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

```
Mounted at /content/drive
```

```
import zipfile
zip_path = '/content/drive/MyDrive/Dental OPG XRAY Dataset.zip'
extract_path ='/content/drive/MyDrive/barubaru'
# Ekstrak file ZIP utama
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```

Cek Duplikat Data

```
import os
import hashlib
import os
```

```
def find_duplicate_images(data_dir):
    hashes = {}

    for root, _, files in os.walk(data_dir):
        for name in files:
            if name.lower().endswith(('.png', '.jpg', '.jpeg')):
                path = os.path.join(root, name)
                try:
                    with open(path, "rb") as f:
                        file_hash = hashlib.md5(f.read()).hexdigest()
                    hashes.setdefault(file_hash, []).append(path)
                except Exception as e:
                    print(f"Gagal memproses {path}: {e}")

    return {h: p for h, p in hashes.items() if len(p) > 1}


data_dir = "/content/drive/MyDrive/barubaru/Dental OPG XRAY Dataset/Dental OPG (Classification)"
duplicates = find_duplicate_images(data_dir)

if duplicates:
    for h, paths in duplicates.items():
        print(f"Hash: {h}")
        for p in paths:
            print(f"  - {p}")
else:
    print("Tidak ada gambar duplikat.")
```

```
Tidak ada gambar duplikat.
```

```
def count_duplicate_images_per_class(data_dir):
    hashes = {}
    dup_count = {}

    for root, _, files in os.walk(data_dir):
        cls = os.path.basename(root)
        dup_count.setdefault(cls, 0)

        for f in files:
            if f.lower().endswith(('.png', '.jpg', '.jpeg')):
                path = os.path.join(root, f)
                try:
                    with open(path, "rb") as img:
                        h = hashlib.md5(img.read()).hexdigest()
                except:
                    continue

                if h in hashes:
                    hashes[h].append(path)
                    dup_count[cls] += 1
                else:
                    hashes[h] = [path]
```

```
    return dup_count


data_dir = "/content/drive/MyDrive/barubaru/Dental OPG XRAY Dataset/Dental OPG (Classification)"
counts = count_duplicate_images_per_class(data_dir)

for cls, n in counts.items():
    print(f"{cls}: {n}")
```

```
Dental OPG (Classification): 0
BDC-BDR: 0
Healthy Teeth: 0
Caries: 0
```

```
def remove_duplicate_images(data_dir, keep_minority=True):
    hashes = {}
    dups = {}
    cls_count = {}

    for root, _, files in os.walk(data_dir):
        cls = os.path.basename(root)
        cls_count.setdefault(cls, 0)

        for f in files:
            if f.lower().endswith(('.png', '.jpg', '.jpeg')):
                path = os.path.join(root, f)
                try:
                    with open(path, "rb") as img:
                        h = hashlib.md5(img.read()).hexdigest()
                except:
                    continue

                if h in hashes:
                    dups.setdefault(h, []).append(path)
                    cls_count[cls] += 1
                else:
                    hashes[h] = path

    minority = min(cls_count, key=cls_count.get)

    for paths in dups.values():
        if keep_minority:
            kept = False
            for p in paths:
                cls = os.path.basename(os.path.dirname(p))
                if cls == minority and not kept:
                    kept = True
                else:
                    os.remove(p)
                    print(f"Removed: {p}")
        else:
            for p in paths[1:]:
                os.remove(p)
                print(f"Removed: {p}")


data_dir = "/content/drive/MyDrive/barubaru/Dental OPG XRAY Dataset/Dental OPG (Classification)"
remove_duplicate_images(data_dir)
```

```
def count_files_per_folder(data_dir):
    return {
        os.path.basename(root): len(files)
        for root, _, files in os.walk(data_dir)
    }
data_dir = "/content/drive/MyDrive/barubaru/Dental OPG XRAY Dataset/Dental OPG (Classification)"
for folder, n in count_files_per_folder(data_dir).items():
    print(f"{folder}: {n}")
```

