LAMPIRAN

Lampiran 1. Source code kombinasi max pooling dan global average pooling.

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

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

# Paths ke folder benign_resize dan malignant_resize
benign_resize_path = "/content/drive/MyDrive/Drive File
Sharing/skin-cancer-dataset/benign_resize"
malignant_resize_path = "/content/drive/MyDrive/Drive File
Sharing/skin-cancer-dataset/malignant_resize"

# List semua gambar dan buat array kosong untuk menyimpan data
X_data = []
y_data = []

# Fungsi untuk memuat dan menormalkan gambar
def load_and_normalize_image(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_normalized = img_rgb.astype('float32') / 255.0
    return img_rgb, img_normalized

# Muat gambar benign
for file in os.listdir(benign_resize_path):
    img_path = os.path.join(benign_resize_path, file)
    img_rgb, img_normalized = load_and_normalize_image(img_path)
    X_data.append(img_normalized)  # Simpan gambar yang telah
dinormalisasi
    y_data.append(0)  # Label untuk benign

# Muat gambar malignant
for file in os.listdir(malignant_resize_path):
    img_path = os.path.join(malignant_resize_path, file)
    img_rgb, img_normalized = load_and_normalize_image(img_path)
```

```
    X_data.append(img_normalized)  # Simpan gambar yang telah
dinormalisasi
    y_data.append(1)  # Label untuk malignant

# Konversi list ke array numpy
X_data = np.array(X_data)
y_data = np.array(y_data)

# Tampilkan beberapa informasi tentang data
print(f"Total gambar: {len(X_data)}")
print(f"Dimensi gambar: {X_data.shape[1:]}")
print(f"Jumlah label benign: {sum(y == 0 for y in y_data)}")
print(f"Jumlah label malignant: {sum(y == 1 for y in y_data)}")

# Fungsi untuk menampilkan distribusi nilai piksel
def plot_pixel_distributions(original_image, normalized_image,
title):
    # Flattening the images to 1D arrays
    original_pixels = original_image.reshape(-1)
    normalized_pixels = normalized_image.reshape(-1)

    plt.figure(figsize=(12, 6))

    # Distribusi nilai piksel sebelum normalisasi
    plt.subplot(1, 2, 1)
    plt.hist(original_pixels, bins=50, color='blue', alpha=0.7)
    plt.title(f'{title} - Sebelum Normalisasi')
    plt.xlabel('Nilai Piksel')
    plt.ylabel('Frekuensi')

    # Distribusi nilai piksel setelah normalisasi
    plt.subplot(1, 2, 2)
    plt.hist(normalized_pixels, bins=50, color='green',
alpha=0.7)
    plt.title(f'{title} - Setelah Normalisasi')
    plt.xlabel('Nilai Piksel')
    plt.ylabel('Frekuensi')

    plt.tight_layout()
    plt.show()

# Ambil satu gambar benign dan satu gambar malignant untuk
visualisasi
```

```
benign_sample_path = os.path.join(benign_resize_path,
os.listdir(benign_resize_path)[0])
malignant_sample_path = os.path.join(malignant_resize_path,
os.listdir(malignant_resize_path)[0])

benign_rgb, benign_normalized =
load_and_normalize_image(benign_sample_path)
malignant_rgb, malignant_normalized =
load_and_normalize_image(malignant_sample_path)

# Visualisasikan distribusi nilai piksel
plot_pixel_distributions(benign_rgb, benign_normalized, "Benign
Sample")
plot_pixel_distributions(malignant_rgb, malignant_normalized,
"Malignant Sample")
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import
ImageDataGenerator

# Membuat objek ImageDataGenerator untuk data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    fill_mode='nearest'
)

# Fungsi untuk menampilkan gambar asli dan hasil augmentasi
def augment_and_display(image_path, title):
    # Load gambar
    img = cv2.imread(image_path)

