DAFTAR PUSTAKA

Isro'in, L. dan Andarmoyo, S. (2012). Personal Hygiene. Yogyakarta: Graha Ilmu, pp. 1–51.

Kemenkes RI. (2010). Profil Kesehatan Indonesia Tahun 2010. Jakarta: Kementrian Kesehatan Republik Indonesia.

Kemenkes RI. (2020). Profil Kesehatan Indonesia Tahun 2020. Jakarta: Kementrian Kesehatan Republik Indonesia.

Sukmawati Tansil Tan, dkk. (2021). Buku Edukasi Ilmu Penyakit Kulit dan Kelamin. Jakarta : Fakultas Kedokteran Universitas Tarumanagara.

Reza Pahlevi. (2016). Perbedaan Jamur dan Eksim. Retrieved from https://m.klikdokter.com/tanya-dokter/read/2808549/perbedaan-jamur-dan-eksim

Oka Sari Utari, I Gusti Ayu; Sudiasa, Sudiasa; Prapyatiningsih, R Yully. (Mei 2019). Diagnosis Dan Penatalaksanaan Angioedema Di Bidang THT-KL. Jurnal Kedokteran, 4(2), 30-49. doi: http://dx.doi.org/10.36679/kedokteran.v4i2.103.

Fitriany, Julia; Alratisda, Fajri. (Mei 2019). Stevens Johnson Syndrome. Jurnal Averrous Vol.5 No.1. https://ojs.unimal.ac.id/averrous/article/download/1632/899.

Evalina, Rita. (2017). Penanganan Luka Pada Stevens-Johnson's Syndrome/ Toxic Epidermal Necrolysis. Medan : Departemen Ilmu Kesehatan Anak Universitas Sumatera Utara. http://repository.usu.ac.id/handle/123456789/69085.

Sinanoto, Melisa. (2017). Diagnosis dan Tatalaksana Urtikaria. CDK-250, Vol. 44 No. 3. https://media.neliti.com/media/publications/401093-diagnosis-dan-tatalaksana-urtikaria-5cabb28c.pdf.

Partogi, Donna. (2008). Fixed Drug Eruption. Medan: Departemen Ilmu Kesehatan Kulit dan Kelamin. https://repository.usu.ac.id/bitstream/handle/123456789/3411/08E00858.pdf

Ramdani, Adi Topan; Sulistiyani; Rosyidah, Devi Usdiyana; Nursanto, Dodik. (2021). Pengaruh Perubahan Kadar pH Kulit terhadap Napkin Eczema. Prociding Call For Paper Thalamus Fakultas Kedokteran, 30–40. https://proceedings.ums.ac.id/index.php/kedokteran/article/view/219/220.

I Putu Gilang Iswara Wijaya, IGK Darmada, Luh Made Mas Rusyati. (2016). Edukasi Dan Penatalaksanaan Dermatitis Kontak Iritan Kronis Di RSUP Sanglah Denpasar Bali Tahun 2014/2015. E-Jurnal Medika, Vol. 5 No.8. https://ojs.unud.ac.id/index.php/eum/article/download/22867/14999.

Trisna Yuliharti Tersinanda, Luh Made Mas Rusyati. (2013). Dermatitis Kontak Alergi. E-Jurnal Medika Udayana, [S.l.], p. 1446-1461. https://ojs.unud.ac.id/index.php/eum/article/view/6113/4604.

Siti Aminah Tri Susila Estri. (2009). Pola Penyebab dan Rekurensi Dermatitis Numularis. Mutiara Medika Edisi Khusus Vol. 9 No. 2: 129 – 135. https://journal.umy.ac.id/index.php/mm/article/viewFile/1616/1661.

Andini Saraswati, Agustyas Tjiptaningrum, Ayla Karyus. (2016). Penatalaksaan Holistik Penyakit Kulit Neurodermatitis Sirkumskripta. JPM Ruwa Jurai, Vol.2, No.1. https://juke.kedokteran.unila.ac.id/index.php/JPM/article/download/1168/pdf

Movita, Theresia. (2014). Tatalaksana Dermatitis Atopik. CDK-222, vol. 41 no. 11. https://www.rsi-ibnusina.com/media/file/ttk.pdf.

SMF Kesehatan Kulit dan Kelamin. (2020). Infeksi Jamur Pada Kulit (Tinea). Surabaya: Instalasi Promosi Kesehatan Rumah Sakit & Hubungan Masyarakat (PKRS & Humas) Rsud Dr. Soetomo.

Veronica. (2016). Tinea Kapitis Tipe Gray Patch Yang Diduga Disebabkan Oleh Microsporum dan Trichophyton. Denpasar: Fakultas Kedokteran Unud.

Hardanti, Syafira Diska. (2021). Gambaran Jamur Dermatofita Penyebab Tinea Unguium Pada Kuku Petani, Tukang Cuci, Kuli Pasir Dan Petambak (Studi Pustaka). Diploma Thesis, Poltekkes Tanjungkarang. https://repository.poltekkes-tjk.ac.id/id/eprint/1728/6/6.%20BAB%20II.pdf.

Kuruvella T, Pandey S. (2022). Tinea Barbae. StatPearls. https://www.ncbi.nlm.nih.gov/books/NBK563204/

Khairina. (2013). Tinea Fasialis Pada Anak. Medan: Fakultas Kedokteran USU. https://repository.usu.ac.id/bitstream/handle/123456789/40778/TINEA%20FASIALIS.pdf.

Ramadhany, Anugrah. (2018). Hubungan Kejadian Tinea Manus Dengan Penggunaan Sarung Tangan Pada Petugas Kebersihan Di Kecamatan Medan Kota. Medan: Fakultas Kedokteran Universitas Muhammadiyah Sumatera Utara.

Minerva Nadia Putri, Fitrianisa Burmana, Azelia Nusadewiarti. (2017). Penatalaksanaan dan Pencegahan Tinea Korporis pada Pasien Wanita dan Anggota Keluarga. J AgromedUnila, Vol.4, No.1.

Putra, Diaz Ananta and Redjeki S, TM Sri. (2014). Pengaruh Higiene Sanitasi Dengan Kejadian Tinea Kruris Pada Santri Laki-Laki Di Pesantren Rhoudlotul Quran Kauman Semarang. Undergraduate thesis, Faculty of Medicine Diponegoro University.

Nuhradi, Stefani. (2017). Tatalaksana Dermatitis Seboroik Terkini. Denpasar: Fakultas Kedokteran UNUD.

Darmada, I Putu Adi (2018) Isolasi Dan Uji Sensitivitas Jamur Candida Albicans Dan Candida Non-Albicans terhadap Flukonazol. Diploma thesis, Jurusan Analis Kesehatan.

Pramono, Annisa Shafira; Soleha, Tri Umiana. (2018). Pitiriasis Versikolor:

Diagnosis dan Terapi. J Agromedicine, Vol. 5, No.1. https://juke.kedokteran.unila.ac.id/index.php/agro/article/download/1981/pdf.

Lowe, D. G. (2004). Distinctive Image Features From Scale-Invariant Keypoints. International Journal Of Computer Vision, 60(2), 91-110.

Pratama, Lingga Eka. (2014). Implementasi Algoritma Sift Untuk Melakukanklasifikasi Bahan Bakar Kendaraan Roda Empat Pada SPBU. Jurnal Ilmiah Komputer dan Informatika, Edisi 1, Vol.1. https://elib.unikom.ac.id/files/disk1/716/jbptunikompp-gdl-linggaekap-35759-6-21.10109-a.pdf.

Shapiro, Linda. (2006). Computer Vision CSE/EE 576. https://courses.cs.washington.edu/courses/cse576/06sp/notes/Interest2.pdf.

Khomaeni, Ayat Tulloh. (2020). Penerapan Metode Scale Invariant Feature Transform (SIFT) pada Augmented Reality dalam Pengenalan Gedung Universitas Islam Negeri Malang. Malang: Fakultas Sains dan Teknologi UIN Malang.

