOG with SVM on test dataset

Below figure shows the accuracy of the model on the validation dataset:

Figure 2.23 : Accuracy on validation dataset



```

precision    recall  f1-score   support

      -1       0.98      0.98      0.98        200
       1       0.98      0.98      0.98        200

 accuracy          0.98
 macro avg          0.98
 weighted avg       0.98
```

```
[13] from sklearn.metrics import accuracy_score

      print(accuracy_score(testLabels, predictions) * 100)
```

98.5

### 2.5 Comparison

As can be seen, both models performed well but CNN provided a slighter higher result. However, SVM is effective in training the model in a shorter time frame. Therefore, since CNN provided a higher accuracy rate, it would be more suitable to use CNN with SoftMax.

### 3. References

**HOG with SVM classification:** <https://www.kaggle.com/code/manikg/training-svm-classifier-with-hog-features/notebook>

## 4. Source Code

### Task 1:

```
# -*- coding: utf-8 -*-
"""Part1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1Baz-1c7oW4WZ4F7r6AUxNHvp8AbtMAi1
"""

#Code to connect your google drive with google colaboratory
from google.colab import drive
drive.mount('/content/drive/')

import cv2 as cv
import numpy as np
import os
import random
from google.colab.patches import cv2_imshow

#Preparing the dataset
imagePath = []

trainingExamples =
'drive/MyDrive/MPAssignment/Data/BuildingSignageDetection/train'

for filename in os.listdir(trainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        imagePath.append(os.path.join(trainingExamples, filename))

valExamples = 'drive/MyDrive/MPAssignment/Data/BuildingSignageDetection/val'

for filename in os.listdir(valExamples):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        imagePath.append(os.path.join(valExamples, filename))

testingImagePath = []

directory = 'drive/MyDrive/MPAssignment/Testing
Data_withLabels/BuildingSinonageDetection'
```

```

for filename in os.listdir(directory):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        testingImagePath.append(os.path.join(directory, filename))

for f in testingImagePath:
    print(f)

#preprocesssing
def preprocessing(image):

    #convert to hsv
    hsv = cv.cvtColor(image.copy(), cv.COLOR_BGR2HSV)

    #create upper and lower mask
    hsv_lower = np.array([0,0,163])
    hsv_upper = np.array([179,45,255])
    mask = cv.inRange(hsv, hsv_lower, hsv_upper)

    #remove mask from original image
    image = cv.bitwise_and(image.copy(), image.copy(), mask=mask)

    #dialate images
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (3,3))
    image = cv.dilate(image , kernel, iterations=1)

    #convert to grayscale
    image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    #smooth
    image = cv.GaussianBlur(image,(5,5),0)

    return image

def showContours(image, contours, text="Output Contours"):
    #copy image
    output_image = image.copy()

    #loop through contours
    for contour in contours:

        #get x,y,w,h of contour
        x, y, w, h = cv.boundingRect(contour)

        #draw contour on image
        cv.rectangle(output_image, (x, y), (x+w, y+h), (0, 255, 0), 1)

    #show image

```

```

cv.imshow(text,output_image)

def filterByAspectRatio(contours):

    filtered = []

    #loop through contours
    for contour in contours:

        #get width and height
        _, _, w, h = cv.boundingRect(contour)

        #calculate ratio
        hw_ratio = h / w

        #target ratio
        targetHeightWidthRatio = hw_ratio >= 1.3 and hw_ratio <= 3

        #check if ratio within target
        if targetHeightWidthRatio:
            filtered.append(contour)

    return filtered

def filterByArea(contours, threshold):

    filtered = []

    #loop through contours
    for contour in contours:

        #get width and height
        _, _, w, h = cv.boundingRect(contour)

        #calculate contour area
        contour_area = w*h

        #threshold
        minimum_area = threshold * 0.001
        maximum_area = threshold * 0.02

        #check area within threshold
        if maximum_area > contour_area > minimum_area:
            filtered.append(contour)

    return filtered

def filterContourInsideContour(contours,hierarchy):

