Cell Counting

Member1 :- Pulkit Agrawal(180050081)
Member2 :- Vipul Agarwal(180050119)

Introduction to the Problem

This project is based on the problem of counting specific type of objects in a digital image.

Specifically, if we talk about cell counting, it is a process of calculating the number of cells in a particular type of cell-images (type of the cell is known).

This is a major problem because it is time consuming and prone to errors due to fatigue of human annotators.

We would like to obtain the count of objects in an input image ‘I’ being given only a few training examples with point annotations of each object. The objects to count are often very small, and the overall image very large. Because counting is labor-intensive, there are often few labeled images in practice.

Strategy

We did not use the classical approach of edge detectors because of the difficulty in getting correct object boundaries in case of overlapping images.

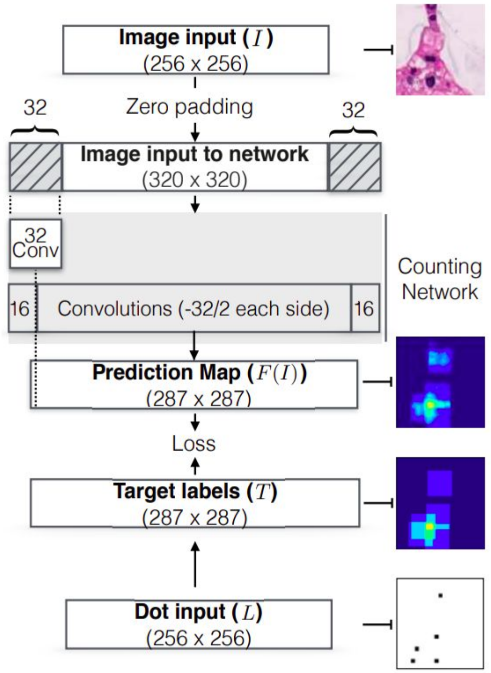
There is also a method to calculate the density map of the image and take its sum to get the total count.

Instead of directly using the density based model, we constructed a model based on redundant counts in order to average over errors. We merged the idea of networks that count everything in their receptive field with the density map of objects using fully convolutional processing.

Our model is based on the receptive field of a smaller regression network which predicts a count of objects that exist inside this frame.

By processing the image in a fully convolutional way each pixel is going to be accounted for some number of times, the number of windows which include it, which is the size of each window.

Algorithm



The diagram on the left shows the basic overview of our algorithm.

We process the image ‘I’ with this network in a fully convolutional way to produce a matrix ‘F(I)’ that represents the counts of objects for a specific receptive field ‘r×r’ of a sub-network that performs the counting.

‘I’ -> Input Image, ‘L’ -> Point annotated Image used for training
‘T’ -> Target Image extracted from ‘L’, ‘F(I)’ -> Predicted Image

Constructing Target Image ‘T’ from Annotated Point Image ‘L’

The target image can be constructed from a point-annotated map L, where each object is annotated by a single pixel. Let R(x, y) be the set of pixel locations in the receptive field corresponding to T[x, y]. Then we can construct the target image ‘T’:

$$T[x, y] = \sum_{X(x',y') \in R(x,y)} L[x', y']$$

(Here T[x, y] is the sum of cells contained in a region the size of the r × r receptive field.)

Then we built a CNN model that yields a fully convolutional network output image larger than the original input. Each pixel in the output will represent the count of targets in that receptive field.