```
Dental OPG (Classification): 0
BDC-BDR: 52
Healthy Teeth: 89
Caries: 88
```

```
import shutil

folders_to_remove = ["Fractured Teeth", "Impacted teeth", "Infection"]
data_dir = "/content/drive/MyDrive/barubaru/Dental OPG XRAY Dataset/Dental OPG (Classification)"

for folder in folders_to_remove:
  folder_path = os.path.join(data_dir, folder)
  if os.path.exists(folder_path):
    try:
      shutil.rmtree(folder_path)
      print(f"Folder '{folder}' removed successfully.")
    except OSError as e:
      print(f"Error removing folder '{folder}': {e}")
  else:
    print(f"Folder '{folder}' not found.")
```

```
Folder 'Fractured Teeth' not found.
Folder 'Impacted teeth' not found.
Folder 'Infection' not found.
```

Training and Validation

```
import os
import cv2
import json
import glob
import numpy as np
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import shutil
```

```
from tensorflow.keras import utils

import tensorflow as tf
# Path ke dataset
data_dir = "/content/drive/MyDrive/barubaru/Dental OPG XRAY Dataset/Dental OPG (Classification)"



# Load training data
train_data = utils.image_dataset_from_directory(
    data_dir,
    labels="inferred",
    label_mode="int",
    validation_split=0.1,
    subset="training",
    shuffle=True,
    color_mode="rgb",
    image_size=(299, 299),
    batch_size=16,
    seed=40,
)



# Load validation data
data_valid = utils.image_dataset_from_directory(
    data_dir,
    labels="inferred",
    label_mode="int",
    validation_split=0.1,
    subset="validation",
    shuffle=True,
    color_mode="rgb",
    image_size=(299, 299),
    batch_size=16,
    seed=40,
)
```

```
# Verifikasi dataset
class_names = train_data.class_names
print("Classes:", class_names)



# Menampilkan beberapa batch data
for images, labels in train_data.take(1):
    print("Shape of image batch:", images.shape)
    print("Shape of label batch:", labels.shape)
```

```
Found 229 files belonging to 3 classes.
Using 207 files for training.
Found 229 files belonging to 3 classes.
Using 22 files for validation.
Classes: ['BDC-BDR', 'Caries', 'Healthy Teeth']
Shape of image batch: (16, 299, 299, 3)
Shape of label batch: (16,)
```

```
# Normalisasi data

def normalize(image, label):
  return image/255.0, label
train_data = train_data.map(normalize)
data_valid= valid_data.map(normalize)
```

```
for img, label in train_data.take(1):

  print(type(img),type(label))
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'> <class 'tensorflow.python.framework.ops.EagerTensor'>
```

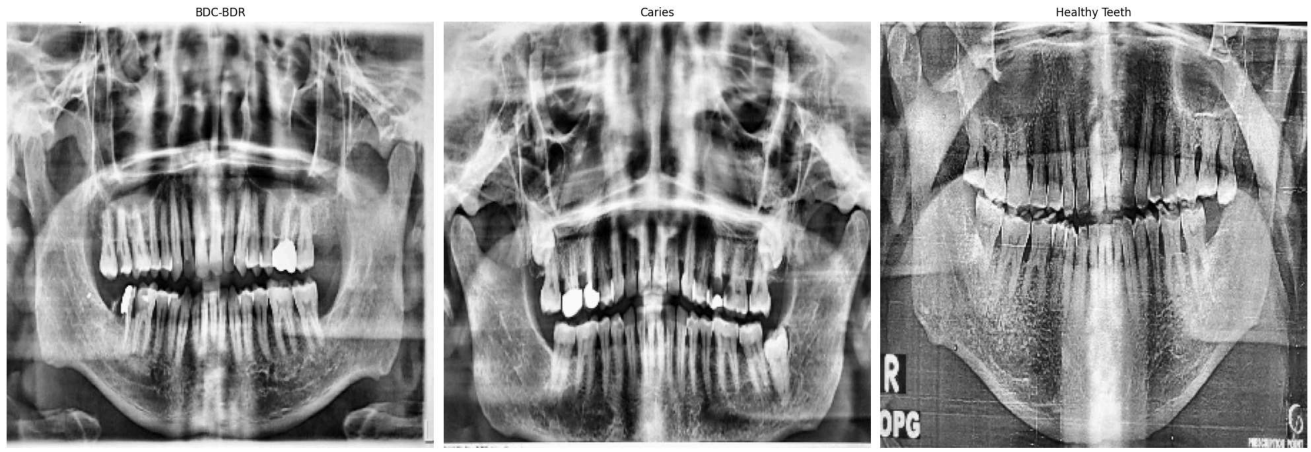
```
train_x=[]
train_y=[]
for image,label in train_data:
   train_x.append(image)
   train_y.append(label)
train_x = tf.concat(train_x, axis=0)
train_y = tf.concat(train_y, axis=0)
```