```

```

filtered = []
index = 0
stats = []

#loop through contours
for contour in contours:

    #calculate center of contour
    M = cv.moments(contour)
    try:
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
    except:

        #if center not detectable set to -1
        cX = -1
        cY = -1

    stats.append([index,cX,cY])

    index = index + 1

duplicate = []
keep = []
duplicateFound = False

#loop through contours
for stat in stats:

    #if stat not already to be found a duplicate
    if stat[0] not in duplicate:

        for second_stat in stats:

            #check if not the same contour
            if stat[0] != second_stat[0]:

                #check if within x-axis
                if stat[1] - 2 < second_stat[1] < stat[1] + 2:
                    #check if within y-axis
                    if stat[2] - 2 < second_stat[2] < stat[2] + 2:
                        #check that second contour not in keep
                        if second_stat[0] not in keep:
                            #append
                            duplicate.append(second_stat[0])

            #keep larger contour

```



```

        if stat[0] not in keep:
            keep.append(stat[0])

        duplicateFound = True

    if not duplicateFound:
        if stat[0] not in keep:
            keep.append(stat[0])
        duplicateFound = False

for stat in stats:
    if stat[0] not in duplicate:
        if stat[0] not in keep:
            keep.append(stat[0])

for k in keep:

    filtered.append(contours[k])

return filtered

def hasNeighbourHorizontally(contours):
    filtered = []

    index = 0
    stats = []
    for contour in contours:

        #calculate coordinates of contour
        x, y, w, h = cv.boundingRect(contour)
        M = cv.moments(contour)
        try:
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
        except:
            cX = -1
            cY = -1

        stats.append([index, cX, cY, x, y, w, h])

        index = index + 1

    hasNeighbour = []
    for stat in stats:
        for second_stat in stats:
            if (stat[0] != second_stat[0]):

```

```

        #check for contours neighbour
        if((stat[1] < second_stat[1] and second_stat[1] <
stat[1]+stat[5]*3) or stat[1]-stat[5]*3 < second_stat[1] and second_stat[1] <
stat[1]):
            if((stat[4] < second_stat[2] and second_stat[2] <
stat[4]+stat[6])):
                hasNeighbour.append([stat[0],second_stat[0]])

filteredHasNeighbour = []
for h in hasNeighbour:
    if h not in filteredHasNeighbour:
        filteredHasNeighbour.append(h)

finalFilter = []

#make sure other contour considers it a neighbour aswell
for h in filteredHasNeighbour:

    c1 = h[0]
    n1 = h[1]

    if [c1,n1] not in filteredHasNeighbour:
        print("does not exist")
    else:
        finalFilter.append(c1)

filteredFinalFilter = []

for h in finalFilter:
    if h not in filteredFinalFilter:
        filteredFinalFilter.append(h)

finalFilter = filteredFinalFilter

for h in finalFilter:
    filtered.append(contours[h])

return filtered

def inSecondContour(second_stat,x1,y1,x2,y2,x3,y3,x4,y4):

    inside = False

    checkXY = [[x1,y1],[x2,y2],[x3,y3],[x4,y4]]

    #check all 4 points to see if in contour

```

```

for xy in checkXY:

    if xy[0] > second_stat[1] and second_stat[1]+second_stat[3] > xy[0]:

        if xy[1] > second_stat[2] and second_stat[2]+second_stat[4] >
xy[1]:
            inside = True

    return inside

def removeOverlappingContours(contours):
    filtered = []

    index = 0
    stats = []
    for contour in contours:

        #get stats of contour
        x, y, w, h = cv.boundingRect(contour)

        stats.append([index,x,y,w,h])

        index = index + 1

    filteredContours = []
    exclude = []

    #check stats of contour against other contours
    for stat in stats:

        x1 = stat[1]
        y1 = stat[2]

        x2 = stat[1] + stat[3]
        y2 = stat[2]

        x3 = stat[1]
        y3 = stat[2] + stat[4]

        x4 = stat[1] + stat[3]
        y4 = stat[2] + stat[4]

        for second_stat in stats:
            if stat[0] != second_stat[0]:

                if not inSecondContour(second_stat,x1,y1,x2,y2,x3,y3,x4,y4):

