DAFTAR PUSTAKA

Agustiani, S., Arifin, Y. T., Juniadi, A., Wildah, S. K., Mustopa, A., & fjksdfjk, s. (2022). Klasifikasi Penyakit Daun Padi Menggunakan Random Forest dan Color Histogram. *Jurnal Komputasi Ilmu Komputer Unila*, Vol 10, No. 1, 65 - 74.

Alpaydin, E. (2020). *Introduction to Machine Learning, Fourth Edition.* Istanbul: MITPress. Dipetik 9 13, 2022, dari MIT Press: https://mitpress.mit.edu/9780262043793/introduction-to-machine-learning/

Amar, A., Nadia, L., & Sartika, D. (2020). *Definisi, Prinsip Dasar dan Perkembangan Bioteknologi Pangan.* Tangerang Selatan: Universitas Terbuka.

Breiman, L. (2001). Random Forest. *Machine Learning*, 45, 5-32. doi:https://doi.org/10.1023/A:1010933404324

Chandra, & Jacktish, L. (2022). Implementasi Deep Learning Menggunakan Convolutional Neural Network untuk Indentifikasi Jenis Bunga Berbasis Mobile Menggunakan Framework TensorFlow Lite. Dalam *S1 Thesis* (hal. 43). Yogyakarta: Universitas Atma Jaya Yogyakarta.

Darmawan, D. (2023). *Implementasi Metode Convolutional Neural Network(CNN) Dalam Mendeteksi Jenis Sampah .* Jambi: Universitas Jambi.

Devetyarov, D., & Nouretdinov, I. (2010). *Prediction with Confidence Based on a Random Forest Classifier.* Berlin, Heidelberg: Springer. doi:https://doi.org/10.1007/978-3-642-16239-8_8

El Naqa, I., & Murphy, M. J. (2015). What is Machine Learning? In Machine Learning in Radiation Oncology. *Springer International*, 3-11.

Estikomah, S. A. (2020). Pemanfaatan Rhizopus Oryzae Dalam Pengembangan Produk Olahan Susu (Keju) Halal Berbasis Bioteknologi. *Pharmaceutical Jurnal of Islamic Pharmacy*.

Fachruddin, M. I. (2015). *Perbandingan Metode Random Forest Classification Dan Support Vector Machine Untuk Deteksi Epilepsi Menggunakan Data Rekaman*

Electroen Cephalograph (EGG). Doctoral Dissertation, Institut Teknologi Sepuluh November.

Fikrie, M. R. (2019). Pengaruh Kerapatan Jarak Tanam Terhadap Intensitas Serangan Hama dan Penyakit Pada Produksi Benih Jagung (Zea mays L.) Hibrida Pioneer. Dalam *Pengaruh Kerapatan Jarak Tanam Terhadap Intensitas Serangan Hama dan Penyakit Pada Produksi Benih Jagung (Zea mays L.) Hibrida Pioneer* (hal. 1-32). Jember: Politeknik Negeri Jember.

Firmansyah, R. (2021). Implementasi Deep Learning Menggunakann Convolutional Neural Network Untuk Klasifikasi Bunga. Dalam *Skripsi* (hal. 129). Jakarta: Universitas Islam Negeri Syarif Hidayatullah.

Fonda, H., Irawan, Y., & Febriani, A. (2020). Klasifikasi Batik Riau Dengan Menggunakan Convolutional Neural Network (CNN). *Jurnal Ilmu Komputer (JIK)*, Vol. 9, No. 1.

Gonzales, C. R., & Woods, R. E. (2002). *Digital Image Processing.* New Jersey, USA: Prentice-Hall Inc.

Goronescu, F. (2011). *Data Mining : Concept, Models, and Techniques.* Verlag Berlin Heidelberg: Springer.

Hijazi, S., Kumar, R., & Rowen, C. (2015). *Using Convolutional Neural Network for Image Recognition.* San Jose, California, USA: Cadance Design System inc. Dipetik 9 22, 2022, dari https://ip.cadence.com/uploads/901/TIP_WP_cnn_FINAL-pdf

Hortikultura, D. J. (2020). *Produktivitas Cabai Rawit Menurut Provinsi, Tahun 2015-2019.*

Hsu, C., Chang, C., & Lin, C. (2003). A Practical Guide to Support Vector Classification. 16. Dipetik 9 9, 2022, dari http://www.datascienceassn.org/sites/default/files/Practical%20Guide%20to%20Support%20Vector%20Classification.pdf

Ikayanti, F. (2018, November 30). *Mengenal Jagung di Indonesia*. Dipetik 9 13, 2022, dari Dinas Pangan, Pertanian, dan Perikanan Kota Potianak: https://dppp.pontianak.go.id/artikel/47-mengenal-jagung-di-indonesia.html

Ilahiyah, S., & Nilogiri, A. (2018). Implementasi Deep Learning Pada Identifikasi Jenis Tumbuhan Berdasarkan Citra Daun Menggunakan Convolutional Neural Network. *Jurnal Sistem & Teknologi Informasi Indonesia (JUSTINDO)*, 49-56.

Khair, H., Pasaribu, M., & Suprapto, E. (2013, April). Respon Pertumbuhan dan Produksi Tanaman Jagung (Zea mays L.) Terhadap Pemberian Pupuk Kandang Ayam dan Pupuk Organik Cair Plus. *Agrium: Jurnal Ilmu Pertanian, Volume 18 No 1*, 13-22. doi:https://doi.org/10.30596/agrium.v18i1.339

Khotimah, B. K., Setiawan, E., Sasmeka, V., Fridayanti, A., Maulana, I., & Zulfida, A. M. (2022). Identifikasi Hama dan Penyakit Tanaman Jagung Dengan Menggunakan Metode Klasifikasi Support Vector Machine (SVM). *Jurnal Ilmiah NERO*, Vol. 7, No. 1.

Kotsiantis, S., Zaharakis, I., & Pintelas, P. (2007). Supervised Machine Learning : A Review of Classification Techniques. *Emerging Artificial Intelligence Applications in Computer Engineering*, 160, 3-24.

