BAB VI
KESIMPULAN

Berdasarkan hasil pengujian dan pembahasan yang telah dilakukan pada bab sebelumnya, maka penulis dapat menyimpulkan bahwa :

1. Transliterasi Aksara Jawa kedalam huruf Latin yang dibangun dengan menggunakan algoritma *Pseudo Zernike Moment* dan *Multi-class SVM* memberikan hasil yang memuaskan dengan persentasi akurasi sebesar 87%, *precision* sebesar 88 %, *recall* sebesar 87%, dan *f1-score* sebesar 87%.
2. Semakin tinggi ordo yang digunakan pada *Pseudo Zernike Moment* membantu meningkatkan akurasi pengenalan, namun hal ini mempengaruhi kecepatan pemrosesan. Ordo terbaik untuk transliterasi Aksara Jawa kedalam huruf Latin adalah sebesar 20 karena penggunaan ordo yang lebih tinggi memberikan hasil yang sama.
3. Nilai parameter yang optimal pada model yang diimplementasikan yaitu γ sebesar 1 dan C sebesar 1.1.

DAFTAR PUSTAKA

Amara, M., Ghedira, K., Zidi, K. and Zidi, S., 2016. A comparative study of multi-class support vector machine methods for Arabic characters recognition. In: *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*. [online] IEEE, pp.1–6.

Atina, V., Palgunadi, S. and Widiarto, W., 2012. Program Transliterasi Antara Aksara Latin dan Aksara Jawa dengan Metode FSA. *Jurnal Teknologi & Informasi ITSmart*, [online] 1(2), p.60.

Budhi, G.S. and Adipranata, R., 2015. Handwritten Javanese Character Recognition Using Several Artificial Neural Network Methods. *Journal of ICT Research and Applications*, [online] 8(3), pp.195–212.

Chong, C.-W., Raveendran, P. and Mukundan, R., 2003. An Efficient Algorithm for Fast Computation of Pseudo-Zernike Moments. *International Journal of Pattern Recognition and Artificial Intelligence*, [online] 17(06), pp.1011–1023.

Clemente, C., Soraghan, J.J., Proudler, I., De Maio, A., Pallotta, L. and Farina, A., 2015. Pseudo-Zernike-based multi-pass automatic target recognition from multi-channel synthetic aperture radar. *IET Radar, Sonar & Navigation*, [online] 9(4), pp.457–466.

Dasgupta, T., Sinha, M. and Basu, A., 2015. Resource creation and development of an English-Bangla back transliteration system. *International Journal of Knowledge-based and Intelligent Engineering Systems*, [online] 19(1), pp.35–46.

Elleuch, M., Maalej, R. and Kherallah, M., 2016. A New design based-SVM of the CNN classifier architecture with dropout for offline Arabic handwritten recognition. In: *Procedia Computer Science*. [online] pp.1712–1723.

Flach, P. and Kull, M., 2015. Precision-Recall-Gain Curves: PR Analysis Done Right. *Advances in Neural Information Processing Systems 28*, [online] 1, pp.838–846.

Himamunanto, A.R. and Setyowati, E., 2018. Text block identification in restoration process of Javanese script damage. *Journal of Physics: Conference Series*, [online] 1013(1), p.012–210.

Hosny, K., 2014. Fast computation of accurate pseudo Zernike moments for binary and gray-level images. *International Arab Journal of Information Technology*, 11(3), pp.243–249.

Jain, A.K., 1989. *Fundamentals of Digital Image Processing*. [online] *Portalacmorg*, Available at: <http://portal.acm.org/citation.cfm?id=59921>.

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2014. An Introduction to Statistical Learning: with Applications in R. *An Introduction to Statistical Learning: with Applications in R*. [online] Available at: <http://dl.acm.org/citation.cfm?id=2517747&coll=DL&dl=GUIDE&CFID=718033788&CFTOKEN=64826262%5Cnpapers3://publication/doi/10.1007/978-1-4614-7138-7.pdf>.

Jan, S., Vu, V.-H. and Koo, I., 2018. Throughput Maximization Using an SVM

for Multi-Class Hypothesis-Based Spectrum Sensing in Cognitive Radio. *Applied Sciences*, [online] 8(3), p.421.

Kanan, H.R. and Salkhordeh, S., 2016. Rotation invariant multi-frame image super resolution reconstruction using Pseudo Zernike Moments. *Signal Processing*, [online] 118, pp.103–114.

Kef, M., Chergui, L. and Chikhi, S., 2016. A novel fuzzy approach for handwritten Arabic character recognition. *Pattern Analysis and Applications*, [online] 19(4), pp.1041–1056.

Prabhakar, D.K. and Pal, S., 2018. Machine transliteration and transliterated text retrieval: a survey. *Sādhanā*, [online] 43(6), p.93.

Pusat Bahasa Departemen Pendidikan Nasional, 2007. *Kamus Besar Bahasa Indonesia*. [online] *Pusat Bahasa Departemen Pendidikan Nasional*. Available at: <http://bahasa.cs.ui.ac.id/kbbi/kbbi.php?keyword=perilaku&varbidang=all&vardialek=all&varragam=all&varkelas=all&submit=tabel>.

Saito, T. and Rehmsmeier, M., 2015. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE*, [online] 10(3), p.e0118432.

Singh, S.P. and Urooj, S., 2016. An Improved CAD System for Breast Cancer Diagnosis Based on Generalized Pseudo-Zernike Moment and Ada-DEWNN Classifier. *Journal of Medical Systems*, [online] 40(4), p.105.

Tanaya, D. and Adriani, M., 2016. Dictionary-based Word Segmentation for Javanese. In: *Procedia Computer Science*. pp.208–213.

