Cover page for answers.pdf
CSE512 Spring 2021 - Machine Learning - Homework 6

Your Name: Kavali shravya

Solar ID: 114404103

NetID email address: Shravya.kavali@stonybrook.edu

Names of people whom you discussed the homework with: None

Question 1

1.1

a) x = tensor([[0.5730, 0.9125],
[0.3517, 0.4131],
[0.9458, 0.9147]])

b) y = tensor([[1., 1.],
[1., 1.],
[1., 1.]])
size = torch.Size([3, 2])

c) out = tensor([[1.5730, 1.9125],
[1.3517, 1.4131],
[1.9458, 1.9147]])

y = tensor([[1.5730, 1.9125],
[1.3517, 1.4131],
[1.9458, 1.9147]])

d)

→ Constructed a randomly initialised numpy array x of size (3, 2) and dtype float.

x = [[0.69335441 0.53018293]
[0.3007379 0.49057505]
[0.83535258 0.48070578]]

type(x) = <class 'numpy.ndarray'>

→ Convert numpy to a tensor using the pytorch function from_numpy()

x_tensor = tensor([[0.6934, 0.5302],
[0.3007, 0.4906],
[0.8354, 0.4807]], dtype=torch.float64)

type(x_tensor) = <class 'torch.Tensor'>

→ Convert it back to a numpy array using the function .numpy().
x_numpy = [[0.69335441 0.53018293]
 [0.3007379 0.49057505]
 [0.83535258 0.48070578]]

type(x_numpy) = <class 'numpy.ndarray'>

1.2

a) x = tensor([[0.5591, 0.3711],
[0.8781, 0.1263],
[0.6891, 0.4883]], requires_grad=True)

b) y = tensor([[5.6913, 3.8113],
[8.8810, 1.3630],
[6.9910, 4.9826]], grad_fn=<AddBackward0>)
out = tensor(8.8810, grad_fn=<MaxBackward1>)

y.grad_fn = <AddBackward0 object at 0×7f3d014671d0>
out.grad_fn = <MaxBackward1 object at 0×7f3d01467cd0>

each tensor variable will keep track of what operation was performed on it in the grad function attribute. Which is used to compute derivatives during back propagation.

y.grad_fn is returning AddBackward0 as we have preformed addition operation. 0 indicates 1 derivation operation during back-propagation.

out.grad_fn is returning maxBackward1 as we have preformed max operation, 1 indicates 2 derivation operation during back-propagation.

c) x.grad = tensor([[0., 0.],
[10., 0.],
[0., 0.]])

d) y.grad_fn = None

out.grad_fn = None.

We got None here as we have used torch.no_grad, because of which we are not tracking any operations performed in between.

1.3

a)

net = Network(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=256, out_features=150, bias=True)
  (fc2): Linear(in_features=150, out_features=70, bias=True)
  (fc3): Linear(in_features=70, out_features=10, bias=True)
)

Number of parameters = 12

b)

output = tensor([[0.0163, 0.1032, 0.1138, 0.0212, 0.0403, 0.0747, -0.0817, 0.0160,
         0.0832, 0.0853]], grad_fn=<AddmmBackward>)

output.size() = torch.Size([1, 10])

c)

bias in **cov1** layer in network after back propagation = tensor([-0.0061, -0.0193, 0.0000, -0.0115, -0.0051, -0.0088, -0.0010, -0.0006, -0.0039, -0.0064, -0.0092, 0.0106, 0.0010, 0.0009, 0.0036, -0.0026, 0.0033, -0.0063, 0.0005, -0.0126, -0.0004, 0.0107, -0.0114, -0.0063, -0.0034, 0.0002, 0.0036, 0.0024, -0.0002, 0.0086, -0.0043, -0.0012])

bias in **fc2** layer in network after back propagation = tensor([0.0000, 0.0000, -0.0485, 0.0420, 0.1040, 0.0000, -0.0049, -0.0154,-0.0460, -0.0162, 0.0000, 0.0549, 0.0000, 0.0144, 0.0000, -0.0420, 0.0000, 0.0350, 0.0000, -0.0724, -0.1165, 0.0345, -0.0292, 0.0093, -0.0958, 0.0082, 0.0000, 0.0000, -0.0314, 0.0000, 0.0000, 0.0429, 0.0000, 0.0241, -0.0426, -0.0288, 0.0162, -0.0091,