Kristina, S., Sianturi, R. D., & Husnadi, R. (2020). Penerapan Model Capacitated Vehicle Routing Problem (CVRP) Menggunakan Google OR-Tools untuk Penentuan Rute Pengantaran Obat pada Perusahaan Pedagang Besar Farmasi (PBF). *Jurnal Telematika*, 104-105.

Kurniawati, H., Yulianingsih, R., & Wahda, L. (2021, April). Upaya Perbaikan Pertumbuhan dan Hasil Tanaman Jagung Manis Dengan Pemberian POC Azolla Microphylla. *PIPER, Vol 17 No 1*, hlmn 1-7. Diambil kembali dari https://jurnal.unka.ac.id/index.php/piper/article/download/507/534

Mangkunegara, L. S., & Purwono. (2022). *Belajar Data Science Dengan Kaggle, Dilengkapi Dengan Praktikum Latihan Data Science Untuk Pemula.* Purwokerto: UHB Press.

Maulana, F., & Rochmawati, N. (2019). Klasifikasi Citra Buah Menggunakan Convolutional Neural Network. *Journal of Informatics and Computer Scince*, Vol. 01, No. 02.

Najira, N., Riandi, R., & Surtikanti, H. K. (2024). Kajian Literatur: Penggunaan Teknologi Pembelajaran Untuk Pengajaran Bioteknologi. *Jurnal Jeumpa*.

Naufal, M. F. (2021). ANALISIS PERBANDINGAN ALGORITMA SVM, KNN, DAN CNN UNTUK KLASIFIKASI CITRA CUACA. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, 311-318.

Nuel Jaya, M. C. (2022). Perancangan Aplikasi Pendeteksi Penyakit Daun Padi Menggunakan Arsitektur MobileNetV2. Dalam *Skripsi* (hal. 1-81). Makassar: Universitas Hasanuddin.

Nugroho, P. A., Fenriana, I., & Arijanto, R. (2020). Implementasi Deep Learning Menggunakan Convolutional Neural Network (CNN) Pada Ekspresi Manusia. *Jurnal Algor*, Vol. 2 No. 1.

Nurcahyati, A. D., Akbar, R. M., & Zahara, S. (2022). Klasifikasi Citra Penyakit Pada Daun Jagung Menggunakan Deep Learning Dengan Metode Convolutional Neural Network (CNN). *Jurnal Ilmiah Teknologi Informasi dan Sains*, 43-51.

Pakki, S. (2005). Epidemiologi dan Pengendalian Penyakit Bercak Daun (Helminthosporium sp.) Pada Tanaman Jagung. *Jurnal Penelitian dan Pengembangan Pertanian*, 24(3), 101-108.

Pane, S. F., & Saputra, Y. A. (2020). *Big Data Classification Behavior Menggunakan Python (1st ed).* Bandung: Kreatif Industri Nusantara.

Paul, A., Mukherjee, D., Das, P., Gangopadhyay, A., Chintha, A., & Kundu, S. (2018). Improved Random Forest for Classification. *IEEE Transactions on Image Processing*, 27(8), 4012–4024. doi:doi:10.1109/tip.2018.2834830

Prasetio, R. T., & Ripandi, E. (2019). Optimasi Klasifikasi Jenis Hutan Menggunakan Deep Learning Berbasis Optimize Selection. *Jurnal Informatika*, Vol. 6, No. 1, pp. 100-106.

Prasetyo, G., Ratih, S., Ivayani, & Akin, H. M. (2017). Efektivitas Pseudomonas Fluorescens dan Paenibacillus Polymyxa Terhadap Keparahan Penyakit Karat dan Hawar Daun Serta Pertumbuhan Tanaman Jagung Manis (Zea mays var. saccharata). *Jurnal Agrotek Tropika*, Vol. 5, No. 2: 102 - 108.

Prof. Drs. Sutarno, M. P. (2016). REKAYASA GENETIK DAN PERKEMBANGAN BIOTEKNOLOGI DI BIDANG PETERNAKAN. *Proceding Biology Education Conference* (hal. 23-27). Universitas Sebelas Maret.

Pujoseno, J. (2018). Implementasi Deep Learning Menggunakann Convolutional Neural Network Untuk Klasifikasi Alat Tulis. Dalam *Skripsi* (hal. 88). Yogyakarta: Universitas Islam Indonesia.

Putra, I. P., Rusbandi, & Alamsyah, D. (2022). Klasifikasi Penyakit Daun Jagung Menggunakan Metode Convolutional Neural Network. *Jurnal Algoritme*, 102-112.

Rachmawanto, E. H., & Hadi, H. P. (2021). Optimasi Ekstraksi Fitur Pada KKN Dalam Klasifikasi Penyakit Daun Jagung. *DINAMIK*, Vol. 22, No. 2. doi:https://doi.org/10.35315/dinamik.v26i2.8673

Reskianty. (2021). Implementasi Metode Random Forest dan Support Vector Machine Untuk *Dataset* Tidak Seimbang . Dalam *Skripsi* (hal. 40). Makassar: Universitas Hasanuddin.

Santosa, B., & Umam, A. (2018). *Data Mining dan Big Data Analytics Edisi 2.* Yogyakarta: Penebar Media Pustaka.

Sari, I. P., Hidayat, B., & Atmaja, R. D. (2016). Perancangan dan Simulasi Deteksi Penyakit Tanaman Jagung Berbasis Pengolahan Citra Digital Menggunakan Metode Color Moments dan GLCM. *SEMINAR NASIONAL INOVASI DAN APLIKASI TEKNOLOGI DI INDUSTRI (SENIATI)*, 6. doi:https://doi.org/10.36040/seniati.vi0.811

Semangun, H. (1993). Penyakit-Penyakit Tanaman Pangan di Indonesia. *Gajah Mada University Press. 499*, 4.

