LAMPIRAN

Lampiran 1 Code Program

1. Import Drive

```
# ============================================================
# 1. Mount Google Drive
# ============================================================
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# ============================================================
# 2. Konfigurasi Path dan Hyperparameter
# ============================================================
# RAW_DIR disesuaikan dengan folder hasil ekstraksi sebelumnya
RAW_DIR  = '/content/drive/MyDrive/Colab Notebooks/dataset/'
SAVE_DIR = '/content/drive/MyDrive/Colab Notebooks/hasil'
CHECKPOINT_FILE = os.path.join(SAVE_DIR, 'training_checkpoint.json')

# Path lokal Colab
AUG_DIR      = '/content/drive/MyDrive/Colab Notebooks/dataset_augmented'
TEMP_LABELS  = '/content/drive/MyDrive/Colab Notebooks/temp_labels'
YOLO_DIR     = '/content/drive/MyDrive/Colab Notebooks/yolo_dataset'
RESULTS_DIR  = '/content/drive/MyDrive/Colab Notebooks/comparison_results'
os.makedirs(RESULTS_DIR, exist_ok=True)

# ---- Deteksi Device & Optimasi CPU ----
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
IS_GPU = torch.cuda.is_available()

# ---- Hyperparameter Optimasi Cepat ----
EPOCHS       = 50
IMGSZ        = 128
BATCH_SIZE   = 16
SEED         = 42
N_FOLDS      = 5
TEST_RATIO   = 0.15
LR           = 0.001
EARLY_STOP_PATIENCE = 3
VALID_EXTS   = ('.jpg', '.jpeg', '.png')

def set_seed(seed: int = SEED):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)

set_seed(SEED)

# Verifikasi RAW_DIR
if not os.path.exists(RAW_DIR):
    # Mencoba folder dataset alternatif jika 'datasetbaru' ada di dalam
subfolder
    alt_path = '/content/drive/MyDrive/Colab Notebooks/dataset'
    if os.path.exists(alt_path):
        RAW_DIR = alt_path
    else:
        raise FileNotFoundError(f"ERROR: Folder {RAW_DIR} tidak ditemukan!")

CLASSES = sorted([d for d in os.listdir(RAW_DIR) if
os.path.isdir(os.path.join(RAW_DIR, d))])
print(f"Device: {device} | Batch Size: {BATCH_SIZE} | Resolution: {IMGSZ}")
print(f"Kelas: {CLASSES}")
```

2. Augmentasi Data

```
augment_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=20),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
```

```
    transforms.RandomAffine(degrees=0, translate=(0.05, 0.05), scale=(0.95,
1.05)),
])

if os.path.exists(AUG_DIR):
    shutil.rmtree(AUG_DIR)
os.makedirs(AUG_DIR, exist_ok=True)

set_seed(SEED)

# Tetapkan target menjadi 820 sesuai permintaan
FIXED_TARGET = 820
augmentation_log = {}

for cls in CLASSES:
    raw_cls_dir = os.path.join(RAW_DIR, cls)
    aug_cls_dir = os.path.join(AUG_DIR, cls)
    os.makedirs(aug_cls_dir, exist_ok=True)

    # Baca gambar asli
    images = [f for f in os.listdir(raw_cls_dir) if
f.lower().endswith(VALID_EXTS)]
    current_count = len(images)

    if current_count == 0:
        print(f"PERINGATAN: kelas '{cls}' tidak punya gambar valid, dilewati.")
        continue

    # Copy gambar asli ke lokal
    for img_name in images:
        shutil.copy(os.path.join(raw_cls_dir, img_name),
os.path.join(aug_cls_dir, img_name))

    # Augmentasi sampai mencapai target 820
    needed = FIXED_TARGET - current_count
    if needed > 0:
        print(f"Proses {cls}: {current_count} asli -> target {FIXED_TARGET}
(tambah {needed})...")
        for i in tqdm(range(needed), desc=cls):
            src_name = images[i % current_count]
            with Image.open(os.path.join(raw_cls_dir, src_name)) as img:
                aug_img = augment_transform(img.convert('RGB'))
                base, ext = os.path.splitext(src_name)
                aug_img.save(os.path.join(aug_cls_dir, f"{base}_aug_{i}{ext}"))
    elif current_count > FIXED_TARGET:
        print(f"Kelas {cls} sudah memiliki {current_count} gambar (melebihi
target {FIXED_TARGET}). Tidak ada augmentasi tambahan.")
    else:
        print(f"Kelas {cls} pas {FIXED_TARGET} gambar.")

    final_count = len([f for f in os.listdir(aug_cls_dir) if
f.lower().endswith(VALID_EXTS)])
    augmentation_log[cls] = {
        "raw_count": current_count,
        "target_fixed": FIXED_TARGET,
        "final_count": final_count,
    }

print("\n=== Ringkasan Augmentasi (Target: 820) ===")
aug_summary_df = pd.DataFrame(augmentation_log).T
display(aug_summary_df)
aug_summary_df.to_csv(os.path.join(RESULTS_DIR, 'augmentation_summary.csv'))
```