-0.0066, 0.0000, -0.0383, 0.0000, 0.0393, -0.0257, 0.0000, 0.0000,
-0.0322, 0.0000, 0.0171, -0.1158, 0.0000, -0.0319, 0.0006, -0.0254, 0.0000,
-0.0669, 0.0000, -0.0380, 0.0000, -0.0329, 0.0000, 0.0019, 0.0312, 0.0000,
0.0000, 0.0629, -0.1191, 0.0000, 0.0000, 0.0000])

1.4

Accuracy of the network on the all test images: 62 %

Accuracy for class plane is: 74.8 %
Accuracy for class car   is: 76.5 %
Accuracy for class bird is: 44.5 %
Accuracy for class cat   is: 52.5 %
Accuracy for class deer is: 55.4 %
Accuracy for class dog   is: 36.2 %
Accuracy for class frog is: 78.2 %
Accuracy for class horse is: 57.2 %
Accuracy for class ship is: 78.5 %
Accuracy for class truck is: 74.9 %

1.5

a)

Accuracy of the network on the all test images: 28 %

Accuracy for class airplane is: 17.0 %
Accuracy for class automobile is: 9.9 %
Accuracy for class bird is: 16.9 %
Accuracy for class cat   is: 57.0 %
Accuracy for class deer is: 6.0 %
Accuracy for class dog   is: 0.0 %
Accuracy for class frog is: 32.7 %
Accuracy for class horse is: 42.7 %
Accuracy for class ship is: 70.0 %
Accuracy for class truck is: 31.7 %

b)

Accuracy of the network on the all test images: 27 %

Accuracy for class airplane is: 30.7 %
Accuracy for class automobile is: 17.2 %
Accuracy for class bird is: 39.9 %
Accuracy for class cat is: 26.9 %
Accuracy for class deer is: 9.5 %
Accuracy for class dog is: 11.5 %
Accuracy for class frog is: 44.6 %
Accuracy for class horse is: 33.8 %
Accuracy for class ship is: 28.3 %
Accuracy for class truck is: 36.9 %

Question 2

2.1 - Summary of Few shot counting task

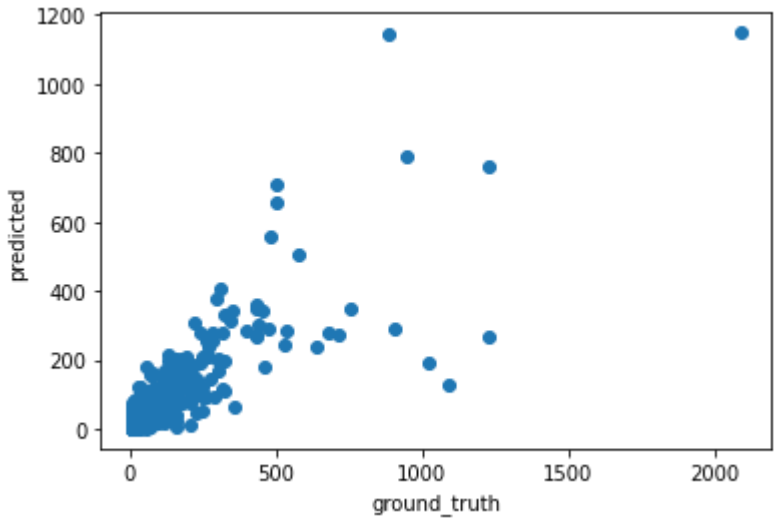
The primary objective of Few-Shot counting is to count the total number of novel class objects in any given image, as well as a few annotated instances in the same image. For this task, the dataset used in the paper consists of 6000 images with 147 object categories. The neural network used is FamNet(Few Shot Adaptation and Matching Network), which essentially has 2 modules a feature extraction module and a density prediction module. Both components can categorise novel items at test time, but we can increase the accuracy of counting objects and density maps with test time adaptation. The feature extraction module, in FamNet, ImageNet pre-trained network is used to handle a wide range of categories. The density prediction module, in FamNet, features obtained from feature extraction are not used instead the various correlation maps between the exemplar features and image features are used. The main idea of test time adaptation is to take advantage of the information provided by the exemplar bounding boxes' locations. min-count loss and perturbation losses are two losses that are combined and used to calculate the loss. In the test time adaptation technique, bounding boxes given for an image are taken to adapt the counting network, with a few gradient descent updates, where the

gradients are calculated using two loss functions that maximise the use of the exemplar positions.