    # Cek jika gambar berhasil dibaca
    if img is None:
        print(f"Error: Tidak bisa membaca gambar di
{image_path}")
        return

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    img_rgb = np.expand_dims(img_rgb, axis=0)  # Perluasan
dimensi untuk generator

    # Men-generate augmented images
    augmented_images = datagen.flow(img_rgb, batch_size=1)

    plt.figure(figsize=(12, 6))

    # Gambar asli
    plt.subplot(1, 5, 1)
    plt.imshow(img_rgb[0].astype('uint8'))
    plt.title("Original")
    plt.axis("off")

    # Menampilkan 4 gambar hasil augmentasi
    for i in range(4):
        aug_img = next(augmented_images)[0]  # Augmentasi gambar
        plt.subplot(1, 5, i + 2)
        plt.imshow(aug_img.astype('uint8'))  # Konversi ke uint8
untuk ditampilkan
        plt.title(f"Augmented {i+1}")
        plt.axis("off")

    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

# Path gambar benign dan malignant
benign_sample_path = os.path.join(benign_resize_path,
os.listdir(benign_resize_path)[3])
malignant_sample_path = os.path.join(malignant_resize_path,
os.listdir(malignant_resize_path)[4])

# Tampilkan augmentasi untuk gambar benign dan malignant
augment_and_display(benign_sample_path, "Benign Augmentation
Example")
augment_and_display(malignant_sample_path, "Malignant
Augmentation Example")
```

```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import StratifiedKFold

def create_model():
    model = Sequential()

    # Regularisasi L2
    weight_decay = 0.0001

    # Blok 1
    model.add(Conv2D(32, (3, 3), padding='same',
activation='relu', kernel_regularizer=l2(weight_decay),
input_shape=(256, 256, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (3, 3), padding='same',
activation='relu', kernel_regularizer=l2(weight_decay)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    # Blok 2
    model.add(Conv2D(64, (3, 3), padding='same',
activation='relu', kernel_regularizer=l2(weight_decay)))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), padding='same',
activation='relu', kernel_regularizer=l2(weight_decay)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.3))

    # Blok 3
    model.add(Conv2D(128, (3, 3), padding='same',
activation='relu', kernel_regularizer=l2(weight_decay)))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3, 3), padding='same',
activation='relu', kernel_regularizer=l2(weight_decay)))
    model.add(BatchNormalization())
```

```
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.4))

    # Global Average Pooling
    model.add(GlobalAveragePooling2D())

    # Fully Connected Layer
    model.add(Dense(256, activation='relu',
kernel_regularizer=l2(weight_decay)))
    model.add(Dropout(0.5))

    # Output Layer (Binary classification)
    model.add(Dense(1, activation='sigmoid'))

    # Optimizer
    optimizer = Adam(learning_rate=0.0005)

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

    return model

model = create_model()

model.summary()
```

```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix
from tensorflow.keras.callbacks import ReduceLROnPlateau,
EarlyStopping, ModelCheckpoint

results_dir = '/content/drive/My Drive/Drive File Sharing/skin-
cancer-dataset/kfold_comb_results'
os.makedirs(results_dir, exist_ok=True)

# K-Fold Cross Validation
k = 5
kf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)
```

```
accuracies = []
precisions = []
recalls = []
f1s = []
confusion_matrices = []
history_list = []

# Tentukan path direktori untuk benign dan malignant
benign_resize_path = "/content/drive/MyDrive/Drive File
Sharing/skin-cancer-dataset/benign_resize"
malignant_resize_path = "/content/drive/MyDrive/Drive File
Sharing/skin-cancer-dataset/malignant_resize"

# Ambil nama file gambar dalam direktori masing-masing
benign_files = os.listdir(benign_resize_path)
malignant_files = os.listdir(malignant_resize_path)

# Gabungkan nama file dan labelnya
X_filenames = benign_files + malignant_files
y_labels = [0] * len(benign_files) + [1] *
len(malignant_files)  # 0 untuk benign, 1 untuk malignant