3. Auto-Labeling YOLO & Persiapan Split (Hold-out Test + 5-Fold CV)

```
# ============================================================
# 1. Auto-Labeling YOLO (Otsu threshold + morphological closing)
# ============================================================
if os.path.exists(TEMP_LABELS):
    shutil.rmtree(TEMP_LABELS)
os.makedirs(TEMP_LABELS, exist_ok=True)

def auto_label_bbox(img_path):
```

```
    img = cv2.imread(img_path)
    if img is None: return None
    h, w = img.shape[:2]
    img_area = h * w
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    kernel = np.ones((7, 7), np.uint8)

    best_bbox, best_area_ratio = None, None
    for flag in (cv2.THRESH_BINARY_INV, cv2.THRESH_BINARY):
        _, thresh = cv2.threshold(blur, 0, 255, flag + cv2.THRESH_OTSU)
        closed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=2)
        contours, _ = cv2.findContours(closed, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        if not contours: continue
        max_cnt = max(contours, key=cv2.contourArea)
        area_ratio = cv2.contourArea(max_cnt) / img_area
        if area_ratio < 0.01 or area_ratio > 0.98: continue
        if best_area_ratio is None or abs(area_ratio - 0.4) <
abs(best_area_ratio - 0.4):
            x, y, bw, bh = cv2.boundingRect(max_cnt)
            best_bbox = ((x + bw / 2) / w, (y + bh / 2) / h, bw / w, bh / h)
            best_area_ratio = area_ratio
    return best_bbox

for idx, cls in enumerate(CLASSES):
    cls_dir = os.path.join(AUG_DIR, cls)
    out_dir = os.path.join(TEMP_LABELS, cls)
    os.makedirs(out_dir, exist_ok=True)
    files = [f for f in os.listdir(cls_dir) if f.lower().endswith(VALID_EXTS)]
    for img_file in tqdm(files, desc=f"Membuat Label YOLO {cls}"):
        bbox = auto_label_bbox(os.path.join(cls_dir, img_file))
        if bbox:
            xc, yc, bw, bh = bbox
            with open(os.path.join(out_dir, os.path.splitext(img_file)[0] +
'.txt'), 'w') as f:
                f.write(f"{idx} {xc:.6f} {yc:.6f} {bw:.6f} {bh:.6f}\n")

# ============================================================
# 2. Persiapan DataFrame & Penyeragaman Ukuran Fold
# ============================================================
all_samples = []
for idx, cls in enumerate(CLASSES):
    img_dir = os.path.join(AUG_DIR, cls)
    images = sorted([f for f in os.listdir(img_dir) if
f.lower().endswith(VALID_EXTS)])
    for fname in images:
        all_samples.append({"filename": fname, "class": cls, "class idx": idx})

samples_df = pd.DataFrame(all_samples)
set_seed(SEED)

# Split Test Set 15%
test_mask = np.zeros(len(samples_df), dtype=bool)
for cls in CLASSES:
    cls_idx = samples_df.index[samples_df['class'] == cls].to_numpy().copy()
    np.random.shuffle(cls_idx)
    n_test = int(len(cls_idx) * TEST_RATIO)
    test_mask[cls_idx[:n_test]] = True

test_df = samples_df[test_mask].reset_index(drop=True)
cv_df = samples_df[~test_mask].reset_index(drop=True)

# --- PENYESUAIAN UKURAN: Pastikan total data CV = 2788 (558 * 5) ---
# Agar setiap fold genap val=558, train=2230
TARGET_CV_TOTAL = 558 * 5 # 2790, namun user minta train 2230 + val 558 = 2788
total
# Jika cv_df saat ini tidak pas 2788, kita sesuaikan sedikit
if len(cv_df) > 2788:
    cv_df = cv_df.sample(n=2788, random_state=SEED).reset_index(drop=True)

# Buat K-Fold manual agar ukuran eksak sama di tiap fold
indices = np.arange(len(cv_df))
np.random.shuffle(indices)
```

```
fold_indices = []
VAL_SIZE = 558
TRAIN_SIZE = 2230

for i in range(N_FOLDS):
    val_idx = indices[i * VAL_SIZE : (i + 1) * VAL_SIZE]
    train_idx = np.setdiff1d(indices, val_idx)
    fold_indices.append((train_idx, val_idx))

print(f"Dataset disesuaikan untuk 5-Fold CV (Total: {len(cv_df)} gambar)")
for i, (tr_idx, val_idx) in enumerate(fold_indices):
    print(f"  Fold {i+1}: train={len(tr_idx)} | val={len(val_idx)}")

def populate_split_dir(df_subset, split_name, base_round_dir):
    cnn_round_dir = os.path.join(base_round_dir, 'cnn')
    yolo_round_dir = os.path.join(base_round_dir, 'yolo')
    for cls in CLASSES:
        os.makedirs(f"{cnn_round_dir}/{split_name}/{cls}", exist_ok=True)
        os.makedirs(f"{yolo_round_dir}/images/{split_name}/{cls}",
exist_ok=True)
        os.makedirs(f"{yolo_round_dir}/labels/{split_name}/{cls}",
exist_ok=True)

    for _, row in df_subset.iterrows():
        cls, fname = row['class'], row['filename']
        src img = os.path.join(AUG DIR, cls, fname)
        shutil.copy(src_img, f"{cnn_round_dir}/{split_name}/{cls}/{fname}")
        shutil.copy(src_img,
f"{yolo_round_dir}/images/{split_name}/{cls}/{fname}")
        lbl_name = os.path.splitext(fname)[0] + '.txt'
        src_lbl = os.path.join(TEMP_LABELS, cls, lbl_name)
        if os.path.exists(src_lbl):
            shutil.copy(src_lbl,
f"{yolo_round_dir}/labels/{split_name}/{cls}/{lbl_name}")
    return cnn_round_dir, yolo_round_dir
```