Csurka, G., Dance, C., Fan, L., Willamowski, J., & Bray, C. (2004). Visual categorization with bags of keypoints. In Workshop on statistical learning in computer vision, ECCV (Vol. 1, No. 1-22, pp. 1-2). http://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/csurka-eccv-04.pdf.

Astuti, Sri. (2020). Algoritma K-Means Dalam Menentukan Penerima Beasiswa Upz (Unit Pengumpulan Zakat) Pada Mahasiswa Uin Sumatera Utara Medan. Medan: Fakultas Sains dan Teknologi UIN Medan.

Parapat, Indri Monika. (2017). Penerapan Metode Support Vector Machine (SVM) pada Klasifikasi Penyimpangan Tumbuh Kembang Anak. Malang: Fakultas Ilmi Komputer Universitas Brawijaya.

Nugroho, Anto Satriyo; Witarto, Arief Budi; Handoko, Dwi (2003). Application of Support Vector Machine in Bioinformatics. https://asnugroho.net/papers/ikcsvm.pdf.

Merianti, Revika Dwi. (2020). Implementasi Metode Support Vector Machine dan Random Forest pada Data Ekspresi Gen Microarray. Yogyakarta: Fakultas MIPA Universitas Islam Indonesia.

Ritonga, Alven Safik; Purwaningsih, Endah Supeni. (2018). Penerapan Metode Support Vector Machine (SVM) dalam Klasifikasi Kualitas Pengelasan SMAW (Shield Metal Arc Welding). Jurnal Ilmiah Edutic (Vol.5, No.1).

Wisudawati, Dinda Tri. (2020). Analisis Sentimen terhadap Dampak Covid-19 pada Performa E-Commerce di Indonesia Menggunakan Support Vector Machine (Review Aplikasi Tokopedia pada Google Play). Semarang: Universitas Muhamadiyah Semarang.

Fadhila Tangguh Admojo; Yudha Islami Sulistya. (2022). Analisis Performa

Algoritma Stochastic Gradient Descent (SGD) dalam Mengklasifikasi Tahu Berformalin. Indonesian Journal of Data and Science (Vol.3, No.1). https://doi.org/10.56705/ijodas.v3i1.42

Połap D, Winnicka A, Serwata K, Kęsik K, Woźniak M. An Intelligent System for Monitoring Skin Diseases. Sensors (Basel). 2018 Aug 4;18(8):2552. doi: 10.3390/s18082552. PMID: 30081540; PMCID: PMC6111999.

Pangestu, Dhimas Galuh. (2021). Identifikasi Penyakit Kulit Menggunakan Gray Level Co-Occurrance Matrix (GLCM) Dan Support Vector Machine (SVM). Bandung: Universitas Telkom.

