CSE512 Spring 2021 - Machine Learning - Homework 6

Name: Sabrina Margetic

Solar ID: 109898930

Net ID Email Address: sabrina.margetic@stonybrook.edu

Names of people whom you discussed the homework with: None (Only the TA)

1

1.1 Question 1.1

a)
X:
tensor([[0.2141, 0.0658], [0.3635, 0.1372], [0.2979, 0.6988]])

b)
Y:
tensor([[1., 1.], [1., 1.], [1., 1.]])
Size:
torch.Size([3, 2])

c)
Out:
tensor([[1.2141, 1.0658], [1.3635, 1.1372], [1.2979, 1.6988]])

In place addition:
tensor([[1.2141, 1.0658], [1.3635, 1.1372], [1.2979, 1.6988]])

d)
Random NP Array:
[[0.22575994 0.63641261] [0.83081888 0.66512089] [0.53414993 0.92871234]]

Tensor Version:
tensor([[0.2258, 0.6364], [0.8308, 0.6651], [0.5341, 0.9287]], dtype=torch.float64)

Converted Back To NP Array:
[[0.22575994 0.63641261] [0.83081888 0.66512089] [0.53414993 0.92871234]]

1.2 Question 1.2

a)
X:
tensor([[0.8785, 0.6608], [0.9554, 0.1180], [0.4257, 0.2175]], requires_grad=True)

b)
Y From x*10:
tensor([[8.7852, 6.6078], [9.5541, 1.1796], [4.2574, 2.1755]], grad_fn=<MulBackward0>)
MulBackward0 means that the multiplication operation happened. This will be remembered later in the backward phase.

Y From y+.1:
tensor([[8.8852, 6.7078], [9.6541, 1.2796], [4.3574, 2.2755]], grad_fn=<AddBackward0>)
AddBackward0 means that the addition operation happened. This will be remembered later in the backward phase.

Out From max(Y):
tensor(9.6541, grad_fn=<MaxBackward1>)
MaxBackward1 means that the max operation happened. This will be remembered later in the backward phase.

c)
D(out)/Dx:
tensor([[0., 0.], [10., 0.], [0., 0.]])

d)
Y after x*10:
tensor([[8.7852, 6.6078], [9.5541, 1.1796], [4.2574, 2.1755]])

Y after y+.1:
tensor([[8.8852, 6.7078], [9.6541, 1.2796], [4.3574, 2.2755]])

Out:
tensor(9.6541)
The difference is the operations wont be tracked, meaning that computing gradients for backpropogation won't be possible. Also, it should be noted that removing this option reduces memory consumption and can be useful when a particular gradient isn't necessary.

1.3 Question 1.3

a)
Network:
Net(
(features1): Sequential(
(0): Conv2d(3, 3, kernel_size=(2, 2), stride=(1, 1))
(1): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(2): ReLU()
(3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(conv2): Conv2d(3, 1, kernel_size=(2, 2), stride=(1, 1))
(features2): Sequential(
(0): BatchNorm2d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(1): ReLU()
(2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(classifier): Sequential(
(0): Linear(in_features=49, out_features=25, bias=True)
(1): Linear(in_features=25, out_features=10, bias=True)
)
)

Number of Parameters:
1570

b)
Network Output:
tensor([[0.2559, -0.2916, -0.8682, 0.6703, 0.6147, -0.1696, 0.6670, 0.3825, -0.2636, 0.3797]], grad$_{fn}$ =< $AddmmBackward$ >)

$ShapeofOutput$:
$torch.Size([1, 10])$

c)

$\underline{Conv2BiasGradAfterBackpropogation:}$
$tensor([-1.2922e-08])$

1.4 Question 1.4

Accuracy for Entire Dataset: 31.56%
Accuracy for Each Class:
plane: 52.6%
car: 46.8%
bird: 19.8%
cat: 16%
deer: 14.2%
dog: 27%
frog: 32.2%
horse: 24.3%
ship: 49.7%
truck: 33%

1.5 Question 1.5

a)
Accuracy for Entire Dataset: 78.85%
Accuracy for Each Class:
plane: 88.5%
car: 72.89999999999999%
bird: 74.6%
cat: 57.4%
deer: 71.2%
dog: 71.6%
frog: 87.1%
horse: 82.89999999999999%
ship: 89.7%
truck: 92.60000000000001%

b)
Accuracy for Entire Dataset: 42.11%
Accuracy for Each Class:
plane: 28.1%
car: 45.2%
bird: 44.2%
cat: 9.9%
deer: 25.0%
dog: 41.5%
frog: 67.10000000000001%
horse: 58.699999999999996%
ship: 61.4%
truck: 40.0%
As you can see, the accuracy has gone down when we used the model as a feature extractor.

