LAMPIRAN

Lampiran 1. Kode sumber *down sampling* citra

```
import cv2
import os
import numpy as np
import pandas as pd
from pathlib import Path

#original image source
bloodyORI = 'C:/Users/AZ/.spyder-py3/DATASET1/bloody/'
non_bloodyORI = 'C:/Users/AZ/.spyder-py3/DATASET1/non_bloody/'

# Folder path for save image resized
bloodyres = 'C:/Users/AZ/.spyder-py3/DATASET1/bloody_resized/'
non_bloodyres = 'C:/Users/AZ/.spyder-py3/DATASET1/non_bloody_resized/'

# Create the folders to store reized images
os.makedirs(bloodyres)
os.makedirs(non_bloodyres)

#resize function
def resize(originalImageFolder, newFolderPath):

    fileList = []
    fileIDList = []

    #search folder
    fileIDList = next(os.walk(originalImageFolder), (None, None,
[]))[2]

    print('Number of Images: ', len(fileIDList))
    i = 0
    for file in fileIDList:

        imgOriginal = cv2.imread(originalImageFolder+file,1)

        image = cv2.imread(originalImageFolder+file)

        width = 400

        height = 400

        dimensi = (width, height)

        #resize image

        resized = cv2.resize(img, dimensi,
interpolation=cv2.INTER_AREA)

     #Write image into the folder

        cv2.imwrite(newFolderPath + fileIDList[i], resized)

        i+=1 #iteration

resize(bloodyORI, bloodyres)

resize(non_bloodyORI, non_bloodyres)

print('resizing done !!!')
```

Lampiran 2. Kode sumber segmentasi citra

```
import cv2
import os
import numpy as np
import pandas as pd
from pathlib import Path

#dataset
bloody_resized = 'C:/Users/AZ/.spyder-py3/DATASET1/bloody_resized/'
non_bloody_resized = 'C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_resized/'

# Folder path for segmented images
bloodyseg = 'C:/Users/AZ/.spyder-py3/DATASET1/bloody_Segmented/'
non_bloodyseg = 'C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_Segmented/'


# Create the folders to store segmented images
os.makedirs(bloodyseg)
os.makedirs(non_bloodyseg)


def segmentation(resizedImageFolder, newFolderPath):

    fileList = []
    fileIDList = []

    fileIDList = next(os.walk(resizedImageFolder), (None, None,
[]))[2]

    print('Number of Images: ', len(fileIDList))
    i = 0
    for file in fileIDList:
        imgResized = cv2.imread(resizedImageFolder+file,1)
        image = cv2.imread(resizedImageFolder+file)

        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        #parameter HSV shade
        lower = np.array([152, 140, 62])
        upper = np.array([180, 255, 255])

        #create hsv mask
        mask = cv2.inRange(hsv, lower, upper)

        kernel = np.ones((3,3), np.uint8)

        #menghilangkan noise
        closing = cv2.morphologyEx(mask, cv2.MORPH_DILATE, kernel)

        #convert image to binary
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        _, th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)
        th_binary = th.copy()

        #invert floodfilled image
        invert = cv2.bitwise_not(th_binary)

        #combine the two images to get the foreground
        out = closing | invert
```

```
        #==========imgfill=========
        contours, hierarchy = cv2.findContours(mask, cv2.RETR_LIST,
cv2.CHAIN_APPROX_NONE)

        for contour in contours:
            area = cv2.contourArea(contour)

            if area > 10:
                cv2.drawContours(closing, [contour], 0, 255, -1)
#untuk menambal hold
        hasil = cv2.bitwise_and(image, image, mask=closing)

        # Write image into the folder
        cv2.imwrite(newFolderPath + fileIDList[i], hasil)
        i+=1

segmentation(bloody_resized, bloodyseg)
segmentation(non_bloody_resized, non_bloodyseg)

print('segmentation done !!!')