Setiawan, B. (2021). PENERAPAN ALGORITMA YOU ONLY LOOK ONCE (YOLO) UNTUK DETEKSI TANAMAN MIANA BERBASIS ANDROID. Dalam *Skripsi* (hal. 1-45). Ponorogo: Universitas Muhammadiyah Ponorogo.

Setyawan, M. A. (2018). Implementasi Deep Learning Menggunakan Convolutional Neural Network Untuk Klasifikasi Alat Tulis. Dalam *Skripsi* (hal. 88). Kediri: Universitas Islam Indonesia.

Setyawan, M. A. (2022). Klasifikasi Penyakit Daun Jagung Berdasarkan Ruang Warna HSV dan Fitur Tekstur Dengan Algoritma K-NN. Dalam *Skripsi* (hal. 28). Kediri: Universitas Nusantara PGRI .

Sudjono, M. S. (1988). Penyakit Jagung dan Pengendaliannya. *Balai Penelitian Tanaman Pangan Bogor*, 381-394.

Syarifudin, A., Hidayat, N., & Fanani, L. (2017, September 17). Sistem Pakar Diagnosis Penyakit Pada Tanaman Jagung Menggunakan Metode Naive Bayes Berbasis

Android. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, vol 2 No(7)*, 2738-2744. Diambil kembali dari https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/1685

Taha, F. P. (2021). Analisis Kinerja Long-Short Term Memory, Peephole Connection LSTM dan Facebook's Prophet dalam Memprediksi Pergerakan Harga Saham PT Telekomunikasi Indonesia Tbk. Dalam *Skripsi* (hal. 57). Makassar: Universitas Hasanuddin.

Talanca, A. H., & Tenrirawe, A. (2015). Respon Beberapa Varietas Terhadap Penyakit Utama Jagung Di Kabupaten Kediri Jawa Timur . *Jurnal Agrotan* , Vol. 1, No. 1: 67 - 78.

Wahyuni, S., Noviani, N., Sartika, A., Habibie, D., & Handayani, L. (2024). Peran Bioteknologi Dalam Ketertarikan Berbagai Aspek Bidang Ilmu Pengetahuan Serta Mewujudkan Inovasi Dan Kreativitas di Kalangan Mahasiswa Universitas Harapan Medan. *Jurnal Pengabdian West Science*.

Wakman, W., & Burhanuddin. (2007). Pengelolaan Penyakit Prapanen Jagung. Dalam *Buku Jagung.* Maros: Pusat Penelitian dan Pengembangan Tanaman Pangan.

Winandar, A. A. (2021). Perbandingan Kinerja Klasifikasi CNN Berdasarkan Strategi Split Data Pada Beragam *Dataset* Citra. Dalam *Skripsi* (hal. 120). Makassar: Universitas Hasanuddin.

Winarto, E. G. (2021). Sistem Smart Greenhouse Pada Budidaya Tanaman Cabai . Dalam *Skripsi* (hal. 105). Makassar: Universitas Hasanuddin.

Xiaofeng, H., & Yan, L. (2015). The Application of Convolution Neural Networks in Handwritten Numeral Recognition. *International Journal of Database Theory and Application*, Vol.9, No.3.

Yana, Y. E., & Nafi`iyah, N. (2021). Klasifikasi Jenis Pisang Berdasarkan Fitur Warna, Tekstur, Bentuk Citra Menggunakan SVM dan KNN. *Journal of Computer, Information System & Technology Management*, Vol. 4, No. 1, Pages 28-36.

Zidane, Z. K. (2019). Mengkombinasikan Teknik Resampling Dengan Algoritma Machine Learning Untuk Mengatasi *Dataset* Tak Seimbang. Dalam *Skripsi* (hal. 33). Makassar: Universitas Hasanuddin.