For calculating the loss between the target image ‘T’ and predicted image ‘F(I)’, we used L1 Loss: $\min \|F(I) - T\|_1$

The above loss is a surrogate objective to the real count that we want. We intentionally count each cell multiple times in order to average over possible errors

# redundant counts = $(r / s)^2$ (r-> receptive field size, s-> stride length)

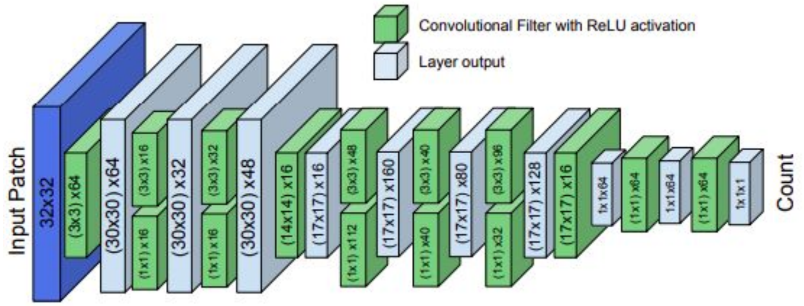
In order to recover the true count we divide the sum of all pixels by the number of redundant counts.

# true counts = $(\sum_{x,y} F(I)[x , y]$ / # redundant counts)

Implementation

Firstly we need to get the target image ‘T’ from the point annotated image ‘L’. For this we first padded the image ‘L’ with pad length 32. Then by iterating through the width and height of the padded image, we constructed receptive fields and calculated the number of points in that region.

We used two types of kernels: “Square” and “Gaussian”. In square kernel, we used a receptive field of size 32*32, each point inside this window has equal contribution in the count. Whereas in the Gaussian kernel, we construct a multivariate gaussian around the points where $L[x,y] \neq 0$ and depending on the distance of the point from this mean, its contribution is noted. ($L[x,y] = 0$ represents those points where there are no cells present)



The model on left shows an overview of the CNN model that we implemented.

At the core of the model we perform 1x1 (pad 0) and 3x3 (pad 1) convolutions at multiple layers without reducing the size of the tensor.

After every convolution a Leaky ReLU activation is applied

We perform down sampling in two locations using large filters to greatly reduce the size of the tensor. A necessity in allowing the model to train is utilizing Batch Normalization layers after every convolution.

Firstly we tried to implement the CNN model using keras. But as keras tends to be more user friendly, it could not provide much modifications that we needed in our training and testing batch size.

Thus we shifted to Pytorch and constructed our model using various parameters and batch size.

To speed up the training process, we trained our model in Google Colab so that we could use the GPU provided by it.

We used the same dataset as mentioned in the paper. There were 3 different types of data:

1. VGG Cells
2. Adipocyte Cells
3. MBM Cells

The size of images in each dataset was different and also the number of images were different. Thus each type required individual pre-processing but we trained all of them on the same model.

Experimental Design

VGG cells Dataset

There are 200 images in this dataset each of size 256*256. We split these 200 images into 3 parts- 40 images for training, 40 images for validation and 120 images for testing. The batch size used is 2 and the number of epochs are 300. The learning rate was taken to be 0.001.

Adipocyte cells Dataset

There are 200 images in this dataset each of size 150*150. We split these into 3 parts- 38 images for training, 38 images for validation and 116 images for testing. (We have removed a few images for whom we felt that the marked annotations did not match with the original image)

The batch size was 2 and the model was run on 1000 epochs. Learning Rate was taken to be 0.005.

MBM Dataset

There were 44 images in this category each of size 600*600. The data was split as- 21 for training, 5 for validation and 18 for testing. It was run on 200 epochs with batch size = 2. Learning Rate was 0.001.

Results

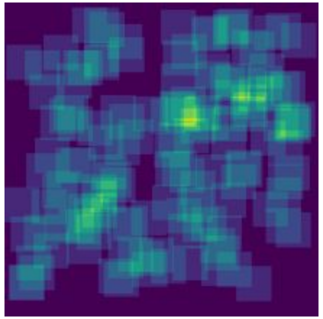
(More Details about the Result can be found in the report at https://github.com/Pulkit-Marlin/Cell-Counting/blob/master/Cell%20Counting%20-%20Report.pdf)