Theodoridis, S., Pikrakis, A., Koutroumbas, K. and Cavouras, D., 2010. *Introduction to Pattern Recognition: A Matlab Approach*. *Introduction to Pattern Recognition: A Matlab Approach*.

Widiarti, A.R., Harjoko, A., Marsono and Hartati, S., 2018a. The model and implementation of javanese script image transliteration. In: *Proceedings - 2017 International Conference on Soft Computing, Intelligent System and Information Technology: Building Intelligence Through IOT and Big Data, ICSIIT 2017*. [online] IEEE, pp.51–57.

Widiarti, A.R., Pulungan, R., Harjoko, A., Marsono and Hartati, S., 2018b. A proposed model for Javanese manuscript images transliteration. *Journal of Physics: Conference Series*, [online] 1098(1), pp.012–014.

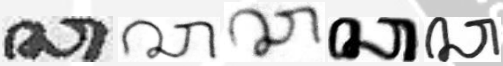
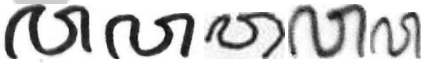
Wisnu, A., Nugraha, W. and Widhiatmoko, H.P., 2012. Jumlah Transisi pada Ciri Transisi dalam Pengenalan Pola Tulisan Tangan Aksara Jawa Nglegeno dengan Multiclass Support Vector Machines Numbers of Transition Features on Basic Jawanesse (Nglegeno) Characters Recognition System with Multiclass Support Vec. *Dinamika Rekayasa*, [online] 8(1), pp.18–24.

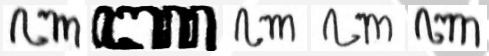
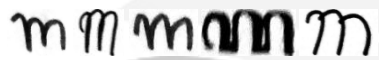
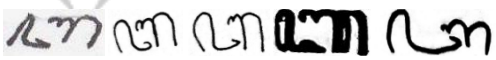
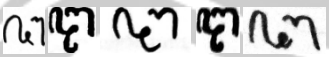
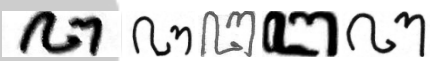
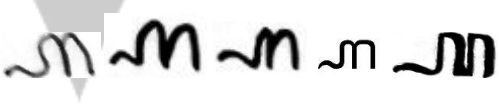
Wong, T.T., 2015. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition*, [online] 48(9), pp.2839–2846.

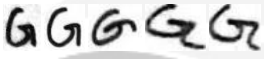
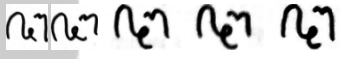
Zareapoor, M., Shamsolmoali, P., Kumar Jain, D., Wang, H. and Yang, J., 2017. Kernelized support vector machine with deep learning: An efficient approach for extreme multiclass dataset. *Pattern Recognition Letters*. [online] Sep. Available at:

<http://linkinghub.elsevier.com/retrieve/pii/S0167865517303276>
[Accessed 20 Mar. 2018].

Lampiran 1 : Contoh Sampel Aksara Jawa

Nama Aksara	Contoh Citra
ha	
na	
ca	
ra	
ka	
da	
ta	
sa	
wa	
la	

pa	
dha	
ja	
ya	
nya	
ma	
ga	
ba	
tha	
nga	
ha (pasangan)	
ca (pasangan)	

da (pasangan)	
ta (pasangan)	
sa (pasangan)	
la (pasangan)	
pa (pasangan)	
dha (pasangan)	
ja (pasangan)	
ma (pasangan)	
ba (pasangan)	
tha (pasangan)	
cecak (-ng)	
pepet (e)	