LAMPIRAN

CNN.ipynb

```
!pip install tensorflow-addons

# Import package

import os

import zipfile

from google.colab import drive

drive.mount('/content/drive')


# Extract the dataset from zip to '/content/dataset'

zip_path = '/content/drive/MyDrive/Feb24/CornDisease.zip'

extract_path = '/content/dataset'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:

    zip_ref.extractall(extract_path)


# Set base path for dataset

base_path = '/content/dataset/CornDisease'


import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from keras.preprocessing.image import load_img, img_to_array


# Visualisasi citra asli

directory = "/content/dataset/CornDisease/Blight/Blight(78).JPG"

img = mpimg.imread(directory)

imgplot = plt.imshow(img)
```

```
plt.show()
#Visualisasi citra yang diresize
img_resize = load_img(directory, grayscale=False, color_mode='rgb',
target_size=(224,224))
plt.figure(figsize=(3,3))
imgplot = plt.imshow(img_resize)

# Mengubah gambar menjadi array
img_arr = img_to_array(img_resize)
# Menampilkan array
img_arr

# Melakukan normalisasi
img_norm = img_arr/255.0
# Menampilkan array hasil normalisasi
img_norm

import pathlib
import numpy as np
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow_hub as hub
import tensorflow as tf
import keras
import tensorflow_addons as tfa
from PIL import Image
import matplotlib.pyplot as plt
```

```
import matplotlib.image as mpimg
import seaborn as sns
from scipy import interp
from itertools import cycle
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score

# Direktori dataset
dir = '/content/dataset/CornDisease'
# Nama-nama kelas pada dataset
File=[]
for file in os.listdir(dir):
 File+=[file]
print(File)

# Visualisasi citra asli
directory = "/content/dataset/CornDisease/Blight/Blight(70).JPG"
img = mpimg.imread(directory)
imgplot = plt.imshow(img)
plt.show()

#Visualisasi citra yang diresize
img_resize = load_img(directory, grayscale=False,
color_mode='rgb',target_size=(224,224))
plt.figure(figsize=(3,3))
imgplot = plt.imshow(img_resize)
# Mengubah gambar menjadi array
```

```
img_arr = img_to_array(img_resize)
# Menampilkan array
img_arr
# Melakukan normalisasi
img_norm = img_arr/255.0
# Menampilkan array hasil normalisasi
img_norm

# Direktori masing-masing kelas
# Directory with our Bacterial Leaf Blight pictures
train_Blight_dir = os.path.join('/content/dataset/CornDisease/Blight')

# Directory with our Brown Spot pictures
train_Rust_dir = os.path.join('/content/dataset/CornDisease/Common_Rust')

# Directory with our Leaf Smut pictures
train_Leaf_dir = os.path.join('/content/dataset/CornDisease/Gray_Leaf_Spot')

# Mengambil nama file masing-masing kelas
train_Blight_names = os.listdir(train_Blight_dir)
print("File names in Bacterial Leaf Blight directory:")
print(train_Blight_names[:10])
train_Rust_names = os.listdir(train_Rust_dir)
print("\nFile names in Brown Spot directory:")
print(train_Rust_names[:10])
train_Leaf_names = os.listdir(train_Leaf_dir)
print("\nFile names in Leaf Smut directory:")
print(train_Leaf_names[:10])
```

```
# Menampilkan jumlah citra pada masing-masing kelas
print('total Blight images:', len(os.listdir(train_Blight_dir)))
print('total Rust images:', len(os.listdir(train_Rust_dir)))
print('total Leaf Spot images:', len(os.listdir(train_Leaf_dir)))
%matplotlib inline

# Parameters for our graph; we'll output images in a 3x3 configuration
nrows = 3
ncols = 3
# Index for iterating over images
pic_index = 0
# Menampilkan 9 sampel citra kelas Blight
# Set up matplotlib fig, and size it to fit 3x3 pics
fig = plt.gcf()
fig.set_size_inches(16, 8)
pic_index += 9
next_Blight_pix = [os.path.join(train_Blight_dir, fname) for fname in
train_Blight_names[pic_index-9:pic_index]]
for i, img_path in enumerate(next_Blight_pix):
 # Set up subplot; subplot indices start at 1
 sp = plt.subplot(nrows, ncols, i + 1)
 sp.axis('Off') # Don't show axes (or gridlines)
 img = mpimg.imread(img_path)
 plt.imshow(img)
 if (i + 1 == 2):
  plt.title("Blight disease", pad = 40, fontsize =20)
plt.show()
pic_index = 0
```

```
# Menampilkan 9 sampel citra kelas Common Rust

# Set up matplotlib fig, and size it to fit 3x3 pics

fig = plt.gcf()

fig.set_size_inches(16, 8)

pic_index += 9

next_Rust_pix = [os.path.join(train_Rust_dir, fname) for fname in
train_Rust_names[pic_index-9:pic_index]]

for i, img_path in enumerate(next_Rust_pix):


 # Set up subplot; subplot indices start at 1

 sp = plt.subplot(nrows, ncols, i+1)

 sp.axis('Off') # Don't show axes (or gridlines)

 img = mpimg.imread(img_path)

 plt.imshow(img)

 if (i + 1 == 2):

  plt.title("Common Rust disease", pad = 40, fontsize = 20)

plt.show()

pic_index = 0


# Menampilkan 9 sampel citra kelas Gray Leaf Spot

# Set up matplotlib fig, and size it to fit 3x3 pics

fig = plt.gcf()

fig.set_size_inches(16, 8)

pic_index += 9

next_Leaf_pix = [os.path.join(train_Leaf_dir, fname) for fname in
train_Leaf_names[pic_index-9:pic_index]]

for i, img_path in enumerate(next_Leaf_pix):
```

```
# Set up subplot; subplot indices start at 1
 sp = plt.subplot(nrows, ncols, i+1)
 sp.axis('Off') # Don't show axes (or gridlines)
 img = mpimg.imread(img_path)
 plt.imshow(img)
 if (i + 1 == 2):
  plt.title("Leaf Smut disease", pad = 40, fontsize = 20)
plt.show()

# Preprocessing data citra
dataset = []
mapping = {'Bacterial_Leaf_Blight':0, 'Brown_Spot':1, 'Leaf_Smut':2}
count = 0
for file in os.listdir(dir):
 path = os.path.join(dir,file)
 for im in os.listdir(path):
  image = load_img(os.path.join(path,im), grayscale=False,color_mode='rgb',
target_size=(224,224))
  image = img_to_array(image)
  image = image/255.0
  dataset.append([image,count])
 count=count+1
# Mengubah data menjadi array
data,labels0=zip(*dataset)
labels1=to_categorical(labels0)
data=np.array(data)
labels=np.array(labels1)
print(data.shape)
print(labels.shape)
```

```
# Pembagian data training dan testing
train_x,test_x,train_y,test_y=train_test_split(data,labels,test_size=0.2,random_state=13
)
# Menampilkan bentuk dimensi data training dan testing
print(train_x.shape)
print(test_x.shape)
print(train_y.shape)
print(test_y.shape)

# Augmentasi data
datagen = ImageDataGenerator(
 horizontal_flip=True,
 vertical_flip=True,
 rotation_range=20,
 zoom_range=0.2,
 width_shift_range=0.2,
 height_shift_range=0.2,
 shear_range=0.1,
 fill_mode="nearest")

# Mengambil model MobileNetV3 dari TensorFlow Hub
module_selection = ("mobilenet_v3", 224, 1280) #@param["(\"mobilenet_v3\", 224,
1280)", "(\"inception_v3\", 299, 2048)"]{type:"raw", allow-input: true}
handle_base, pixels, FV_SIZE = module_selection
# MODULE_HANDLE ="https://tfhub.dev/google/tf2-
preview/{}/feature_vector/4".format(handle_base)

# IMAGE_SIZE = (pixels, pixels)
MODULE_HANDLE =
"https://tfhub.dev/google/imagenet/mobilenet_v3_large_100_224/feature_vector/5"
```

```
IMAGE_SIZE = (224, 224)
print("Using {} with input size {} and output dimension{}".format(MODULE_HANDLE,
IMAGE_SIZE, FV_SIZE))
do_fine_tuning = False #@param {type:"boolean"}

# Membuat lapisan untuk MobileNetV3
feature_extractor = hub.KerasLayer(MODULE_HANDLE,
 input_shape=IMAGE_SIZE + (3,),
 output_shape=[FV_SIZE],
 trainable=do_fine_tuning,
 name="MobileNetV3",)

# Membangun model akhir
print("Building model with", MODULE_HANDLE)
model = tf.keras.Sequential([feature_extractor,
 tf.keras.layers.Dense(256, activation='relu', name='Fully_Connected_Layer'),
 tf.keras.layers.Dropout(0.5, name='Dropout_Layer'),
 tf.keras.layers.Dense(3, activation='softmax', name='Output_Layer')])
model.summary()

# Membekukan lapisan agar bobot tidak berubah
NUM_LAYERS = 10 #@param {type:"slider", min:1, max:50, step:1}
if do_fine_tuning:
 feature_extractor.trainable = True
 for layer in model.layers[-NUM_LAYERS:]:
  layer.trainable = True
else:
 feature_extractor.trainable = False
```

```
# Compile model
METRICS = [
 'accuracy',
 keras.metrics.Precision(name='precision'),
 keras.metrics.Recall(name='recall'),
 tfa.metrics.F1Score(num_classes=3)]
if do_fine_tuning:
 model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.002,
  momentum=0.9),
loss=tf.keras.losses.SparseCategoricalCrossentropy(),
 metrics=METRICS)
else:
 model.compile(optimizer='adam',
  loss='categorical_crossentropy',
  metrics=METRICS)

# plot the model including the sizes of the model
tf.keras.utils.plot_model(model, show_shapes=True)
# Train the model.
history = model.fit(
 datagen.flow(train_x, train_y, batch_size=8),
 validation_data = (test_x, test_y),
 epochs = 50)
# Menampilkan nilai presisi, recall, dan F1-score
pred = model.predict(test_x, verbose=2)
model_predicted = np.argmax(pred, axis = 1)
print(classification_report(test_y.argmax(axis=1), model_predicted, target_names=File))
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
```

```
# Menampilkan kurva akurasi
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(len(acc))

plt.plot(epochs, acc, color=colors[0], label='Train')
plt.plot(epochs, val_acc, color=colors[0], linestyle="--", label='Val')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.show()

# Menampilkan kurva F1-score untuk masing-masing kelas
f1_score = history.history['f1_score']
val_f1_score = history.history['val_f1_score']
f1_numpy = np.array(f1_score)
val_f1_numpy = np.array(val_f1_score)
epochs = range(len(f1_score))

n_classes = 3
colors = cycle(['orange', 'green', 'blue'])
for i, color in zip(range(n_classes), colors):
 plt.plot(epochs, f1_numpy[:, i], color=color,label='Train F1-score of class {0}'.format(i))
for i, color in zip(range(n_classes), colors):
 plt.plot(epochs, val_f1_numpy[:, i], color=color, linestyle="--", label='Val F1-score of
class {0}'.format(i))
plt.xlabel('Epoch')
```

```
plt.ylabel('F1-score')
plt.title('Training and validation f1-score')
plt.legend(loc=0)
plt.figure(figsize=(12,12))
plt.show()

# Menampilkan Confusion Matrix
disease_types = ["Blight", "Common Rust", "Gray Leaf Spot"]
Y_pred = model.predict(test_x)
Y_pred = np.argmax(Y_pred, axis=1)
Y_true = np.argmax(test_y, axis=1)
fig, ax = plt.subplots(figsize=(12,12))
cm = confusion_matrix(Y_true, Y_pred)

#plt.figure(figsize=(12, 12))
ax = sns.heatmap(cm, cmap=plt.cm.Blues, annot=True, square=True,
xticklabels=disease_types, yticklabels=disease_types, ax=ax)
ax.set_ylabel('Actual', fontsize=15)
ax.set_xlabel('Predicted', fontsize=15)
ax.set_title('Confusion Matrix', fontsize=20, pad=40)
```