LAMPIRAN

Lampiran 1 *Source code* model klasifikasi menggunakan SVM dengan pengelompokan *K-Means*

```
import os
import cv2
import itertools
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
f1_score, classification_report
from sklearn.utils import shuffle


# Preparing the dataset
path = './data'
image_path = []
image_label = []
for sub_folder in os.listdir(os.path.join(path)):
    if sub_folder == '.DS_Store' or sub_folder ==
'.ipynb_checkpoints':
        continue
    sub_folder_files = os.listdir(os.path.join(path,
sub_folder))
    len_sub_folder = len(sub_folder_files) - 1
    for i, filename in enumerate(sub_folder_files):
        if filename == '.ipynb_checkpoints' or filename ==
'.DS_Store':
            continue
        image_path.append(os.path.join(path, sub_folder,
filename))
        image_label.append(sub_folder)


# Split data to train and test
image_path_train, image_path_test =
train_test_split(image_path, test_size=0.2, random_state=19,
stratify=image_label)


# Function for Feature Extraction
def CalcFeatures(img, th):
    sift = cv2.SIFT_create(th)
    kp, des = sift.detectAndCompute(img, None)
    return des

def getSIFTFeature(thresh, path):
```

```
    features = []
    for file in path:
        img = cv2.imread(file, 0)
        img_des = CalcFeatures(img, thresh)
        if img_des is not None:
            features.append(img_des)
    features = np.vstack(features)
    return features

def bag_of_features(features, centres, k = 500):
    vec = np.zeros((1, k))
    for i in range(features.shape[0]):
        feat = features[i]
        diff = np.tile(feat, (k, 1)) - centres
        dist = pow(((pow(diff, 2)).sum(axis = 1)), 0.5)
        idx_dist = dist.argsort()
        idx = idx_dist[0]
        vec[0][idx] += 1
    return vec


# RBF SVM function
class RBFKernelSVM:
    def __init__(self, kernel='rbf', gamma=None, C=1):
        self.kernel = kernel
        self.gamma = gamma
        self.C = C

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Compute the kernel matrix
        if self.kernel == 'rbf':
            if self.gamma is None:
                self.gamma = 1 / n_features
            K = np.zeros((n_samples, n_samples))
            for i in range(n_samples):
                for j in range(n_samples):
                    K[i,j] = np.exp(-self.gamma *
np.linalg.norm(X[i]-X[j])**2)
        else:
            raise ValueError("Unsupported kernel function")

        # Solve the dual optimization problem using quadratic
programming
        P = np.outer(y, y) * K
        q = -np.ones(n_samples)
        G = np.vstack((-np.eye(n_samples), np.eye(n_samples)))
```

```
        h = np.hstack((np.zeros(n_samples), np.ones(n_samples)
* self.C))
        A = y.reshape(1, -1)
        A = A.astype('float')
        b = np.zeros(1)

        alpha = np.zeros(n_samples)
        from cvxopt import matrix, solvers
        solvers.options['show_progress'] = None
        sol = solvers.qp(matrix(P), matrix(q), matrix(G),
matrix(h), matrix(A), matrix(b))
        alpha = np.ravel(sol['x'])

        # Find the support vectors
        sv_indices = alpha > 1e-5
        self.support_vectors = X[sv_indices]
        self.support_vector_labels = y[sv_indices]
        self.support_vector_alpha = alpha[sv_indices]

        # Compute the bias term
        sv_index = np.argmax(self.support_vector_alpha)
        self.b = self.support_vector_labels[sv_index]
        for i in range(n_samples):
            if alpha[i] > 1e-5:
                self.b -= alpha[i] * y[i] * K[i,sv_index]
        self.b /= len(self.support_vector_alpha)

    def predict(self, X):
        if self.kernel == 'rbf':
            K = np.zeros((len(X), len(self.support_vectors)))
            for i in range(len(X)):
                for j in range(len(self.support_vectors)):
                    K[i,j] = np.exp(-self.gamma *
np.linalg.norm(X[i]-self.support_vectors[j])**2)
        else:
            raise ValueError("Unsupported kernel function")

        predictions = np.sign(np.dot(K,
self.support_vector_alpha * self.support_vector_labels) +
self.b)
        return predictions.astype(int)


# Linear SVM Function
def compute_cost(W, X, Y):
    # calculate hinge loss
    N = X.shape[0]
    distances = 1 - Y * (np.dot(X, W))
```

```
    distances[distances < 0] = 0  # equivalent to max(0,
distance)
    hinge_loss = reg_strength * (np.sum(distances) / N)

    # calculate cost
    cost = 1 / 2 * np.dot(W, W) + hinge_loss
    return cost

def calculate_cost_gradient(W, X_batch, Y_batch):
    # if only one example is passed (eg. in case of SGD)
    if type(Y_batch) == np.float64 or type(Y_batch) ==
np.int64:
        Y_batch = np.array([Y_batch])
        X_batch = np.array([X_batch])
    distance = 1 - (Y_batch * np.dot(X_batch, W))
    dw = np.zeros(len(W))
    for ind, d in enumerate(distance):
        if max(0, d) == 0:
            di = W
        else:
            di = W - (reg_strength * Y_batch[ind] *
X_batch[ind])
        dw += di
    dw = dw/len(Y_batch)  # average
    return dw

def sgd(features, outputs):
    max_epochs = 5000
    weights = np.zeros(features.shape[1])
    nth = 0
    prev_cost = float("inf")
    cost_threshold = 0.01  # in percent
    # stochastic gradient descent
    for epoch in range(1, max_epochs):
        # shuffle to prevent repeating update cycles
        X, Y = shuffle(features, outputs)
        for ind, x in enumerate(X):
            ascent = calculate_cost_gradient(weights, x,
Y[ind])
            weights = weights - (learning_rate * ascent)
        # convergence check on 2^nth epoch
        if epoch == 2 ** nth or epoch == max_epochs - 1:
            cost = compute_cost(weights, features, outputs)

            # stoppage criterion
            if abs(prev_cost - cost) < cost_threshold *
prev_cost:
                return weights
            prev_cost = cost
```

```
            nth += 1
    return weights

def SVMClassifier(X_train, y_train, reg_strength,
learning_rate):
    W = sgd(X_train, y_train)
    # print("weights are: {}".format(W))
    return W


# Training for Model "Bukan Alergi dan Jamur" vs "Alergi atau
Jamur"
thresh = [10, 15, 20, 25, 30, 40]
k_value = [0.05, 0.1, 0.2]
C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
gamma = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

centers = {}
models = {}
features = {}

for th in thresh:
    centers[th] = {}
    models[th] = {}
    features[th] = getSIFTFeature(th, image_path_train)
    for k in k_value:
        k = int(k * th * len(image_path_train))

        criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 10, 0.1)
        flags = cv2.KMEANS_RANDOM_CENTERS
        compact, labels, centres = cv2.kmeans(features[th], k,
None, criteria, 10, flags)
        centers[th][k] = centres

        y_train = []
        vec = []
        for file in image_path_train:
            img = cv2.imread(file, 0)
            img_des = CalcFeatures(img, th)
            if img_des is not None:
                img_vec = bag_of_features(img_des, centres, k)
                vec.append(img_vec)
                if file.split('/')[2] == 'bukan_alergi_jamur':
                    y_train.append(1)     # 1 =
bukan_alergi_jamur
                else:
                    y_train.append(-1)    # -1 = alergi or
jamur
```

```
        X_train = np.vstack(vec)
        y_train = np.array(y_train)

        y_test = []
        vec = []
        for file in image_path_test:
            img = cv2.imread(file, 0)
