Name: Chirapat Suninhong ID: 5881018

EGCI486
Image Processing - Final Examination

Part 1

Red is answer that I believe to be the most possible

1.
   1.1. A because we can use contrast stretching to enhance the low contrast image, we can clearly see from the image that the contrast has been increased. Therefore the method of contrast stretching is possible
   1.2. B because it is also possible that the image is enhanced to fit the new display screen. The dark area in the result image is obviously darker than the dark area in the input image because gamma is 2(makes shadow darker)
   1.3. C isn't the most possible answer, because Gray-level slicing is used to highlight a specific range of gray-levels[A,B] in the photo. In this case, there isn't any particular area in the photo that is highlighted, but it could be that the line in the sky is slightly darkened.
   1.4. D Bit Plane slicing is one of the most possible answers in this case because 8-bit layers are stacked on top of each other to form an image. If one layer is removed from the image then that will cause the intensity of the colors in the image to change. Since most significant bits contain the majority of the visually significant data. The Figure 1(b) still looks mostly like the original image, but if we extract the lesser significant bit instead, the image might not look as clear as what it is looking like right now in Figure 1(b)
2. 2a
   2.1. A Contrast stretching enhances the image by increasing the contrast to a poor quality image. In this case we can see the lighter area of the image gets significantly brighter. While the darker area could as well was made darker. Thus, this method could have been used to highlight this particular part of the image
   2.2. B Power-law transformation level is controlled based on the level of gamma, in this case the gamma is only 0.3, therefore the dark areas become brighter as shown in the photo. Additionally, smokescreen that can be seen in the image is a common result of power-law transformations
   2.3. C Gray-level slicing with the background isn't the most possible answer
   2.4. D Histogram equalization is used to increase the dynamic range of the gray-levels in an image. The process helps the image to be clearer and the intensity is distributed more equally throughout the whole image. We can see from this image that the lighter part makes the rest of the image brighter as well. Thus, this answer is possible, but it's not the most possible
3.

3.1. A Laplacian filter is an edge detector used to compute the second derivatives of an image, while average filtering helps in smoothing the photo. We can see from the photo that the edges are detected and the noise is removed(but average filter is more suitable for gaussian noise) so this answer is also possible, However we can see that the output image is very smooth, therefore we can assume that it is made smooth before the edges are detected.

3.2. B Median filter is used well to remove salt and pepper, therefore this option is also possible, while median helps to make the image smooth. However, just like a) it is more likely that the image is made smoother before the edge detection because the noise can mess up with the edge detection. We can see the edge is detected sharply so we can assume that the image is made smoother before.

3.3. C This is the most possible answer, because median filter is suitable for the removal of noise and pepper noise, while Prewitt filtering will help in edge detection and these two processes combined will give a result like the figure 3(b). We smooth the image before detecting the edge because the noise could mess up with the process of edge detection.

3.4. D This answer is also possible, however the average filter would be a greater fit for gaussian noise removal, even though Laplacian is used to smooth the image.

4.

4.1. Multiple global thresholding could have been used for the resulting image because we can clearly see that multiple colors and shades were extracted from the image. However, since the result image in 4(b) is in monochrome we can apply single global thresholding in this case to get a sharper and cleaner image.

4.2. a threshold of around 125-127 works best in this case

5.

5.1. This program will use cv2, numpy and matplotlib libraries. Firstly, we will import the image and then we will convert the color system into HSV for the purpose of extracting and masking. Then we will define the ‘green_low = np.array([0,113,0])’ and ‘green_high = np.array([0,241,0])’. The numbers will create the range of green colors that we will extract in this program. Then we will mask all the green colors in this image to be calculated for the area of the leaves.

5.2. In order for us to be able to calculate the area of the leaves only, we will need to create a loop “CalculateArea” to find the mask that is bigger than a certain threshold (that is big enough to be the leaves but not smaller than the stem itself). We then add all of the values together to get the total value of the area of the leaves.

Pseudocode to extract the leaves

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

image = cv2.imread(path+'Fig5_plant.jpg')
hsv_img = cv2.cvtColor(image,
cv2.COLOR_BGR2HSV)

plt.imshow(hsv_img)
green_low = np.array([0,113,0] )
green_high = np.array([0,241,0])

curr_mask = cv2.inRange(hsv_img, green_low,
green_high)
hsv_img[curr_mask > 0] = ([75,255,200])
plt.imshow(hsv_img)
RGB_again = cv2.cvtColor(hsv_img,
cv2.COLOR_HSV2RGB)
gray = cv2.cvtColor(RGB_again,
cv2.COLOR_RGB2GRAY)

plt.imshow(gray)
ret, threshold = cv2.threshold(gray, 90, 255, 0)

contours, hierarchy =
cv2.findContours(threshold,cv2.RETR_TREE,cv2.CHAI
N_APPROX_SIMPLE)
cv2.drawContours(image, contours, -1, (0, 0, 255), 3)
plt.imshow(image)
```

Pseudocode to find the area

