LAMPIRAN

Lampiran 1 Kode Augmentasi Data

```
import cv2
import numpy as np
import random
import os

def augment_image_opencv(image_path, output_dir, count):
    # Buka gambar
    image = cv2.imread(image_path)
    height, width = image.shape[:2]
    aspect_ratio = width / height

    # Loop untuk membuat beberapa augmentasi
    for i in range(count):
        aug_image = image.copy()

        # Penyesuaian kecerahan
        brightness_factor = random.uniform(0.7, 1.05)
        aug_image = cv2.convertScaleAbs(aug_image, alpha=brightness_factor, beta=0)

        # Penyesuaian hue
        hsv_image = cv2.cvtColor(aug_image, cv2.COLOR_BGR2HSV)
        hue_shift = random.uniform(-20, 20)  # dalam derajat, diperluas rentangnya
        hsv_image[:, :, 0] = (hsv_image[:, :, 0] + hue_shift) % 180  # Hue di OpenCV
adalah [0, 180]
        aug_image = cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)

        # Zoom dan unzoom dengan menjaga orientasi
        scale = random.uniform(0.8, 1.1)  # Mengubah skala gambar
        new_width = int(width * scale)
        new_height = int(new_width / aspect_ratio)
        aug_image = cv2.resize(aug_image, (new_width, new_height),
interpolation=cv2.INTER_LANCZOS4)
# Crop atau pad untuk mengembalikan ukuran semula
if scale > 1.0:
    x_start = (new_width - width) // 2
    y_start = (new_height - height) // 2
    aug_image = aug_image[y_start:y_start + height, x_start:x_start + width]
else:
    # Tambah padding dengan mode BORDER_REFLECT_101 untuk menghindari area
hitam
    top = (height - new_height) // 2
    bottom = height - new_height - top
    left = (width - new_width) // 2
    right = width - new_width - left
    aug_image = cv2.copyMakeBorder(aug_image, top, bottom, left, right,
cv2.BORDER_REFLECT_101)

# Simpan gambar hasil augmentasi
output_path = os.path.join(output_dir, f"aug_{i}_{os.path.basename(image_path)}")
```

(Lanjutan lampiran 1) Kode Augmentasi Data

```
input_dir = r'C:\Users\riyof\Pictures\cuk\DATASET_SIPATEX\bagus'
output_dir = r'C:\Users\riyof\Pictures\cuk\DATASET_SIPATEX\bagus/AUGMENTED'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Mengambil semua gambar dari folder input
image_paths = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith(('png', 'jpg', 'jpeg', 'bmp', 'gif'))]

for image_path in image_paths:
    augment_image_opencv(image_path, output_dir, 1)  # Membuat 4 augmentasi per gambar
```

Lampiran 2 Kode Pelatihan CNN tanpa *tuning*

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import MaxPooling2D, Flatten, Dense, ZeroPadding2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import os
import cv2
import seaborn as sns
import matplotlib.pyplot as plt

# Fungsi untuk memuat dan preprocess gambar
def load_and_preprocess_images(image_folder):
    images = []
    labels = []
    categories = ['bagus', 'pakan_hilang', 'Snarling', 'pakan_dobel', 'lusi_putus', 'pakan_bulu']

    for category in categories:
        category_path = os.path.join(image_folder, category)
        label = categories.index(category)
        for img_name in os.listdir(category_path):
            img_path = os.path.join(category_path, img_name)
            img = cv2.imread(img_path)
            if img is not None:  # Memastikan gambar terbaca dengan benar
                img = cv2.resize(img, (224, 224))  # Ubah ukuran gambar
                img = img / 255.0  # Normalisasi gambar
                images.append(img)
                labels.append(label)

    return np.array(images), np.array(labels)
```

(Lanjutan Lampiran 2 Kode Pelatihan CNN Tanpa *Tuning*)

```
# Memuat gambar
image_folder = r'D:\cuk\DATASET_SIPATEX'
images, labels = load_and_preprocess_images(image_folder)