```
val_x=[]
val_y=[]
for image,label in train_data:
   val_x.append(image)
   val_y.append(label)
val_x = tf.concat(val_x, axis=0)
val_y = tf.concat(val_y, axis=0)
```

```
#one hot encode

num_classes = 3
train_y = tf.keras.utils.to_categorical(train_y, num_classes=num_classes)
val_y = tf.keras.utils.to_categorical(val_y, num_classes=num_classes)
```

```
import matplotlib.pyplot as plt
import numpy as np

# Class labels
class_labels = [
    'BDC-BDR', 'Caries', 'Healthy Teeth'
]

# Create the figure and axes
fig, axes = plt.subplots(1, 3, figsize=(20, 8))

# Loop untuk setiap kelas
for idx, (ax, kelas) in enumerate(zip(axes.flat, range(len(class_labels)))):
    # Pilih gambar dari kelas yang dipilih
    gambar_kelas = train_x[train_y[:, kelas] == 1]

    # Plotkan gambar pertama dari kelas yang dipilih
    ax.imshow(gambar_kelas[0], cmap='gray')
    ax.set_title(class_labels[kelas])
    ax.axis('off')
```

```
# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```



Eksekusiiiii

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras import layers, models, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler
from tensorflow.keras.preprocessing.image import ImageDataGenerator

input_shape = (299, 299, 3)
num_classes = 3

def build_inceptionv3(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    base = InceptionV3(weights='imagenet', include_top=False,
                       input_shape=input_shape)(inputs)
    base.trainable = False

    x = layers.GlobalAveragePooling2D()(base)
    x = layers.Dense(1024, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs, outputs)
    model.compile(
        optimizer=Adam(1e-4),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

def scheduler(epoch, lr):
    return lr * 0.1 if epoch > 15 else lr

callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    LearningRateScheduler(scheduler)
]
```

```
model = build_inceptionv3(input_shape, num_classes)