4. Model CNN

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
from tqdm.auto import tqdm
import os
import shutil

# ============================================================
# 0 Definisi Arsitektur PureCNN
# ============================================================
class PureCNN(nn.Module):
    def __init__(self, num_classes):
        super(PureCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d((1, 1)),
```

```
            nn.Flatten(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x

def evaluate_cnn(model, loader, device, criterion):
    model.eval()
    all_preds, all_labels = [], []
    loss_sum = 0.0
    with torch.no_grad():
        for imgs, labels in loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            loss_sum += criterion(outputs, labels).item()
            preds = outputs.argmax(dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    accuracy = accuracy_score(all_labels, all_preds)
    precision = precision_score(all_labels, all_preds, average='macro',
zero_division=0)
    recall = recall_score(all_labels, all_preds, average='macro',
zero_division=0)
    f1 = f1_score(all_labels, all_preds, average='macro', zero_division=0)
    cm = confusion_matrix(all_labels, all_preds)

    return {
        "loss": loss_sum / max(len(loader), 1),
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1": f1,
        "confusion_matrix": cm
    }

# --- Transformasi Ringan ---
train_tf = transforms.Compose([
    transforms.Resize((IMGSZ, IMGSZ)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

cnn_fold_results = []

for fold_i in range(N_FOLDS):
    tr_idx, val_idx = fold_indices[fold_i]
    print(f"\n{'='*50}\n[CNN] STARTING FOLD {fold_i+1}/{N_FOLDS}\n{'='*50}")

    fold_dir = f'/content/drive/MyDrive/Colab
Notebooks/hasil/cnn_fold_{fold_i+1}'
    if os.path.exists(fold_dir): shutil.rmtree(fold_dir)
    os.makedirs(fold_dir, exist_ok=True)

    cnn_round_dir = os.path.join(fold_dir, 'cnn')

    # Copy data sesuai split fold
    for split_name, indices_subset in {'train': tr_idx, 'val': val_idx}.items():
        df_subset = cv_df.iloc[indices_subset]
        for cls in CLASSES: os.makedirs(f"{cnn_round_dir}/{split_name}/{cls}",
exist_ok=True)
        for _, row in df_subset.iterrows():
            shutil.copy(os.path.join(AUG_DIR, row['class'], row['filename']),
f"{cnn_round_dir}/{split_name}/{row['class']}/{row['filename']}")

    train_loader = DataLoader(datasets.ImageFolder(f'{cnn_round_dir}/train',
train_tf), batch_size=BATCH_SIZE, shuffle=True)
```