taling	
wignyan(-h)	
wulu(i)	
layar (-r)	
tarung	

Lampiran 2 : Kode Program

ML_transliterasi.ipnyb

```
#organize import
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import mahotas
import cv2
import os
import h5py
from skimage.morphology import skeletonize, thin
from pyzernikemoment import Zernikemoment

# fixed-sizes for image
fixed_size = 100

# path to training data
train_path        =        "C:/Users/maydf/character-
classification/datasets/train"

# no.of.trees for Random Forests
num_trees = 100

# bins for histogram
bins = 8

# train_test_split size
test_size = 0.10

# seed for reproducing same results
seed = 7

pad = 10

from skimage.transform import resize, pyramid_reduce


def get_square(image, square_size):

    height, width = image.shape
    if(height > width):
      differ = height
    else:
      differ = width
```

```
    differ += 4

    # square filler
    mask = np.zeros((differ, differ), dtype = "uint8")
    x_pos = int((differ - width) / 2)
    y_pos = int((differ - height) / 2)
    # center image inside the square
    mask[y_pos: y_pos + height, x_pos: x_pos + width]
= image[0: height, 0: width]

    # downscale if needed
    if differ / square_size > 1:
      mask    =    pyramid_reduce(mask,    differ    /
square_size)
    else:
      mask     =     cv2.resize(mask,     (square_size,
square_size), interpolation = cv2.INTER_AREA)
    return mask

#feature_descriptor-2 : Zernike Texture
def fd_zernike(image):

    gray =  cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #thresh
    thresh                                              =
cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THR
ESH_OTSU)[1]
    thresh[thresh == 255] = 1
    resized = get_square(thresh,fixed_size)

    kernel = np.ones((3,3),np.uint8)
    erosion = cv2.erode(resized,kernel,iterations = 1)
    #zernike                                             =
mahotas.features.zernike_moments(erosion,  radius=100,
degree=20, cm= mahotas.center_of_mass(erosion))
    zernike = Zernikemoment(erosion, 15)
    #return the result
    return Zernike

#get the training labels
train_labels = os.listdir(train_path)

#sort the training labels
train_labels.sort()
print(train_labels)

#empty lists to hold fectors and labels
```

```
global_features=[]
labels=[]

i,j = 0, 0
k = 0

#num of images per class
images_per_class = 500

%%time
# loop over the training data sub-folders
for training_name in train_labels:
    # join the training data path and each species
training folder
    #dir = os.path.join(train_path, training_name)
    dir = train_path + "/" + training_name
    #print (dir)

    # get the current training label
    current_label = training_name
    from matplotlib import pyplot
    #image1 = cv2.imread(dir + "/image_20.png")
    #pyplot.imshow(image1)
    #pyplot.show()
    k = 1
    # loop over the images in each sub-folder
    for x in range(1,images_per_class+1):
        # get the image file name

        file = dir + "/image_" + str(x) + ".png"
        #print(file)
        # read the image and resize it to a fixed-size
        image = cv2.imread(file)


        #image_pad=
cv2.copyMakeBorder(resized,pad,pad,pad,pad,cv2.BORDER_
CONSTANT,value= [255,255,255])

        #image = cv2.resize(image, (100, 100))
        #image = get_square(image, 100)

        ####################################
        # Global Feature extraction
        ####################################
        #fv_hu_moments = fd_hu_moments(image)
        fv_zernike   = fd_zernike(image)
```

```
        #fv_histogram  = fd_histogram(image)

        ####################################
        # Concatenate global features
        ####################################
        global_feature = fv_zernike

        # update the list of labels and feature
vectors
        labels.append(current_label)
        global_features.append(global_feature)

        i += 1
        k += 1
    print ("[STATUS] processed folder:
{}".format(current_label))
    j += 1

print ("[STATUS] completed Global Feature
Extraction...")

%%time
# get the overall feature vector size
print ("[STATUS] feature vector size
{}".format(np.array(global_features).shape))

# get the overall training label size
print ("[STATUS] training Labels
{}".format(np.array(labels).shape))

# encode the target labels
targetNames = np.unique(labels)
le = LabelEncoder()
target = le.fit_transform(labels)
print ("[STATUS] training labels encoded...")

# normalize the feature vector in the range (0-1)
#scaler = MinMaxScaler(feature_range=(0, 1))
#global_features2 = global_features
#rescaled_features =
scaler.fit_transform(global_features2)
print ("[STATUS] feature vector normalized...")

print ("[STATUS] target labels: {}".format(target))
print ("[STATUS] target labels shape:
{}".format(target.shape))
```

```
# save the feature vector using HDF5
h5f_data = h5py.File('output/data_mhl.h5', 'w')
h5f_data.create_dataset('dataset_mhl',
data=np.array(global_features))

h5f_label = h5py.File('output/labels_mhl.h5', 'w')
h5f_label.create_dataset('dataset_mhl',
data=np.array(target))

h5f_data.close()
h5f_label.close()

print ("[STATUS] end of training..")

# Classification
%%time
# import the necessary packages
import glob
from matplotlib import pyplot
from sklearn.model_selection import cross_val_score
from sklearn.cross_validation import train_test_split
from     sklearn.model_selection     import     KFold,
StratifiedKFold
from    sklearn.metrics    import    confusion_matrix,
accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from        sklearn.discriminant_analysis        import
LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn import preprocessing
# create all the machine learning models


# variables to hold the results and names
results = []
names = []
scoring = "accuracy"

# import the feature vector and trained labels
h5f_data = h5py.File('output/data_mhl.h5', 'r')
h5f_label = h5py.File('output/labels_mhl.h5', 'r')

global_features_string = h5f_data['dataset_mhl']
global_labels_string = h5f_label['dataset_mhl']

global_features = np.array(global_features_string)
global_labels = np.array(global_labels_string)
```

```
h5f_data.close()
h5f_label.close()

clf  =SVC(kernel='rbf',  decision_function_shape='ovr',
cache_size=1000, class_weight='balanced')
# verify the shape of the feature vector and labels
print        ("[STATUS]        features        shape:
{}".format(global_features.shape))
print        ("[STATUS]        labels        shape:
{}".format(global_labels.shape))

print ("[STATUS] training started...")

%%time
# split the training and testing data
#(X_train, X_validation, y_train, y_validation)
scaler = MinMaxScaler(feature_range=(0, 1))
scaler.fit(global_features)
rescaled_features = scaler.transform(global_features)

(trainDataGlobal,  testDataGlobal,  trainLabelsGlobal,
testLabelsGlobal)                                    =
train_test_split(np.array(rescaled_features),

np.array(global_labels),

test_size=test_size,

random_state=seed)

print ("[STATUS] splitted train and test data...")
print        ("Train        data                    :
{}".format(trainDataGlobal.shape))
print        ("Test        data                     :
{}".format(testDataGlobal.shape))
print                  ("Train                  labels:
{}".format(trainLabelsGlobal.shape))
print            ("Test            labels          :
{}".format(testLabelsGlobal.shape))

%%time
# filter all the warnings
import warnings
warnings.filterwarnings('ignore')

# 10-fold cross validation
```

```
kfold = KFold(n_splits=10, random_state=7)
cv_results = cross_val_score(clf, trainDataGlobal,
trainLabelsGlobal, cv=kfold, scoring=scoring)
msg = "SVM Score: %f (%f)" % (cv_results.mean(),
cv_results.std())
print(msg)

X = trainDataGlobal  # we only take the first two
features.
y = trainLabelsGlobal
pyplot.scatter(X[:,     0],     X[:,     1],     c=y,
cmap=pyplot.cm.coolwarm)
pyplot.xlabel('fitur')
pyplot.ylabel('label')
pyplot.title('Sepal Width & Length')
pyplot.show()

%%time
# fit the training data to the model
clf.fit(trainDataGlobal,           trainLabelsGlobal)
#(n_samples, n_features)

%%time
#predictions = clf.predict(rescaledY)
predictions = clf.predict(testDataGlobal)

print(confusion_matrix(testLabelsGlobal, predictions))

print(classification_report(testLabelsGlobal,
predictions))

# path to test data
test_path        =        "C:/Users/maydf/character-
classification/datasets/test"

# loop through the test images
for file in glob.glob(test_path + "/*.png"):
    print(file)
          # read the image
    image = cv2.imread(file)
    image = cv2.resize(image, (100, 100))
    # resize the image
    #gray =  cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #resized = get_square(gray,fixed_size)
    #image_pad=
cv2.copyMakeBorder(resized,pad,pad,pad,pad,cv2.BORDER_
CONSTANT,value= [255,255,255])
```

```
    ####################################
    # Global Feature extraction
    ####################################
    #fv_hu_moments = fd_hu_moments(image)
    fv_zernike   = fd_zernike(image)
    #fv_histogram  = fitur.fd_histogram(image)

    ####################################
    # Concatenate global features
    ####################################
    global_feature = fv_zernike

    # predict label of test image
    print(global_feature)
    new_data                                                =
scaler.transform(global_feature.reshape(1,-1))
    #print(new_data)

    prediction = clf.predict(new_data)[0]
    # show predicted label on image
    cv2.putText(image,      train_labels[prediction],
(20,30),  cv2.FONT_HERSHEY_SIMPLEX,  0.5,  (0,255,255),
2)
    print (train_labels[prediction])
    # display the output image
    pyplot.imshow(image)
    pyplot.show()
```

