```
# -*- coding: utf-8 -*-
"""Augmentasi_data fix.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1dWcyuT5tAzvY43Zic9p4DtvSUBUeUCMQ

IMAGE DATA GENERATOR
"""

from google.colab import drive
drive.mount('/content/drive')

cd /content/drive/MyDrive

ls

from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import random
import cv2

# Definisikan path direktori gambar asli
images_path = '/content/drive/MyDrive/REVISI BATIK MANUAL/Data Val/Megamendung'

# Definisikan path direktori untuk menyimpan hasil augmentasi
augmented_path = '/content/drive/MyDrive/SHEAR 35 AUG/Val Aug /Megamendung'

# Inisialisasi objek ImageDataGenerator dengan pengaturan augmentasi yang
diinginkan
datagen = ImageDataGenerator(
    shear_range= 3.5)

# Membaca daftar path gambar asli
image_paths = [os.path.join(images_path, im) for im in os.listdir(images_path)]

# Mengatur jumlah gambar yang ingin di-generate
images_to_generate = 1

i = 1
while i <= images_to_generate:
    # Memilih gambar asli secara acak
    image_path = random.choice(image_paths)

    # Membaca gambar asli
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = image.reshape((1,) + image.shape)

    # Menghasilkan data augmentasi dengan mengaplikasikan transformasi pada gambar
    augmented_images = []
    for batch in datagen.flow(image, batch_size=1, save_to_dir=augmented_path,
save_prefix='NonGMB_Aug_', save_format='jpg'):
        augmented_images.append(batch[0])
        if len(augmented_images) == 1:  # Menghentikan setelah menghasilkan 1
gambar augmentasi
            break
```

```
i += 1
```

```
# -*- coding: utf-8 -*-
"""Sourcecode_BatiQu_MobilenetV1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/15tZG-EkKIusIYFIjzJXZJPmKWi_2mXyk
"""

#mengaitkan gdrive dengan google colab
from google.colab import drive
import os
drive.mount('/content/drive/')

from PIL import Image
import numpy as np
from tensorflow.keras.models import Sequential
import os, sys
import sklearn.metrics as metrics
import seaborn as sn
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from numpy import expand_dims
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras import models
import pandas as pd
from imutils import paths

#memanggil seluruh citra dan menampungnya ke dalam array yang sudah disediakan,
serta merize sesuai kebutuhan
train_datagen = ImageDataGenerator(rescale=1./255,
                                   featurewise_std_normalization= True,
                                   horizontal_flip= True,
                                   vertical_flip= True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('/content/drive/MyDrive/SHEAR
35 AUG/Train Aug',
                                                    target_size=(300, 300),
                                                    batch_size= 64,
                                                    shuffle= False,
                                                    class_mode='categorical')

validation_generator =
test_datagen.flow_from_directory('/content/drive/MyDrive/SHEAR 35 AUG/Val Aug ',
                                                        target_size=(300, 300),
                                                        batch_size= 64,
                                                        shuffle= True,
                                                        class_mode='categorical')

test_datagen = ImageDataGenerator(rescale = 1./255)  #Image normalization.

test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/SHEAR 35
AUG/Test Aug',
                                            target_size = (300, 300),
                                            batch_size = 32,
                                            shuffle= False,
```

```
                                                class_mode = 'categorical')

class_indices = train_generator.class_indices
print (class_indices)

labels_train = train_generator.classes
print(labels_train)

from tensorflow import keras
image_path = '/content/drive/MyDrive/SHEAR 35 AUG/Train
Aug/Megamendung/NonGMB_Aug__0_1130.jpg'

img = keras.preprocessing.image.load_img(image_path, target_size= (300, 300))
img_tensor = keras.preprocessing.image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)

#Uses ImageDataGenerator to flip the images
datagen = ImageDataGenerator(horizontal_flip=True)

#Creates our batch of one image
pic = datagen.flow(img_tensor, batch_size= 32)
plt.figure(figsize=(10,5))

#Plots our figures
for i in range(1, 4):
  plt.subplot(1, 3, i)
  batch = pic.next()
  image_ = batch[0].astype('uint8')
  plt.imshow(image_)
plt.show()

#Uses ImageDataGenerator to flip the images
datagen = ImageDataGenerator(brightness_range=[0.1, 1.5])
#Creates our batch of one image
pic = datagen.flow(img_tensor, batch_size= 32)
plt.figure(figsize=(10,5))

#Plots our figures
for i in range(1, 4):
  plt.subplot(1, 3, i)
  batch = pic.next()
  image_ = batch[0].astype('uint8')
  plt.imshow(image_)
plt.show()

#merancang model CNN sendiri#