```
    val_loader = DataLoader(datasets.ImageFolder(f'{cnn_round_dir}/val',
train_tf), batch_size=BATCH_SIZE)

    model = PureCNN(len(CLASSES)).to(device)
    optimizer = optim.Adam(model.parameters(), lr=LR)
    criterion = nn.CrossEntropyLoss()

    best_val_f1 = 0.0
    epochs_no_improve = 0

    fold_train_losses = [] # Track training loss per epoch
    fold_val_losses = []   # Track validation loss per epoch
    fold_val_accs = []     # Track validation accuracy per epoch

    for ep in range(EPOCHS):
        model.train()
        train_loss = 0.0
        pbar = tqdm(train_loader, desc=f"Fold {fold_i+1} Ep {ep+1}/{EPOCHS}",
leave=False)
        for imgs, labels in pbar:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
            pbar.set_postfix({'loss': f"{loss.item():.4f}"})

        avg_train_loss = train_loss / len(train_loader)
        fold_train_losses.append(avg_train_loss)

        m = evaluate_cnn(model, val_loader, device, criterion)
        fold_val_losses.append(m['loss'])
        fold_val_accs.append(m['accuracy'])

        print(f"Ep {ep+1:02d} | Val Loss: {m['loss']:.3f} | Acc:
{m['accuracy']:.3f} | F1: {m['f1']:.3f} | Prec: {m['precision']:.3f} | Rec:
{m['recall']:.3f}")

        if m['f1'] > best_val_f1:
            best_val_f1 = m['f1']
            epochs_no_improve = 0
            torch.save(model.state_dict(), os.path.join(RESULTS_DIR,
f'cnn_fold{fold_i+1}_best.pth'))
        else:
            epochs_no_improve += 1
            if epochs_no_improve >= EARLY_STOP_PATIENCE:
                print(f"Early stopping dipicu pada epoch {ep+1}")
                break

    model.load_state_dict(torch.load(os.path.join(RESULTS_DIR,
f'cnn_fold{fold_i+1}_best.pth')))
    final_m = evaluate_cnn(model, val_loader, device, criterion)
    final_m['train_losses'] = fold_train_losses # Add train loss history
    final_m['val_losses'] = fold_val_losses     # Add validation loss history
    final_m['val_accuracies'] = fold_val_accs   # Add validation accuracy
history
    cnn_fold_results.append(final_m)
    shutil.rmtree(fold_dir)

print("\n[COMPLETE] CNN Training selesai untuk semua fold.")
```

5. Model YOLOv8s

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
from tqdm.auto import tqdm
```

```
import os
import shutil

# ============================================================
# 🎯 Training YOLOv8s — Progress per Fold
# ============================================================
yolo_fold_results = []

for fold_i in range(N_FOLDS):
    tr_idx, val_idx = fold_indices[fold_i]
    print(f"\n{'='*50}\n[YOLOv8s] STARTING FOLD {fold_i+1}/{N_FOLDS}\n{'='*50}")

    fold_dir = f'/content/drive/MyDrive/Colab
Notebooks/hasil/yolo_fold_{fold_i+1}'
    if os.path.exists(fold_dir): shutil.rmtree(fold_dir)

    _, yolo_round_dir = populate_split_dir(cv_df.iloc[tr_idx], 'train',
fold_dir)
    populate_split_dir(cv_df.iloc[val_idx], 'val', fold_dir)

    yaml_path = os.path.join(fold_dir, 'data.yaml')
    with open(yaml_path, 'w') as f:
        json.dump({'path': os.path.abspath(yolo_round_dir), 'train':
'images/train', 'val': 'images/val',
                   'nc': len(CLASSES), 'names': {i: c for i, c in
enumerate(CLASSES)}}, f)

    # Menggunakan yolov8s.pt (Small) sesuai permintaan
    model = YOLO('yolov8s.pt')
    results = model.train(
        data=yaml_path, epochs=EPOCHS, imgsz=IMGSZ, batch=BATCH_SIZE,
        project=RESULTS_DIR, name=f'yolo_fold_{fold_i+1}', exist_ok=True,
verbose=True,
        patience=EARLY_STOP_PATIENCE # Menambahkan early stopping untuk YOLO
    )

    # Validasi metrik deteksi & proxy klasifikasi
    metrics = model.val(data=yaml_path, imgsz=IMGSZ, batch=BATCH_SIZE)
    p, r = float(metrics.box.mp), float(metrics.box.mr)
    f1 = (2 * p * r / (p + r)) if (p + r) > 0 else 0.0

    print(f"Fold {fold_i+1} Result -> mAP50: {metrics.box.map50:.4f} | F1-Proxy:
{f1:.4f}")

    yolo_fold_results.append({"fold": fold_i + 1, "f1": f1, "mAP50":
metrics.box.map50})
    shutil.rmtree(fold_dir)

print("\n[COMPLETE] YOLOv8s Training selesai.")
```

