DM-Count: Distribution Matching for Crowd Counting

Priyanshu Raj Jindal
Roll Number: 210787

October 22, 2024

1 Code Implementation Details

Run details are provided in the README of the GitHub repository. Other code implementation details are as follows:

1. **preprocess.py** - Handles large images from the UCF-QNRF and NWPU datasets by calling `preprocess_dataset_qnrf.py` and `preprocess_dataset_nwpu.py`.
2. **preprocess_dataset_qnrf.py**: Resizes images to a minimum of 512 pixels and a maximum of 2048 pixels (as per height or width (whichever is minimum)) while maintaining aspect ratio.
3. **preprocess_dataset_nwpu.py**: Resizes images to a minimum of 384 pixels and a maximum of 1920 pixels (height or width (whichever is minimum)) while preserving aspect ratio.
4. **train.py** - Supports command-line arguments for training across multiple datasets, acting as the user interface to start training with customizable parameters.
5. **datasets/crowd.py** - Defines dataset loading and processing for QNRF, NWPU, and ShanghaiTech using PyTorch's `Dataset` class. It handles random cropping, horizontal flipping, and generates discrete density maps based on ground truth. The transformations ensure resizing, normalization, and downsampling for model training.
6. **losses/ot_loss.py** - Implements an Optimal Transport (OT) loss function for density map estimation, computing the Wasserstein distance (OT distance) between predicted and ground-truth density maps using Sinkhorn iterations to calculate the optimal transport plan.
7. **losses/bregmann_pytorch.py** - Implements OT solvers using Sinkhorn's algorithm, including methods like `sinkhorn`, `sinkhorn_stabilized`, and `sinkhorn_epsilon_scaling`, to compute OT matrices between distributions while minimizing transport costs. Based on the paper "Sinkhorn Distances: Lightspeed Computation of Optimal Transport" by M. Cuturi.
8. **utils/log_utils.py**: Implements a simple logger to track the status of training and test runs.
9. **utils/pytorch_utils.py**: Provides utility functions for adjusting learning rate, saving model checkpoints (`Save_Handle`), tracking metrics (`AverageMeter`), setting trainable model layers, and counting the number of trainable parameters in a model.
10. **models.py** - Loads a pre-trained VGG-19 model with configuration 'E' (optional batch normalization), adds a custom regression layer for processing extracted features, and includes a final layer for density map prediction. The model input passes through feature layers, is upsampled, and then normalized in the output layer.
11. **train_helper.py** - Integrates all components, loading datasets, training models, validating for the best model, saving checkpoints, and computing losses (OT, MAE, and TV) to update weights. Validation measures MAE and MSE.
12. **test.py** - Evaluates a pre-trained VGG-19-based crowd counting model on a specified dataset, computes Mean Absolute Error (MAE) and Mean Squared Error (MSE) between predicted and actual crowd counts, and saves predicted density maps as images.

Notes:

1. MAE refers to Mean Absolute Error, and RMSE are the only evaluation metrics used throughout, integrated within the test and training processes as appropriate.

2 Comparison with Paper Results

Notes:

1. The paper does not specify the number of epochs per dataset. In the official implementation, UCF-QNRF trains for 1000 epochs. The best model is selected based on the 2×RMSE + MAE metric.
2. Due to computational constraints, training was stopped once results approached those in the paper. Similarly, the number of epochs for ShanghaiTech A and B isn't explicitly mentioned.

Table 1: Comparison of DM-Count (obtained) and DM-Count (paper) across datasets (100 Sinkhorn iterations).

Method	**UCF-QNRF**			**ShanghaiTech A**			**ShanghaiTech B**		
	MAE	**RMSE**	**Epochs**	**MAE**	**RMSE**	**Epochs**	**MAE**	**RMSE**	**Epochs**
DM-Count (obtained)	88.97	154.8	154	61.45	98.4	567	8.44	13.97	570
DM-Count (paper)	85.6	148.3	*	59.7	95.7	*	7.4	11.8	*

3 Ablation Studies

3.1 Effect of Sinkhorn Iterations

Table 2: Ablation study on the effect of Sinkhorn iterations on model performance.