Lampiran 3. Draft Publikasi

Transliteration of Javanese Characters using Pseudo Zernike Moment and Multi-class SVM

Amaya Andri Damaini, Pranowo, and Djoko Budiyanto Setyohadi
Magister Teknik Informatika, Universitas Atma Jaya Yogyakarta, Yogyakarta, Indonesia
amayaandridamaini@gmail.com, pranowo@uajy.ac.id, djoko.bdy@gmail.com

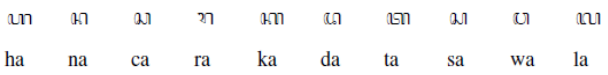
***Abstract*—** Javanese script or Hanacaraka is one of the original Indonesian traditional characters which has become a cultural heritage of Indonesia, and can still be found today in several regions on the island of Java. Currently there are several applications and research that help transliterate scripts from Latin letters to Javanese script or vice versa. The purpose of this study is to evaluate the accuracy of recognition of Javanese script using PZM and Multi-class SVM algorithms. The research was carried out by implementing a learning machine to be able to recognize Javanese characters and transliterate them into Latin letters. The selection of higher order in PZM was able to increase the success in the introduction, but resulted in a longer processing time. The test results provide a fairly good recognition accuracy of 87% by using the best parameters that have been tested.

Keywords*— *Javanese Letters, OCR, Image Processing, Pseudo Zernike Moment, Multi-Class SVM

I. Introduction

Indonesia has a variety of letters which is a cultural heritage in the past, one of which is Javanese script. In the past, Javanese script was used to write story scripts, historical notes, ancient songs, or fortune telling. To preserve this time Javanese script is included in local content subjects at schools in Yogyakarta. Javanese script can also be found on street signboards and in public places in several cities in Java.

Javanese script has a way of writing characters by combining consonants and vowels into one syllable in one unit [1]. Javanese script has 20 forms of basic characters that have not received an affix (Fig. 1). This script is called *nglegena* characters.



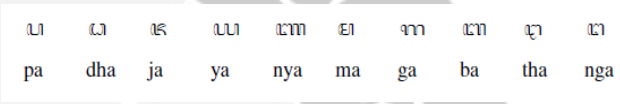


Fig. 1 Nglegena Characters

Each character has a pair. This pair is used to eliminate the vowel sounds in the characters in front of them so that they become consonants. The pair is placed under the basic character except for the pair *ha*, *sa*, *pa*, and *nya*. Whereas for the *na* and *wa* pairs it is located hanging on the basic script. Fig. 2 shows the shape of *nglegena* characters pair.

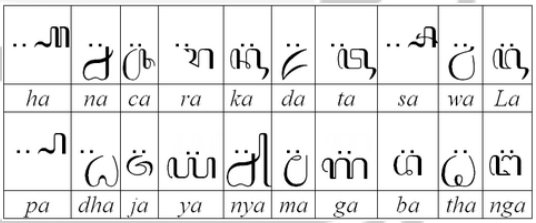


Fig. 2 Pair of nglegena characters
Source : http://pinterjawa.weebly.com/aksara-jawa.html

Sandhangan characters are additional symbols to change the sound of syllables. The symbols can be seen on Fig. 3.

	wulu = ...i		cecak = ...ng
◌ُ	suku = ...u		layar = ...r
◌ေ	pepet = ...e	◌ಾ	pangkon = ...paten
◌ೆ	taling = ...é	◌ಚ	cakra = ...r...
◌ೆ ◌ಾ	taling tarung = ...o	◌ಘ	cakra keret = ...re
◌ೈ	wignyan = ...h	◌ಾ	pengkal = ...y...

Fig. 3 Sandhangan

Research related to Javanese script has been carried out today. Generally, the research conducted regarding recognition of basic characters [2][3] or handwriting recognition[4] where the recognition is done per syllable. The research also evolves to transliteration of manuscripts of Javanese script into Latin letters [5].