```
totalvalue =0
def CalculateArea(contours):
 treshold=30
 if(contours>treshold):
   totalvalue=totalvalue+contours

print(totalvalue)
```

Finding the area of the biggest leaves

```
def findGreatesContour(contours):
 largest_area = 0
   largest_contour_index = -1
   i = 0
   total_contours = len(contours)
   print("tc\n")
   print(total_contours)
   while (i < total_contours ):
       area =
cv2.contourArea(contours[i])
       if(area > largest_area):
           largest_area = area
           largest_contour_index = i
       i+=1

   return largest_area,
largest_contour_index
cnt = contours[13]
M = cv2.moments(cnt)
cX = int(M["m10"] / M["m00"])
cY = int(M["m01"] / M["m00"])
largest_area, largest_contour_index =
findGreatesContour(contours)
print(largest_area)
print(largest_contour_index)
print(len(contours))
```

	print(cX) print(cY)

6.

6.1. 6.1) we could use image enhancement processes like negative transformation(this would work like the breast cancer example in the class), then we can use power law to enhance the visibility of lung damage and it could help increase the accuracy of the result. (Additionally, we could try to generate more images by using the process of data generation: rotation, width shift, height shift, shear, zoom, channel shift, horizontal flip, vertical flip because the images from Covid-19 case itself is not big enough)

6.2. 6.2) I am going to use the same approach to enhance the image results just like those of the breast cancer example. Firstly, we invert the image: inverted[i,j] = 255 - Input[i,j]. Secondly, we use power-law transformation to enhance the result of the image(tested and the best value is getting gamma to be = 0.5).
Pre-Processed-Data[i,j] = Power-Law[i,j]( ) = 0.5.

6.2 code

```
import tensorflow
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from IPython.display import Image, display
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2
import os
```

```
import sys
from imutils import paths
from os import path

args={}
args["dataset"]='covid/dataset'
args["plot"]='plot.png'
args["model"]='covid19.model'
args["testset"]='covid/testset'


directory = input("Select the path to save the model to ")
try:
   os.path.isfile('model.h5')
   print("Path is valid")
except IOError:
   print("Error: Path is invalid")

#learning rate
LR = 1e-3
#epochs for training
EPOCHS = 25
#batch size
BS = 8


def Imagepreprocess(image):
 image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
 # negative transformation to make everything looks clearer
 invert_neg = (255 - image)
 # Power Law transformation
 powerlaw = np.array(255*(invert_neg/255)**0.5, dtype='uint8')
 resultimage = cv2.cvtColor(powerlaw, cv2.COLOR_GRAY2RGB)
 return resultimage


print("Images loading")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []


for imagePath in imagePaths:
   label = imagePath.split(os.path.sep)[-2]
```

```
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (224, 224))
    image = Imagepreprocess(image)
    data.append(image)
    labels.append(label)
data = np.array(data) / 255.0
labels = np.array(labels)
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,test_size=0.20,
stratify=labels, random_state=7)
trainAug = ImageDataGenerator(rotation_range=15,fill_mode="nearest")

print("Images are loaded\n")

baseModel = VGG16(weights="imagenet", include_top=False,
input_tensor=Input(shape=(224, 224, 3)))

headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

model = Model(inputs=baseModel.input, outputs=headModel)

for layer in baseModel.layers:
    layer.trainable = False

print("Models are being compiled\n")
opt = Adam(lr=LR, decay=LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy"])


print("Model head Training\n")
H = model.fit_generator(
trainAug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,
validation_data=(testX, testY),
validation_steps=len(testX) // BS,
```

```
epochs=EPOCHS)

print("Model evalution\n")
predIdxs = model.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)
print(classification_report(testY.argmax(axis=1), predIdxs,
target_names=lb.classes_))

confusionmatrix = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(confusionmatrix))
acc = (confusionmatrix[0, 0] + confusionmatrix[1, 1]) / total
sensitivity = confusionmatrix[0, 0] / (confusionmatrix[0, 0] +
confusionmatrix[0, 1])
spec = confusionmatrix[1, 1] / (confusionmatrix[1, 0] + confusionmatrix[1, 1])

print(confusionmatrix)
print("accuracy: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(spec))

N = EPOCHS
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])

model.save(directory)

print("Loading images")
imagePaths = list(paths.list_images(args["testset"]))
testX = []
testY = []
for imagePath in imagePaths:
    label = imagePath.split(os.path.sep)[-2]
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (224, 224))
    image = transform(image)
    testX.append(image)
```

```
    testY.append(label)
testX = np.array(testX) / 255.0
testY = np.array(testY)

lb = LabelBinarizer()
testY = lb.fit_transform(testY)
testY = to_categorical(testY)
print("Done")

print("Evaluating model")
predIdxs = model.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)
print(classification_report(testY.argmax(axis=1), predIdxs,
target_names=lb.classes_))

confusionmatrix = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(confusionmatrix))
acc = (confusionmatrix[0, 0] + confusionmatrix[1, 1]) / total
sensitivity = confusionmatrix[0, 0] / (confusionmatrix[0, 0] +
confusionmatrix[0, 1])
spec = confusionmatrix[1, 1] / (confusionmatrix[1, 0] + confusionmatrix[1, 1])

print(confusionmatrix)
print("accuracy: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(spec))
```