VGG Cells

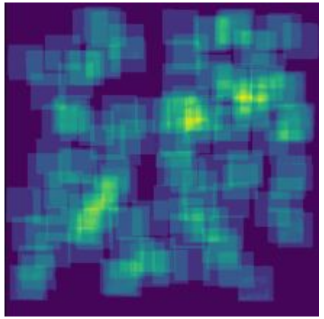
Mean Difference in Counts = 2.83

(square kernel)

Target Image T



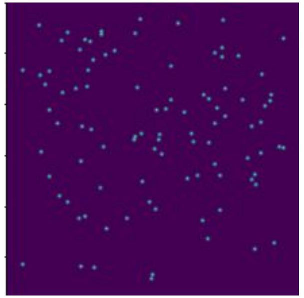
Predicted Image F(I)



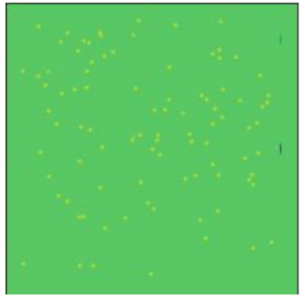
Mean Difference in Counts = 23.43

(gaussian kernel)

Target Image T



Predicted Image F(I)



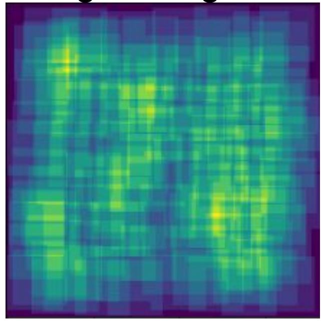
Mean Difference in Counts is taken over the whole test dataset

Adipocyte Cells

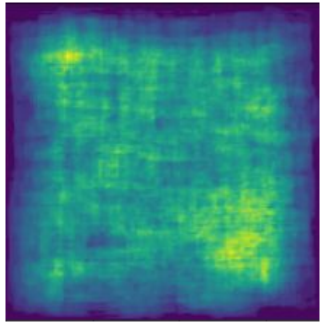
Mean Difference in Counts = 13.85

(square kernel)

Target Image T



Predicted Image F(I)

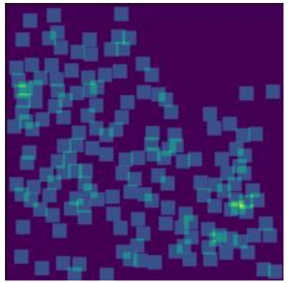


MBM Cells

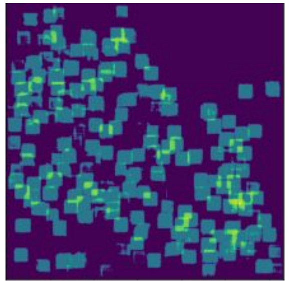
Mean Difference in Counts = 9.55

(square kernel)

Target Image T



Predicted Image F(I)



As mentioned in the paper, we found out that the method based on redundant counts outperforms the existing methods like object detection and density map.

By increasing the stride we can reduce double counting until there is none. As we increase the stride to equal the window size where no redundant counting is occurring, the accuracy is reduced.

The run-time of this algorithm is not trivial. We explored models with less parameters and found they could not achieve the same performance. Shorter models (fewer layers) or narrower models (less filters per layer) tended to not have enough representational power to count correctly. Making the network wider would cause the model to overfit.

While analysing the predicted images (‘F(I)’), we found that although the model predicts count very accurately, its prediction about the x,y coordinates of the cell is not very precise. It just gives a rough idea. Thus, viewing the predicted count map can localize where the detection came from but not to a specific coordinate.

For many applications accurate counting is more important than exact localization.

Conclusions

We implemented two kernels - “Square” and “Gaussian”, and Square kernel outperformed for all the 3 datasets. We feel this is due to the fact that although we assign maximum value in gaussian kernel at the point where cell is present, but it also gives a less value to nearby points where cell might not be present. Thus, it leads to some errors in our counting, giving less accuracy.

This approach is promising for tasks with different sizes of objects which have complicated structure. However, the method has some limitations. The count map can be used for localization but it cannot easily provide x, y locations of objects.