```

```

        filteredContours.append(stat[0])
    else:
        exclude.append(stat[0])

f = []
for c in filteredContours:
    if c not in f and c not in exclude:
        f.append(c)

filteredContours = f

for c in filteredContours:
    filtered.append(contours[c])

return filtered

def filterBorderContours(contours,image_width,image_height):

    filtered = []

    index = 0
    stats = []
    for contour in contours:

        #get stats
        x, y, w, h = cv.boundingRect(contour)

        stats.append([index,x,y,w,h])

        index = index + 1

    exclude = []

    for stat in stats:

        #touching left hand side
        if stat[1] <= 0:
            exclude.append(stat[0])

        #touching top
        if stat[2] <= 0:
            exclude.append(stat[0])

        #touching right hand side
        if image_width <= stat[1]+stat[3]:
            exclude.append(stat[0])

```

```

        #touching bottom
        if image_height <= stat[2]+stat[4]:
            exclude.append(stat[0])

    for stat in stats:
        if stat[0] not in exclude:
            filtered.append(contours[stat[0]])

    return filtered

def grab_second(index):
    return index[1]

def filterByMaximumRelativeArea(contours, tolerance=1.5):
    filtered = []

    index = 0

    stats = []
    for contour in contours:

        #get stats
        x, y, w, h = cv.boundingRect(contour)
        area = w*h

        stats.append([index,area])

        index = index + 1

    stats.sort(key=grab_second, reverse=True)

    #get the 3 largest contours
    largest3 = [stats[0],stats[1],stats[2]]

    #calculate average
    average = 0
    for stat in largest3:
        average = average + stat[1]

    average = int(average/3)

    maximum_average = int(average * tolerance)

    filteredContours = []

    for contour in contours:

```

```

        x, y, w, h = cv.boundingRect(contour)
        area = w*h

        #check within max average
        if area <= maximum_average:

            filtered.append(contour)

    return filtered

def filterByMinimumRelativeArea(contours, tolerance="0.9"):
    filtered = []

    index = 0

    stats = []
    for contour in contours:

        #get stats
        x, y, w, h = cv.boundingRect(contour)
        area = w*h

        stats.append([index,area])

        index = index + 1

    stats.sort(key=grab_second, reverse=True)

    #get largest
    largest3 = [stats[0],stats[1],stats[2]]

    #calculate average
    average = 0
    for stat in largest3:
        average = average + stat[1]

    average = int(average/3)

    min_average = int(average * tolerance)

    filteredContours = []
    for contour in contours:
        x, y, w, h = cv.boundingRect(contour)
        area = w*h

```

```

        #check larger than min average
        if min_average <= area:
            filtered.append(contour)

    return filtered

def FilterByRelativeHeight(contours,tolerance=1.5):

    filtered = []

    index = 0

    stats = []
    for contour in contours:

        #get stats
        x, y, w, h = cv.boundingRect(contour)
        stats.append([index,w,h])

        index = index + 1

    stats.sort(key=grab_second, reverse=True)

    #get 3 largest contours
    largest3 = [stats[0],stats[1],stats[2]]

    #calculate average height
    average = 0
    for stat in largest3:
        average = average + stat[2]

    average = int(average/3)

    max_average = int(average * tolerance)

    filteredContours = []
    for contour in contours:
        x, y, w, h = cv.boundingRect(contour)

        #check height within max average height
        if h < max_average:
            filtered.append(contour)

    return filtered

def final3Selection(contours):

    filtered = []

```

```

index = 0

stats = []
for contour in contours:

    #get stats
    x, y, w, h = cv.boundingRect(contour)
    area = w*h

    stats.append([index,h,y])

    index = index + 1

filteredHeight = []
mostCommonContour = []
max_counter = 0

#check which is the most common height
for stat in stats:
    counter = 0
    for second_stat in stats:
        if stat[0] != second_stat[0]:
            if stat[1]-(stat[1]*0.25) < second_stat[1] <
stat[1]+(stat[1]*0.25):
                counter = counter + 1

    if counter > max_counter:
        mostCommonContour = stat
        max_counter = counter

index = 0

#filter by most common height
for contour in contours:
    x, y, w, h = cv.boundingRect(contour)

    tolerance = mostCommonContour[1]*0.25
    if mostCommonContour[1]- tolerance < h < mostCommonContour[1] +
tolerance:
        filteredHeight.append(index)

    index = index + 1

filteredYAxis = []
mostCommonContour = []
max_counter = 0