```

Lampiran 3. Kode sumber ekstraksi fitur

```
import cv2
import os
import numpy as np
import pandas as pd
from pathlib import Path
from statistics import mean
from statistics import stdev
from matplotlib import pyplot as plt
from skimage.feature import greycomatrix, greycoprops
from scipy.cluster.vq import whiten
from scipy.cluster.vq import kmeans
from scipy.stats import skew
from scipy.stats import kurtosis
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

bloodyseg = 'C:/Users/AZ/.spyder-py3/DATASET1/bloody_Segmented/'
non_bloodyseg = 'C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_Segmented/'

# File path for extracted color histogram features
bloodyex_colhist = 'C:/Users/AZ/.spyder-
py3/DATASET1/bloody_features_colhist.csv'
non_bloodyex_colhist = 'C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_features_colhist.csv'
```

```
# File path for extracted GLCM features
bloodyex_glcm = 'C:/Users/AZ/.spyder-
py3/DATASET1/bloody_features_glcm.csv'
non_bloodyex_glcm = 'C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_features_glcm.csv'

def ColourHistogram(folderPath,filename):

    # Read image
    img = cv2.imread(folderPath+filename)

    # Create df
    df = pd.DataFrame()
    df['ID'] = [filename]

    # Colour Histogram
    Blue = cv2.calcHist([img], [0], None, [256], [0,256])
    Green = cv2.calcHist([img], [1], None, [256], [0,256])
    Red = cv2.calcHist([img], [2], None, [256], [0,256])

    result_R = [i for i in range(0,15)]
    result_G = [i for i in range(0,15)]
    result_B = [i for i in range(0,15)]

    start = 0
    end = 17
    for i in range(0,15):
        r = np.sum(Red[start:end])
        g = np.sum(Green[start:end])
        b = np.sum(Blue[start:end])
        start = end
        end = end + 17
        result_R[i] = r
        result_G[i] = g
        result_B[i] = b

    finalResult = [filename]
    finalResult = result_R + result_G + result_B + finalResult

    mean_r= np.mean(result_R)
    df['mean_r'] = mean_r
```

```
    std_r= np.std(result_R)
    df['std_r'] = std_r
    skew_r= skew(result_R)
    df['skew_r'] = skew_r
    kurt_r= kurtosis(result_R)
    df['kurt_r'] = kurt_r

    mean_g= np.mean(result_G)
    df['mean_g'] = mean_g
    std_g= np.std(result_G)
    df['std_g'] = std_g
    skew_g= skew(result_G)
    df['skew_g'] = skew_g
    kurt_g= kurtosis(result_G)
    df['kurt_g'] = kurt_g

    mean_b= np.mean(result_B)
    df['mean_b'] = mean_b
    std_b= np.std(result_B)
    df['std_b'] = std_b
    skew_b= skew(result_B)
    df['skew_b'] = skew_b
    kurt_b= kurtosis(result_B)
    df['kurt_b'] = kurt_b

    df = df.set_index('ID')
    return df

def GLCM(folderPath,filename):
    # Read image
    img = cv2.imread(folderPath+filename, cv2.IMREAD_GRAYSCALE)

    df = pd.DataFrame()
    df['ID'] = [filename]

    #Full image
