Enhanced Implementation of DecideNet for Crowd Counting Using the Mall Dataset

Priyanshu Kumar Bhushan

November 1, 2024

1 Introduction

Crowd counting is one of the computer vision tasks that estimates the number of people in a scene. The DecideNet model employs both detection and regression approaches with an attention mechanism to dynamically select the most appropriate estimation mode for each pixel based on crowd density. This report explains the updated DecideNet implementation with attention feature fusion and pre-trained backbone for improved accuracy on the Mall dataset.

2 Updated DetNet Formulation

Crowd density estimation at some selected points is done by means of DetNet in DecideNet. For an input image I_i, a DetNet produces a low-density region individual density map as D_{det}.

$$D_{\text{det}}(p \mid I_i) = \sum_{P \in P_{\text{det}}} N_{\text{det}}(p \mid \mu = P, \sigma^2) \tag{1}$$

where:

- D_{det} is the density map from detection,
- P_{det} represents identified head positions,
- N_{det} is a Gaussian function centered on P with variance σ^2.

3 Pre-trained Backbone Advantage

A pre-trained backbone like ResNet or MobileNet efficiently captures hierarchical features. Integration of this backbone has enhanced the ability of DetNet to identify subtle edges and textures toward the high head localization accuracy in low-density areas.

3.1 Mathematical Representation of the Pre-trained Backbone

Define a feature map computed by pre-trained backbone as follows related to input imageI_i, with,

$$F_{\text{backbone}} = f_{\text{backbone}}(I_i; \theta_{\text{pre-trained}}) \tag{2}$$

where $\theta_{\text{pre-trained}}$ denote learned weights for pre-training. The features F_{backbone} are consumed by downstream DetNet layers to produce the density map, using convolutional layer(s) as front end that acts as initial layers for the DetNet architecture in order to feed much richer features, improving at the same time detection and faster training.

4 Implementation Details

It is done using TensorFlow with the below steps :

- **Data Preparation:** Images are resized to 320×240, normalized, and split into training, validation, and test sets.
- **Backbone and Attention Fusion:**ResNet50 backbone extracts the hierarchical features. DetNet and RegNet branches use adaptive attention-guided ASKCFuse layers.

5 Attention-Guided Feature Fusion with ASKC-Fuse

The ASKCFuse uses local and global attention on features to adaptively fuse them:

- **Local Attention:**It utilizes two convolutional layers to make the model focus more accurately on fine-grained information
- **Global Attention:** It uses convolutional and global average pooling layers. These layers tend to gather more general information for larger patterns.

This combination does the final fusion with an added sigmoid layer in combination with local and global attentions, allowing scaling over feature maps through which model achieves weighted output across variances in crowd densities to improve adaptability.

6 Training and Evaluation

The model is trained with:

- **Optimizer:** Adam with a learning rate of 1×10^{-4}.
- **Loss Function:** Mean Squared Error (MSE).
- **Evaluation Metrics:** Mean Absolute Error (MAE) and Mean Squared Error (MSE).

7 Results

Quantitative and qualitative results are shown on the Mall dataset, which confirm the benefits of adaptive feature fusion in enhancing counting accuracy in the face of varying densities.

Model	MAE	MSE
DecideNet (combined with Attention Fusion)	1.07176	3.314796

Table 1: Quantitative results on the Mall dataset

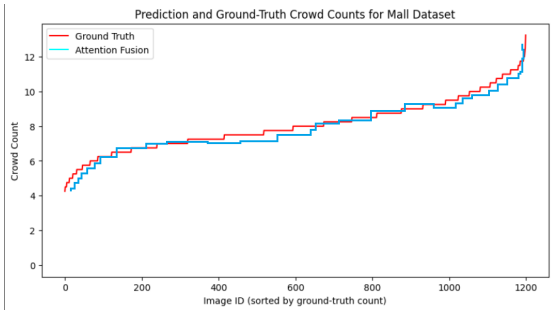

Figure 1: Ground Truth vs. Prediction plot

8 Conclusion

The updated DecideNet model demonstrates the advantage of using a pre-trained backbone and attention-guided fusion. The ASKCFuse layer enhances model performance by adaptively combining local and global features. This approach improves counting accuracy in varying density scenarios.