2.2

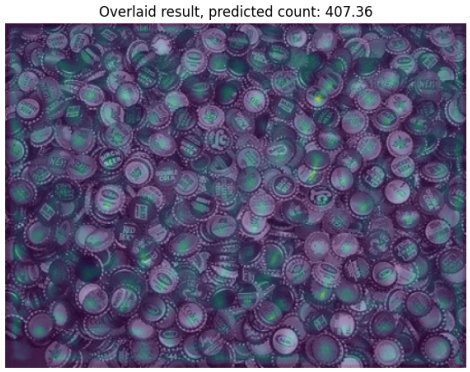
Mean Absolute Error (MAE): 24.32

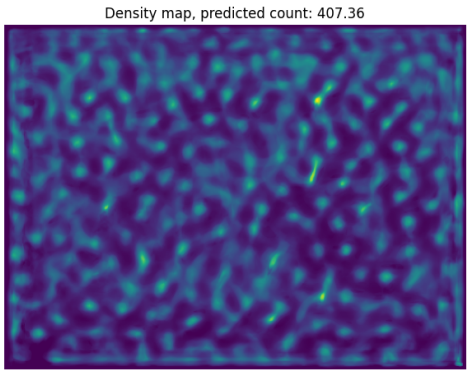
Root Mean Squared Error (RMSE): 70.94

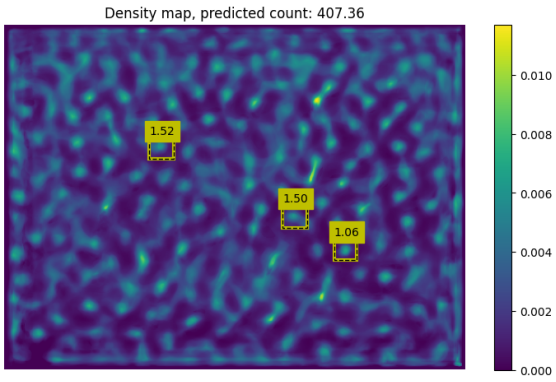


images with highest over count error :