Transliteration is the process of translating phonetically a word from the source language into the same word in the target language that represents the pronunciation of the source language [6]. Transliteration only changes the text from one script to another. It does not have to represent original pronunciation, but rather focuses on representing characters as accurately and clearly as possible [7].

Widiarti et al. [8] proposed a model for transliteration of Javanese script. In this model there are three main parts in the Javanese script script transliteration system: creating a database from manuscript, creating a feature database, and translating manuscript. In the manuscript transliteration section, there are three sub-processes to get transliteration results: image segmentation, transliteration of image segmentation pieces, and word formation. The result is a row of words that have been transliterated into Latin characters.

In the transliteration process, there is a classification model built to recognize the Javanese script pattern. The pattern recognition of local script has been widely carried out, such as the recognition of Arabic letters [9][10][11], recognition of Gurmukhi characters [12], and recognition of Devanagari characters[13]. In study [12], Gurmukhi characters recognition gets 96,7% recognition rate by using a voting-based classifier, where the comparison algorithm of Hidden Markov Model (HMM) and Support Vector Machine (SVM) is done. In research on hindi handwriting recognition [13] based on K-means Clustering and SVM gives 95,86% accuracy result.

In this study we built an image pattern recognition system for transliteration of Javanese characters into Latin letters. Using the Pseudo Zernike Moment (PZM) and multi-class SVM algorithms, we examine how much accuracy the Java Literacy recognition is obtained by trying to find an efficient maximum order on PZM and optimizing the γ and C parameters in the multi-class SVM.

II. Related Works

The pattern recognition of digital image can be divided into 3 stages; preprocessing, feature extraction, and classification. Feature extraction and classification are two stages that are very important in digital image pattern recognition. Feature extraction will affect system performance in object classification, especially character recognition. The extracted features must be able to classify characters in a unique way.

As a feature descriptor, PZM has high description capability because it is resistant to noise [14], translational property and PZM rotation independence make the algorithm strong [15]. The Pseudo-Zernike Moment is used in several pattern recognition applications for feature extraction in images [16].

Zernike feature extraction defines complex orthogonal sequences of complex polynomials above the polar coordinate space within the unit circle, e.g. $x^2 + y^2 = 1$. The two-dimensional Pseudo-Zernike moment of order n with the repetition of m for the continuous image function f (x, y) is defined as:

$$Z_{nm} = \frac{n+1}{\pi} \sum_x \sum_y f(x,y)\, V_{nm}^*(\rho,\theta), \quad x^2 + y^2 \le 1 \tag{1}$$

where * denotes the complex conjugate, and $V_{nm}(\rho,\theta)$ is Pseudo-Zernike polynomial of order p and repetition q given by :

$$V_{nm}(\rho,\theta) = R_{nm}(\rho) e^{jq\theta} \tag{2}$$

Information :

n : integer numbers are positive or 0

m : integer numbers with positive or negative values, $|m| \le n, n - |m|$ is even.

ρ : The length of the vector from the original image to the pixel (x, y); has $\sqrt{x^2 + y^2}$ value.

θ : The angle between the vectors ρ and the x-axis in the counterclockwise direction which has $\tan^{-1}(\frac{y}{x})$ value.

j : $\sqrt{-1}$

Radial polynomials with real values, defined as :

$$R_{nm}(\rho) = \sum_{k=0}^{n-|m|} (-1)^k \frac{(2n+1-k)!}{k!\,(n+|m|+1-k)!\,(n-|m|-k)} \rho^{n-k} \tag{3}$$

where $0 \le |m| \le n$. Because the Pseudo-Zernike Moment is defined in polar coordinates (ρ, θ) with $|\rho| \le 1$, then the computation of the Pseudo-Zernike polynomial requires a linear transformation of the image with coordinates $(i,j), i,j = 0.1,2,\dots,N-1$ to the $x, y \in R^2$ domain inside the unit circle.

The image can be reconstructed by applying inverse transforms as:

$$\hat{f}(x_i, y_k) = \sum_{n=0}^{n_{max}} \sum_{m=-n}^{n} Z_{nm} V_{nm}(x_i, y_k), i, k = 0.1,2, \dots, N-1 \quad (4)$$

All of Z_{nm} moments from *f(x,y)* to the maximum order of n_{max} can be used as feature vectors.

SVM is a statistical classification method that is generally used for two class (binary) classification problems, but now it has evolved so that it can solve multi-class problems[17], where in this study the multi-class method used is one-vs-rest.

One way to solve non-linear problems is to transform data into feature space so that data can be separated linearly with the help of the kernel [18]. Thus, the function that results from training becomes :

$$f(x) = \sum_{i \in S}^{N} \alpha_i y_i K\langle x, x_i \rangle + \beta_0 \quad (5)$$

Where K is the base radial kernel function, which has a formula :

$$K(x_i, x_{i'}) = \exp(-\gamma \|x - x'\|^2) \quad (6)$$

III. Methods

The purpose of this paper is to test the accuracy of recognition of Javanese script using PZM and Multi-class SVM algorithms. The sequence of research methods can be seen in Fig. 4. The study began by conducting literature studies and collecting datasets. The collected dataset is 20,000 datasets, where the dataset is in the form of Javanese script syllables. The image collected is Javanese scripts in the form of *nglegena* syllables, *pairs* and sandhangan which are not connected with basic script. All scripts are divided into 40 classes with 500 images for each class. The dataset will be used during the training, testing and validation processes when evaluating the system.

System design is based on analysis of literature studies that have been carried out. There are three main steps in the pattern recognition system, namely preprocessing, feature extraction, and classification.