```



```

#check which is the most common y-axis value
for stat in stats:
    counter = 0
    for second_stat in stats:

        if stat[0] != second_stat[0]:

            if stat[2]-(stat[2]*0.25) < second_stat[2] <
stat[2]+(stat[2]*0.25):
                counter = counter + 1

    if counter > max_counter:
        mostCommonContour = stat
        max_counter = counter

index = 0

#filter contours by y-axis value
for contour in contours:
    x, y, w, h = cv.boundingRect(contour)

    tolerance = mostCommonContour[2]*0.25
    if mostCommonContour[2]- tolerance < y < mostCommonContour[2] +
tolerance:
        filteredYAxis.append(index)

    index = index + 1

for i in filteredHeight:
    if i in filteredYAxis:
        filtered.append(contours[i])

#if length of contours is greater than 3 then grab the 3 largest
if len(filtered) > 3:

    stats = []
    index = 0
    for contour in filtered:
        x, y, w, h = cv.boundingRect(contour)
        area = w*h

        stats.append([index,area])
        index = index+1

    stats.sort(key=grab_second, reverse=True)

```

```

        largest3 = [stats[0],stats[1],stats[2]]

        fFiltered = []
        for c in largest3:
            fFiltered.append(filtered[c[0]])

        filtered = fFiltered

    return filtered

def extractDigits(original_image, contours):
    digits = []
    sign = []

    minX = -1
    maxX = -1
    minY = -1
    maxY = -1
    for contour in contours:
        image = original_image.copy()
        x, y, w, h = cv.boundingRect(contour)

        if minX == -1 or x < minX:
            minX = x

        if maxX == -1 or x+w > maxX:
            maxX = x+w

        if minY == -1 or y < minY:
            minY = y

        if maxY == -1 or y+h > maxY:
            maxY = y+h

        digit = image[y:y+h,x:x+w]
        digits.append(digit)

    image = original_image.copy()

    #add buffer
    minX = int(minX * 0.95)
    minY = int(minY * 0.95)
    maxX = int(maxX * 1.05)
    maxY = int(maxY * 1.05)
    sign = image[minY:maxY,minX:maxX]

    i = 0
    for digit in digits:

```

```

        text = "Digit: {}".format(i)
        cv2_imshow(digit)
        i = i + 1

    cv2_imshow(sign)

    return digits, sign

def drawDigits(original_image, contours):
    digits = []
    sign = []
    image = original_image.copy()

    minX = -1
    maxX = -1
    minY = -1
    maxY = -1
    for contour in contours:

        x, y, w, h = cv.boundingRect(contour)

        if minX == -1 or x < minX:
            minX = x

        if maxX == -1 or x+w > maxX:
            maxX = x+w

        if minY == -1 or y < minY:
            minY = y

        if maxY == -1 or y+h > maxY:
            maxY = y+h

        cv.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 1)

    #add buffer
    minX = int(minX * 0.95)
    minY = int(minY * 0.95)
    maxX = int(maxX * 1.05)
    maxY = int(maxY * 1.05)

    cv.rectangle(image, (minX, minY), (maxX, maxY), (0, 0, 255), 1)

    cv2_imshow(image)

    return image

```

```

#detectBuildingNumber

def detectBuildingNumber(image, refine=False):

    original_image = image.copy()
    image_width = image.shape[0]
    image_height = image.shape[1]
    image_area = image_width * image_height
    backup_contours = []
    skip_rest = False

    image = preprocessing(image)

    #cv.imshow("processed image",image)

    canny_output = cv.Canny(image, 150, 255)

    #cv.imshow("edges ",canny_output)

    if refine:
        #sharpen image
        kernel = np.array([[0, -1, 0],
                           [-1, 5, -1],
                           [0, -1, 0]])
        image_sharp = cv.filter2D(src=image, ddepth=-1, kernel=kernel)

        #re-run canny
        image = preprocessing(image)

        canny_output = cv.Canny(image, 200, 255)

    #get contours
    contours, hierarchy = cv.findContours(canny_output, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)

    #showContours(original_image,contours,text="unprocessed")
    backup_contours = contours

    contours = filterByAspectRatio(contours)

    if(len(contours) < 3):
        contours = backup_contours
        skip_rest = True

    backup_contours = contours

