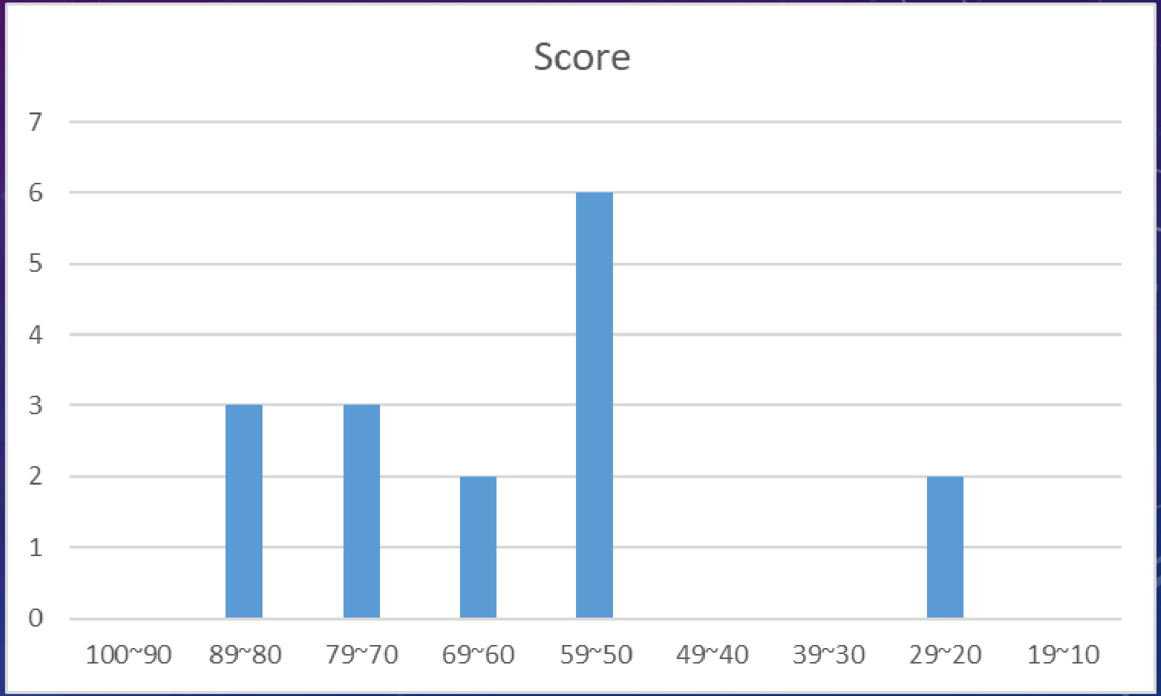
DIGITAL SURVEILLANCE SYSTEM AND APPLICATION

EXAM GRADES

Jison Hsu
Artificial Vision Laboratory
Taiwan Tech

GRADES OVERVIEW

p1 [40/12(	p2 [15/12(	p3 [15/12(	p4 [25/12(	p5 [25/12(	Total
38	14	2	23	3	80
35	0	10	6	0	51
32	8	15	10	12	77
33	0	12	2	7	54
26	15	4	4	5	54
15	6	2	0	3	26
28	6	12	9	12	67
33	15	15	6	5	74
37	8	10	20	10	85
35	15	2	4	2	58
30	2	6	5	11	54
33	6	3	9	12	63
35	14	15	17	3	84
29	9	15	12	8	73
33	11	0	8	5	57
19	4	0	0	0	23
					61.25



Average score

Problem 1 [40/120]

1. Please modify the Prob1.ipynb with the following requirements and set the Cross Entropy Loss, Adam Optimizer 0.002 learning rate and betas [0.5,0.999] to train a classifier :

   A. Design a model with the following structure.

      - First Conv. layer: Input: RGB, Output Channel 8, second Conv. layer: Output Channel 16, third Conv. layer: Output Channel 32.
      - FC-Layer1: Input: Defined by the third convolutional Layer, Output: 300
      - FC-Layer2: Input: From FC- Layer1, Output: 150
      - FC-Layer3: Input: From FC- Layer2, Output: equal to your class size [8/40]