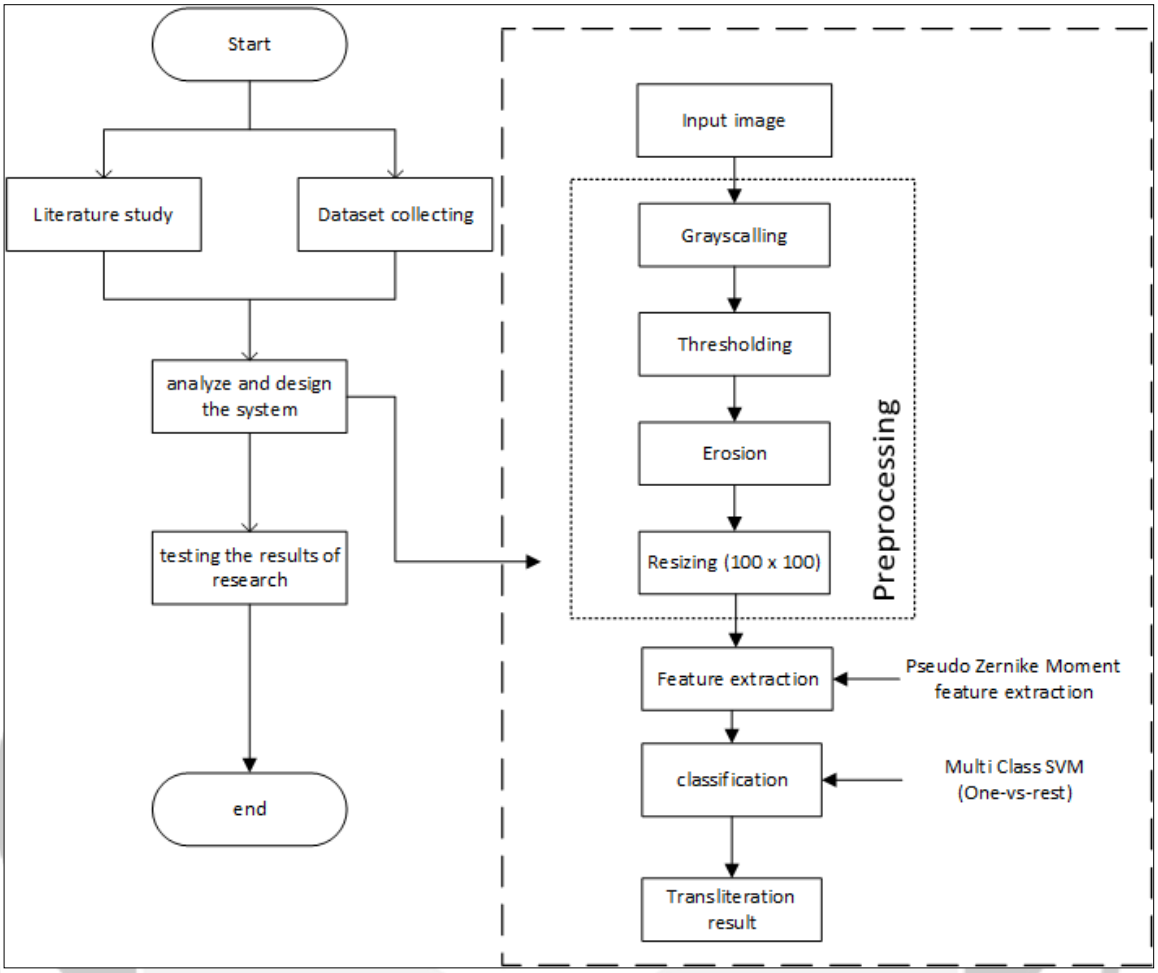


Fig. 4 Research Method

Before going through the training or testing process, each image goes through the preprocessing stage. The image preprocessing results are shown in Fig. 5. The original image (a) is converted into a gray / grayscale image (b), then the thresholding process is carried out (c) to separate the image object from the background where the object image is 1 and the background is 0. Each image is converted to 100 x 100 by maintaining aspect ratio so that when feature extraction, there is no image mapping error caused by the shape of the image that is not included entirely in the circular image region [19]. To simplify the line thickness, erosion (e) is carried out to ease the feature extraction process.

Image samples that have passed the preprocessing stage will go through feature extraction to then be used as datasets during training. Feature extraction is done using the Pseudo Zernike Moment to describe the shape features of the image.

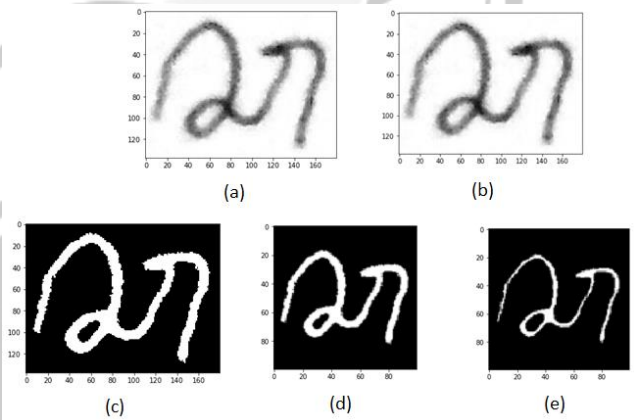


Fig. 5 Image Preprocessing

(b) *Original Image*; (b) *Grayscaled*; (c) *Thresholded*; (d) *resized*; (e) *erode*

Classification consists of training and testing with 90% separation for training data, and 10% for test data. Thus, data for training amounted to 18,000 samples, and data for testing

amounted to 2000 samples. Before being classified, all data was normalized using the min-max scaler method so that the vector values ranged from 0 to 1. Classification was done using the SVM multi-class algorithm with the one-vs-rest method. In this study, data cannot be separated linearly, so to make the data separate linearly the base radial kernel function is used.