2 Question 2

2.1 Question 2.1

The method proposed here is used as an alternative to the object counting problem. Instead of training a network on labeled examples, and then counting the number of those objects within an image, this method alternatively only takes a few labeled objects and generalizes it to the whole image. This method is more generally applicable to non observed categories and doesn't rely on extensive labeled data.
The architecture used for this model is called FamNet. FamNet works in two parts, the first is feature extraction and the second is density prediction. The output, a density map, can be summed over to obtain the count value. An important note about the feature extraction is that in order to make the network generic to all possible categories, the features obtained are not directly used for the density mapping. Rather, a correlation between the input exempler features and the image features are used for the density prediction. Additionally, this correlation is done on varying scales in order to account for varying object sizes.
At training, the loss function used is the Mean Square error. Adaptions are made at test time. The errors calculated at test time are Perturbation and Min-Count Loss.
Since this was a quiet unique solution, and problem, data was specifically collected. I think as a step further with this research it would be interesting to see variations of this data set. For example, the data set had constraints that it was high image quality, appearance similarity, and only one category per image. It would be interesting to see modifications made to the original network to compensate for this.
Overall, the network outperforms it competitors in both speed, and accuracy. The closest competitor is MAML, but FamNet is approximately three times faster than this.

2.2 Question 2.2

On val data, MAE: 24.32, RMSE: 70.94
Scatter Plot:
The scatter plot can be seen at Fig. 1.

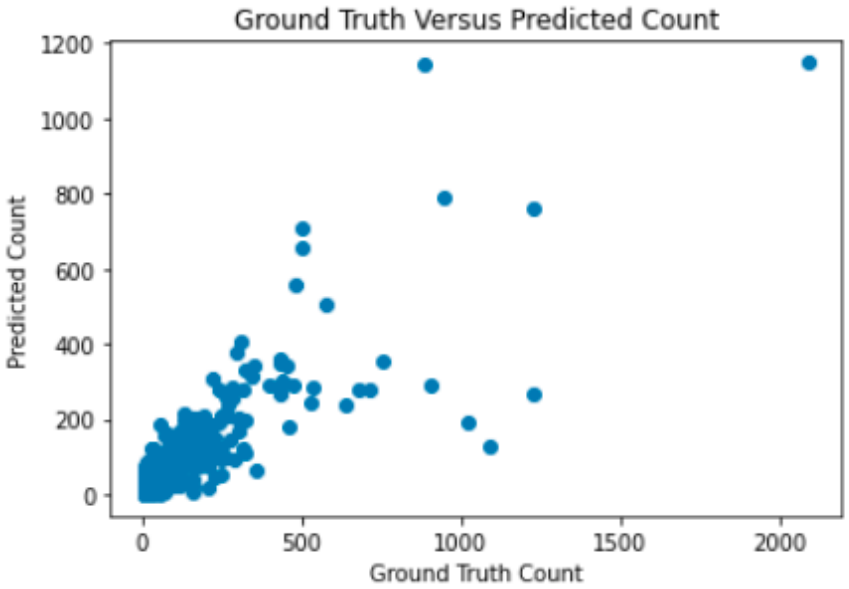

Figure 1

Highest Over Count Error:
The images with the highest over count values can be seen at Fig. 2. Its interesting to see that for 4/5 of these images, the objects counted were circular.

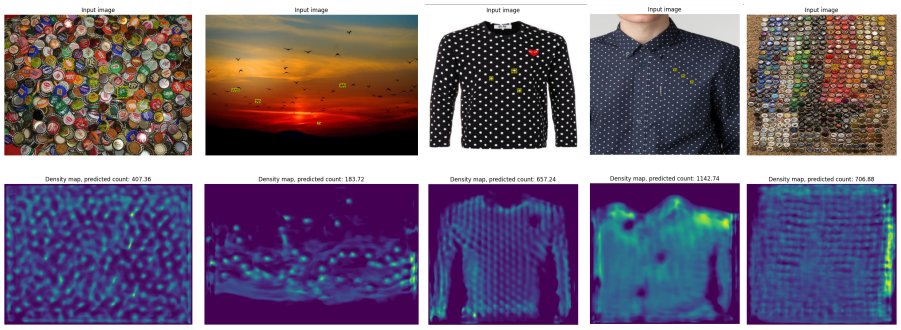

Figure 2

Highest Under Count Error:
The images with the highest under count values can be seen at Fig. 3. This under count in way makes sense since in the ground truth there is an extremely high amount of objects to be counted, obviously making counting all of them difficult.