```

```

#showContours(original_image,contours,text="Filtered by aspect ratio")
if not skip_rest:
    contours = filterByArea(contours,image_area)

if(len(contours) < 3):
    contours = backup_contours
    skip_rest = True

backup_contours = contours

#showContours(original_image,contours,text="filterByArea")
if not skip_rest:
    contours = filterContourInsideContour(contours,hierarchy)

if(len(contours) < 3):
    contours = backup_contours
    skip_rest = True

backup_contours = contours

#showContours(original_image,contours,text="filterContourInsideContour")

if not skip_rest:
    contours = hasNeighbourHorizontally(contours)

#showContours(original_image,contours,text="has Neighbours (just before
relative checking)")

if(len(contours) < 3):
    contours = backup_contours
    skip_rest = True

previousNumberOfContours = -1

#print("Starting relative checking")

while not skip_rest:

    if not skip_rest:
        contours = filterByMaximumRelativeArea(contours, tolerance=1.5)

    #showContours(original_image,contours,text="1")

    if(len(contours) < 3):
        contours = backup_contours
        skip_rest = True

```

```

    if(len(contours) == 3):
        skip_rest = True

    backup_contours = contours

    if not skip_rest:
        contours = hasNeighbourHorizontally(contours)

    #showContours(original_image,contours,text="has neighbour 2")

    if(len(contours) < 3):
        contours = backup_contours
        skip_rest = True
    if(len(contours) == 3):
        skip_rest = True

    backup_contours = contours

    if not skip_rest:
        contours = removeOverlappingContours(contours)

    if(len(contours) < 3):
        contours = backup_contours
        skip_rest = True
    if(len(contours) == 3):
        skip_rest = True

    backup_contours = contours

    #showContours(original_image,contours,text="removeOverlappingContours"
)

    if not skip_rest:
        contours = hasNeighbourHorizontally(contours)

    #showContours(original_image,contours,text="has neighbour 3")

    if(len(contours) < 3):
        contours = backup_contours
        skip_rest = True
    if(len(contours) == 3):
        skip_rest = True

    backup_contours = contours

    if not skip_rest:
        contours = filterBorderContours(contours,image_width,image_height)

```

```

if(len(contours) < 3):
    contours = backup_contours
    skip_rest = True
if(len(contours) == 3):
    skip_rest = True

backup_contours = contours

#showContours(original_image,contours,text="filterBorderContours")

if not skip_rest:
    contours = filterByMinimumRelativeArea(contours, tolerance=0.6)

#showContours(original_image,contours,text="2")

if(len(contours) < 3):
    contours = backup_contours
    skip_rest = True
if(len(contours) == 3):
    skip_rest = True

backup_contours = contours

if not skip_rest:
    contours = hasNeighbourHorizontally(contours)

#showContours(original_image,contours,text="has neighbour 4")

if(len(contours) < 3):
    contours = backup_contours
    skip_rest = True
if(len(contours) == 3):
    skip_rest = True

backup_contours = contours

if not skip_rest:
    contours = FilterByRelativeHeight(contours, tolerance = 1.2)

#showContours(original_image,contours,text="4")

if(len(contours) < 3):
    contours = backup_contours
    skip_rest = True
if(len(contours) == 3):
    skip_rest = True

backup_contours = contours

```

```

        if len(contours) == previousNumberOfContours:
            skip_rest = True
            previousNumberOfContours = -1
        else:
            previousNumberOfContours = len(contours)

    #showContours(original_image,contours,text="hasNeighbour")

    #print("filtered contour length : ",len(contours))

    contours = final3Selection(contours)

    #showContours(original_image,contours,text="Final 3")

    digits, sign = extractDigits(original_image,contours)

    image = drawDigits(original_image,contours)

    return image

#read all images and detect the building numbers
for path in imagePath:

    print(path)

    image = cv.imread(path)

    width = image.shape[0]
    height = image.shape[1]

    ratio = height/width

    newWidth = 1000
    newHeight = int(newWidth * ratio)

    resized_image = cv.resize(image, (newHeight,newWidth),
interpolation=cv.INTER_CUBIC)

    #cv.imshow(path,resized_image)

    detectBuildingNumber(resized_image)

#read all images and detect the building numbers

```



```

for path in testingImagePath:

    print(path)

    image = cv.imread(path)

    width = image.shape[0]
    height = image.shape[1]

    ratio = height/width

    newWidth = 1000
    newHeight = int(newWidth * ratio)

    resized_image = cv.resize(image, (newHeight,newWidth),
interpolation=cv.INTER_CUBIC)

    image = detectBuildingNumber(resized_image)