            img_des = CalcFeatures(img, th)
            if img_des is not None:
                img_vec = bag_of_features(img_des, centres, k)
                vec.append(img_vec)
                if file.split('/')[2] == 'bukan_alergi_jamur':
                    y_test.append(1)     # 1 =
bukan_alergi_jamur
                else:
                    y_test.append(-1)    # -1 = alergi or
jamur
        X_test = np.vstack(vec)
        y_test = np.array(y_test)

        models[th][k] = {}

        for g in gamma:
            models[th][k][g] = {}
            for c in C:

                rbf = RBFKernelSVM(gamma=g, C=c)
                rbf.fit(X_train, y_train)
                models[th][k][g][c] = rbf

                train_preds = rbf.predict(X_train)

                train_acc = accuracy_score(y_train,
np.array(train_preds))
                train_f1 = f1_score(y_train, train_preds)
                train_conf_mat = confusion_matrix(y_train,
train_preds)

                test_preds = rbf.predict(X_test)

                test_acc = accuracy_score(y_test,
np.array(test_preds))
                test_f1 = f1_score(y_test, test_preds)
                test_conf_mat = confusion_matrix(y_test,
test_preds)
                if(test_acc > 0.70 and test_f1 > 0.70):
                    print('Calc for threshold = {} and k = {}
|| gamma = {} and C = {}'.format(th, k, g, c))
                    print('Train Data')
```

```
                    print('Accuracy = {}\nF1 Score =
{}\nConfusion matrix :\n{}'.format(train_acc*100,
train_f1*100, train_conf_mat))
                    print('Test Data')
                    print('Accuracy = {}\nF1 Score =
{}\nConfusion matrix :\n{}'.format(test_acc*100, test_f1*100,
test_conf_mat))


# Training for Model "Alergi Kulit" vs "Jamur Kulit"
reg_strengths = [1e4, 2e4, 5e4, 1e5]
learning_rates = [1e-8, 1e-7, 2e-7]

AvJcenters = {}
AvJweights = {}
AvJfeatures = {}

for th in thresh:
    AvJcenters[th] = {}
    AvJweights[th] = {}
    AvJfeatures[th] = getSIFTFeature(th, image_path_train)

    for k in k_value:
        k = int(k * th * len(image_path_train))

        criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 10, 0.1)
        flags = cv2.KMEANS_RANDOM_CENTERS
        compact, labels, centres = cv2.kmeans(AvJfeatures[th],
k, None, criteria, 10, flags)
        AvJcenters[th][k] = centres

        y_train = []
        vec = []
        for file in image_path_train:
            img = cv2.imread(file, 0)
            img_des = CalcFeatures(img, th)
            if img_des is not None:
                img_vec = bag_of_features(img_des, centres, k)

                if file.split('/')[2] == 'alergi_kulit':
                    vec.append(img_vec)
                    y_train.append(1)     # 1 = alergi_kulit
                elif file.split('/')[2] == 'jamur_kulit':
                    vec.append(img_vec)
                    y_train.append(-1)    # -1 = jamur_kulit
        vec = np.vstack(vec)
        intercept_col = np.ones((vec.shape[0], 1))
        X_train = np.append(vec, intercept_col, axis=1)
```

```
        y_train = np.array(y_train)

        # X_train, X_val, y_train, y_val =
train_test_split(X_data, y_data, test_size=0.25,
random_state=12, stratify=y_data)

        y_test = []
        vec = []
        for file in image_path_test:
            img = cv2.imread(file, 0)
            img_des = CalcFeatures(img, th)
            if img_des is not None:
                img_vec = bag_of_features(img_des, centres, k)

                if file.split('/')[2] == 'alergi_kulit':
                    vec.append(img_vec)
                    y_test.append(1)     # 1 = bukan alergi
dan jamur
                elif file.split('/')[2] == 'jamur_kulit':
                    vec.append(img_vec)
                    y_test.append(-1)    # -1 = alergi dan
jamur
        vec = np.vstack(vec)
        intercept_col = np.ones((vec.shape[0], 1))
        X_test = np.append(vec, intercept_col, axis=1)
        y_test = np.array(y_test)

        AvJweights[th][k] = {}

        for reg_strength in reg_strengths:
            AvJweights[th][k][reg_strength] = {}
            for learning_rate in learning_rates:
                print('Calculating for threshold = {} and k =
{} || rs: {}, lr: {}'.format(th, k, reg_strength,
learning_rate))
                W = SVMClassifier(X_train, y_train,
reg_strength, learning_rate)
                AvJweights[th][k][reg_strength][learning_rate]
= W

                train_preds = np.array([])
                for i in range(X_train.shape[0]):
                    yp = np.sign(np.dot(W, X_train[i])) #model
                    train_preds = np.append(train_preds, yp)

                train_acc = accuracy_score(y_train,
train_preds)
                train_f1 = f1_score(y_train, train_preds)
```

```
                train_conf_mat = confusion_matrix(y_train,
train_preds)


                test_preds = np.array([])
                for i in range(X_test.shape[0]):
                    yp = np.sign(np.dot(W, X_test[i])) #model
                    test_preds = np.append(test_preds, yp)

                test_acc = accuracy_score(y_test, test_preds)
                test_f1 = f1_score(y_test, test_preds)
                test_conf_mat = confusion_matrix(y_test,
test_preds)
                if(test_acc > 0.70 and test_f1 > 0.70):
                    print('Train Data')
                    print('Accuracy = {}\nF1 Score =
{}\nConfusion matrix :\n{}'.format(train_acc*100,
train_f1*100, train_conf_mat))
                    print('Test Data')
                    print('Accuracy = {}\nF1 Score =
{}\nConfusion matrix :\n{}'.format(test_acc*100, test_f1*100,
test_conf_mat))
                else:
                    print('Accuracy and F1 score below 60%')
            print('-----------------------------------------
-------')


# Pengujian model SVM
def predict_label(image_path):
    img = cv2.imread(image_path, 0)

    th = 40
    k = 334
    center_penyakit_vs_non = centers[th][k]
    model_penyakit_vs_non = models[th][k][0.1][100] #
threshold = 40 and k = 334 || gamma = 0.1 and C = 100
    img_des = CalcFeatures(img, th)
    if img_des is not None:
        X = bag_of_features(img_des, center_penyakit_vs_non,
k)
        yp = model_penyakit_vs_non.predict(X)

        if yp == -1: # Alergi or Jamur
            th = 40
            k = 668
            center_alergi_vs_jamur = AvJcenters[th][k]
```

```
            model_alergi_vs_jamur =
AvJweights[th][k][50000.0][1e-07] # threshold = 40 and k = 668
|| rs: 50000.0, lr: 1e-07
            img_des = CalcFeatures(img, th)
            if img_des is not None:
                img_vec = bag_of_features(img_des,
center_alergi_vs_jamur, k)
                X = np.transpose(np.append(img_vec,
np.ones((1,1)), axis=1))
                yp = np.sign(np.dot(model_alergi_vs_jamur, X))
                if yp == 1:
                    return 'alergi_kulit'
                else:
                    return 'jamur_kulit'

        else:
            return 'bukan_alergi_jamur'


train_preds = np.array([])
y_test = np.array([])
for image_path in image_path_test:
    yp = predict_label(image_path)
    if yp is not None:
        train_preds = np.append(train_preds, yp)
        y_test = np.append(y_test, image_path.split('/')[2])

acc = accuracy_score(y_test, train_preds)
conf_mat = confusion_matrix(y_test, train_preds)
print()
print('Accuracy : {}\nConfusion Matrix: \n{}\nResult
:\n{}'.format(acc*100, conf_mat, classification_report(y_test,
train_preds)))
```