Sinkhorn Iterations	**DM-Count (obtained)**			**DM-Count (paper)**		
	MAE	**RMSE**	**Epochs**	**MAE**	**RMSE**	**Epochs**
50	103.81	176.10	155	85.6	148.3	*
100	88.97	154.8	154	85.6	148.3	*
120	108.07	177.03	106	85.5	151.5	*

Notes:

1. Training was stopped early due to computational constraints. Although the obtained results are not identical, they show a trend suggesting that 100 Sinkhorn iterations are optimal.

4 Datasets

Table 3: Dataset Characteristics

Dataset	Total Images	Annotations	Crowd Range	Average Crowd	Availability	GDrive Link
UCF-QNRF	1,535	1.25M	49 - 12,865	815	Available	Link
ShanghaiTech A	482	241K	33 - 3,139	501	Available	Link
ShanghaiTech B	716	88K	9 - 578	123.6	Available	Link

5 Relevant Links

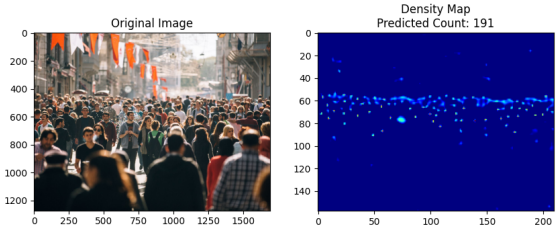
Description	**Link**
GitHub Repository	GitHub Repository Link
Pre-Trained Models for Results	Pre-Trained Model Link
Pre-Trained Models from Paper	Pre-Trained Models from Paper
Kaggle Notebook for Reproducibility	Kaggle Notebook Link

Table 4: Links to relevant resources

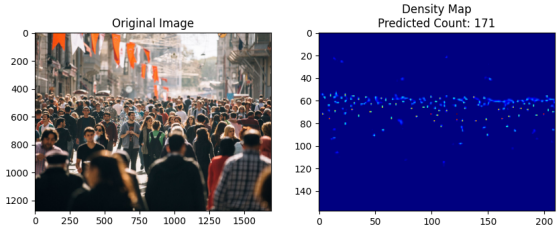
Note:

1. I have added "Hidangmayum Bebina" kaggle account as collaborator, still if it is not visible Kindly message me I will immediately make it public.

6 Example Run on Some Images with Results on Pre-Trained and Self-Trained Models

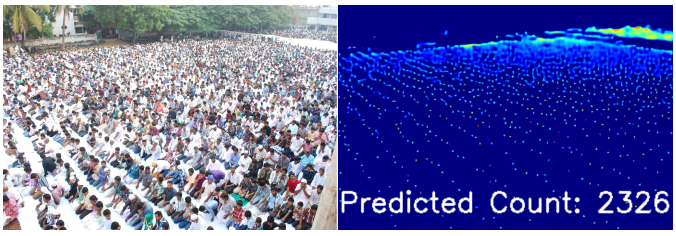


(a) Model Trained by Self

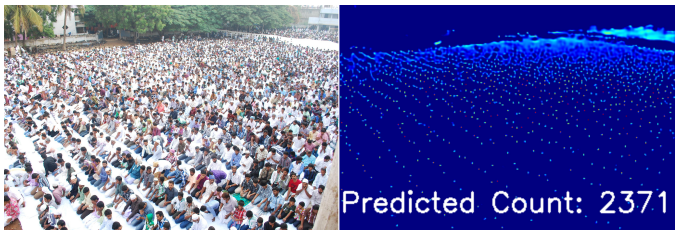


(b) Trained Model provided by authors

Figure 1: Comparison of Results from Self-Trained Model and Author-Provided Model



(a) Model Trained by Self



(b) Trained Model provided by authors

Figure 2: Comparison of Results from Self-Trained Model and Author-Provided Model