RandomForest.ipynb

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from PIL import Image
import zipfile
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Extract the dataset from zip to '/content/dataset'
zip_path = '/content/drive/MyDrive/Feb24/CornDisease.zip'
extract_path = '/content/dataset'
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Set base path for dataset
base_path = '/content/dataset/CornDisease'

# Load dataset with color histogram feature
def load_dataset_with_histogram(base_path):
    X = []
    y = []
    # labels = os.listdir(base_path)
```

```
    labels = ['Blight', 'Common_Rust', 'Gray_Leaf_Spot']  # Urutan label yang diinginkan
    label_to_id = {label: idx for idx, label in enumerate(labels)}  # Mapping from label name to label id
    for label in labels:
        label_path = os.path.join(base_path, label)
        for image_file in os.listdir(label_path):
            image_path = os.path.join(label_path, image_file)
            image = Image.open(image_path)
            image = image.resize((224, 224))  # Resize images if necessary

# Calculate color histogram
            hist = np.array(image.histogram()).reshape(3, 256)  # Red, Green, Blue channels
            hist = hist / hist.sum()  # Normalize histogram
            hist = hist.flatten()
            X.append(np.hstack([np.array(image).flatten(), hist]))  # Combine pixel values and histogram
            y.append(label_to_id[label])  # Use label id instead of label name
    return np.array(X), np.array(y), label_to_id
X, y, label_to_id = load_dataset_with_histogram(base_path)

# Split dataset into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize lists to store accuracy values
train_accuracy = []
val_accuracy = []
```

```
# Train Random Forest Classifier with varying number of trees
num_trees_range = range(1, 101)  # Vary the number of trees from 1 to 100
for num_trees in num_trees_range:
    clf = RandomForestClassifier(n_estimators=num_trees, random_state=42)
    clf.fit(X_train, y_train)

    # Predictions
    train_predictions = clf.predict(X_train)
    val_predictions = clf.predict(X_val)

    # Accuracy
    train_accuracy.append(accuracy_score(y_train, train_predictions))
    val_accuracy.append(accuracy_score(y_val, val_predictions))

    print(f"Number of trees: {num_trees}, Training Accuracy Random Forest:
{train_accuracy[-1]*100:.4f}%, Validation Accuracy Random Forest: {val_accuracy[-
1]*100:.4f}%")