# K-fold cross-validation
for fold, (train_index, val_index) in
enumerate(kf.split(X_filenames, y_labels)):
    fold_dir = os.path.join(results_dir, f'fold_{fold+1}')

    # Cek apakah hasil fold ini sudah ada
    if os.path.exists(os.path.join(fold_dir, 'metrics.npz')):
        print(f"Fold {fold+1} sudah selesai, melanjutkan ke fold
berikutnya.")
        continue

    os.makedirs(fold_dir, exist_ok=True)

    X_train_fold, X_valid_fold =
np.array(X_filenames)[train_index],
np.array(X_filenames)[val_index]
    y_train_fold, y_valid_fold = np.array(y_labels)[train_index],
np.array(y_labels)[val_index]

    # Proses data train
    X_train_fold_resized = [
```

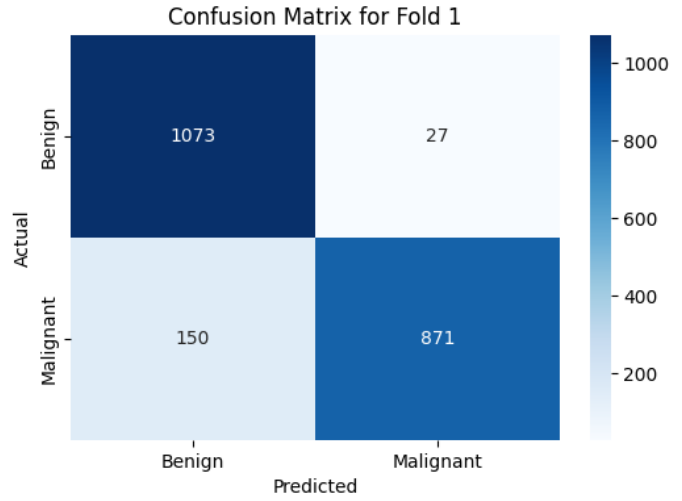
```
        load_and_normalize_image(os.path.join(benign_resize_path
if y == 0 else malignant_resize_path, x))[1]
        for x, y in zip(X_train_fold, y_train_fold)
    ]
    X_train_fold_resized = np.array(X_train_fold_resized)

    # Proses data validasi
    X_valid_fold_resized = [
        load_and_normalize_image(os.path.join(benign_resize_path
if y == 0 else malignant_resize_path, x))[1]
        for x, y in zip(X_valid_fold, y_valid_fold)
    ]
    X_valid_fold_resized = np.array(X_valid_fold_resized)

    # Model dan callbacks
    model = create_model()
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=10, min_lr=0.00001)
    early_stopping = EarlyStopping(monitor='val_loss',
patience=40, restore_best_weights=True, verbose=1)

    checkpoint = ModelCheckpoint(os.path.join(fold_dir,
'best_model.keras'),
                                 monitor='val_loss',
                                 save_best_only=True,
                                 mode='min',
                                 verbose=1)

    augmented_images = datagen.flow(X_train_fold_resized,
y_train_fold, batch_size=64)

    # Pelatihan model
    history = model.fit(augmented_images, epochs=50,
validation_data=(X_valid_fold_resized, y_valid_fold),
                        callbacks=[reduce_lr, early_stopping],
verbose=2)
    history_list.append(history)

