Full Preprocessing Script

```
import os
import numpy as np
import scipy.io as sio
from PIL import Image
from scipy.ndimage import gaussian_filter
from pathlib import Path
from tqdm import tqdm
import matplotlib.pyplot as plt
import csv


# ------------------- SETTINGS -------------------
SIGMA = 15  # Gaussian blur value for density map

DATASETS = [
    "data/part_A_final/train_data",
    "data/part_B_final/train_data"
]
# ------------------------------------------------


def generate_density_map(img_path, gt_path):
    """Generate a density map using Gaussian filtering."""
    img = np.array(Image.open(img_path))
    h, w = img.shape[0], img.shape[1]

    # Load GT .mat file
    mat = sio.loadmat(gt_path)
    points = mat["image_info"][0][0][0][0][0]

    density = np.zeros((h, w), dtype=np.float32)

    for p in points:
        x, y = int(p[0]), int(p[1])
        if x < w and y < h:
            density[y, x] = 1

    density = gaussian_filter(density, sigma=SIGMA)
    return density, points



def save_visual(img_path, density_map, points, save_path):
    """Generate visualization of image, GT points, and density map."""
    img = np.array(Image.open(img_path))

    gt_count = len(points)
    pred_count = density_map.sum()

    plt.figure(figsize=(15, 5))

    # Original Image
    plt.subplot(1, 3, 1)
    plt.imshow(img)
    plt.title(f"Original Image\nGT Count = {gt_count}")
    plt.axis("off")

    # GT Points on Image
    plt.subplot(1, 3, 2)
    plt.imshow(img)
    plt.scatter(points[:, 0], points[:, 1], s=5, c='red')
    plt.title("Ground Truth Points")
    plt.axis("off")

    # Density Map
    plt.subplot(1, 3, 3)
```

```
    plt.imshow(density_map, cmap='jet')
    plt.title(f"Density Map\nPredicted Count = {pred_count:.1f}")
    plt.axis("off")

    plt.tight_layout()
    plt.savefig(save_path, dpi=150)
    plt.close()



def append_csv_row(csv_path, row):
    """Append a row to CSV report, create header if needed."""
    write_header = not csv_path.exists()

    with open(csv_path, 'a', newline='') as f:
        writer = csv.writer(f)

        if write_header:
            writer.writerow(["Image Name", "GT Count", "Predicted Count", "Absolute Error", "Dataset"])

        writer.writerow(row)



def process_folder(base_dir):
    """Process each dataset: generate density maps, visuals, and CSV report."""
    base_dir = Path(base_dir)

    img_dir = base_dir / "images"
    gt_dir = base_dir / "ground_truth"
    dm_dir = base_dir / "density_maps"
    vis_dir = base_dir / "visualizations"
    report_path = base_dir / "report.csv"

    dm_dir.mkdir(exist_ok=True)
    vis_dir.mkdir(exist_ok=True)

    img_files = list(img_dir.glob("*.jpg"))

    print(f"\n Processing Dataset: {base_dir}")
    print(f" Total Images: {len(img_files)}")

    for img_path in tqdm(img_files):
        img_name = img_path.stem
        gt_path = gt_dir / f"GT_{img_name}.mat"

        if not gt_path.exists():
            print(f" Missing GT for {img_path.name}")
            continue

        # Generate density map & get GT points
        density_map, points = generate_density_map(img_path, gt_path)

        # Save density map as .npy
        np.save(dm_dir / f"{img_name}.npy", density_map)

        # Save visualization PNG
        save_visual(
            img_path,
            density_map,
            points,
            vis_dir / f"{img_name}_viz.png"
        )

        # Save metrics to CSV
        gt_count = len(points)
        pred_count = float(density_map.sum())
        abs_error = abs(gt_count - pred_count)

        append_csv_row(
            report_path,
            [img_name, gt_count, pred_count, abs_error, base_dir.name]
```

```
        )

    print(f" Completed: {base_dir}")
    print(f" CSV Report saved at: {report_path}")
    print(f" Visualizations saved in: {vis_dir}")
    print(f" Density maps saved in: {dm_dir}")



def main():
    """Run preprocessing on all datasets."""
    for dataset in DATASETS:
        process_folder(dataset)

    print("\n ALL DONE — Entire Dataset Preprocessed Successfully!")



if __name__ == "__main__":
    main()
```