# Train Random Forest Classifier with final number of trees
final_num_trees = num_trees_range[np.argmax(val_accuracy)]
final_clf = RandomForestClassifier(n_estimators=final_num_trees, random_state=42)
final_clf.fit(X_train, y_train)

# Evaluate on training set
train_predictions = final_clf.predict(X_train)
train_accuracy_final = accuracy_score(y_train, train_predictions)
# Evaluate on validation set
val_predictions = final_clf.predict(X_val)
val_accuracy_final = accuracy_score(y_val, val_predictions)
```

```
# Convert accuracies to percentage and print
train_accuracy_percentage = train_accuracy_final * 100
val_accuracy_percentage = val_accuracy_final * 100

print(f"Final Training Accuracy Random Forest: {train_accuracy_percentage:.4f}%")
print(f"Final Validation Accuracy Random Forest: {val_accuracy_percentage:.4f}%")
```

Support Vector Machine

```
!pip install Augmentor
from google.colab import drive
drive.mount('/content/drive')

import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from skimage.feature import local_binary_pattern
import zipfile
import Augmentor
from sklearn.utils import shuffle

# Extract the dataset from zip to '/content/dataset'
zip_path = '/content/drive/MyDrive/Skripsi/CornDisease.zip'
extract_path = '/content/dataset'
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Set base path for dataset
base_path = '/content/dataset/CornDisease'
```

```
# Function to extract LBP features
def extract_lbp_features(image, num_points=24, radius=8):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = local_binary_pattern(gray, num_points, radius, method='uniform')
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, num_points + 3), range=(0, num_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist

# Function to perform data augmentation
def augment_data(image_path, output_path, num_images):
    p = Augmentor.Pipeline(image_path, output_directory=output_path)
    p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
    p.flip_left_right(probability=0.5)
    p.flip_top_bottom(probability=0.5)
    p.zoom_random(probability=0.5, percentage_area=0.8)
    p.sample(num_images)

# Define paths for augmented data
augmented_path = '/content/augmented_data'
os.makedirs(augmented_path, exist_ok=True)
# Perform data augmentation
num_augmented_images = 3000  # Increased number of augmented images
for class_folder in os.listdir(base_path):
    class_path = os.path.join(base_path, class_folder)
    if os.path.isdir(class_path):
        augment_data(class_path, os.path.join(augmented_path, class_folder), num_augmented_images)
```

```
# Load images and labels
data = []
labels = []

for folder in os.listdir(base_path):
    folder_path = os.path.join(base_path, folder)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, file_name)
            image = cv2.imread(image_path)
            lbp_features = extract_lbp_features(image)
            data.append(lbp_features)
            labels.append(folder)

# Load augmented data
augmented_data = []
augmented_labels = []

for folder in os.listdir(augmented_path):
    folder_path = os.path.join(augmented_path, folder)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, file_name)
            image = cv2.imread(image_path)
            lbp_features = extract_lbp_features(image)
            augmented_data.append(lbp_features)
            augmented_labels.append(folder)
```

```
# Convert data and labels to numpy arrays
data = np.array(data)
labels = np.array(labels)
augmented_data = np.array(augmented_data)
augmented_labels = np.array(augmented_labels)

# Combine augmented data with original data
X_combined = np.concatenate((data, augmented_data), axis=0)
y_combined = np.concatenate((labels, augmented_labels), axis=0)
# Split combined dataset into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_combined, y_combined,
test_size=0.2, random_state=42)

# Shuffle data
X_train, y_train = shuffle(X_train, y_train, random_state=42)
X_val, y_val = shuffle(X_val, y_val, random_state=42)

# Normalize combined data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Perform PCA for feature selection
pca = PCA(n_components=0.95, whiten=True)
X_train_pca = pca.fit_transform(X_train_scaled)
X_val_pca = pca.transform(X_val_scaled)

# Define parameter grid for grid search
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1], 'kernel': ['linear', 'rbf']}
```

```
# Initialize SVM classifier
svm = SVC()

# Perform grid search
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train_pca, y_train)
# Get best parameters
best_params = grid_search.best_params_

# Train SVM classifier with best parameters
svm_best = SVC(**best_params)
svm_best.fit(X_train_pca, y_train)

# Predict on training set
y_train_pred = svm_best.predict(X_train_pca)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Predict on validation set
y_val_pred = svm_best.predict(X_val_pca)
print("Validation Classification Report:")
print(classification_report(y_val, y_val_pred))
# Check accuracy
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_val = accuracy_score(y_val, y_val_pred)
print("Training Accuracy:", accuracy_train)
print("Validation Accuracy:", accuracy_val)
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve


# Function to plot learning curve
def plot_learning_curve(estimator, title, X_train, y_train, X_val, y_val, ylim=None,
cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Accuracy")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X_train, y_train, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    plt.grid()
    # plt.plot(train_sizes, train_scores_mean, 'o-', color="blue",
    #          label=f"Training accuracy : {train_scores_mean[-1]:.2f}")
    # plt.plot(train_sizes, test_scores_mean, 'o-', color="orange",
    #          label=f"Validation accuracy : {test_scores_mean[-1]:.2f}")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="blue",
             label=f"Training accuracy")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="orange",
             label=f"Validation accuracy")
    plt.legend(loc="best")
    return plt
```

```
# Plot learning curve

title = "Training and Validation Accuracy SVM"

plot_learning_curve(svm_best, title, X_train_pca, y_train, X_val_pca, y_val, ylim=(0.7,
1.01), cv=5, n_jobs=-1)

plt.show()


from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Confusion matrix for training set

train_cm = confusion_matrix(y_train, y_train_pred, normalize='true')

plt.figure(figsize=(10, 8))

sns.heatmap(train_cm, annot=True, fmt=".2%", cmap="Blues",
xticklabels=grid_search.classes_, yticklabels=grid_search.classes_)

plt.title("Training Confusion Matrix")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()