import tensorflow as tf
import tensorflow.keras as keras
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNet

mobilenet = tf.keras.applications.MobileNet(input_shape= (300, 300, 3),
```

```
                                            include_top= False,
                                            weights='imagenet')

def create_model():
  model = Sequential()
  model.add(mobilenet)
  model.add(GlobalAveragePooling2D())
  model.add(Flatten())
  model.add(BatchNormalization())
  # model.add(Dense(1024, activation='relu'))
  # model.add(Dropout(0.2))
  # model.add(Dense(512, activation='relu'))
  # model.add(Dropout (0.02))
  model.add(Dense(256, activation='relu'))
  model.add(Dropout (0.35))
  model.add(Dense(128, activation='relu'))
  model.add(Dropout (0.35))
  model.add(Dense(6, activation='softmax' ,name="classification"))

  return model

from keras.utils.vis_utils import plot_model
model = create_model()
plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)

model.summary()

from keras.callbacks import EarlyStopping, ModelCheckpoint

#setting optimizer
filepath='model.h5'
opt  = tf.keras.optimizers.legacy.Adam(learning_rate=0.0001) #bikin variabel
namanya opt, dengan learning rate biasanya antara 0.001 - 0.1. jika terlalu besar
akan terjadi overshooting
model.compile(loss='categorical_crossentropy', optimizer= opt ,
metrics=['accuracy'])

es = EarlyStopping(monitor='val_loss', mode='min', verbose= 1, patience= 10)
checkpointer = ModelCheckpoint(filepath,
                               monitor = 'val_loss',
                               verbose=1,
                               save_best_only=True,
                               mode='min')
callbacks_list = [checkpointer, es]

model_history = []
history = model.fit(
            train_generator,
            validation_data = validation_generator,
            batch_size= 64,
            epochs= 100,
            callbacks= [checkpointer,es])

# Evaluasi model pada data pelatihan
train_score = model.evaluate(train_generator)
print('Train Score: ', train_score)

val_score = model.evaluate(validation_generator)
```

```
print('Val Score: ', val_score)

test_score = model.evaluate(test_set)
print('Test Score: ', test_score)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# Prediksi label menggunakan model
y_pred = model.predict_generator(test_set, steps=len(test_set))

# Ambil label yang sebenarnya
y_true = test_set.classes

# Hitung confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
conf_matrix = confusion_matrix(y_true, np.argmax(y_pred, axis=1))
print(conf_matrix)

# # plot confusion matrix
# plt.imshow(conf_matrix, cmap=plt.cm.Blues)
# plt.title('Confusion Matrix')

df_cm = pd.DataFrame(conf_matrix, index=test_set.class_indices,
columns=train_generator.class_indices)

# plot confusion matrix using seaborn heatmap
sns.heatmap(df_cm, annot=True, fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()

# membuat classification report
from sklearn.metrics import classification_report
y_pred_prob = model.predict_generator(test_set)
y_pred = np.argmax(y_pred_prob, axis=1)

report = classification_report(y_true, y_pred, target_names=test_set.class_indices)

print(report)

from PIL import Image
```

```
import numpy as np
from keras.models import load_model


classes = ['Ceplok', 'Kawung', 'Megamendung', 'Nitik', 'Parang', 'Tambal']
loaded_model = load_model ('model.h5')

image_testing = Image.open('/content/drive/MyDrive/SHEAR 35 AUG/Test
Aug/Parang/NonGMB_Aug__0_1440.jpg')
image_testing = np.array(image_testing.resize((300, 300)))/255.0
plt.imshow(image_testing)
image_testing.shape

image_testing = np.expand_dims(image_testing, axis=0)
print(image_testing.shape)

output = model.predict(image_testing)
best_index = np.argmax(output)
class_name = classes[best_index]

print(output)
print(best_index)

print(classes[best_index])

def representative_dataset_gen():
  num_calibration_steps = 100
  input_data = np.random.random((1, 300, 300, 3)).astype(np.float32)
  for _ in range(num_calibration_steps):
    # Get sample input data as a numpy array in a method of your choosing.
    yield [input]

converter = tf.lite.TFLiteConverter.from_saved_model("/content/model.pb")
converter.experimental_new_converter = True
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS]
converter.allow_custom_ops = False
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset_gen
converter.target_spec.supported_types = [tf.float16]
converter.experimental_new_quantizer = True
converter.dynamic_range_quantization = True
tflite_model = converter.convert()

with open("model.tflite", "wb") as f:
    f.write(tflite_model)
```

```
# -*- coding: utf-8 -*-
"""Sourcecode_BatiQu_Resnet152_V2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/18iDR4cfcn-ncw0wj9UvRGC9Q_Fkw4o4Z
"""

#mengaitkan gdrive dengan google colab
from google.colab import drive
import os
drive.mount('/content/drive/')