# Membagi dataset menjadi data pelatihan, validasi, dan pengujian
X_train, X_temp, y_train, y_temp = train_test_split(images, labels, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Augmentasi data
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(X_train)

# Menggunakan model VGG16 pra-latih
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
predictions = Dense(6, activation='softmax')(x)  # Jumlah kelas sesuai dengan jumlah kategori cacat kain

model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Melatih model dengan early stopping
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=50, validation_data=(X_val, y_val), callbacks=[early_stoppin

# Evaluasi model pada data pengujian
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'Akurasi data pengujian: {test_acc:.2f}')

# Menyimpan model
model.save('model_cacat_kain_maxpooling_padding.h5')                # Menyimpan model ke file
```

(Lanjutan Lampiran 2 Kode Pelatihan CNN Tanpa *Tuning*)

```
# Menyimpan grafik pelatihan (loss dan akurasi)
plt.figure(figsize=(12, 6))

# Grafik Akurasi
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Akurasi (Training)')
plt.plot(history.history['val_accuracy'], label='Akurasi (Validasi)')
plt.xlabel('Epoch')
plt.ylabel('Akurasi')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('Grafik Akurasi Pelatihan MaxPooling dengan Padding')

# Grafik Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Loss (Training)')
plt.plot(history.history['val_loss'], label='Loss (Validasi)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Grafik Loss Pelatihan MaxPooling dengan Padding')

plt.savefig('grafik_pelatihan_maxpooling_padding.png')  # Menyimpan grafik
pelatihan ke file
```

Lampiran 3 Kode Pelatihan CNN dengan *maxpooling*

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import MaxPooling2D, Flatten, Dense, ZeroPadding2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import os
import cv2
import seaborn as sns
import matplotlib.pyplot as plt

# Fungsi untuk memuat dan preprocess gambar
def load_and_preprocess_images(image_folder):
    images = []
    labels = []
    categories = ['bagus', 'pakan_hilang', 'Snarling', 'pakan_dobel', 'lusi_putus',
'pakan_bulu']
```

(Lanjutan Lampiran 3 Kode Pelatihan CNN dengan *maxpooling*)

```
    for category in categories:
        category_path = os.path.join(image_folder, category)
        label = categories.index(category)
        for img_name in os.listdir(category_path):
            img_path = os.path.join(category_path, img_name)
            img = cv2.imread(img_path)
            if img is not None:  # Memastikan gambar terbaca dengan benar
                img = cv2.resize(img, (224, 224))  # Ubah ukuran gambar
                img = img / 255.0  # Normalisasi gambar
                images.append(img)
                labels.append(label)
    return np.array(images), np.array(labels)
# Memuat gambar
image_folder = r'D:\cuk\DATASET_SIPATEX'
images, labels = load_and_preprocess_images(image_folder)
# Membagi dataset menjadi data pelatihan, validasi, dan pengujian
X_train, X_temp, y_train, y_temp = train_test_split(images, labels, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
# Augmentasi data
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(X_train)
# Menggunakan model VGG16 pra-latih
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
# Membekukan beberapa lapisan awal
for layer in base_model.layers:
    layer.trainable = False
# Menambahkan lapisan baru untuk klasifikasi dengan MaxPooling2D dan padding
x = base_model.output
x = ZeroPadding2D(padding=(1, 1))(x)  # Menambahkan padding sebelum MaxPooling
x = MaxPooling2D(pool_size=(2, 2), padding='valid')(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(6, activation='softmax')(x)  # Jumlah kelas sesuai dengan jumlah kategori cacat kain

model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy'
```