   B. Change the learning rate to 0.0002 when epoch=2. [5/40]
      hint: torch.optim.lr_scheduler.StepLR

   C. Save the model and name it as ‘Prob1.pth’ [3/40]

   D. Save the optimizer and name it as ‘Prob1_1.pth’[3/40]

Net ( )

Layer type	Input channel	Output channel	Filter size	Stride	padding	Negative slope
Conv1	A	B	3	1	2	
Leaky_ReLU						0.01
AvgPool			2	1	1	
Conv2	B	C	2	1	1	
ELU						
MaxPool			2	2	1	
Conv3	C	D	2	1	1	
Leaky_ReLU						0.02
MaxPool			2	3	1	
Linear1	E	F				
ELU						
Linear2	F	G				
ELU						
Linear3	G	H				

Please crop the parts that you modify in Prob1.ipynb and paste to the solution .docx.

Solution 1A

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 3, 1, 2)
        self.pool = nn.AvgPool2d(2, 1, 1)
        self.conv2 = nn.Conv2d(8, 16, 2, 1, 1)
        self.pool2 = nn.MaxPool2d(2, 2, 1)
        self.conv3 = nn.Conv2d(16, 32, 2, 1, 1)
        self.pool3 = nn.MaxPool2d(2, 3, 1)
        self.fc1 = nn.Linear(1568, 300)
        self.fc2 = nn.Linear(300, 150)
        self.fc3 = nn.Linear(150, 10)

    def forward(self, x):
        x = self.pool(F.leaky_relu(self.conv1(x),negative_slope=0.01))
        x = self.pool2(F.elu(self.conv2(x)))
        x = self.pool3(F.leaky_relu(self.conv3(x),negative_slope=0.02))
        x = x.view(-1,1568)
        x = F.elu(self.fc1(x))
        x = F.elu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```

A B C D E F G H

Solution 1B

```
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.1, last_epoch=-1)
```

Result

```
Epoch : 1 steps : 1000 Training Loss : 1.947866315215826
Epoch : 1 steps : 2000 Training Loss : 1.764978513419628
Epoch : 1 steps : 3000 Training Loss : 1.6969225649237634
Epoch : 1 steps : 4000 Training Loss : 1.6640090248584747
Epoch : 1 steps : 5000 Training Loss : 1.559449948579073
Epoch : 1 steps : 6000 Training Loss : 1.5539788318276406
Epoch : 1 steps : 7000 Training Loss : 1.5530633701384067
Epoch : 1 steps : 8000 Training Loss : 1.512206330806017
Epoch : 1 steps : 9000 Training Loss : 1.519058351173997
Epoch : 1 steps : 10000 Training Loss : 1.5114509925097228
Epoch : 1 steps : 11000 Training Loss : 1.4792182659208775
Epoch : 1 steps : 12000 Training Loss : 1.448660266853869
epoch: 1 lr: [0.002]
Epoch : 2 steps : 1000 Training Loss : 1.416085939258337
Epoch : 2 steps : 2000 Training Loss : 1.4364076300412416
Epoch : 2 steps : 3000 Training Loss : 1.4133284003362059
Epoch : 2 steps : 4000 Training Loss : 1.40917020855844
Epoch : 2 steps : 5000 Training Loss : 1.3903609827756882
Epoch : 2 steps : 6000 Training Loss : 1.4354739887416363
Epoch : 2 steps : 7000 Training Loss : 1.3700204498693347
Epoch : 2 steps : 8000 Training Loss : 1.3779636757671834
Epoch : 2 steps : 9000 Training Loss : 1.4121912475116551
Epoch : 2 steps : 10000 Training Loss : 1.3958124362006783
Epoch : 2 steps : 11000 Training Loss : 1.3583056082800031
Epoch : 2 steps : 12000 Training Loss : 1.3853848991133273
epoch: 2 lr: [0.0002]
Epoch : 3 steps : 1000 Training Loss : 1.1073712862990797
Epoch : 3 steps : 2000 Training Loss : 1.076918856561184
Epoch : 3 steps : 3000 Training Loss : 1.0175932760313153
⋮
```