    #GLCM = greycomatrix(img, [1], [0, np.pi/4, np.pi/2, 3*np.pi/4])

    #distance 1
    GLCM0 = greycomatrix(img, [1], [0])
    GLCM_Energy0 = greycoprops(GLCM0, 'energy')[0]
    df['Energy_1_0'] = GLCM_Energy0
```

```
GLCM_corr0 = greycoprops(GLCM0, 'correlation')[0]
df['Corr_1_0'] = GLCM_corr0
GLCM_diss0 = greycoprops(GLCM0, 'dissimilarity')[0]
df['Diss_sim_1_0'] = GLCM_diss0
GLCM_hom0 = greycoprops(GLCM0, 'homogeneity')[0]
df['Homogen_1_0'] = GLCM_hom0
GLCM_contr = greycoprops(GLCM0, 'contrast')[0]
df['Contrast_1_0'] = GLCM_contr

GLCM45 = greycomatrix(img, [1], [np.pi/4])
GLCM_Energy45 = greycoprops(GLCM45, 'energy')[0]
df['Energy_1_45'] = GLCM_Energy45
GLCM_corr45 = greycoprops(GLCM45, 'correlation')[0]
df['Corr_1_45'] = GLCM_corr45
GLCM_diss45 = greycoprops(GLCM45, 'dissimilarity')[0]
df['Diss_sim_1_45'] = GLCM_diss45
GLCM_hom45 = greycoprops(GLCM45, 'homogeneity')[0]
df['Homogen_1_45'] = GLCM_hom45
GLCM_contr45 = greycoprops(GLCM45, 'contrast')[0]
df['Contrast_1_45'] = GLCM_contr45

GLCM90 = greycomatrix(img, [1], [np.pi/2])
GLCM_Energy90 = greycoprops(GLCM90, 'energy')[0]
df['Energy_1_90'] = GLCM_Energy90
GLCM_corr90 = greycoprops(GLCM90, 'correlation')[0]
df['Corr_1_90'] = GLCM_corr90
GLCM_diss90 = greycoprops(GLCM90, 'dissimilarity')[0]
df['Diss_sim_1_90'] = GLCM_diss90
GLCM_hom90 = greycoprops(GLCM90, 'homogeneity')[0]
df['Homogen_1_90'] = GLCM_hom90
GLCM_contr90 = greycoprops(GLCM90, 'contrast')[0]
df['Contrast_1_90'] = GLCM_contr90

GLCM135 = greycomatrix(img, [1], [3*np.pi/4])
GLCM_Energy135 = greycoprops(GLCM135, 'energy')[0]
df['Energy_1_135'] = GLCM_Energy135
GLCM_corr135 = greycoprops(GLCM135, 'correlation')[0]
df['Corr_1_135'] = GLCM_corr135
GLCM_diss135 = greycoprops(GLCM135, 'dissimilarity')[0]
df['Diss_sim_1_135'] = GLCM_diss135
GLCM_hom135 = greycoprops(GLCM135, 'homogeneity')[0]
df['Homogen_1_135'] = GLCM_hom135
```

```
GLCM_contr135 = greycoprops(GLCM135, 'contrast')[0]
df['Contrast_1_135'] = GLCM_contr135

#distance 2
GLCM0 = greycomatrix(img, [2], [0])
GLCM_Energy0 = greycoprops(GLCM0, 'energy')[0]
df['Energy_2_0'] = GLCM_Energy0
GLCM_corr0 = greycoprops(GLCM0, 'correlation')[0]
df['Corr_2_0'] = GLCM_corr0
GLCM_diss0 = greycoprops(GLCM0, 'dissimilarity')[0]
df['Diss_sim_2_0'] = GLCM_diss0
GLCM_hom0 = greycoprops(GLCM0, 'homogeneity')[0]
df['Homogen_2_0'] = GLCM_hom0
GLCM_contr = greycoprops(GLCM0, 'contrast')[0]
df['Contrast_2_0'] = GLCM_contr

GLCM45 = greycomatrix(img, [2], [np.pi/4])
GLCM_Energy45 = greycoprops(GLCM45, 'energy')[0]
df['Energy_2_45'] = GLCM_Energy45
GLCM_corr45 = greycoprops(GLCM45, 'correlation')[0]
df['Corr_2_45'] = GLCM_corr45
GLCM_diss45 = greycoprops(GLCM45, 'dissimilarity')[0]
df['Diss_sim_2_45'] = GLCM_diss45
GLCM_hom45 = greycoprops(GLCM45, 'homogeneity')[0]
df['Homogen_2_45'] = GLCM_hom45
GLCM_contr45 = greycoprops(GLCM45, 'contrast')[0]