(Lanjutan Lampiran 3 Kode Pelatihan CNN dengan *Maxpooling*)

```
# Melatih model dengan early stopping
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=50, validation_data=(X_val, y_val), callbacks=[early_stopping])

# Evaluasi model pada data pengujian
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'Akurasi data pengujian: {test_acc:.2f}')

# Menyimpan model
model.save('model_cacat_kain_maxpooling_padding.h5')  # Menyimpan model ke file

# Menyimpan grafik pelatihan (loss dan akurasi)
plt.figure(figsize=(12, 6))

# Grafik Akurasi
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Akurasi (Training)')
plt.plot(history.history['val_accuracy'], label='Akurasi (Validasi)')
plt.xlabel('Epoch')
plt.ylabel('Akurasi')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('Grafik Akurasi Pelatihan MaxPooling dengan Padding')

# Grafik Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Loss (Training)')
plt.plot(history.history['val_loss'], label='Loss (Validasi)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Grafik Loss Pelatihan MaxPooling dengan Padding')

plt.savefig('grafik_pelatihan_maxpooling_padding.png')  # Menyimpan grafik pelatihan ke file
```

Lampiran 4 Kode Penambahan Metadata ke Dalam Model tflite

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import os
from absl import app
from absl import flags
import flatbuffers
import tensorflow as tf
from tflite_support import metadata_schema_py_generated as _metadata_fb
from tflite_support import metadata as _metadata
FLAGS = flags.FLAGS


def define_flags():
  flags.DEFINE_string("model_file", None,
                      "Path and file name to the TFLite model file.")
  flags.DEFINE_string("label_file", None, "Path to the label file.")
  flags.DEFINE_string("export_directory", None,
                      "Path to save the TFLite model files with metadata.")
  flags.mark_flag_as_required("model_file")
  flags.mark_flag_as_required("label_file")
  flags.mark_flag_as_required("export_directory")


class ModelSpecificInfo(object):
  """Holds information that is specificly tied to an image classifier."""

  def __init__(self, name, version, image_width, image_height, image_min,
               image_max, mean, std, num_classes, author):
    self.name = name
    self.version = version
    self.image_width = image_width
    self.image_height = image_height
    self.image_min = image_min
    self.image_max = image_max
    self.mean = mean
    self.std = std
    self.num_classes = num_classes
    self.author = author
```

```
_MODEL_INFO = {
"model_cacat_kain_maxpooling_padding_tanpaearlystopping.tflite":
        ModelSpecificInfo(
            name="Klasifikasi dan Deteksi Cacat Kain",
            version="1.1",
            image_width=224,
            image_height=224,
            image_min=0,
            image_max=255,
            mean=[127.5],
            std=[127.5],
            num_classes=6,
            author="Syam Agung Widiarna")
}

class MetadataPopulatorForImageClassifier(object):
  """Populates the metadata for an image classifier."""

  def __init__(self, model_file, model_info, label_file_path):
    self.model_file = model_file
    self.model_info = model_info
    self.label_file_path = label_file_path
    self.metadata_buf = None

  def populate(self):
    """Creates metadata and then populates it for an image classifier."""
    self._create_metadata()
    self._populate_metadata()

  def _create_metadata(self):
    """Creates the metadata for an image classifier."""
# Creates model info.
model_meta = _metadata_fb.ModelMetadataT()
model_meta.name = self.model_info.name
model_meta.description = ("Mengklasifikasikan jenis cacat kain "
                          "dari %d kategori." %
                          self.model_info.num_classes)
model_meta.version = self.model_info.version
model_meta.author = self.model_info.author
model_meta.license = ("no license")
# Creates input info.
input_meta = _metadata_fb.TensorMetadataT()
input_meta.name = "image"
input_meta.description = (
    "Input image to be classified. The expected image is {0} x {1},
    "three channels (red, blue, and green) per pixel. Each value in
    "tensor is a single byte between {2} and {3}.".format(
        self.model_info.image_width, self.model_info.image_height,
        self.model_info.image_min, self.model_info.image_max))
input_meta.content = _metadata_fb.ContentT()
input_meta.content.contentProperties = _metadata_fb.ImageProperties
input_meta.content.contentProperties.colorSpace = (
```