history = model.fit(
    datagen.flow(train_x, train_y, batch_size=16),
    epochs=40,
    validation_data=(val_x, val_y),
    callbacks=callbacks
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ord
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDatase
  self._warn_if_super_not_called()
Epoch 1/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 147s 6s/step - accuracy: 0.3752 - loss: 1.1781 - val_accuracy: 0.4155 - val_loss: 1.1457 - learning_
Epoch 2/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 431ms/step - accuracy: 0.5218 - loss: 1.0570 - val_accuracy: 0.4734 - val_loss: 1.0386 - learning
Epoch 3/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 495ms/step - accuracy: 0.5082 - loss: 0.9562 - val_accuracy: 0.5024 - val_loss: 1.0006 - learning
Epoch 4/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 433ms/step - accuracy: 0.5948 - loss: 0.9369 - val_accuracy: 0.5604 - val_loss: 0.9955 - learning
Epoch 5/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 430ms/step - accuracy: 0.6310 - loss: 0.8735 - val_accuracy: 0.4444 - val_loss: 1.2875 - learning
Epoch 6/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 461ms/step - accuracy: 0.6596 - loss: 0.7278 - val_accuracy: 0.4976 - val_loss: 1.1279 - learning
Epoch 7/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 420ms/step - accuracy: 0.6631 - loss: 0.7210 - val_accuracy: 0.5121 - val_loss: 1.2977 - learning
Epoch 8/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 7s 520ms/step - accuracy: 0.7887 - loss: 0.5283 - val_accuracy: 0.6377 - val_loss: 0.7859 - learning
Epoch 9/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 422ms/step - accuracy: 0.7735 - loss: 0.5214 - val_accuracy: 0.5942 - val_loss: 1.1046 - learning
Epoch 10/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 7s 505ms/step - accuracy: 0.8271 - loss: 0.4706 - val_accuracy: 0.6232 - val_loss: 1.0851 - learning
Epoch 11/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 430ms/step - accuracy: 0.8370 - loss: 0.4460 - val_accuracy: 0.6812 - val_loss: 0.9249 - learning
Epoch 12/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 7s 500ms/step - accuracy: 0.7915 - loss: 0.5343 - val_accuracy: 0.6522 - val_loss: 0.9442 - learning
Epoch 13/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 423ms/step - accuracy: 0.8859 - loss: 0.2742 - val_accuracy: 0.6618 - val_loss: 1.1318 - learning
Epoch 14/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 455ms/step - accuracy: 0.9121 - loss: 0.2328 - val_accuracy: 0.6570 - val_loss: 1.0105 - learning
Epoch 15/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 454ms/step - accuracy: 0.8998 - loss: 0.2661 - val_accuracy: 0.7295 - val_loss: 0.7374 - learning
Epoch 16/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 449ms/step - accuracy: 0.9080 - loss: 0.2106 - val_accuracy: 0.8261 - val_loss: 0.4439 - learning
Epoch 17/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 10s 443ms/step - accuracy: 0.9000 - loss: 0.2623 - val_accuracy: 0.8599 - val_loss: 0.3658 - learni
Epoch 18/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 7s 524ms/step - accuracy: 0.9066 - loss: 0.2629 - val_accuracy: 0.8841 - val_loss: 0.3194 - learning
Epoch 19/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 440ms/step - accuracy: 0.9332 - loss: 0.1809 - val_accuracy: 0.8889 - val_loss: 0.2924 - learning
Epoch 20/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 7s 533ms/step - accuracy: 0.9043 - loss: 0.2836 - val_accuracy: 0.8986 - val_loss: 0.2701 - learning
Epoch 21/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 437ms/step - accuracy: 0.9762 - loss: 0.1309 - val_accuracy: 0.9082 - val_loss: 0.2503 - learning
Epoch 22/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 7s 524ms/step - accuracy: 0.9153 - loss: 0.1947 - val_accuracy: 0.9179 - val_loss: 0.2357 - learning
Epoch 23/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 444ms/step - accuracy: 0.8869 - loss: 0.2992 - val_accuracy: 0.9227 - val_loss: 0.2206 - learning
Epoch 24/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 497ms/step - accuracy: 0.8807 - loss: 0.2220 - val_accuracy: 0.9275 - val_loss: 0.2051 - learning
Epoch 25/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 10s 440ms/step - accuracy: 0.9322 - loss: 0.2034 - val_accuracy: 0.9275 - val_loss: 0.1932 - learni
Epoch 26/40
13/13 ━━━━━━━━━━━━━━━━━━━━ 7s 534ms/step - accuracy: 0.8867 - loss: 0.2964 - val_accuracy: 0.9324 - val_loss: 0.1820 - learning
Epoch 27/40
```

```
jumlah_variasi = len(datagen.flow(train_x, train_y, batch_size=16))
print(jumlah_variasi)
```

```
13
```

```
import matplotlib.pyplot as plt

# Evaluasi model pada data validasi
val_loss, val_acc = model.evaluate(val_x, val_y, verbose=0)
print(f"Validation Loss: {val_loss:.4f}")
print(f"Validation Accuracy: {val_acc:.4f}")