Solution 1C

```
Path='Prob1.pth'
torch.save(net.state_dict(),Path)
```

Result

Prob1.pth

Solution 1D

```
Path='Prob1_1.pth'
torch.save(optimizer.state_dict(),Path)
```

Result

Prob1_1.pth

Problem 1

E. Change the dataset to CIFAR100 [2/40]

F. Load the 'Prob1.pth' obtained from C. and design a model Net2( ) which is ONLY different from the last layer of Net( ). [4/40]

G. Save the model and name it as 'Prob1_2.pth'. [4/40]

H. Extract the features from the g1.jpg and the g2.jpg using Prob1.pth and Prob1_2.pth, respectively. The extracted features will be obtained from the second last layer (Linear2 shown in Table). [3/40]

I. Please calculate the cosine distance between the two latent vectors extracted from each model. [3/40]

Solution 1E

```
transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)
```

Solution 1F

```
checkpoint = torch.load('./Prob1.pth')
net.load_state_dict(checkpoint,strict=False)
```

Solution 1G

```
Path='Prob1_2.pth'
torch.save(net.state_dict(),Path)
```

Result

Prob1_2.pth

```
class Net2(nn.Module):
    def __init__(self):
        super(Net2, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 3, 1, 2)
        self.pool = nn.AvgPool2d(2, 1, 1)
        self.conv2 = nn.Conv2d(8, 16, 2, 1, 1)
        self.pool2 = nn.MaxPool2d(2, 2, 1)
        self.conv3 = nn.Conv2d(16, 32, 2, 1, 1)
        self.pool3 = nn.MaxPool2d(2, 3, 1)
        self.fc1 = nn.Linear(1568, 300)
        self.fc2 = nn.Linear(300, 150)
        self.fc4 = nn.Linear(150, 100)

    def forward(self, x):
        x = self.pool(F.leaky_relu(self.conv1(x),negative_slope=0.01))
        x = self.pool2(F.elu(self.conv2(x)))
        x = self.pool3(F.leaky_relu(self.conv3(x),negative_slope=0.02))
        x = x.view(-1,1568)
        x = F.elu(self.fc1(x))
        x = F.elu(self.fc2(x))
        x = self.fc4(x)
        return x

net = Net2()
```

Given by page 5 architecture

Solution 1H

You need to modify Net() and Net2() first and return the feature vector of fc2.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 3, 1, 2)
        self.pool = nn.AvgPool2d(2, 1, 1)
        self.conv2 = nn.Conv2d(8, 16, 2, 1, 1)
        self.pool2 = nn.MaxPool2d(2, 2, 1)
        self.conv3 = nn.Conv2d(16, 32, 2, 1, 1)
        self.pool3 = nn.MaxPool2d(2, 3, 1)
        self.fc1 = nn.Linear(1568, 300)
        self.fc2 = nn.Linear(300, 150)
        self.fc3 = nn.Linear(150, 10)
```

Given by page 5 architecture

```
    def forward(self, x):
        x = self.pool(F.leaky_relu(self.conv1(x),negative_slope=0.01))
        x = self.pool2(F.elu(self.conv2(x)))
        x = self.pool3(F.leaky_relu(self.conv3(x),negative_slope=0.02))
        x = x.view(-1,1568)
        x = F.elu(self.fc1(x))
        x1 = F.elu(self.fc2(x))
        x = self.fc3(x1)
        return x,x1

net = Net()
```

```
class Net2(nn.Module):
    def __init__(self):
        super(Net2, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 3, 1, 2)
        self.pool = nn.AvgPool2d(2, 1, 1)
        self.conv2 = nn.Conv2d(8, 16, 2, 1, 1)
        self.pool2 = nn.MaxPool2d(2, 2, 1)
        self.conv3 = nn.Conv2d(16, 32, 2, 1, 1)
        self.pool3 = nn.MaxPool2d(2, 3, 1)
        self.fc1 = nn.Linear(1568, 300)
        self.fc2 = nn.Linear(300, 150)
        self.fc4 = nn.Linear(150, 100)
```

Given by page 10 architecture