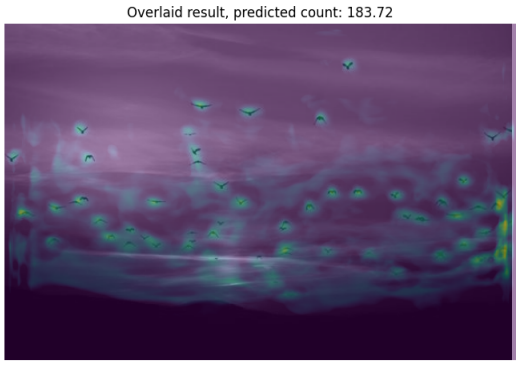
error = 94.35775756835938

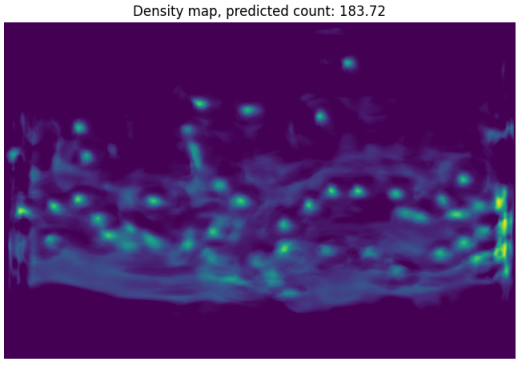


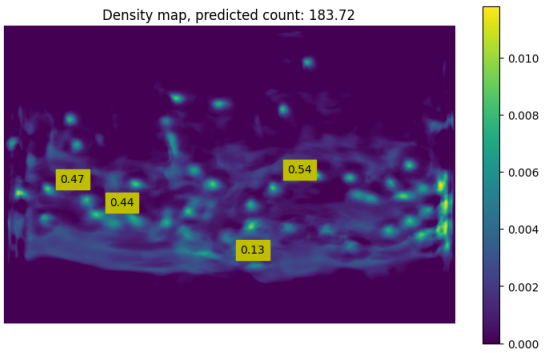




error = 123.72071838378906





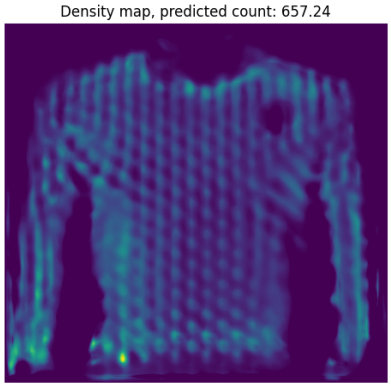


error = 156.2415771484375

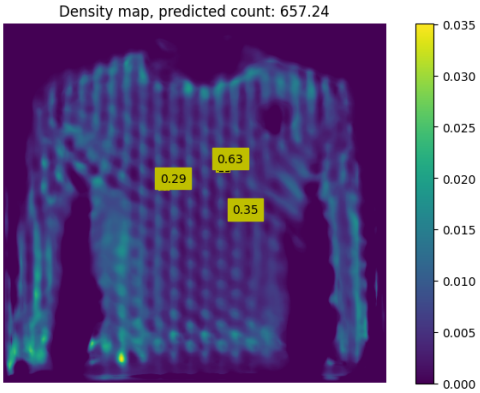
Input image

Overlaid result, predicted count: 657.24

Density map, predicted count: 657.24



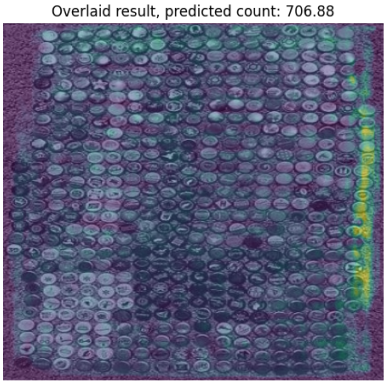
Density map, predicted count: 657.24



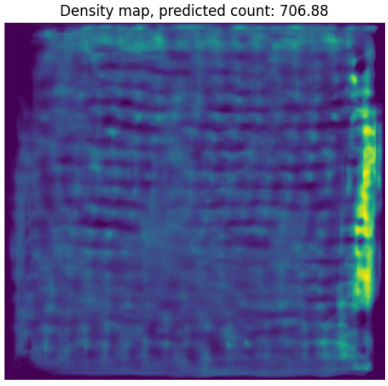
error = 205.88055419921875

Input image

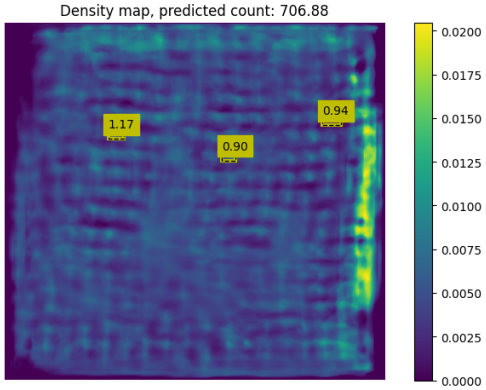
Overlaid result, predicted count: 706.88