# Confusion matrix for validation set

val_cm = confusion_matrix(y_val, y_val_pred, normalize='true')

plt.figure(figsize=(10, 8))

sns.heatmap(val_cm, annot=True, fmt=".2%", cmap="Blues",
xticklabels=grid_search.classes_, yticklabels=grid_search.classes_)

plt.title("Confusion Matrix SVM")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score

# Initialize lists to store F1-Scores
train_f1_scores = {disease_class: [] for disease_class in np.unique(y_train)}
val_f1_scores = {disease_class: [] for disease_class in np.unique(y_val)}
num_examples = []
# Define a range of number of examples for plotting
max_num_examples = 25000
batch_size = 1000
example_range = range(batch_size, max_num_examples+1, batch_size)

# Calculate F1-Scores for different number of examples
for num in example_range:
    # Train SVM classifier with the same parameters but with a subset of training examples
    svm_subset = SVC(**best_params)
    svm_subset.fit(X_train_pca[:num], y_train[:num])

# Predict on training set
    y_train_subset_pred = svm_subset.predict(X_train_pca[:num])
# Predict on validation set
    y_val_pred = svm_subset.predict(X_val_pca)

# Calculate F1-Scores for each class on training and validation sets
    for disease_class in np.unique(y_train):
        train_f1 = f1_score(y_train[:num], y_train_subset_pred, average=None, labels=[disease_class])[0]
```

```
        val_f1 = f1_score(y_val, y_val_pred, average=None, labels=[disease_class])[0]
        train_f1_scores[disease_class].append(train_f1)
        val_f1_scores[disease_class].append(val_f1)
    num_examples.append(num)

# Plot F1-Score curves for each class
plt.figure(figsize=(10, 6))
for disease_class in np.unique(y_train):
    plt.plot(num_examples, train_f1_scores[disease_class], label=f"Train F1-Score {disease_class}")
    plt.plot(num_examples, val_f1_scores[disease_class], label=f"Validation F1-Score {disease_class}")

plt.title('Training and Validation F1-Score for SVM')
plt.xlabel('Training Examples')
plt.ylabel('F1-Score')
plt.legend()
plt.grid(True)
plt.show()

import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from skimage.feature import local_binary_pattern
```

```
import zipfile
import Augmentor
from sklearn.utils import shuffle

# Extract the dataset from zip to '/content/dataset'
zip_path = '/content/drive/MyDrive/Feb24/CornDisease.zip'
extract_path = '/content/dataset'
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
# Set base path for dataset
base_path = '/content/dataset/CornDisease'

# Function to extract LBP features
def extract_lbp_features(image, num_points=24, radius=8):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = local_binary_pattern(gray, num_points, radius, method='uniform')
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, num_points + 3), range=(0,
num_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist
# Function to perform data augmentation
def augment_data(image_path, output_path, num_images):
    p = Augmentor.Pipeline(image_path, output_directory=output_path)
    p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
    p.flip_left_right(probability=0.5)
    p.flip_top_bottom(probability=0.5)
    p.zoom_random(probability=0.5, percentage_area=0.8)
    p.sample(num_images)
```

```
# Define paths for augmented data
augmented_path = '/content/augmented_data'
os.makedirs(augmented_path, exist_ok=True)

# Perform data augmentation
num_augmented_images = 2000  # Increased number of augmented images
for class_folder in os.listdir(base_path):
    class_path = os.path.join(base_path, class_folder)
    if os.path.isdir(class_path):
        augment_data(class_path, os.path.join(augmented_path, class_folder),
num_augmented_images)

# Load images and labels
data = []
labels = []

for folder in os.listdir(base_path):
    folder_path = os.path.join(base_path, folder)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, file_name)
            image = cv2.imread(image_path)
            lbp_features = extract_lbp_features(image)
            data.append(lbp_features)
            labels.append(folder)
# Load augmented data
augmented_data = []
augmented_labels = []
```

```
for folder in os.listdir(augmented_path):
    folder_path = os.path.join(augmented_path, folder)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, file_name)
            image = cv2.imread(image_path)
            lbp_features = extract_lbp_features(image)
            augmented_data.append(lbp_features)
            augmented_labels.append(folder)

# Convert data and labels to numpy arrays
data = np.array(data)
labels = np.array(labels)
augmented_data = np.array(augmented_data)
augmented_labels = np.array(augmented_labels)

# Combine augmented data with original data
X_combined = np.concatenate((data, augmented_data), axis=0)
y_combined = np.concatenate((labels, augmented_labels), axis=0)
# Split combined dataset into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_combined, y_combined,
test_size=0.2, random_state=42)

# Shuffle data
X_train, y_train = shuffle(X_train, y_train, random_state=42)
X_val, y_val = shuffle(X_val, y_val, random_state=42)
# Normalize combined data
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
# Perform PCA for feature selection
pca = PCA(n_components=0.95, whiten=True)
X_train_pca = pca.fit_transform(X_train_scaled)
X_val_pca = pca.transform(X_val_scaled)

# Define parameter grid for grid search
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1], 'kernel': ['linear', 'rbf']}
# Initialize SVM classifier
svm = SVC()
# Perform grid search
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train_pca, y_train)
# Get best parameters
best_params = grid_search.best_params_
# Train SVM classifier with best parameters
svm_best = SVC(**best_params)
svm_best.fit(X_train_pca, y_train)

# Predict on training set
y_train_pred = svm_best.predict(X_train_pca)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))
# Predict on validation set
y_val_pred = svm_best.predict(X_val_pca)
print("Validation Classification Report:")
print(classification_report(y_val, y_val_pred))
```

```
# Check accuracy
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_val = accuracy_score(y_val, y_val_pred)
print("Training Accuracy:", accuracy_train)
print("Validation Accuracy:", accuracy_val)

import matplotlib.pyplot as plt
from sklearn.metrics import f1_score

# Initialize lists to store F1-Scores
train_f1_scores = {disease_class: [] for disease_class in np.unique(y_train)}
val_f1_scores = {disease_class: [] for disease_class in np.unique(y_val)}
num_examples = []