```
    # Creates output info.
    output_meta = _metadata_fb.TensorMetadataT()
    output_meta.name = "probability"
    output_meta.description = "Probabilities of the %d labels respectively." %
self.model_info.num_classe
    output_meta.content = _metadata_fb.ContentT()
    output_meta.content.content_properties                                    =
_metadata_fb.FeaturePropertiesT()
    output_meta.content.contentPropertiesType = (
        _metadata_fb.ContentProperties.FeatureProperties)
    output_stats = _metadata_fb.StatsT()
    output_stats.max = [1.0]
    output_stats.min = [0.0]
    output_meta.stats = output_stats
    label_file = _metadata_fb.AssociatedFileT()
    label_file.name = os.path.basename(self.label_file_path)
    label_file.description = "Labels for objects that the model can recognize."
    label_file.type                                                           =
_metadata_fb.AssociatedFileType.TENSOR_AXIS_LABELS
    output_meta.associatedFiles = [label_file]

    # Creates subgraph info.
    subgraph = _metadata_fb.SubGraphMetadataT()
    subgraph.inputTensorMetadata = [input_meta]
    subgraph.outputTensorMetadata = [output_meta]
    model_meta.subgraphMetadata = [subgraph]

    b = flatbuffers.Builder(0)
    b.Finish(
        model_meta.Pack(b),
        _metadata.MetadataPopulator.METADATA_FILE_IDENTIFIER)
    self.metadata_buf = b.Output()

  def _populate_metadata(self):
    """Populates metadata and label file to the model file."""
    populator = _metadata.MetadataPopulator.with_model_file(self.model_file)
    populator.load_metadata_buffer(self.metadata_buf)
    populator.load_associated_files([self.label_file_path])
    populator.populate()


def main(_):
    model_file = FLAGS.model_file
    model_basename = os.path.basename(model_file)
    if model_basename not in _MODEL_INFO:
        raise ValueError(
            "The model info for, {0}, is not defined yet.".format(model_basename))
```

```
    export_model_path = os.path.join(FLAGS.export_directory,
os.path.splitext(model_basename)[0] + "_metadata.tflite")

    # Copies model_file to export_path.
    tf.io.gfile.copy(model_file, export_model_path, overwrite=False)

    # Generate the metadata objects and put them in the model file
    populator = MetadataPopulatorForImageClassifier(
        export_model_path, _MODEL_INFO.get(model_basename),
FLAGS.label_file)
    populator.populate()

    # Validate the output model file by reading the metadata and
produce
    # a json file with the metadata under the export path
    displayer = _metadata.MetadataDisplayer.with_model_file(export_model_path)
    export_json_file = os.path.join(FLAGS.export_directory,
                                    os.path.splitext(model_basename)[0] +
"_metadata.json")
    json_file = displayer.get_metadata_json()
    with open(export_json_file, "w") as f:
        f.write(json_file)

    print("Finished populating metadata and associated file to the
model:")
    print(model_file)
    print("The metadata json file has been saved to:")
    print(export_json_file)
    print("The associated file that has been packed to the model is:")
    print(displayer.get_packed_associated_file_list())



if __name__ == "__main__":
  define_flags()
  app.run(main)
```

Lampiran 5 Kode untuk Membuat Antarmuka Sederhana dengan Gradio

```
import gradio as gr
import tensorflow as tf
import numpy as np
from PIL import Image

model = tf.keras.models.load_model("C:/Users/ACER/Downloads/Coding_Codingan-20240825T122849Z-001/Coding_Codingan/model_cacat_kain_maxpooling_padding.h5")
categories = [
    'bagus',
    'pakan_hilang',
    'Snarling',
    'pakan_dobel',
    'lusi_putus',
    'pakan_bulu'
]

def predict_image(image):
    if not isinstance(image, Image.Image):
        image = Image.fromarray(image)
    image = image.resize((224, 224))
    image_array = np.array(image) / 255.0
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension
    predictions = model.predict(image_array)
    predicted_class_index = np.argmax(predictions, axis=-1)[0]
    predicted_class = categories[predicted_class_index]

    return predicted_class

# Create Gradio interface
interface = gr.Interface(
    fn=predict_image,
    inputs=gr.Image(),  # No shape argument here
    outputs=gr.Text(),
    live=True
)

interface.launch()
```