Lampiran 2 *Source code* model klasifikasi menggunakan SVM tanpa pengelompokan *K-Means*

```
import os
import cv2
import itertools
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
f1_score, classification_report
from sklearn.utils import shuffle


# Preparing the dataset
path = './data'
image_path = []
image_label = []
for sub_folder in os.listdir(os.path.join(path)):
    if sub_folder == '.DS_Store' or sub_folder ==
'.ipynb_checkpoints':
        continue
    sub_folder_files = os.listdir(os.path.join(path,
sub_folder))
    len_sub_folder = len(sub_folder_files) - 1
    for i, filename in enumerate(sub_folder_files):
        if filename == '.ipynb_checkpoints' or filename ==
'.DS_Store':
            continue
        image_path.append(os.path.join(path, sub_folder,
filename))
        image_label.append(sub_folder)


# Split data to train and test
image_path_train, image_path_test =
train_test_split(image_path, test_size=0.2, random_state=19,
stratify=image_label)


# Function for Feature Extraction
def CalcFeatures(img, th):
    sift = cv2.SIFT_create(th)
    kp, des = sift.detectAndCompute(img, None)
    return des

def getSIFTFeature(thresh, path):
    features = []
    for file in path:
```

```
        img = cv2.imread(file, 0)
        img_des = CalcFeatures(img, thresh)
        if img_des is not None:
            features.append(img_des)
    features = np.vstack(features)
    return features


# RBF SVM function
class RBFKernelSVM:
    def __init__(self, kernel='rbf', gamma=None, C=1):
        self.kernel = kernel
        self.gamma = gamma
        self.C = C

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Compute the kernel matrix
        if self.kernel == 'rbf':
            if self.gamma is None:
                self.gamma = 1 / n_features
            K = np.zeros((n_samples, n_samples))
            for i in range(n_samples):
                for j in range(n_samples):
                    K[i,j] = np.exp(-self.gamma *
np.linalg.norm(X[i]-X[j])**2)
        else:
            raise ValueError("Unsupported kernel function")

        # Solve the dual optimization problem using quadratic
programming
        P = np.outer(y, y) * K
        q = -np.ones(n_samples)
        G = np.vstack((-np.eye(n_samples), np.eye(n_samples)))
        h = np.hstack((np.zeros(n_samples), np.ones(n_samples)
* self.C))
        A = y.reshape(1, -1)
        A = A.astype('float')
        b = np.zeros(1)

        alpha = np.zeros(n_samples)
        from cvxopt import matrix, solvers
        solvers.options['show_progress'] = None
        sol = solvers.qp(matrix(P), matrix(q), matrix(G),
matrix(h), matrix(A), matrix(b))
        alpha = np.ravel(sol['x'])

        # Find the support vectors
```

```
        sv_indices = alpha > 1e-5
        self.support_vectors = X[sv_indices]
        self.support_vector_labels = y[sv_indices]
        self.support_vector_alpha = alpha[sv_indices]

        # Compute the bias term
        sv_index = np.argmax(self.support_vector_alpha)
        self.b = self.support_vector_labels[sv_index]
        for i in range(n_samples):
            if alpha[i] > 1e-5:
                self.b -= alpha[i] * y[i] * K[i,sv_index]
        self.b /= len(self.support_vector_alpha)

    def predict(self, X):
        if self.kernel == 'rbf':
            K = np.zeros((len(X), len(self.support_vectors)))
            for i in range(len(X)):
                for j in range(len(self.support_vectors)):
                    K[i,j] = np.exp(-self.gamma *
np.linalg.norm(X[i]-self.support_vectors[j])**2)
        else:
            raise ValueError("Unsupported kernel function")

        predictions = np.sign(np.dot(K,
self.support_vector_alpha * self.support_vector_labels) +
self.b)
        return predictions.astype(int)


# Linear SVM Function
def compute_cost(W, X, Y):
    # calculate hinge loss
    N = X.shape[0]
    distances = 1 - Y * (np.dot(X, W))
    distances[distances < 0] = 0  # equivalent to max(0,
distance)
    hinge_loss = reg_strength * (np.sum(distances) / N)

    # calculate cost
    cost = 1 / 2 * np.dot(W, W) + hinge_loss
    return cost

def calculate_cost_gradient(W, X_batch, Y_batch):
    # if only one example is passed (eg. in case of SGD)
    if type(Y_batch) == np.float64 or type(Y_batch) ==
np.int64:
        Y_batch = np.array([Y_batch])
        X_batch = np.array([X_batch])
    distance = 1 - (Y_batch * np.dot(X_batch, W))
```

```
    dw = np.zeros(len(W))
    for ind, d in enumerate(distance):
        if max(0, d) == 0:
            di = W
        else:
            di = W - (reg_strength * Y_batch[ind] *
X_batch[ind])
        dw += di
    dw = dw/len(Y_batch)  # average
    return dw

def sgd(features, outputs):
    max_epochs = 5000
    weights = np.zeros(features.shape[1])
    nth = 0
    prev_cost = float("inf")
    cost_threshold = 0.01  # in percent
    # stochastic gradient descent
    for epoch in range(1, max_epochs):
        # shuffle to prevent repeating update cycles
        X, Y = shuffle(features, outputs)
        for ind, x in enumerate(X):
            ascent = calculate_cost_gradient(weights, x,
Y[ind])
            weights = weights - (learning_rate * ascent)
        # convergence check on 2^nth epoch
        if epoch == 2 ** nth or epoch == max_epochs - 1:
            cost = compute_cost(weights, features, outputs)

            # stoppage criterion
            if abs(prev_cost - cost) < cost_threshold *
prev_cost:
                return weights
            prev_cost = cost
            nth += 1
    return weights

def SVMClassifier(X_train, y_train, reg_strength,
learning_rate):
    W = sgd(X_train, y_train)
    # print("weights are: {}".format(W))
    return W


# Training for Model “Alergi Jamur” vs “All”
reg_strengths = [1e4, 2e4, 5e4, 1e5]
learning_rates = [1e-8, 1e-7, 2e-7]

AJvAllweights = {}
```

```
AJvAllfeatures = {}

for th in thresh:
    # AvJcenters[th] = {}
    AJvAllweights[th] = {}
    AJvAllfeatures[th] = getSIFTFeature(th, image_path_train)

    y_train = []
    vec = []
    for file in image_path_train:
        img = cv2.imread(file, 0)
        img_des = CalcFeatures(img, th)
        if img_des is not None:
            img_vec = np.resize(img_des, (1, 128*th))
            if file.split('/')[2] == 'alergi_kulit' or
file.split('/')[2] == 'jamur_kulit':
                vec.append(img_vec)
                y_train.append(1)     # 1 = alergi_kulit or
jamur_kulit
            else:
                vec.append(img_vec)
                y_train.append(-1)    # -1 = other
    vec = np.vstack(vec)
    intercept_col = np.ones((vec.shape[0], 1))
    X_train = np.append(vec, intercept_col, axis=1)
    y_train = np.array(y_train)

    # X_train, X_val, y_train, y_val =
train_test_split(X_data, y_data, test_size=0.25,
random_state=12, stratify=y_data)

    y_test = []
    vec = []
    for file in image_path_test:
        img = cv2.imread(file, 0)
        img_des = CalcFeatures(img, th)
        if img_des is not None:
            img_vec = np.resize(img_des, (1, 128*th))
            if file.split('/')[2] == 'alergi_kulit' or
file.split('/')[2] == 'jamur_kulit':
                vec.append(img_vec)
                y_test.append(1)     # 1 = alergi_kulit or
jamur_kulit
            else:
                vec.append(img_vec)
                y_test.append(-1)    # -1 = other
    vec = np.vstack(vec)
    intercept_col = np.ones((vec.shape[0], 1))
    X_test = np.append(vec, intercept_col, axis=1)
```

```
    y_test = np.array(y_test)

    AJvAllweights[th] = {}

    for reg_strength in reg_strengths:
        AJvAllweights[th][reg_strength] = {}
        for learning_rate in learning_rates:
            print('Calculating for threshold = {} || rs: {},
lr: {}'.format(th, reg_strength, learning_rate))
            W = SVMClassifier(X_train, y_train, reg_strength,
learning_rate)
            AJvAllweights[th][reg_strength][learning_rate] = W

            train_preds = np.array([])
            for i in range(X_train.shape[0]):
                yp = np.sign(np.dot(W, X_train[i])) #model
                train_preds = np.append(train_preds, yp)

            train_acc = accuracy_score(y_train, train_preds)
            train_f1 = f1_score(y_train, train_preds)
            train_conf_mat = confusion_matrix(y_train,
train_preds)


            test_preds = np.array([])
            for i in range(X_test.shape[0]):
                yp = np.sign(np.dot(W, X_test[i])) #model
                test_preds = np.append(test_preds, yp)

            test_acc = accuracy_score(y_test, test_preds)
            test_f1 = f1_score(y_test, test_preds)
            test_conf_mat = confusion_matrix(y_test,
test_preds)

            if(test_acc > 0.70 or test_f1 > 0.70):
                print('Train Data')
                print('Accuracy = {}\nF1 Score = {}\nConfusion
matrix :\n{}'.format(train_acc*100, train_f1*100,
train_conf_mat))
                print('Test Data')
                print('Accuracy = {}\nF1 Score = {}\nConfusion
matrix :\n{}'.format(test_acc*100, test_f1*100,
test_conf_mat))
            else:
                print('Accuracy and F1 score below 70%')
        print('-----------------------------------------------
---')
```

```
# Training for Model “Alergi Kulit” vs “Jamur Kulit”
reg_strengths = [1e4, 2e4, 5e4, 1e5]
learning_rates = [1e-8, 1e-7, 2e-7]

AvJweights = {}
AvJfeatures = {}

for th in thresh:
    # AvJcenters[th] = {}
    AvJweights[th] = {}
    AvJfeatures[th] = getSIFTFeature(th, image_path_train)

    y_train = []
    vec = []
    for file in image_path_train:
        img = cv2.imread(file, 0)
        img_des = CalcFeatures(img, th)
        if img_des is not None:
            img_vec = np.resize(img_des, (1, 128*th))
            if file.split('/')[2] == 'alergi_kulit':
                vec.append(img_vec)
                y_train.append(1)     # 1 = alergi_kulit
            elif file.split('/')[2] == 'jamur_kulit':
                vec.append(img_vec)
                y_train.append(-1)    # -1 = jamur_kulit
    vec = np.vstack(vec)
    intercept_col = np.ones((vec.shape[0], 1))
    X_train = np.append(vec, intercept_col, axis=1)
    y_train = np.array(y_train)

    # X_train, X_val, y_train, y_val =
train_test_split(X_data, y_data, test_size=0.25,
random_state=12, stratify=y_data)

    y_test = []
    vec = []
    for file in image_path_test:
        img = cv2.imread(file, 0)
        img_des = CalcFeatures(img, th)
        if img_des is not None:
            img_vec = np.resize(img_des, (1, 128*th))
            if file.split('/')[2] == 'alergi_kulit':
                vec.append(img_vec)
                y_test.append(1)     # 1 = bukan alergi dan
jamur
            elif file.split('/')[2] == 'jamur_kulit':
                vec.append(img_vec)
                y_test.append(-1)    # -1 = alergi dan jamur
    vec = np.vstack(vec)
```