# Define a range of number of examples for plotting
max_num_examples = 7000
batch_size = 1000
example_range = range(batch_size, max_num_examples+1, batch_size)

# Calculate F1-Scores for different number of examples
for num in example_range:
# Train SVM classifier with the same parameters but with a subset of training examples
    svm_subset = SVC(**best_params)
    svm_subset.fit(X_train_pca[:num], y_train[:num])

# Predict on training set
    y_train_subset_pred = svm_subset.predict(X_train_pca[:num])
```

```
# Predict on validation set
    y_val_pred = svm_subset.predict(X_val_pca)


# Calculate F1-Scores for each class on training and validation sets
    for disease_class in np.unique(y_train):
        train_f1 = f1_score(y_train[:num], y_train_subset_pred, average=None,
labels=[disease_class])[0]
        val_f1 = f1_score(y_val, y_val_pred, average=None, labels=[disease_class])[0]
        train_f1_scores[disease_class].append(train_f1)
        val_f1_scores[disease_class].append(val_f1)
    num_examples.append(num)


# Plot F1-Score curves for each class
plt.figure(figsize=(10, 6))
for disease_class in np.unique(y_train):
    plt.plot(num_examples, train_f1_scores[disease_class], label=f"Train F1-Score
{disease_class}")
    plt.plot(num_examples, val_f1_scores[disease_class], label=f"Validation F1-Score
{disease_class}", linestyle='--')
plt.title('Training and Validation F1-Score for SVM')
plt.xlabel('Training Examples')
plt.ylabel('F1-Score')
plt.legend()
plt.grid(True)
plt.show()


import os
import cv2
import numpy as np
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from skimage.feature import local_binary_pattern
import zipfile
import Augmentor
from sklearn.utils import shuffle

# Extract the dataset from zip to '/content/dataset'
zip_path = '/content/drive/MyDrive/Feb24/CornDisease.zip'
extract_path = '/content/dataset'
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Set base path for dataset
base_path = '/content/dataset/CornDisease'

# Function to extract LBP features
def extract_lbp_features(image, num_points=24, radius=8):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = local_binary_pattern(gray, num_points, radius, method='uniform')
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, num_points + 3), range=(0,
num_points + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist
```

```
# Function to perform data augmentation
def augment_data(image_path, output_path, num_images):
    p = Augmentor.Pipeline(image_path, output_directory=output_path)
    p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
    p.flip_left_right(probability=0.5)
    p.flip_top_bottom(probability=0.5)
    p.zoom_random(probability=0.5, percentage_area=0.8)
    p.sample(num_images)

# Define paths for augmented data
augmented_path = '/content/augmented_data'
os.makedirs(augmented_path, exist_ok=True)

# Perform data augmentation
num_augmented_images = 3000  # Increased number of augmented images
for class_folder in os.listdir(base_path):
    class_path = os.path.join(base_path, class_folder)
    if os.path.isdir(class_path):
        augment_data(class_path, os.path.join(augmented_path, class_folder),
num_augmented_images)

# Load images and labels
data = []
labels = []
for folder in os.listdir(base_path):
    folder_path = os.path.join(base_path, folder)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, file_name)
```

```
            image = cv2.imread(image_path)
            lbp_features = extract_lbp_features(image)
            data.append(lbp_features)
            labels.append(folder)

# Load augmented data
augmented_data = []
augmented_labels = []

for folder in os.listdir(augmented_path):
    folder_path = os.path.join(augmented_path, folder)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            image_path = os.path.join(folder_path, file_name)
            image = cv2.imread(image_path)
            lbp_features = extract_lbp_features(image)
            augmented_data.append(lbp_features)
            augmented_labels.append(folder)

# Convert data and labels to numpy arrays
data = np.array(data)
labels = np.array(labels)
augmented_data = np.array(augmented_data)
augmented_labels = np.array(augmented_labels)

# Combine augmented data with original data
X_combined = np.concatenate((data, augmented_data), axis=0)
y_combined = np.concatenate((labels, augmented_labels), axis=0)
```

```
# Split combined dataset into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_combined, y_combined,
test_size=0.2, random_state=42)

# Shuffle data
X_train, y_train = shuffle(X_train, y_train, random_state=42)
X_val, y_val = shuffle(X_val, y_val, random_state=42)

# Normalize combined data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Perform PCA for feature selection
pca = PCA(n_components=0.95, whiten=True)
X_train_pca = pca.fit_transform(X_train_scaled)
X_val_pca = pca.transform(X_val_scaled)

# Define parameter grid for grid search
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1], 'kernel': ['linear', 'rbf']}
# Initialize SVM classifier
svm = SVC()

# Perform grid search
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train_pca, y_train)
# Get best parameters
best_params = grid_search.best_params_
```

```
# Train SVM classifier with best parameters
svm_best = SVC(**best_params)
svm_best.fit(X_train_pca, y_train)

# Predict on training set
y_train_pred = svm_best.predict(X_train_pca)
print("Training Classification Report:")
print(classification_report(y_train, y_train_pred))

# Predict on validation set
y_val_pred = svm_best.predict(X_val_pca)
print("Validation Classification Report:")
print(classification_report(y_val, y_val_pred))

# Check accuracy
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_val = accuracy_score(y_val, y_val_pred)
print("Training Accuracy:", accuracy_train)
print("Validation Accuracy:", accuracy_val)

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion matrix for training set
train_cm = confusion_matrix(y_train, y_train_pred, normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(train_cm, annot=True, fmt=".2%", cmap="Blues",
xticklabels=grid_search.classes_, yticklabels=grid_search.classes_)
```

```
plt.title("Training Confusion Matrix")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()


# Confusion matrix for validation set

val_cm = confusion_matrix(y_val, y_val_pred, normalize='true')

plt.figure(figsize=(10, 8))

sns.heatmap(val_cm, annot=True, fmt=".2%", cmap="Blues",
xticklabels=grid_search.classes_, yticklabels=grid_search.classes_)

plt.title("Confusion Matrix SVM")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()
```