df['Contrast_2_45'] = GLCM_contr45

GLCM90 = greycomatrix(img, [2], [np.pi/2])
GLCM_Energy90 = greycoprops(GLCM90, 'energy')[0]
df['Energy_2_90'] = GLCM_Energy90
GLCM_corr90 = greycoprops(GLCM90, 'correlation')[0]
df['Corr_2_90'] = GLCM_corr90
GLCM_diss90 = greycoprops(GLCM90, 'dissimilarity')[0]
df['Diss_sim_2_90'] = GLCM_diss90
GLCM_hom90 = greycoprops(GLCM90, 'homogeneity')[0]
df['Homogen_2_90'] = GLCM_hom90
GLCM_contr90 = greycoprops(GLCM90, 'contrast')[0]
df['Contrast_2_90'] = GLCM_contr90

GLCM135 = greycomatrix(img, [2], [3*np.pi/4])
GLCM_Energy135 = greycoprops(GLCM135, 'energy')[0]
```

```
    df['Energy_2_135'] = GLCM_Energy135
    GLCM_corr135 = greycoprops(GLCM135, 'correlation')[0]
    df['Corr_2_135'] = GLCM_corr135
    GLCM_diss135 = greycoprops(GLCM135, 'dissimilarity')[0]
    df['Diss_sim_2_135'] = GLCM_diss135
    GLCM_hom135 = greycoprops(GLCM135, 'homogeneity')[0]
    df['Homogen_2_135'] = GLCM_hom135
    GLCM_contr135 = greycoprops(GLCM135, 'contrast')[0]
    df['Contrast_2_135'] = GLCM_contr135

    df = df.set_index('ID')
    return df

def FeaturesExtractionsColHist(folderPath):
    df = pd.DataFrame()
    for fname in sorted(os.listdir(folderPath)):
        colourHist = ColourHistogram(folderPath,fname)
        dfRow = colourHist
        df = df.append(dfRow)
    return df

def FeaturesExtractionsGLCM(folderPath):
    df = pd.DataFrame()
    for fname in sorted(os.listdir(folderPath)):
        glcm = GLCM (folderPath,fname)
        dfRow = glcm
        df = df.append(dfRow)
    return df

dfbloody = FeaturesExtractionsColHist(bloodyseg)
dfbloody.to_csv(bloodyex_colhist)

dfnon_bloody = FeaturesExtractionsColHist(non_bloodyseg)
dfnon_bloody.to_csv(non_bloodyex_colhist)

print('Color Histogram Extraction, Done !!!')

dfbloody1 = FeaturesExtractionsGLCM(bloodyseg)
#glcm_pca bloody
scalar1 = StandardScaler()
df1_scaled = pd.DataFrame(scalar1.fit_transform(dfbloody1),
columns=dfbloody1.columns)
```

```
pca1 = PCA()
df1_glcm_pca = pd.DataFrame(pca1.fit_transform(df1_scaled))
glcm_pca1 = df1_glcm_pca.iloc[:, 0:6].to_csv(bloodyex_glcm,
index=False)

pd.DataFrame(pca1.explained_variance_ratio_).plot.bar()
plt.legend('')
plt.xlabel('bloody Principal Components')
plt.ylabel('bloody Explained Varience');

dfnon_bloody1 = FeaturesExtractionsGLCM(non_bloodyseg)
#glcm_pca non_bloody
scalar2 = StandardScaler()
df2_scaled = pd.DataFrame(scalar2.fit_transform(dfnon_bloody1),
columns=dfnon_bloody1.columns)

pca2 = PCA()
df2_glcm_pca = pd.DataFrame(pca2.fit_transform(df2_scaled))
glcm_pca2 = df2_glcm_pca.iloc[:, 0:6].to_csv(non_bloodyex_glcm,
index=False)

pd.DataFrame(pca2.explained_variance_ratio_).plot.bar()
plt.legend('')
plt.xlabel('non_bloody Principal Components')
plt.ylabel('non_bloody Explained Varience');

print('GLCM Extraction, Done !!!')