```
    def forward(self, x):
        x = self.pool(F.leaky_relu(self.conv1(x),negative_slope=0.01))
        x = self.pool2(F.elu(self.conv2(x)))
        x = self.pool3(F.leaky_relu(self.conv3(x),negative_slope=0.02))
        x = x.view(-1,1568)
        x = F.elu(self.fc1(x))
        x1 = F.elu(self.fc2(x))
        x = self.fc4(x1)
        return x,x1

net = Net2()
```

Solution 1H

You need to resize the image to 32*32 and turn it into a tensor value before it can be sent to the model.

```
from PIL import *
from torch.autograd import Variable
transform = transforms.Compose([transforms.Resize((32,32)),transforms.ToTensor()])

net1 = Net()
net2 = Net2()

checkpoint1 = torch.load('Prob1.pth')
checkpoint2 = torch.load('Prob1_2.pth')

net1.load_state_dict(checkpoint1,strict=False)
net2.load_state_dict(checkpoint2,strict=False)

image1 = transform(Image.open('g1.jpg'))
image2 = transform(Image.open('g2.jpg'))

image1=Variable(torch.unsqueeze(image1, dim=0).float(), requires_grad=False)
image2=Variable(torch.unsqueeze(image2, dim=0).float(), requires_grad=False)

output1_1,feature1_1 = net1(image1)
output1_2,feature1_2 = net1(image2)
output2_1,feature2_1 = net2(image1)
output2_2,feature2_2 = net2(image2)
```

Refer to exercise2-7

Since the image sent to the model is single sheets, the batch size needs to be increased.

According to the results of the previous changes to the architecture, return the final and fc2 layer output.

Correct answer rate: 0/16

Solution 1I

```
cos= nn.CosineSimilarity(dim=1)
cosine_dist1 = 1 - cos(feature1_1,feature1_2)
print("The prob1.pth cosine distance:{}".format(cosine_dist1))
cosine_dist2 = 1 - cos(feature2_1,feature2_2)
print("The prob1_2.pth cosine distance:{}".format(cosine_dist2))
```

Refer to Prob2B in the sample problems

Result

```
The prob1.pth cosine distance:tensor([0.4697], grad_fn=<RsubBackward1>)
The prob1_2.pth cosine distance:tensor([0.5660], grad_fn=<RsubBackward1>)
```

Correct answer rate: 4/16

Problem 2 [15/120]

2. Prob2.ipynb gives you a ResNet-18 trained on ImageNet. Use Prob2.ipynb to show the following:

A. The feature maps extracted from Layer1, 0-BasicBlock, Conv1 and Layer2, 1-BasicBlock, Conv2. The ResNet-18 architecture will be shown when you are running the initialization (__init__) of "FeatureVisualization". [8/15]

B. Calculate the Euclidean distance and Cosine similarity between the images g1.jpg and g2.jpg, which are given with the code. [3/15]

C. Please point out at least three differences between ResNet and VGG in structures. [4/15]

Solution 2A

```
self.pretrained_model = models.resnet18(pretrained=True)
print(self.pretrained_model)
self.pretrained_model.eval()
```

- Extracted from Layer1, 0-BasicBlock, Conv1

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

Solution 2A

- Extracted from Layer1, 0-BasicBlock, Conv1