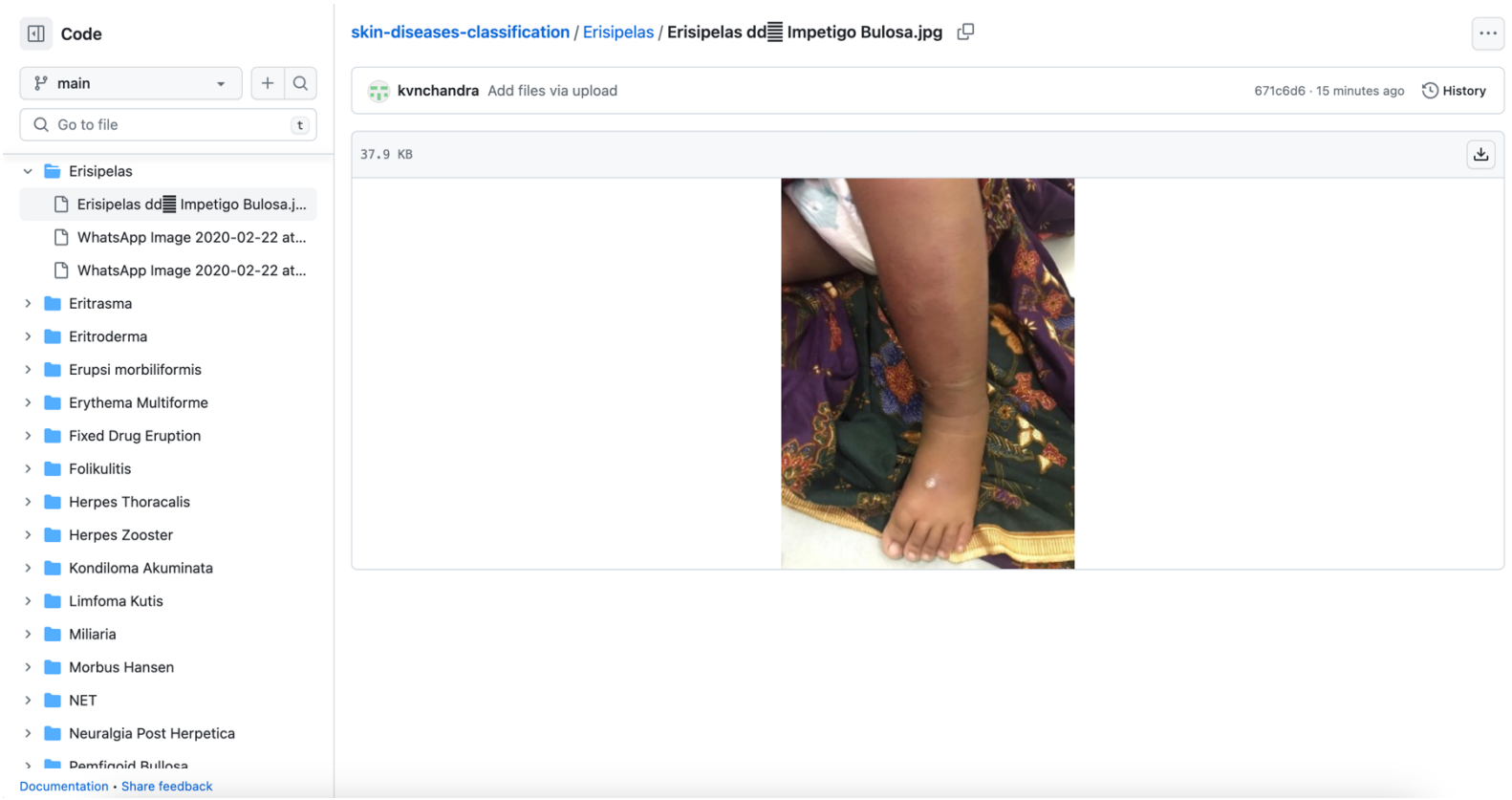
```
    intercept_col = np.ones((vec.shape[0], 1))
    X_test = np.append(vec, intercept_col, axis=1)
    y_test = np.array(y_test)

    AvJweights[th] = {}

    for reg_strength in reg_strengths:
        AvJweights[th][reg_strength] = {}
        for learning_rate in learning_rates:
            print('Calculating for threshold = {} || rs: {},
lr: {}'.format(th, reg_strength, learning_rate))
            W = SVMClassifier(X_train, y_train, reg_strength,
learning_rate)
            AvJweights[th][reg_strength][learning_rate] = W

            train_preds = np.array([])
            for i in range(X_train.shape[0]):
                yp = np.sign(np.dot(W, X_train[i])) #model
                train_preds = np.append(train_preds, yp)

            train_acc = accuracy_score(y_train, train_preds)
            train_f1 = f1_score(y_train, train_preds)
            train_conf_mat = confusion_matrix(y_train,
train_preds)


            test_preds = np.array([])
            for i in range(X_test.shape[0]):
                yp = np.sign(np.dot(W, X_test[i])) #model
                test_preds = np.append(test_preds, yp)

            test_acc = accuracy_score(y_test, test_preds)
            test_f1 = f1_score(y_test, test_preds)
            test_conf_mat = confusion_matrix(y_test,
test_preds)
            if(test_acc > 0.70 and test_f1 > 0.70):
                print('Train Data')
                print('Accuracy = {}\nF1 Score = {}\nConfusion
matrix :\n{}'.format(train_acc*100, train_f1*100,
train_conf_mat))
                print('Test Data')
                print('Accuracy = {}\nF1 Score = {}\nConfusion
matrix :\n{}'.format(test_acc*100, test_f1*100,
test_conf_mat))
            else:
                print('Accuracy and F1 score below 70%')
        print('-----------------------------------------------
---')
```

```
# Pengujian model SVM
def predict_label(image_path):
    img = cv2.imread(image_path, 0)

    # threshold = 10 || rs: 100000.0, lr: 1e-08
    th = 10
    rs = 100000.0
    lr = 1e-08
    model_penyakit_vs_non = AJvAllweights[th][rs][lr]
    img_des = CalcFeatures(img, th)
    if img_des is not None:
        img_vec = np.resize(img_des, (1, 128*th))
        X = np.transpose(np.append(img_vec, np.ones((1,1)),
axis=1))
        yp = np.sign(np.dot(model_penyakit_vs_non, X))

        if yp == 1: # Alergi or Jamur
            th = 10
            model_alergi_vs_jamur =
AvJweights[th][20000.0][1e-08] # threshold = 10 || rs:
20000.0, lr: 1e-08
            img_des = CalcFeatures(img, th)
            if img_des is not None:
                img_vec = np.resize(img_des, (1, 128*th))
                X = np.transpose(np.append(img_vec,
np.ones((1,1)), axis=1))
                yp = np.sign(np.dot(model_alergi_vs_jamur, X))
                if yp == 1:
                    return 'alergi_kulit'
                else:
                    return 'jamur_kulit'

        else:
            return 'bukan_alergi_jamur'


train_preds = np.array([])
y_test = np.array([])
for image_path in image_path_test:
    yp = predict_label(image_path)
    if yp is not None:
        train_preds = np.append(train_preds, yp)
        y_test = np.append(y_test, image_path.split('/')[2])

acc = accuracy_score(y_test, train_preds)
conf_mat = confusion_matrix(y_test, train_preds)
print()
```

```
print('Accuracy : {}\nConfusion Matrix: \n{}\nResult
:\n{}'.format(acc*100, conf_mat, classification_report(y_test,
train_preds)))
```