# Ringkasan arsitektur model
```

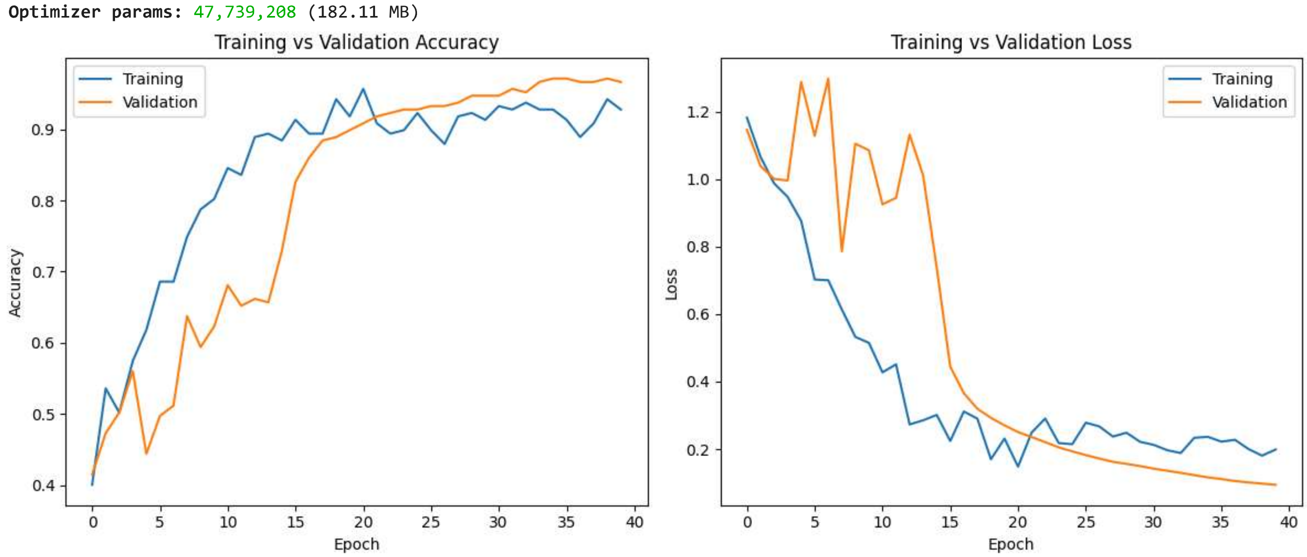
```
model.summary()

# Visualisasi akurasi dan loss
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training')
plt.plot(history.history['val_loss'], label='Validation')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Validation Loss: 0.0941
Validation Accuracy: 0.9662
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 299, 299, 3)	0
inception_v3 (Functional)	(None, 8, 8, 2048)	21,802,784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 1024)	2,098,176
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 3)	3,075

Total params: 71,643,243 (273.30 MB)
Trainable params: 23,869,603 (91.06 MB)
Non-trainable params: 34,432 (134.50 KB)
Optimizer params: 47,739,208 (182.11 MB)




```
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

class_names = ['BDC-BDR', 'Caries', 'Healthy Teeth']
```

```
# Prediksi pada data validasi
y_pred = model.predict(val_x, verbose=0)
y_pred_cls = np.argmax(y_pred, axis=1)
y_true = np.argmax(val_y, axis=1)

# Classification report
print("Classification Report:")
print(classification_report(y_true, y_pred_cls, target_names=class_names))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred_cls)

plt.figure(figsize=(8, 6))
plt.imshow(cm)
plt.colorbar()
plt.xticks(range(len(class_names)), class_names, rotation=45)
plt.yticks(range(len(class_names)), class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")

for i in range(len(class_names)):
    for j in range(len(class_names)):
        plt.text(j, i, cm[i, j], ha="center", va="center")

plt.tight_layout()
plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

      BDC-BDR       1.00      0.96      0.98        46
       Caries       0.95      0.96      0.96        81
Healthy Teeth       0.96      0.97      0.97        80

     accuracy                           0.97       207
    macro avg       0.97      0.96      0.97       207
 weighted avg       0.97      0.97      0.97       207
```