#penggabungan fitur
bloodyex_colhist = pd.read_csv('C:/Users/AZ/.spyder-
py3/DATASET1/bloody_features_colhist.csv')
non_bloodyex_colhist = pd.read_csv ('C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_features_colhist.csv')

bloodyex_glcm = pd.read_csv('C:/Users/AZ/.spyder-
py3/DATASET1/bloody_features_glcm.csv')
non_bloodyex_glcm = pd.read_csv('C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_features_glcm.csv')

df_new1 = pd.concat([bloodyex_colhist, bloodyex_glcm], axis=1)
df_new2 = pd.concat([non_bloodyex_colhist, non_bloodyex_glcm], axis=1)
```

```
bloody_features = df_new1.to_csv("C:/Users/AZ/.spyder-
py3/DATASET1/bloody_features.csv", index=False)
non_bloody_features = df_new2.to_csv("C:/Users/AZ/.spyder-
py3/DATASET1/non_bloody_features.csv", index=False)
```

Lampiran 4. Kode sumber tahap klasifikasi

```
import pandas as pd
import cv2
from matplotlib import pyplot as plt
import math
import os
import shutil
from pathlib import Path
import glob
import itertools
from itertools import cycle
from scipy import interp
from statistics import mean
from statistics import stdev
from array import *
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import label_binarize
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
import matplotlib.pyplot as plt
import matplotlib.image as img
from sklearn.svm import SVC
import seaborn as sns

df1 = "C:/Users/AZ/.spyder-py3/DATASET1/bloody_features.csv"
df2 = "C:/Users/AZ/.spyder-py3/DATASET1/non_bloody_features.csv"

def CombineAllDF(folderPaths, labels):
```

```
    dfAll = pd.DataFrame()
    i=0
    for paths, label in zip(folderPaths, labels):
        dfs = pd.read_csv(paths)
        dfs['Labels'] = labels[i]
        dfs = dfs.set_index('ID')
        dfAll = dfAll.append(dfs)
        i=i+1
    dfAll = dfAll.reset_index()
    return dfAll

bloody = df1
non_bloody = df2

folderPaths = [bloody, non_bloody]
classes = ['bloody','non_bloody']

dfALL = CombineAllDF(folderPaths, classes)
#fitur = dfALL.to_csv('C:/Users/AZ/.spyder-py3/DATASET1/all.csv')
print (dfALL.shape)

#Generate X_train, without ID and Labels
# Y_train only contains labels
X = dfALL.drop(['ID','Labels'], axis = 1)
y = dfALL.loc[:,'Labels']

# split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state = 0)
print('Training set shape: ', X_train.shape, y_train.shape)
print('Testing set shape: ', X_test.shape, y_test.shape)

cols = X_train.columns

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])

# instantiate classifier with rbf kernel and C=100
```

```
svc=SVC(C = 100, kernel = 'rbf')

# fit classifier to training set
svc.fit(X_train,y_train)

print('Training score: {:.4f}'.format(svc.score(X_train, y_train)))

# make predictions on test set
y_pred=svc.predict(X_test)

# compute and print accuracy score
print('Testing score : {:.4f}'.format(accuracy_score(y_test, y_pred)))

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1',
'Actual Negative:0'],
                         index=['Predict Positive:1', 'Predict
Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')

print('classification report')
print(classification_report(y_test, y_pred))

TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
```

```
print('Classification accuracy :
{0:0.4f}'.format(classification_accuracy))

classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error : {0:0.4f}'.format(classification_error))

precision = TP / float(TP + FP)
print('Precision : {0:0.4f}'.format(precision))

recall = TP / float(TP + FN)
print('Recall or Sensitivity : {0:0.4f}'.format(recall))

true_positive_rate = TP / float(TP + FN)
print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

false_positive_rate = FP / float(FP + TN)
print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

specificity = TN / (TN + FP)
print('Specificity : {0:0.4f}'.format(specificity))
```