```

## Task 2: HOG with SVM

```

# -*- coding: utf-8 -*-
"""Copy of Part2_Svm_Hog_Refined_Final.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1_Y7hMsEmrA_EJcd47pF5qh29aZ10YUqs
"""

#Code to connect your google drive with google colaboratory
from google.colab import drive
drive.mount('/content/drive/')

import os
from skimage.feature import hog
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
import numpy as np
from sklearn.metrics import classification_report
import cv2 as cv
from scipy.stats import norm

#Prepare the dataset

```

```

#train dataset

positiveTrainImagePath = []

positiveTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/coralImageClassification/train/ImagesL
abelledPositive'

for filename in os.listdir(positiveTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpeg"):
        positiveTrainImagePath.append(os.path.join(positiveTrainingExamples,
filename))

positiveTrainImagePath.sort(key=lambda f: int(''.join(filter(str.isdigit,
f))))

negativeTrainImagePath = []

negativeTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/coralImageClassification/train/Images
LabelledNegative'

for filename in os.listdir(negativeTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpeg"):
        negativeTrainImagePath.append(os.path.join(negativeTrainingExamples,
filename))

negativeTrainImagePath.sort(key=lambda f: int(''.join(filter(str.isdigit,
f))))

#val dataset

positiveValImagePath = []

positiveTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/coralImageClassification/val/ImagesL
abelledPositive'

for filename in os.listdir(positiveTrainingExamples):

```

```

    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpg"):
        positiveValImagePath.append(os.path.join(positiveTrainingExamples,
filename))

positiveValImagePath.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))

negativeValImagePath = []

negativeTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/coralImageClassification/val/ImagesL
abelledNegative'

for filename in os.listdir(negativeTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpg"):
        negativeValImagePath.append(os.path.join(negativeTrainingExamples,
filename))

negativeValImagePath.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))

positiveTestingImagePath = []

positiveTrainingExamples = 'drive/MyDrive/MPAssignment/Testing
Data_withLabels/CoralImageClassification/Coral Images'

for filename in os.listdir(positiveTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpg"):
        positiveTestingImagePath.append(os.path.join(positiveTrainingExamples,
filename))

positiveTestingImagePath.sort(key=lambda f: int(''.join(filter(str.isdigit,
f))))

negativeTestingImagePath = []

negativeTrainingExamples = 'drive/MyDrive/MPAssignment/Testing
Data_withLabels/CoralImageClassification/Non-Coral Images'

for filename in os.listdir(negativeTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpg"):
        negativeTestingImagePath.append(os.path.join(negativeTrainingExamples,
filename))

```

```

negativeTestingImagePath .sort(key=lambda f: int(''.join(filter(str.isdigit,
f))))

# compute HOG features and label them:

trainData = []
trainLabels = []

for file in positiveTrainImagePath: #this loop enables reading the files in
the pos_im_listing variable one by one
    img = cv.imread(file) # open the file
    img = cv.resize(img, (128*4, 64*4))
    # calculate HOG for positive features
    fd, hog_image = hog(img, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, multichannel=True)# fd= feature
descriptor
    trainData.append(fd)
    trainLabels.append(1)

for file in negativeTrainImagePath: #this loop enables reading the files in
the pos_im_listing variable one by one
    img = cv.imread(file) # open the file
    img = cv.resize(img, (128*4, 64*4))
    # calculate HOG for positive features
    fd, hog_image = hog(img, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, multichannel=True)# fd= feature
descriptor
    trainData.append(fd)
    trainLabels.append(-1)

valData = []
valLabels = []

for file in positiveValImagePath: #this loop enables reading the files in the
pos_im_listing variable one by one
    img = cv.imread(file) # open the file
    img = cv.resize(img, (128*4, 64*4))
    # calculate HOG for positive features
    fd, hog_image = hog(img, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, multichannel=True)# fd= feature
descriptor
    valData.append(fd)
    valLabels.append(1)

for file in negativeValImagePath: #this loop enables reading the files in the
pos_im_listing variable one by one
    img = cv.imread(file) # open the file
    img = cv.resize(img, (128*4, 64*4))