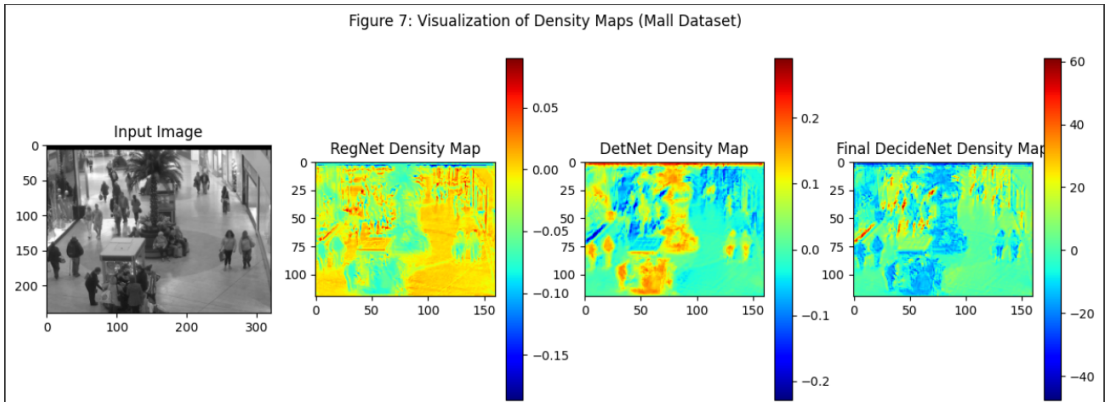

Figure 2: Visualization of Density Map

Model: "functional_16"

Layer (type)	Output Shape	Param #	Connected to
inputs_image (InputLayer)	(None, 240, 320, 1)	0	-
inputs_detection (InputLayer)	(None, 240, 320, 1)	0	-
grayscale_to_rgb_42 (GrayscaleToRGB)	(None, 240, 320, 3)	0	inputs_image[0][0]
grayscale_to_rgb_43 (GrayscaleToRGB)	(None, 240, 320, 3)	0	inputs_detection[0][0]
resnet50 (Functional)	(None, 8, 10, 2048)	23,587,712	grayscale_to_rgb_43[0… grayscale_to_rgb_42[0…
up_sampling2d_47 (UpSampling2D)	(None, 64, 80, 2048)	0	resnet50[0][0]
up_sampling2d_46 (UpSampling2D)	(None, 64, 80, 2048)	0	resnet50[0][0]
conv2d_192 (Conv2D)	(None, 64, 80, 32)	1,638,432	up_sampling2d_47[0][0]
conv2d_190 (Conv2D)	(None, 64, 80, 48)	2,457,648	up_sampling2d_46[0][0]
batch_normalization_43 (BatchNormalization)	(None, 64, 80, 32)	128	conv2d_192[0][0]
batch_normalization_42 (BatchNormalization)	(None, 64, 80, 48)	192	conv2d_190[0][0]
dropout_45 (Dropout)	(None, 64, 80, 32)	0	batch_normalization_4…
dropout_44 (Dropout)	(None, 64, 80, 48)	0	batch_normalization_4…
conv2d_193 (Conv2D)	(None, 64, 80, 1)	33	dropout_45[0][0]
conv2d_191 (Conv2D)	(None, 64, 80, 1)	49	dropout_44[0][0]
quality_net_22 (QualityNet)	(None, 64, 80, 1)	71,617	conv2d_193[0][0], conv2d_191[0][0], grayscale_to_rgb_42[0…
subtract_14 (Subtract)	(None, 64, 80, 1)	0	quality_net_22[0][0]
multiply_28 (Multiply)	(None, 64, 80, 1)	0	quality_net_22[0][0], conv2d_191[0][0]
multiply_29 (Multiply)	(None, 64, 80, 1)	0	subtract_14[0][0], conv2d_193[0][0]
add_14 (Add)	(None, 64, 80, 1)	0	multiply_28[0][0], multiply_29[0][0]
up_sampling2d_48 (UpSampling2D)	(None, 256, 320, 1)	0	add_14[0][0]
conv2d_198 (Conv2D)	(None, 256, 320, 1)	2	up_sampling2d_48[0][0]
resizing_25 (Resizing)	(None, 240, 320, 1)	0	conv2d_198[0][0]

Total params: 27,755,813 (105.88 MB)
Trainable params: 4,167,941 (15.90 MB)
Non-trainable params: 23,587,872 (89.98 MB)

Figure 3: Model Architecture