Lampiran 3 Data citra mentah yang digunakan

Github: https://github.com/kvnchandra/skin-diseases-classification-raw-images.git


Code
main
Go to file
t
Erisipelas
Erisipelas dd Impetigo Bulosa.j...
WhatsApp Image 2020-02-22 at...
WhatsApp Image 2020-02-22 at...
Eritrasma
Eritroderma
Erupsi morbiliformis
Erythema Multiforme
Fixed Drug Eruption
Folikulitis
Herpes Thoracalis
Herpes Zooster
Kondiloma Akuminata
Limfoma Kutis
Miliaria
Morbus Hansen
NET
Neuralgia Post Herpetica
Documentation • Share feedback
skin-diseases-classification / Erisipelas / Erisipelas dd Impetigo Bulosa.jpg
kvnchandra Add files via upload
671c6d6 · 15 minutes ago
History
37.9 KB

Code

main

Go to file t

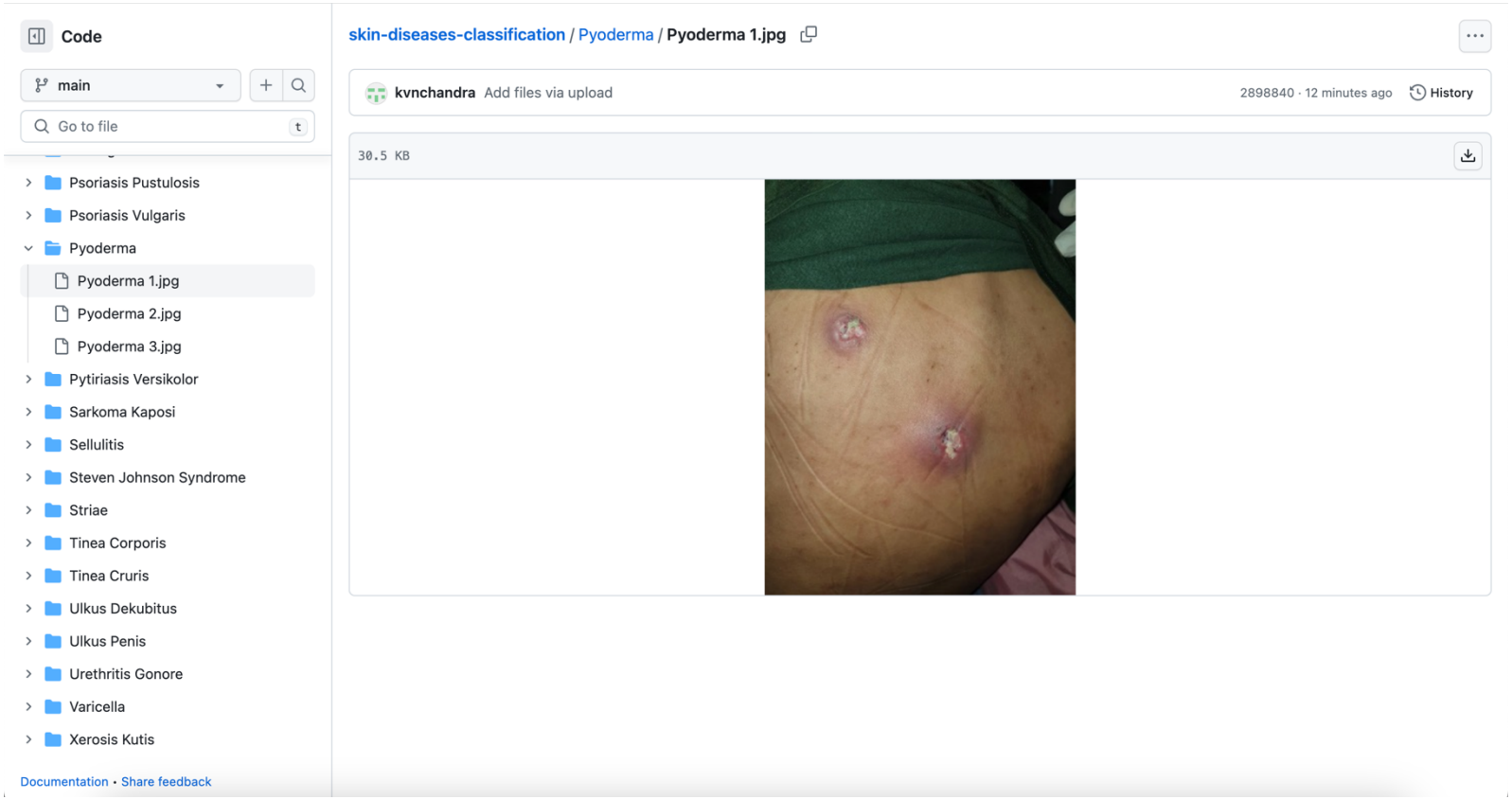
- Pemfigoid Bullosa
  - Andi Tenri Bullous Pemphigoid 2....
  - Andi Tenri Bullous Pemphigoid 3....
  - Andi Tenri Bullous Pemphigoid 4....
  - Andi Tenri Bullous Pemphigoid 5....
  - Andi Tenri Bullous Pemphigoid.jpg
  - Pemfigus bullosa 1.jpg
  - Pemfigus bullosa 2.jpg
  - Pemfigus bullosa 3.jpg
  - Pemfigus bullosa 5.jpg
  - Saleh Pemfigoid bulosa 2.jpg
  - Saleh Pemfigoid bulosa 3.jpg
  - Saleh Pemfigoid bulosa.jpg
  - WhatsApp Image 2020-07-18 at ...
  - WhatsApp Image 2020-07-18 at ...
  - WhatsApp Image 2020-07-18 at ...
  - WhatsApp Image 2020-07-18 at ...
  - WhatsApp Image 2020-07-18 at ...
  - WhatsApp Image 2020-07-18 at

Documentation • Share feedback

skin-diseases-classification / Pemfigoid Bullosa / Andi Tenri Bullous Pemphigoid 2.jpg

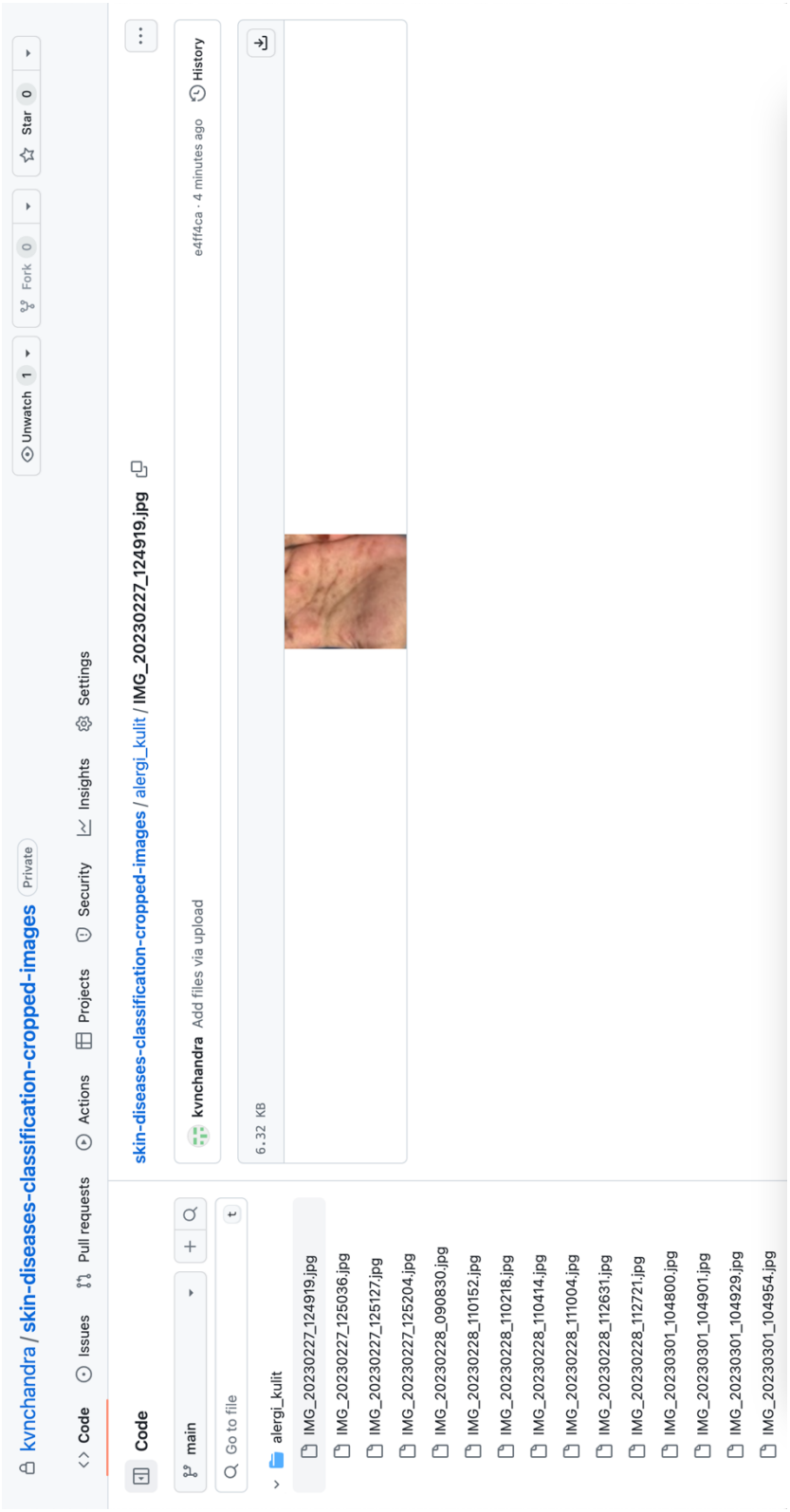
kvnchandra Add files via upload 2898840 · 11 minutes ago History