# #setting direktori dataset
# base_dir = '/content/drive/MyDrive/DATASET BATIK MANUAL'
# !ls "/content/drive/MyDrive/DATASET BATIK MANUAL"

from PIL import Image
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation, Flatten,
Dense, Dropout
import os, sys
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
import sklearn.metrics as metrics
import seaborn as sn
import matplotlib.pyplot as plt
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.image import load_img, img_to_array,
ImageDataGenerator
from numpy import expand_dims
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras import models
import pandas as pd
from imutils import paths
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras

#memanggil seluruh citra dan menampungnya ke dalam array yang sudah disediakan,
serta merize sesuai kebutuhan
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   featurewise_std_normalization= True,
                                   horizontal_flip=True,
                                   vertical_flip= True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('/content/drive/MyDrive/SHEAR
AUG BATIK/DATA TRAIN',
                                                    target_size=(300, 300),
                                                    batch_size=64,
                                                    shuffle= False,
                                                    class_mode= 'categorical')
```

```
validation_generator =
test_datagen.flow_from_directory('/content/drive/MyDrive/SHEAR AUG BATIK/DATA VAL',
                                                         batch_size= 64,
                                                         target_size=(300, 300),
                                                         shuffle= True,
                                                         class_mode= 'categorical')

test_datagen = ImageDataGenerator(rescale = 1./255)  #Image normalization.

test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/SHEAR AUG
BATIK/DATA TEST',
                                            target_size = (300, 300),
                                            batch_size = 64,
                                            shuffle= False,
                                            class_mode = 'categorical')

class_indices = train_generator.class_indices
print (class_indices)

labels_train= train_generator.classes
print(labels_train)

#Menampilkan data prepocessing
image_path = '/content/drive/MyDrive/SHEAR AUG BATIK/DATA
TRAIN/MEGAMENDUNG/NonGMB_Aug__0_1124.jpg'

img = keras.preprocessing.image.load_img(image_path, target_size= (300, 300))
img_tensor = keras.preprocessing.image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)

#Uses ImageDataGenerator to flip the images
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip= True)
#Creates our batch of one image
pic = datagen.flow(img_tensor, batch_size =64)
plt.figure(figsize=(20,10))

#Plots our figures
for i in range(1,4):
  plt.subplot(1, 3, i)
  batch = pic.next()
  image_ = batch[0].astype('uint8')
  plt.imshow(image_)
plt.show()

#Uses ImageDataGenerator to flip the images
datagen = ImageDataGenerator(brightness_range=[0.1, 2.5])
#Creates our batch of one image
pic = datagen.flow(img_tensor, batch_size =64)
plt.figure(figsize=(20,10))

#Plots our figures
for i in range(1, 4):
  plt.subplot(1, 3, i)
  batch = pic.next()
  image_ = batch[0].astype('uint8')
  plt.imshow(image_)
plt.show()
```

```
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Flatten,
Dense, Activation, Input
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.regularizers import l2
import tensorflow as tf


model = tf.keras.Sequential([
    ResNet152V2(
        include_top=False,
        weights='imagenet',
        input_shape=(300, 300, 3)),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(6, activation='softmax', name='classification')
])
model.layers[0].trainable = False
plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)

model.summary()
plt.savefig("ModelSummary.jpg")
plt.show()

# Compile the Neural network
opt = tf.keras.optimizers.RMSprop(learning_rate=0.0001)
model.compile(optimizer = opt, loss = 'categorical_crossentropy', metrics =
['accuracy'])

from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

filepath = 'model_Batik.h5'
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience= 10)
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
callbacks_list = [checkpoint, es]

#save the model history in a list after fitting so taht we can plot later
model_history = []
history = model.fit(
        train_generator,
        validation_data=validation_generator,
        batch_size=64,
        epochs=100,
        callbacks=[checkpoint, es])

# Evaluasi model pada data pelatihan
train_score = model.evaluate(train_generator)
print('Train Score:', train_score)

val_score = model.evaluate(validation_generator)
print('Val Score:', val_score)

test_score = model.evaluate(test_set)
print('Test Score:', test_score)
```