```

```

    # calculate HOG for positive features
    fd, hog_image = hog(img, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, multichannel=True)# fd= feature
descriptor
    valData.append(fd)
    valLabels.append(-1)

testData = []
testLabels = []

for file in positiveTestingImagePath: #this loop enables reading the files in
the pos_im_listing variable one by one
    img = cv.imread(file) # open the file
    img = cv.resize(img, (128*4, 64*4))
    # calculate HOG for positive features
    fd, hog_image = hog(img, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, multichannel=True)# fd= feature
descriptor
    testData.append(fd)
    testLabels.append(1)

for file in negativeTestingImagePath: #this loop enables reading the files in
the pos_im_listing variable one by one
    img = cv.imread(file) # open the file
    img = cv.resize(img, (128*4, 64*4))
    # calculate HOG for positive features
    fd, hog_image = hog(img, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True, multichannel=True)# fd= feature
descriptor
    testData.append(fd)
    testLabels.append(-1)

#based on: https://www.kaggle.com/code/manikg/training-svm-classifier-with-hog-features/notebook

#convert to np array
trainData = np.array(trainData)
valData = np.array(valData)

#svm
model = SVC(gamma='scale',kernel='rbf', C=1)

model.fit(trainData, trainLabels)

#predict on validation data
predictions = model.predict(valData)
print(classification_report(valLabels, predictions))

```

```

from sklearn.metrics import accuracy_score

print(accuracy_score(valLabels, predictions) * 100)

#predict on validation data
predictions = model.predict(testData)
print(classification_report(testLabels, predictions))

print(accuracy_score(testLabels, predictions) * 100)

with open(r'drive/MyDrive/MPAssignment/Testing
Data_withLabels/CorallImageClassification/predictedLabels.txt', 'w') as fp:
    for item in predictions: ## labels is a list of the predicted labels, 1
    for corals, -1 for non-corals
        # write each item on a new line
        fp.write("%s\n" % item)

```

## Task 2: CNN with SoftMax

```

# -*- coding: utf-8 -*-
"""Part2_CNN_ResNet50_Softmax_Final.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1D4hoE_rmuhptIbYCBKm-eTC3YS4VwYB7
"""

#Code to connect your google drive with google colaboratory
from google.colab import drive
drive.mount('/content/drive/')

!pip install d2l
!pip install matplotlib==3.1.3
!pip install matplotlib_inline

import os
import torch
import torchvision
from torch import nn
from torch.utils.data import (DataLoader, Dataset)
from d2l import torch as d2l
import torchvision.transforms as transforms
import numpy as np
import cv2 as cv

```

```

#Prepare the dataset

#train dataset

trainImagePath = []

positiveTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/CoralImageClassification/train/ImagesL
abelledPositive'

for filename in os.listdir(positiveTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpeg"):
        trainImagePath.append([os.path.join(positiveTrainingExamples,
filename),1],)

negativeTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/CoralImageClassification/train/ImagesL
abelledNegative'

for filename in os.listdir(negativeTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpeg"):
        trainImagePath.append([os.path.join(negativeTrainingExamples,
filename),0],)

for path in trainImagePath:
    print(path[0],path[1])

#val dataset

valImagePath = []

positiveTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/CoralImageClassification/val/ImagesL
abelledPositive'

for filename in os.listdir(positiveTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpeg"):
        valImagePath.append([os.path.join(positiveTrainingExamples, filename),1],)

```

```

negativeTrainingExamples =
'/content/drive/MyDrive/MPAssignment/Data/CoralImageClassification/val/ImagesL
abelledNegative'

for filename in os.listdir(negativeTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpeg"):
        valImagePath.append([os.path.join(negativeTrainingExamples, filename),0],)

trainImagePath = np.array(trainImagePath)
valImagePath = np.array(valImagePath)

# Hyperparameters
num_classes = 2
learning_rate = 0.001
batch_size = 16
num_epochs = 10
num_folds = 3

class CoralDataset(Dataset):
    def __init__(self, data, root_dir , transform=None):
        self.data = data
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        img_path = self.data[index, 0]
        image = cv.imread(img_path)
        #print(self.data[index, 1])

        y_label = torch.tensor(int(self.data[index, 1]))

        if self.transform:
            image = self.transform(image)

        return(image, y_label)

# z-scale the data
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225])

preprocess = transforms.Compose([