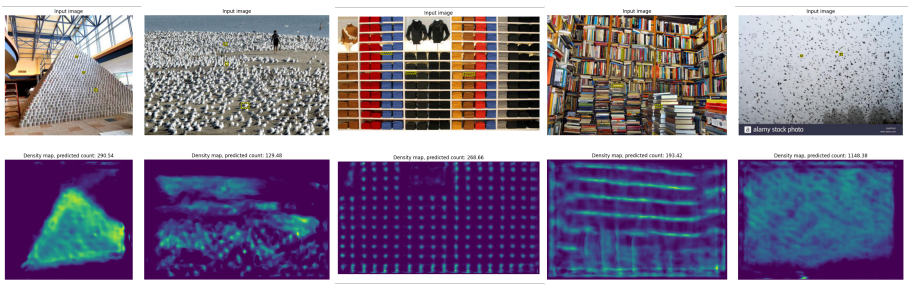

Figure 3

2.3 Question 2.3

For this part of the assignment, I implemented it by making modifications to the test.py file, I saved the altered version as test2.py and attached it as well with the assignment.
The way that I implemented this part was by allowing the original images to pass through the network, then at the Density Prediction Module, where loss was calculated, I took into account the masks provided. I did this by summing over density values for when the mask had a value of 1. Before adding this to the Perturbation and Min-Count Loss, I multiplied it by a factor of 1e-09. The paper had mentioned that the loss needed to be relatively small so that the adaption loss was similar to the training loss. The weight provided did not alter the original order of magnitude. I kept the Min-Count learning rate and the number of gradient steps, ie, the weights for the Min-Count and Perturbation Losses, as their original values. I tried multiplying them by various factors and found that the original values performed best on the validation set.

With Adaption:
MAE: 15.10, RMSE: 24.73

Without Adaption:
MAE: 16.11, RMSE: 25.56

Scatter Plots: The scatter plot for when adaption was used, and not used, can be seen in Fig 4, and 5, respectively. As you can see, with the adaption, the spread of values has lessened and the curve has become more linear, which is ideal.

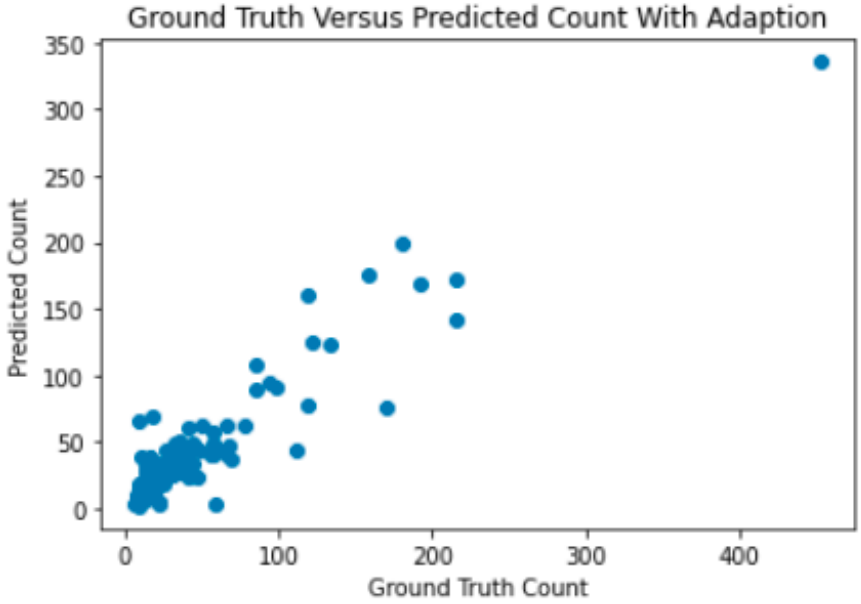

Figure 4

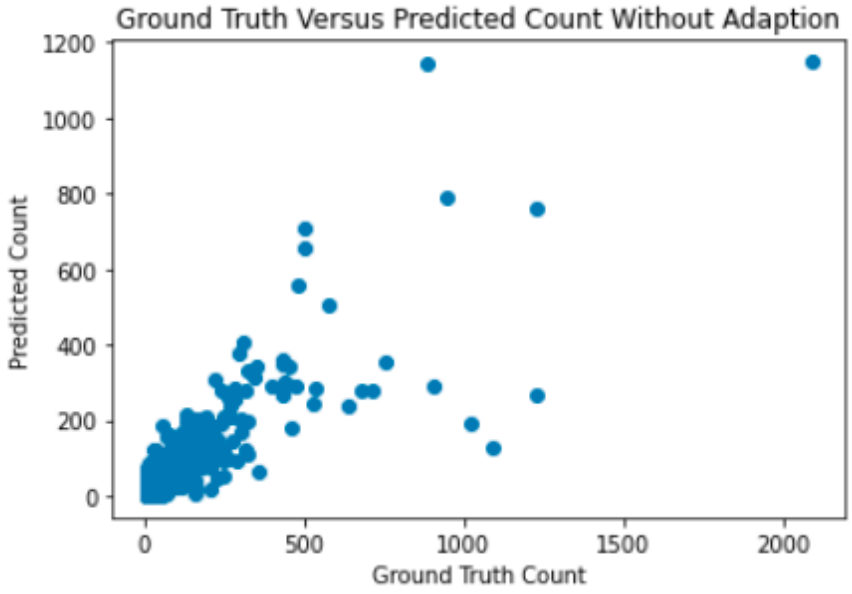

Figure 5

2.4 Question 2.4

The submission was made to Kaggle accordingly.

2.5 Question 2.5

The submission was made to Kaggle accordingly.