LAMPIRAN

Lampiran 1. Script training & testing data dengan Python

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import matplotlib.pyplot as plt
import seaborn as sea
import cv2
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Dropout, Flatten,
Input, Conv2D, MaxPool2D
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tqdm.notebook import tqdm
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet import
preprocess_input
from tensorflow.keras import layers

# Load Data
df = pd.read_csv('/kaggle/input/dataset-klasifikasi-
kebersihan-jalanan/dataset/metadata.csv')
df.head()

# Data Augmentation using Albumentations
!pip install albumentations
import albumentations as A

PATH = '/kaggle/input/dataset-klasifikasi-kebersihan-
jalanan/dataset/Images/Images/'

df['filename'] = df['filename'].map(lambda x : PATH + x)

# Augmentasi
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.Rotate(limit=30, p=0.5),
    A.ElasticTransform(p=0.5),
    A.Blur(blur_limit=3, p=0.5),
])

# Split Data
df_train, df_val = train_test_split(df, test_size=0.2,
random_state=42)
```

```
# Image Processing Functions with augmentation
def map_function(img, label, training):
    img = plt.imread(img.decode())[:, :, :3]
    img = cv2.resize(img, (224, 224))
    if training:
        img = transform(image=img)['image']
    img = preprocess_input(img)
    label = to_categorical(label, num_classes=2)
    return img, label

# Constants
IMG_SHAPE = (224, 224, 3)
NUM_CLASSES = 2
BATCH_SIZE = 64

def create_dataset(df, training=False):
    dataset =
tf.data.Dataset.from_tensor_slices((df['filename'],
df['label']))
    dataset = dataset.shuffle(1000)
    dataset = dataset.map(lambda img, label:
tf.numpy_function(
        map_function, [img, label, training], [tf.float32,
tf.float32]),

num_parallel_calls=tf.data.experimental.AUTOTUNE).batch(BATCH_
SIZE)
    dataset =
dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return dataset

# Create Datasets
train_dataset = create_dataset(df_train, training=True)
val_dataset = create_dataset(df_val)

# Base MobileNetV2 Model
base_model = MobileNetV2(weights='imagenet',
input_shape=IMG_SHAPE, include_top=False)
base_model.trainable = True

n = int(0.70 * len(base_model.layers))
for i in range(n):
    base_model.layers[i].trainable = False

# Build Model
def make_model():
    inp = Input(shape=IMG_SHAPE)
    x = base_model(inp)
    x = Dropout(0.2)(x)
    x = Conv2D(256, (3, 3), activation='relu')(x)
    x = MaxPool2D(2)(x)
    x = Dropout(0.2)(x)
    x = Flatten()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.2)(x)
    out = Dense(NUM_CLASSES, activation='softmax')(x)

    model = Model(inputs=inp, outputs=out)
    return model

# Instantiate Model
model = make_model()

# Compile Model
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

# Model Summary
model.summary()

lossfxn = keras.losses.CategoricalCrossentropy()
optimizer = keras.optimizers.Adam(learning_rate = 1e-4)

train_acc_metric = keras.metrics.CategoricalAccuracy()
val_acc_metric = keras.metrics.CategoricalAccuracy()

@tf.function
def train_step(x, y):
    with tf.GradientTape() as tape:
        y_pred = model(x, training = True)
        loss = lossfxn(y_pred, y)

    grads = tape.gradient(loss, model.trainable_weights)
    optimizer.apply_gradients(zip(grads,
model.trainable_weights))
    train_acc_metric.update_state(y, y_pred)
    acc = train_acc_metric.result()

    return loss, acc

@tf.function
def val_step(x, y):
    y_pred = model(x, training = False)
    loss = lossfxn(y_pred, y)
    val_acc_metric.update_state(y, y_pred)
    acc = train_acc_metric.result()
    return loss, acc

def append_metrics(train_loss, train_acc, val_loss, val_acc):
    TRAIN_LOSS.append(train_loss)
    TRAIN_ACC.append(train_acc)
    VAL_LOSS.append(val_loss)
    VAL_ACC.append(val_acc)

import time

start_time = time.time()

EPOCHS = 10
```

```
NUM_BATCHES = len(train_dataset)
TEST_BATCHES = len(val_dataset)
TRAIN_LOSS, TRAIN_ACC, VAL_LOSS, VAL_ACC = [], [], [], []

for epoch in range(EPOCHS):
    train_loss, test_loss, train_acc, test_acc = 0, 0, 0, 0

    for x, y in tqdm(train_dataset):
        batch_loss, batch_acc = train_step(x, y)
        train_loss += batch_loss
        train_acc += batch_acc

    for x, y in val_dataset:
        batch_loss, batch_acc = val_step(x, y)
        test_loss += batch_loss
        test_acc += batch_acc

    train_loss = train_loss/NUM_BATCHES
    val_loss = test_loss/TEST_BATCHES

    train_acc = train_acc/NUM_BATCHES
    val_acc = test_acc/TEST_BATCHES

    train_acc_metric.reset_states()
    val_acc_metric.reset_states()

    append_metrics(train_loss, train_acc, val_loss, val_acc)

    print("Epoch: {} Training: [Loss:{:.3f} Acc:{:.3f}]
Validation: [Loss:{:.3f} Acc:{:.3f}]".format(
            epoch, train_loss, train_acc, val_loss, val_acc))

end_time = time.time()
inference_time = end_time - start_time
print("Waktu training:", inference_time, "detik")

plt.figure(figsize = (16, 5))

plt.subplot(1,2,1)
plt.title('Loss')
plt.plot(TRAIN_LOSS, marker = 'o', label = 'Training')
plt.plot(VAL_LOSS, '--r',label = 'Validation')
plt.legend()

plt.subplot(1,2,2)
plt.title('Accuracy')
plt.plot(TRAIN_ACC, marker = 'o', label = 'Training')
plt.plot(VAL_ACC, '--r', label = 'Validation')
plt.legend()

test_pred, true = [], []

for x, y in val_dataset:
    y_pred = np.argmax(model(x, training = False), axis = 1)
    test_pred.extend(y_pred)
    true.extend(np.argmax(y, axis = 1))
```

```
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(true, test_pred)

# Membuat heatmap dari confusion matrix
plt.figure(figsize=(8, 6))
sea.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='g',
xticklabels=['Bersih', 'Kotor'], yticklabels=['Bersih',
'Kotor'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

d_names = {
    0: 'bersih',
    1: 'kotor',
}

samples = df_val.sample(frac = 1.0)

plt.figure(figsize = (20, 10))

for i in range(10):
    img = plt.imread(samples.iloc[i, 0])[:, :, :3]
    img = cv2.resize(img, (224, 224))
    img2 = preprocess_input(img)
    img2 = np.resize(img2, (1, 224, 224, 3))
    pred = d_names[np.argmax(model.predict(img2), axis =
1)[0]]
    l = d_names[samples.iloc[i,1]]

    plt.subplot(2, 5,i + 1)
    plt.axis('off')
    plt.imshow(img)
    plt.title('Fakta: {}\nPrediksi: {}\n{}'.format(l, pred,
"Benar" if l == pred else "Salah"))

plt.show()
```