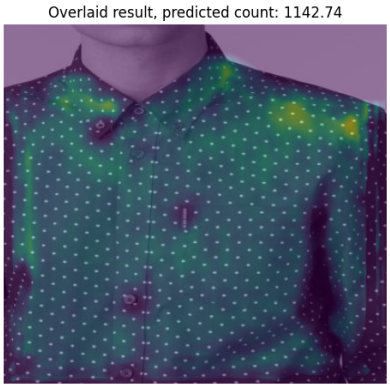
Density map, predicted count: 706.88

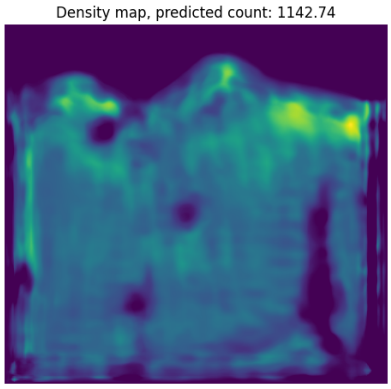


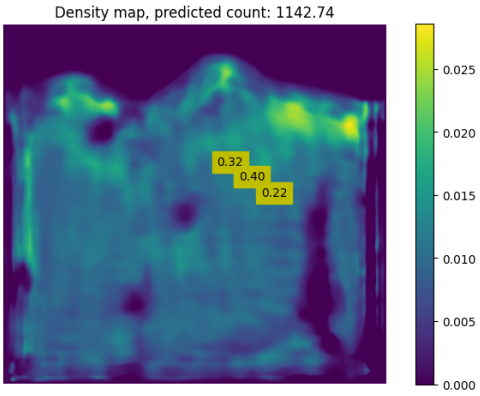
Density map, predicted count: 706.88



error = 257.7421875

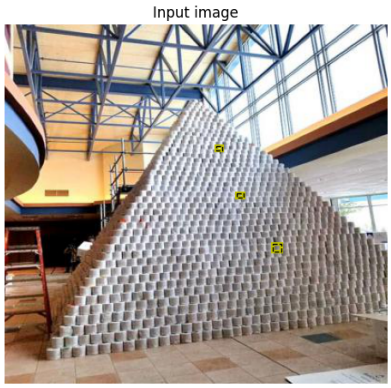


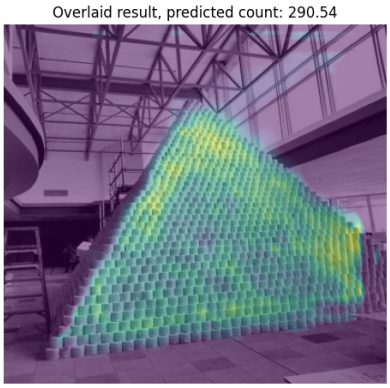


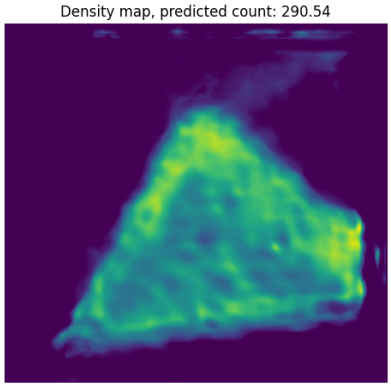


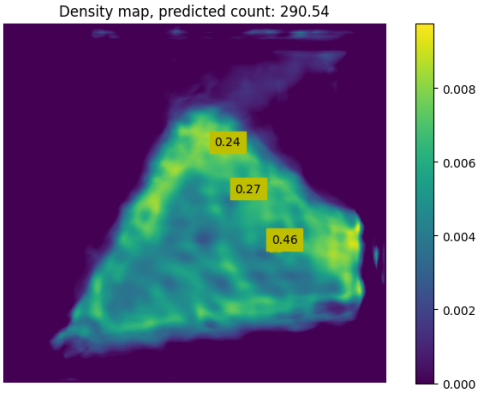
Images with highest under count error :