    # Evaluasi model
    y_pred = (model.predict(X_valid_fold_resized) >
0.5).astype("int32")
    accuracy = accuracy_score(y_valid_fold, y_pred)
    precision = precision_score(y_valid_fold, y_pred,
average='binary')
```

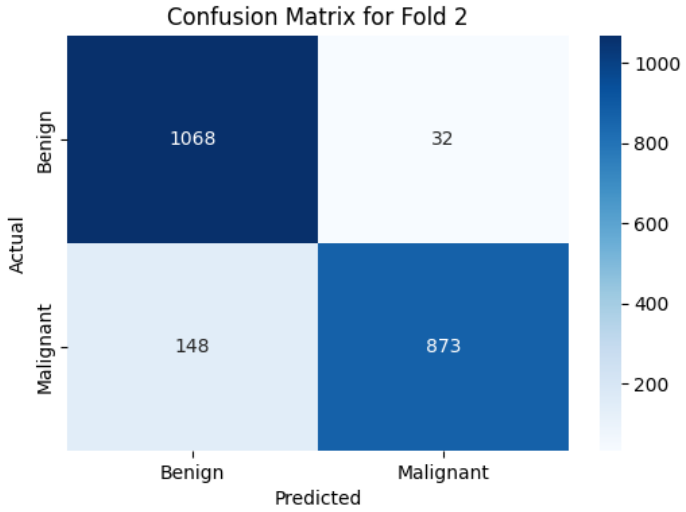
```
    recall = recall_score(y_valid_fold, y_pred, average='binary')
    f1 = f1_score(y_valid_fold, y_pred, average='binary')
    cm = confusion_matrix(y_valid_fold, y_pred)
    confusion_matrices.append(cm)
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1s.append(f1)

    # Save results for this fold
    fold_dir = os.path.join(results_dir, f'fold_{fold+1}')
    os.makedirs(fold_dir, exist_ok=True)
    np.savez(os.path.join(fold_dir, 'metrics.npz'),
accuracy=accuracy, precision=precision, recall=recall, f1=f1)
    np.savez(os.path.join(fold_dir, 'confusion_matrix.npz'),
confusion_matrix=cm)

    print(f"Fold {fold+1} Accuracy: {accuracy}")
    print(f"Fold {fold+1} Precision: {precision}")
    print(f"Fold {fold+1} Recall: {recall}")
    print(f"Fold {fold+1} F1 Score: {f1}\n")

print("Rata-rata Hasil K-Fold:")
print(f"Average Accuracy: {np.mean(accuracies)}")
print(f"Average Precision: {np.mean(precisions)}")
print(f"Average Recall: {np.mean(recalls)}")
print(f"Average F1 Score: {np.mean(f1s)}")

# Visualize confusion matrix for each fold
for i, cm in enumerate(confusion_matrices):
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Benign', 'Malignant'], yticklabels=['Benign',
'Malignant'])
    plt.title(f'Confusion Matrix for Fold {i+1}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.savefig(os.path.join(results_dir,
f'confusion_matrix_fold_{i+1}.png'))
    plt.close()
```

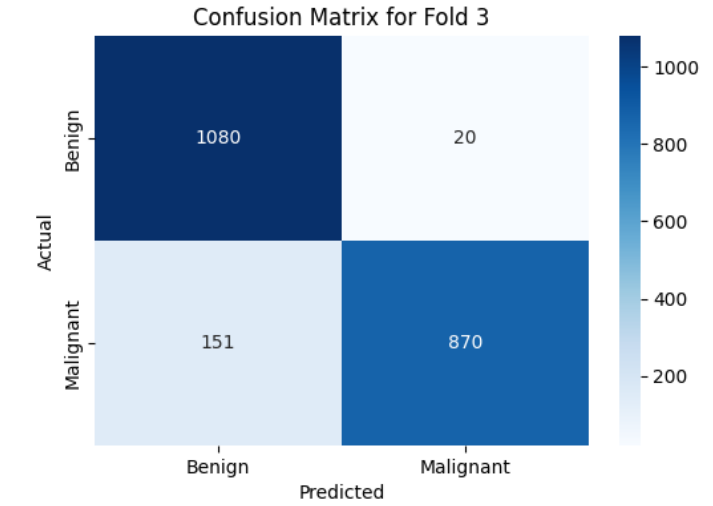
```
import matplotlib.pyplot as plt
```

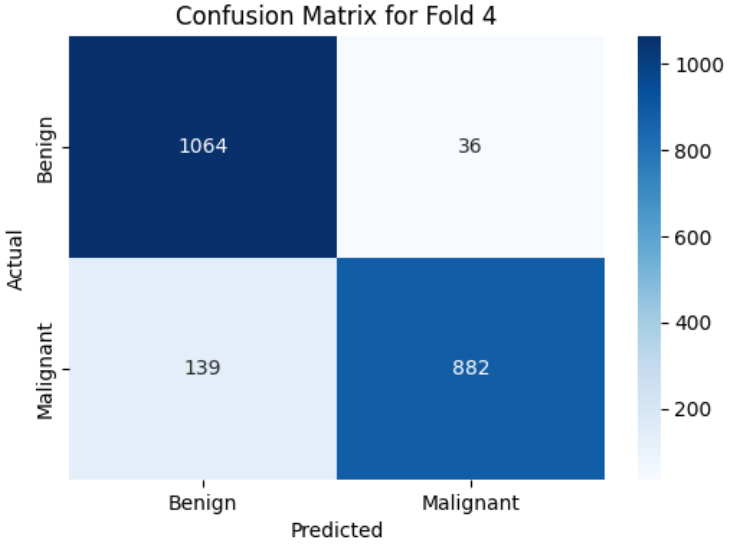
```
# Visualisasi training & validation accuracy dan loss
for i, hist in enumerate(history_list):
    # Ambil nilai accuracy dan loss dari history
    acc = hist.history['accuracy']
    val_acc = hist.history['val_accuracy']
    loss = hist.history['loss']
    val_loss = hist.history['val_loss']