The validation process is done using the K-fold Cross Validation method. K-fold Cross Validation is a popular validation method used to estimate accuracy by dividing a set of data folds as much as K and repeating the experiment as much as K[20]. Based on the results of the test it can be evaluated how accurately the proposed method can recognize Javanese script.

IV. Experimental Result

Evaluation of research results is done by trying different parameters in the Pseudo Zernike Moment algorithm and MSVM. In the first test, we tried to compare the level of recognition accuracy if using a different order in the Pseudo Zernike Moment algorithm. The feature vector contains the entire set of Pseudo Zernike moments from order 0 to order n, so that the maximum value of the order (n_{max}) affects the number of moments used as features. Table 1 shows the percentage of recognition accuracy in Pseudo Zernike moments with the maximum value of different orders, using the default parameters in the multi-class SVM.

Table 1 Accuracy based on n_{max}

n_{max}	Moments	Accuracy
1	2	17 %
5	12	57 %
10	36	65 %
15	72	66%
20	121	71 %

The maximum order in the Pseudo Zernike Moment affects the level of recognition accuracy. The test is done using orders from 1 to 20. The results show the percentage of accuracy increases with the increasing value of n_{max}. The highest level of accuracy obtained was 71% in the 20th order. In the higher order there was no significant increase. The higher the order used, the more time spent on calculating the moments.

The second test is done by finding the best parameters in the classification method. In the SVM algorithm, there are parameters γ and C. The best γ and C parameters are sought to produce an optimal level of recognition. For this reason, the search for the best parameter values from the range 0.01 - 1 for parameters 1, and 0.01 - 1 for parameter C. Table 2 shows the percentage of accuracy for the parameters γ and C. The test results show the best parameters are γ of 1 and C of 1 , 1.

Table 2. Multi-class SVM parameter optimization results

γ	C	Accuracy
0.01	0.01	0.3 %
0.1	0.01	9 %
1	0.01	10 %
0.01	0.1	13 %
0.1	0.1	63 %
1	0.1	62 %
0.01	1	66 %
0.1	1	82 %
1	1	86 %
0.01	1.1	67 %
0.1	1.1	82 %
1	1.1	87%

Performance evaluation of the method used is based on precision, recall, and F-Measure (f1-score) values generated [21]. Precision is used to determine the exact answer given by the system based on the information requested. Recall or sensitivity is used to measure the level of success of the system in recovering the requested information, while the f1-score is the average value that is aligned between precision and recall[22].

Table 3 shows the overall classification results for each character. The results show that the taling character has the highest f1-score value, which is 100%. The lowest level of recognition is in the character ha, which has a score of 87%. Overall, the Javanese script transliteration system to Latin letters has a precision value of 88%, a recall of 87% and a f1-score of 87%. From 2000 test data, there are 255 data that have misclassification.

Table 3. classification results of each character

No	Character	precision	recall	f1-score	support	missclasified
1	ba	0.80	0.80	0.80	40	8
2	ba (pair)	0.89	0.82	0.86	51	9
3	ca	0.88	0.86	0.87	50	7
4	ca (pair)	0.75	0.88	0.81	41	5
5	cecak (-ng)	1.00	0.98	0.99	41	1
6	da	0.74	0.82	0.78	55	10
7	da (pair)	0.98	0.96	0.97	46	2
8	dha	0.84	0.77	0.80	47	11
9	dha (pair)	0.89	1.00	0.94	41	0
10	pepet (e)	0.91	0.86	0.88	49	7
11	taling (ê)	1.00	1.00	1.00	49	0
12	ga	0.97	0.81	0.88	42	8
13	wignyan (-h)	1.00	0.98	0.99	54	1
14	ha	0.91	0.65	0.76	46	16
15	ha (pair)	0.73	0.80	0.77	51	10
16	wulu (i)	0.98	1.00	0.99	48	0
17	ja	0.81	0.94	0.87	53	3
18	ja (pair)	0.89	0.75	0.81	55	14
19	ka	0.78	0.80	0.79	56	11
20	ka (pair)	0.88	0.79	0.84	48	10
21	la	0.94	0.90	0.92	50	5
22	la (pair)	0.90	0.91	0.90	57	5
23	ma	0.82	0.89	0.86	66	7
24	ma (pair)	0.88	0.86	0.87	51	7
25	na	0.93	0.76	0.83	49	12
26	nga	0.89	0.89	0.89	46	5
27	nya	0.92	0.89	0.90	63	7
28	pa	095	0.82	0.88	66	12
29	pa (pair)	0.93	0.88	0.90	57	7

30	layar (-r)	0.96	1.00	0.98	46	0
31	ra	0.92	0.88	0.90	51	6
32	sa	0.65	0.85	0.74	41	6
33	sa (pair)	0.76	0.86	0.81	56	8
34	ta	0.81	0.87	0.84	53	7
35	ta (pair)	076	0.86	0.81	37	5
36	tarung	0.95	0.96	0.95	54	2
37	tha	0.93	0.85	0.89	48	7
38	tha (pasangan)	0.89	0.92	0.90	51	4
39	wa	0.76	0.82	0.79	55	10
40	ya	0.98	1.00	0.99	40	0
	Average/Total	0.88	0.87	0.87	2.000	255

V. Conclusion

Transliteration of Javanese script into Latin letters built using the PZM algorithm and Multi-class SVM gives quite satisfying results with a percentage of accuracy of 87%, precision of 88%, recall of 87%, and f1-score of 87%. The higher the order used in the Pseudo Zernike Moment helps improve recognition accuracy, but this affects the processing speed. The best n_{max} value for transliteration of Javanese characters into Latin letters is 20 because the use of higher order gives results that are not much different. The optimal parameter values on the model implemented are γ of 1 and C of 1.1.