```
def get_feature(self):
        # Image preprocessing
        input=self.process_image()
        #print("input.shape:{}".format(input.shape))
        print("input.shape:{}".format(input.shape))
        x=input
        x0 = self.pretrained_model2.conv1(x)
        x = self.pretrained_model2.bn1(x0)
        x = self.pretrained_model2.relu(x)
        x = self.pretrained_model2.maxpool(x)
        ###your code

        feature_1 = self.pretrained_model2.layer1[0].conv1(x)
        x1 = self.pretrained_model2.layer1(x)
        x2 = self.pretrained_model2.layer2[0](x1)
        x2 = self.pretrained_model2.layer2[1].conv1(x2)
        x2 = self.pretrained_model2.layer2[1].bn1(x2)
        x2 = self.pretrained_model2.layer2[1].relu(x2)
        feature_2 = self.pretrained_model2.layer2[1].conv2(x2)
        ###end code
        # x3 = self.pretrained_model2.layer3(x2)
        # x4 = self.pretrained_model2.layer4(x3)
        # x = self.pretrained_model2.avgpool(x4)
        # x = x.view(-1,512)

        return feature_1
```

Solution 2A

- Extracted from Layer1, 0-BasicBlock, Conv1

Input:

Result:

Input:

Result:

Solution 2A

- Extracted from Layer2, 1-BasicBlock, Conv2

```
(layer2): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

Solution 2A

- Extracted from Layer2, 1-BasicBlock, Conv2

```
def  get_feature(self):
        #  Image    preprocessing
        input=self.process_image()
        #print("input.shape:{}".format(input.shape))
        print("input.shape:{}".format(input.shape))
        x=input
        x0  =  self.pretrained_model2.conv1(x)
        x  =  self.pretrained_model2.bn1(x0)
        x  =  self.pretrained_model2.relu(x)
        x  =  self.pretrained_model2.maxpool(x)
        ###your  code

        feature_1  =  self.pretrained_model2.layer1[0].conv1(x)
        x1  =  self.pretrained_model2.layer1(x)
        x2  =  self.pretrained_model2.layer2[0](x1)
        x2  =  self.pretrained_model2.layer2[1].conv1(x2)
        x2  =  self.pretrained_model2.layer2[1].bn1(x2)
        x2  =  self.pretrained_model2.layer2[1].relu(x2)
        feature_2  =  self.pretrained_model2.layer2[1].conv2(x2)
        ###end  code
        #  x3  =  self.pretrained_model2.layer3(x2)
        #  x4  =  self.pretrained_model2.layer4(x3)
        #  x  =  self.pretrained_model2.avgpool(x4)
        #  x  =  x.view(-1,512)

        return  feature_2
```

Solution 2A

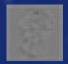
- Extracted from Layer2, 1-BasicBlock, Conv2

Input:

Result:

Input:

Result:



Correct answer rate:7/16

Solution 2B

```
#Define cosine similarity
cos= nn.CosineSimilarity(dim=1)
#Define Euclidean distance
euclidean_dist = torch.dist(first_vector,second_vector,p=2)
cosine_similarity = cos(first_vector,second_vector)

print(“Verification:”)
print(“Their cosine_similarity:{}”.format(cosine_similarity))
print(“Their euclidean_dist:{}”.format(euclidean_dist))
```

Refer to Prob2B in the sample problems and exercise 2-5.

Result:

```
Verification:
Their cosine_similarity:tensor([0.4720], grad_fn=<DivBackward0>)
Their euclidean_dist:67.58879852294922
```

Solution 2C

- ResNet uses the residual block in the CNN structure in order to solve the vanishing gradient problem.
- ResNet uses the Average Pooling after the last convolution layer instead of Max Pooling in VGG structure.
- There are three fully connected layer in VGG instead of only a fully connected layer in ResNet.

Problem 3 [15/120]

3. Consider Table 3 (on next page), the dimensions of the feature maps made by AvgPoo11 and Conv6 are 20*8*3414*3428 and 20*256*52*52, please compute the following:
   - The stride at Conv2.
   - The dimension of the input.
   - The dimensions of the output feature maps from Conv2, Conv3, Conv4, Conv6, Conv7.