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# Prediksi label menggunakan model
y_pred = model.predict_generator(test_set, steps=len(test_set))

# Ambil label yang sebenarnya
y_true = test_set.classes

# Hitung confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
conf_matrix = confusion_matrix(y_true, np.argmax(y_pred, axis=1))
print(conf_matrix)

# # plot confusion matrix
# plt.imshow(conf_matrix, cmap=plt.cm.Blues)
# plt.title('Confusion Matrix')

df_cm = pd.DataFrame(conf_matrix, index= test_set.class_indices, columns=
test_set.class_indices)

# plot confusion matrix using seaborn heatmap
sns.heatmap(df_cm, annot=True, fmt= 'd')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()

# Step 1: Predict on test data
y_pred_prob = model.predict_generator(test_set)

# Step 2: Get class with highest probability
y_pred = np.argmax(y_pred_prob, axis=1)

# Step 3: Calculate accuracy score
from sklearn.metrics import accuracy_score
y_true = test_set.classes
accuracy = accuracy_score(y_true, y_pred)
print("Accuracy:", accuracy)

# membuat classification report
from sklearn.metrics import classification_report
report = classification_report(y_true, y_pred, target_names=test_set.class_indices)
```

```
print(report)

from PIL import Image
import numpy as np
from keras.models import load_model


classes = ['Ceplok', 'Kawung', 'Megamendung', 'Nitik', 'Parang', 'Tambal']
loaded_model = load_model ('model_Batik.h5')

image_testing = Image.open('/content/drive/MyDrive/SHEAR AUG BATIK/DATA
TEST/CEPLOK/NonGMB_Aug__0_152.jpg')
image_testing = np.array(image_testing.resize((300, 300)))/255.0
plt.imshow(image_testing)
image_testing.shape

image_testing = np.expand_dims(image_testing, axis=0)
print(image_testing.shape)

output = model.predict(image_testing)
best_index = np.argmax(output)
class_name = classes[best_index]

print(output)
print(best_index)

print(classes[best_index])
```