Acknowledgment

This research is supported by Atma Jaya University Yogyakarta. We would like to thank Pranowo and Djoko Budiyanto Setyohadi for help on the guidance and comments that greatly improve this manuscript.

References

[1] D. Tanaya and M. Adriani, "Dictionary-based Word Segmentation for Javanese," in *Procedia Computer Science*, 2016, pp. 208–213.

[2] A. Wisnu, W. Nugraha, and H. P. Widhiatmoko, "Jumlah Transisi pada Ciri Transisi dalam Pengenalan Pola Tulisan Tangan Aksara Jawa Nglegeno dengan Multiclass Support Vector Machines Numbers of Transition Features on Basic Jawanesse (Nglegeno) Characters Recognition System with Multiclass Support Vec," *Din. Rekayasa*, vol. 8, no. 1, pp. 18–24, 2012.

[3] A. R. Himamunanto and E. Setyowati, "Text block identification in restoration process of Javanese script damage," *J. Phys. Conf. Ser.*, vol. 1013, no. 1, p. 012210. May 2018.

[4] G. S. Budhi and R. Adipranata, "Handwritten Javanese Character Recognition Using Several Artificial Neural Network Methods," *J. ICT Res. Appl.*, vol. 8, no. 3, pp. 195–212, Aug. 2015.

[5] A. R. Widiarti, A. Harjoko, Marsono, and S. Hartati, "The model and implementation of javanese script image transliteration," in *Proceedings - 2017 International Conference on Soft Computing, Intelligent System and Information Technology: Building Intelligence Through IOT and Big Data, ICSIIT 2017*, 2018, vol. 2018–Janua, pp. 51–57.

[6] T. Dasgupta, M. Sinha, and A. Basu, "Resource creation and development of an English-Bangla back transliteration system," *Int. J. Knowledge-based Intell. Eng. Syst.*, vol. 19, no. 1, pp. 35–46, Jun. 2015.

[7] D. K. Prabhakar and S. Pal, "Machine transliteration and transliterated text retrieval: a survey," *Sādhanā*, vol. 43, no. 6, p. 93, Jun. 2018.

[8] A. R. Widiarti, R. Pulungan, A. Harjoko, Marsono, and S. Hartati, "A proposed model for Javanese manuscript images transliteration," *J. Phys. Conf. Ser.*, vol. 1098, no. 1, pp. 012–014, Sep. 2018.

[9] M. Kef, L. Chergui, and S. Chikhi, "A novel fuzzy approach for handwritten Arabic character recognition," *Pattern Anal. Appl.*, vol. 19, no. 4, pp.

1041–1056, 2016.

[10] M. Elleuch and M. Kherallah, “An Improved Arabic Handwritten Recognition System using Deep Support Vector Machines,” *Int. J. Multimed. Data Eng. Manag.*, 2016.

[11] M. Elleuch, R. Maalej, and M. Kherallah, “A New design based-SVM of the CNN classifier architecture with dropout for offline Arabic handwritten recognition,” in *Procedia Computer Science*, 2016, vol. 80. pp. 1712–1723.

[12] K. Verma and R. K. Sharma, “Comparison of HMM- and SVM-based stroke classifiers for Gurmukhi script,” *Neural Comput. Appl.*, vol. 28, pp. 51–63, 2017.

[13] A. Gaur and S. Yadav, “Handwritten Hindi character recognition using k-means clustering and SVM,” in *2015 4th International Symposium on Emerging Trends and Technologies in Libraries and Information Services, ETTLIS 2015 - Proceedings*, 2015, pp. 65–70.

[14] H. R. Kanan and S. Salkhordeh, “Rotation invariant multi-frame image super resolution reconstruction using Pseudo Zernike Moments,” *Signal Processing*, vol. 118, pp. 103–114, Jan. 2016.

[15] C. Clemente, J. J. Soraghan, I. Proudler, A. De Maio, L. Pallotta, and A. Farina, “Pseudo-Zernike-based multi-pass automatic target recognition from multi-channel synthetic aperture radar,” *IET Radar, Sonar Navig.*, vol. 9, no. 4, pp. 457–466, Apr. 2015.

[16] C.-W. Chong, P. Raveendran, and R. Mukundan, “An Efficient Algorithm for Fast Computation of Pseudo-Zernike Moments,” *Int. J. Pattern Recognit. Artif. Intell.*, vol. 17, no. 06, pp. 1011–1023, Sep. 2003.

[17] M. Zareapoor, P. Shamsolmoali, D. Kumar Jain, H. Wang, and J. Yang, “Kernelized support vector machine with deep learning: An efficient approach for extreme multiclass dataset,” *Pattern Recognition Letters*, Sep-2017.

[18] S. Theodoridis, A. Pikrakis, K. Koutroumbas, and D. Cavouras, *Introduction to Pattern Recognition: A Matlab Approach*. 2010.

[19] K. Hosny, “Fast computation of accurate pseudo Zernike moments for binary and gray-level images,” *Int. Arab J. Inf. Technol.*, vol. 11, no. 3, pp. 243–249, 2014.

[20] T. T. Wong, “Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation,” *Pattern Recognit.*, vol. 48, no. 9, pp. 2839–2846, Sep. 2015.

[21] T. Saito and M. Rehmsmeier, “The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets,” *PLoS One*, vol. 10. no. 3, p. e0118432, Mar. 2015.

[22] P. Flach and M. Kull, “Precision-Recall-Gain Curves: PR Analysis Done Right,” *Adv. Neural Inf. Process. Syst. 28*, vol. 1, pp. 838–846, 2015.

serviens in lumine veritatis