error = 616.4561462402344

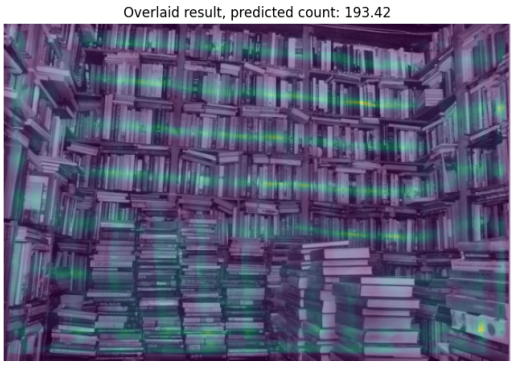


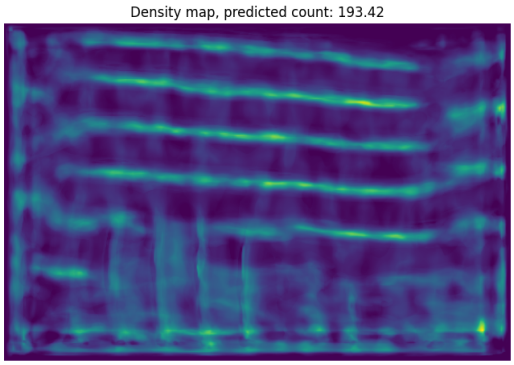


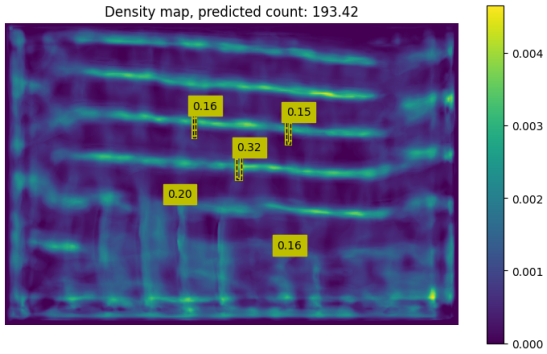




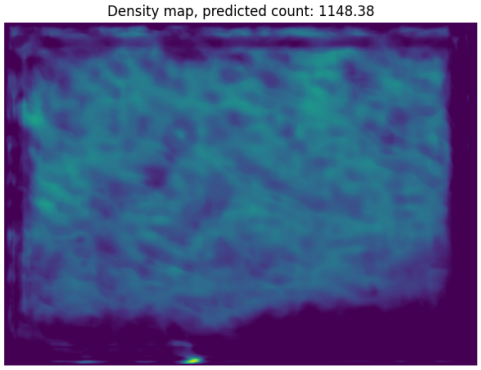
error = 828.5813751220703

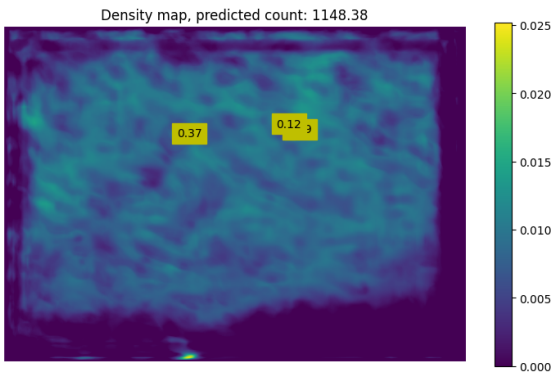




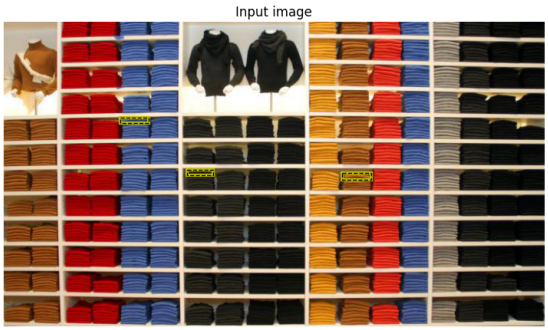


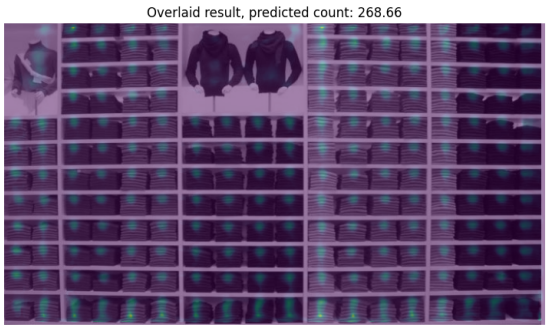
error = 943.616455078125

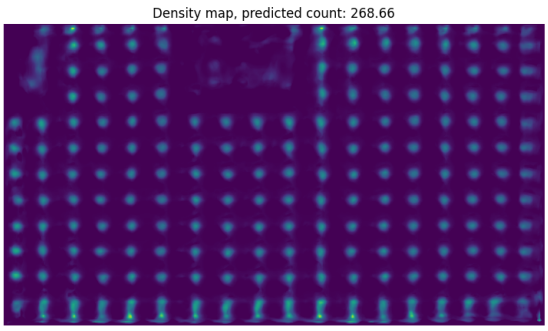