    # Plot training & validation accuracy
    plt.figure(figsize=(12, 6))

    # Plot accuracy
    plt.subplot(1, 2, 1)
    plt.plot(acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title(f'Fold {i+1} - Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title(f'Fold {i+1} - Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

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

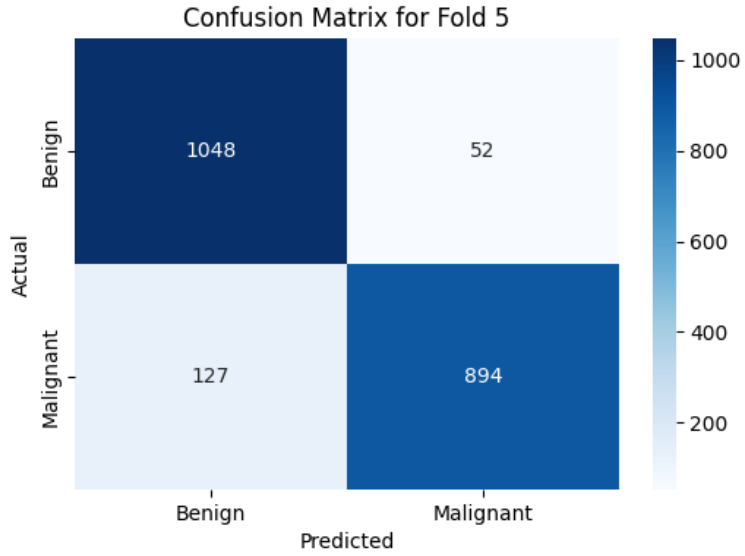
Lampiran 2. Visualisasi confusion matrix fold 1 sampai fold 5 kombinasi max pooling dan global average pooling.






Confusion Matrix for Fold 3
Benign
Malignant
Actual
1080
20
151
870
Benign
Malignant
Predicted
1000
800
600
400
200


Confusion Matrix for Fold 4
Benign
Malignant
Actual
1064
36
139
882
Benign
Malignant
Predicted
1000
800
600
400
200


Confusion Matrix for Fold 5
1048
52
127
894
Benign
Malignant
Actual
Benign
Malignant
Predicted
1000
800
600
400
200