```



```

        transforms.ToPILImage(),
        transforms.Resize((224, 224)),
        transforms.ToTensor(), # also scales to [0, 1]
        normalize
    ])

#create dataset
train_dataset = CoralDataset(data = trainImagePath, root_dir="", transform =
preprocess)

train_loader = DataLoader(dataset=train_dataset,
batch_size=batch_size,shuffle=True)

val_dataset = CoralDataset(data = valImagePath, root_dir="", transform =
preprocess)

val_loader = DataLoader(dataset=val_dataset,
batch_size=batch_size,shuffle=True)

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

finetune_net = torchvision.models.resnet18(pretrained=True)
finetune_net.fc = nn.Linear(finetune_net.fc.in_features, 2)
nn.init.xavier_uniform_(finetune_net.fc.weight);

#from D2L book
def evaluate_accuracy_gpu(net, data_iter, device=None):
    """Compute the accuracy for a model on a dataset using a GPU."""
    if isinstance(net, nn.Module):
        net.eval() # Set the model to evaluation mode
        if not device:
            device = next(iter(net.parameters())).device
    # No. of correct predictions, no. of predictions
    metric = d2l.Accumulator(2)

    with torch.no_grad():
        for X, y in data_iter:
            if isinstance(X, list):
                # Required for BERT Fine-tuning (to be covered later)
                X = [x.to(device) for x in X]
            else:
                X = X.to(device)
            y = y.to(device)
            metric.add(d2l.accuracy(net(X), y), y.numel())
    return metric[0] / metric[1]

#from D2L book

```

```

def train_ch6(net, train_iter, test_iter, num_epochs, lr, device):
    """Train a model with a GPU (defined in Chapter 6)."""

    print('training on', device)
    net.to(device)
    optimizer = torch.optim.SGD(net.parameters(), lr=lr)
    loss = nn.CrossEntropyLoss()
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs],
                           legend=['train loss', 'train acc', 'test acc'])
    timer, num_batches = d2l.Timer(), len(train_iter)
    for epoch in range(num_epochs):
        # Sum of training loss, sum of training accuracy, no. of examples
        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate(train_iter):
            timer.start()
            optimizer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            l.backward()
            optimizer.step()
            with torch.no_grad():
                metric.add(l * X.shape[0], d2l.accuracy(y_hat, y), X.shape[0])
            timer.stop()
            train_l = metric[0] / metric[2]
            train_acc = metric[1] / metric[2]
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (train_l, train_acc, None))
            test_acc = evaluate_accuracy_gpu(net, test_iter)
            animator.add(epoch + 1, (None, None, test_acc))
        print(f'loss {train_l:.3f}, train acc {train_acc:.3f}, '
              f'test acc {test_acc:.3f}')
        print(f'{metric[2] * num_epochs / timer.sum():.1f} examples/sec '
              f'on {str(device)}')

#run train method
train_ch6(finetune_net, train_loader, val_loader, num_epochs, learning_rate,
device)

positiveTestingImagePath = []

positiveTrainingExamples = 'drive/MyDrive/MPAssignment/Testing
Data_withLabels/CoralImageClassification/Coral Images'

for filename in os.listdir(positiveTrainingExamples):

```

```

    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpg"):
        positiveTestingImagePath.append(os.path.join(positiveTrainingExamples,
filename))

positiveTestingImagePath.sort(key=lambda f: int(''.join(filter(str.isdigit,
f))))

negativeTestingImagePath = []

negativeTrainingExamples = 'drive/MyDrive/MPAssignment/Testing
Data_withLabels/CoralImageClassification/Non-Coral Images'

for filename in os.listdir(negativeTrainingExamples):
    if filename.endswith(".jpg") or filename.endswith(".png") or
filename.endswith(".jpg"):
        negativeTestingImagePath .append(os.path.join(negativeTrainingExamples,
filename))

negativeTestingImagePath .sort(key=lambda f: int(''.join(filter(str.isdigit,
f))))

testingImagePath = []

for path in positiveTestingImagePath:
    testingImagePath.append([path,1])

for path in negativeTestingImagePath:
    testingImagePath.append([path,0])

testingImagePath = np.array(testingImagePath)

test_dataset = CoralDataset(data = testingImagePath, root_dir="", transform =
preprocess)

test_loader = DataLoader(dataset=test_dataset,
batch_size=batch_size,shuffle=True)

#run train method
train_ch6(finetune_net, train_loader, test_loader, num_epochs, learning_rate,
device)

```