6. Hold-Out Test Set

```
# ============================================================
# Siapkan Test Set (hold-out, 15%) untuk evaluasi akhir
# ============================================================
TEST_ROOT = '/content/drive/MyDrive/Colab Notebooks/hasil/test_eval'
if os.path.exists(TEST_ROOT): shutil.rmtree(TEST_ROOT)

# Definisi transformasi evaluasi
eval_tf = transforms.Compose([
    transforms.Resize((IMGSZ, IMGSZ)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

cnn_test_dir, yolo_test_dir = populate_split_dir(test_df, 'test', TEST_ROOT)

# ---- 1) Evaluasi CNN pada test set ----
cnn_df_res = pd.DataFrame(cnn_fold_results)
best_cnn_fold_idx = cnn_df_res['f1'].idxmax()
cnn_save_path = os.path.join(RESULTS_DIR,
f'cnn_fold{best_cnn_fold_idx+1}_best.pth')
```

```
best_cnn_model = PureCNN(len(CLASSES)).to(device)
if os.path.exists(cnn_save_path):
    best_cnn_model.load_state_dict(torch.load(cnn_save_path,
map_location=device))

test_set_cnn = datasets.ImageFolder(f'{cnn_test_dir}/test', eval_tf)
test_loader_cnn = DataLoader(test_set_cnn, batch_size=BATCH_SIZE, shuffle=False)

criterion = nn.CrossEntropyLoss()
cnn_test_metrics = evaluate_cnn(best_cnn_model, test_loader_cnn, device,
criterion)
cnn_param_count = sum(p.numel() for p in best_cnn_model.parameters())

# Kecepatan inferensi CNN
best_cnn_model.eval()
n_speed_samples = min(50, len(test_set_cnn))
with torch.no_grad():
    dummy = torch.randn(1, 3, IMGSZ, IMGSZ).to(device)
    _ = best_cnn_model(dummy) # Warm-up

    start = time.time()
    for i in range(n_speed_samples):
        img, _ = test_set_cnn[i]
        img = img.unsqueeze(0).to(device)
        _ = best_cnn_model(img)
    cnn_infer_time = time.time() - start
cnn_fps = n_speed_samples / cnn_infer_time if cnn_infer_time > 0 else 0

# ---- 2) Evaluasi YOLO pada test set ----
yolo_df_res = pd.DataFrame(yolo_fold_results)
best_yolo_fold_idx = yolo_df_res['f1'].idxmax()
best_yolo_fold = int(yolo_df_res.iloc[best_yolo_fold_idx]['fold'])
best_yolo_weights = os.path.join(RESULTS_DIR, f'yolo_fold_{best_yolo_fold}',
'weights', 'best.pt')

yolo_model = YOLO(best_yolo_weights)
yolo_param_count = sum(p.numel() for p in yolo_model.model.parameters())

# Kecepatan inferensi YOLO
sample_img_path = os.path.join(AUG_DIR, test_df.iloc[0]['class'],
test_df.iloc[0]['filename'])
_ = yolo_model(sample_img_path, verbose=False) # Warm-up

start_y = time.time()
for i in range(n_speed_samples):
    img_p = os.path.join(AUG_DIR, test_df.iloc[i]['class'],
test_df.iloc[i]['filename'])
      = yolo model(img p, verbose=False)
yolo_infer_time = time.time() - start_y
yolo_fps = n_speed_samples / yolo_infer_time

print(f"=== EVALUASI TEST SET SELESAI ===")
print(f"CNN - Acc: {cnn_test_metrics['accuracy']:.4f} | FPS: {cnn_fps:.1f}")
print(f"YOLO - F1: {yolo_df_res.iloc[best_yolo_fold_idx]['f1']:.4f} | FPS:
{yolo_fps:.1f}")
```