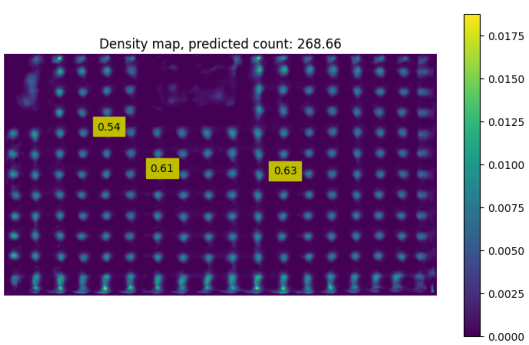


error = 962.3380737304688

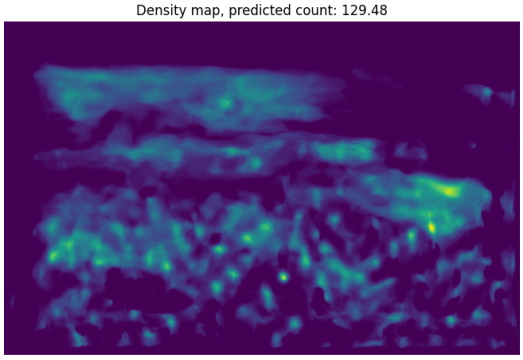


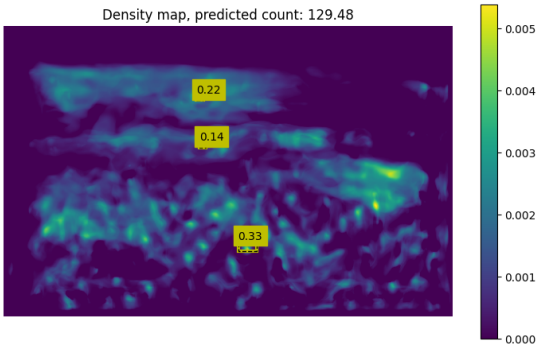






error = 962.5200958251953



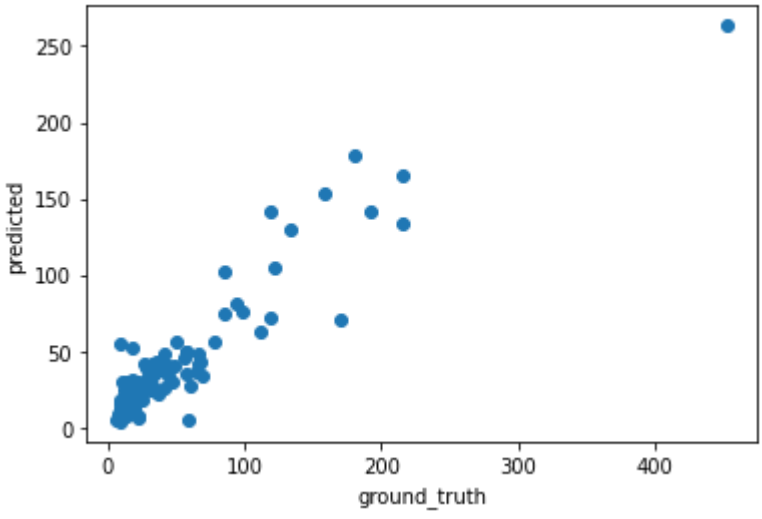


2.3

MAE : 14.92

RMSE: 28.32

Scatter plot for val-partA with test time adaptation using negative stroke loss function



Scatter plot for val-partA Without any test time adaptation