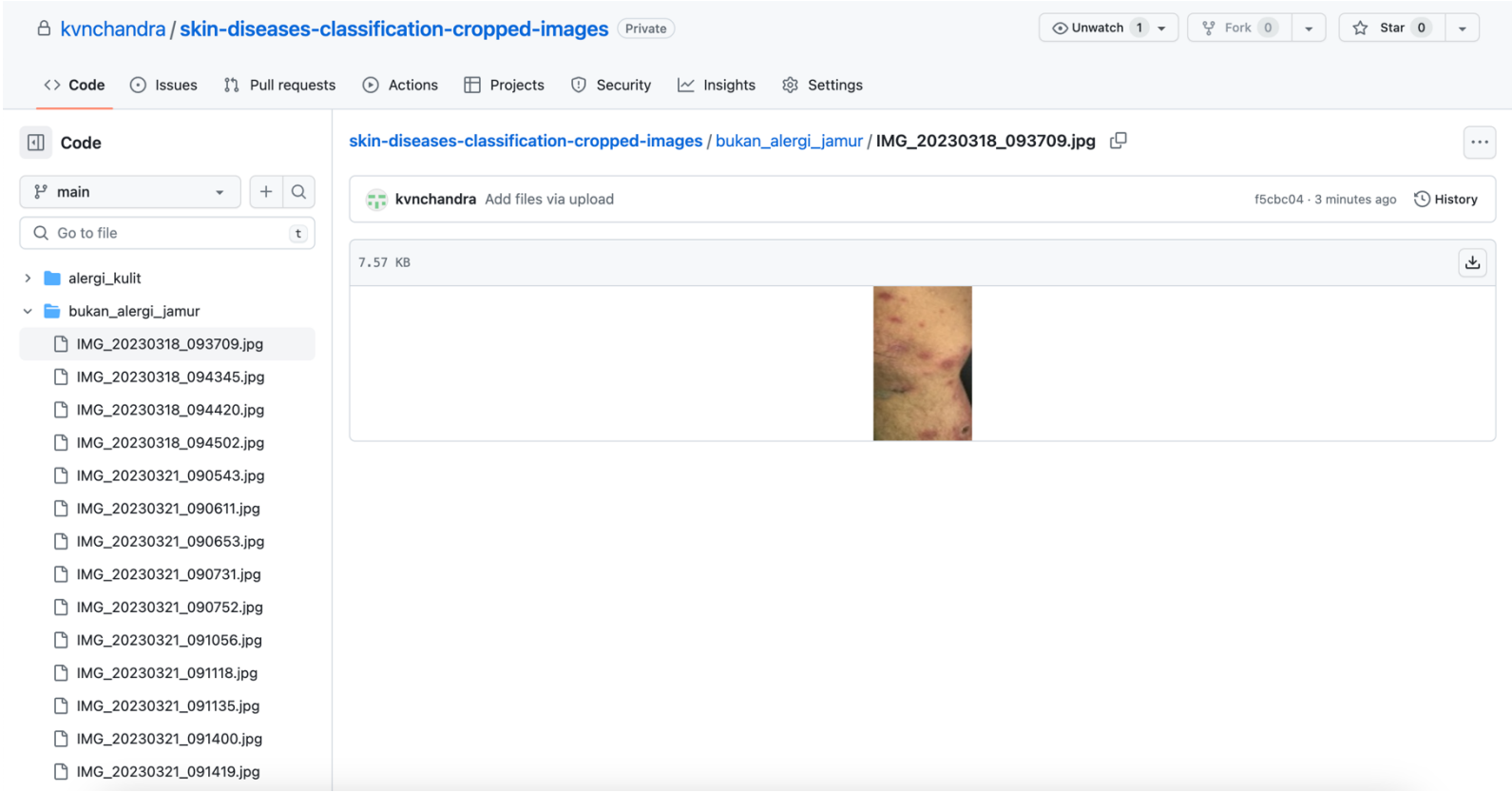
30 KB


Code
main
Go to file
t
Psoriasis Pustulosis
Psoriasis Vulgaris
Pyoderma
Pyoderma 1.jpg
Pyoderma 2.jpg
Pyoderma 3.jpg
Pytiriasis Versikolor
Sarkoma Kaposi
Sellulitis
Steven Johnson Syndrome
Striae
Tinea Corporis
Tinea Cruris
Ulkus Dekubitus
Ulkus Penis
Urethritis Gonore
Varicella
Xerosis Kutis
Documentation • Share feedback
skin-diseases-classification / Pyoderma / Pyoderma 1.jpg
kvnchandra Add files via upload
2898840 · 12 minutes ago
History
30.5 KB

Lampiran 4 Data citra yang telah dilakukan proses *cropping*

Github: https://github.com/kvnchandra/skin-diseases-classification-cropped-images.git




kvnchandra / skin-diseases-classification-cropped-images
Private
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
Code
main
Go to file
t
alergi_kulit
bukan_alergi_jamur
IMG_20230318_093709.jpg
IMG_20230318_094345.jpg
IMG_20230318_094420.jpg
IMG_20230318_094502.jpg
IMG_20230321_090543.jpg
IMG_20230321_090611.jpg
IMG_20230321_090653.jpg
IMG_20230321_090731.jpg
IMG_20230321_090752.jpg
IMG_20230321_091056.jpg
IMG_20230321_091118.jpg
IMG_20230321_091135.jpg
IMG_20230321_091400.jpg
IMG_20230321_091419.jpg
skin-diseases-classification-cropped-images / bukan_alergi_jamur / IMG_20230318_093709.jpg
kvnchandra Add files via upload
f5cbc04 · 3 minutes ago
History
7.57 KB

Code

main

Go to file t

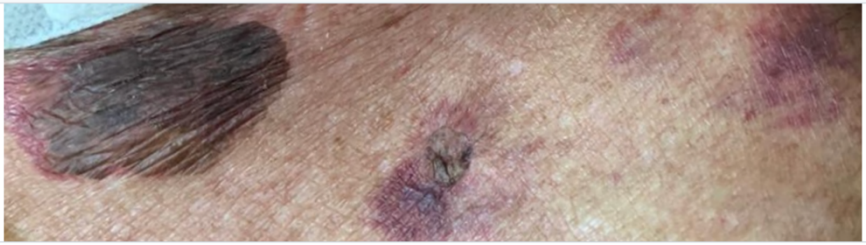
- alergi_kulit
- bukan_alergi_jamur
- jamur_kulit
  - IMG_20230306_115756.jpg
  - IMG_20230306_115828.jpg
  - IMG_20230306_115907.jpg
  - IMG_20230306_115935.jpg
  - IMG_20230306_115953.jpg
  - IMG_20230306_120033.jpg
  - IMG_20230306_120048.jpg
  - IMG_20230306_120117.jpg
  - IMG_20230306_120138.jpg
  - IMG_20230306_120201.jpg
  - IMG_20230306_120230.jpg
  - IMG_20230306_120304.jpg
  - IMG_20230306_120325.jpg
  - IMG_20230306_120359.jpg
  - IMG_20230306_120425.jpg
  - IMG_20230306_120456.jpg

Documentation • Share feedback

skin-diseases-classification-cropped-images / jamur_kulit / IMG_20230306_120117.jpg

kvnchandra Add files via upload b4a5328 · 4 minutes ago History

83 KB



LEMBAR PERBAIKAN SKRIPSI

"KLASIFIKASI PENYAKIT PADA CITRA KULIT MENGGUNAKAN METODE SUPPORT VECTOR MACHINE"

OLEH:

KEVIN CHANDRA
D42116311

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 24 Juli 2023.

Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Dr. Ir. Ingrid Nurtanio, M.T.	
Sekretaris	Anugrayani Bustamin, S.T., M.T.	
Anggota	Dr. Ir. Zahir Zainuddin, M.Sc.	
	Dr. Amil Ahmad Ilham, S.T., M.IT.	

Persetujuan Perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Dr. Ir. Ingrid Nurtanio, M.T.	
II	Anugrayani Bustamin, S.T., M.T.	