Refer to Prob3
in the SAMPLE PROBLEMS

Table 3

Layer type	Input channel	Output channel	Filter size	Stride
Conv1	3	8	(3,2)	1
AvgPool1			4	1
Conv2	8	16	(4,1)	?
MaxPool1			3	2
Conv3	16	32	(2,3)	1
MaxPool2			2	1
Conv4	32	64	(4,5)	2
AvgPool2			3	1
Conv5	64	128	2	1
MaxPool3			2	2
Conv6	128	256	2	2
AvgPool3			2	2
Conv7	256	512	(7,6)	1

Solution3

$$Output = \frac{Input - kernel\ size + 2 \times Padding}{Stride} + 1$$

$$Input = (Output - 1) \times Stride\ - 2 \times Padding + kernel\ size$$

- Input-> [20, 3, 3419, 3432] $3419 = (3417 - 1) \times 1\ - 2 \times 0{+}3$, $3432 = (3431 - 1) \times 1\ - 2 \times 0{+}2$
- Conv1-> [20, 8, 3417, 3431] $3417 = (3414 - 1) \times 1\ - 2 \times 0{+}4$, $3431 = (3428 - 1) \times 1\ - 2 \times 0{+}4$
- AvgPool1-> [20, 8, 3414, 3428]
- Conv2-> [20, 16, 853, 857] $853 = (426 - 1) \times 2\ - 2 \times 0{+}3$, $857 = (428 - 1) \times 2\ - 2 \times 0{+}3$
- MaxPool1-> [20, 16, 426, 428] $426 = (425 - 1) \times 1\ - 2 \times 0{+}2$, $428 = (426 - 1) \times 1\ - 2 \times 0{+}3$
- Conv3-> [20, 32, 425, 426] $425 = (424 - 1) \times 1\ - 2 \times 0{+}2$, $426 = (425 - 1) \times 1\ - 2 \times 0{+}2$
- MaxPool2-> [20, 32, 424, 425] $424 = (211 - 1) \times 2\ - 2 \times 0{+}4$, $425 = (211 - 1) \times 2\ - 2 \times 0{+}5$
- Conv4-> [20, 64, 211, 211] $211 = (209 - 1) \times 1\ - 2 \times 0{+}3$, $211 = (209 - 1) \times 1\ - 2 \times 0{+}3$
- AvgPool2->[20, 64, 209, 209] $209 = (208 - 1) \times 1\ - 2 \times 0{+}2$, $3431 = (208 - 1) \times 1\ - 2 \times 0{+}2$

Solution3

- Conv5-> [20, 128, 208, 208]
- MaxPool3-> [20, 128, 104, 104]
- Conv6->[20, 256, 52, 52]
- AvgPool3->[20, 256, 26, 26]
- Conv7->[20, 512, 20, 21]

$208 = (104 - 1) \times 2 \ - 2 \times 0{+}2 \,,$ $3431 = (104 - 1) \times 2 \ - 2 \times 0{+}2$

$104 = (52 - 1) \times 2 \ - 2 \times 0{+}2 \,,$ $104 = (52 - 1) \times 1 \ - 2 \times 0{+}2$

$26 = \frac{52 - 2 + 2 \times 0}{2} + 1 \,,$ $26 = \frac{52 - 2 + 2 \times 0}{2} + 1$

$20 = \frac{26 - 7 + 2 \times 0}{1} + 1 \,,$ $21 = \frac{26 - 6 + 2 \times 0}{1} + 1$

Correct answer rate: 8/16

Problem 4 [25/120]

4. Please modify the Prob4.ipynb :

   A. Test the given images on the Colab. [2/25]
   B. Consider the five images shown in DSS_TEST folder , add in the part for computing the IOU. Calculate the average IOU with the prediction bbox and ground truth bbox. [4/25]
   C. Please compute the MSE between the prediction bbox and ground truth bbox. [8/25]
   D. Given the five images from B. please draw the PR-Curve with threshold =[0.1, 0.3, 0.5, 0.7, 0.8]. [8/25]
   E. If the number of classes is 5, please compute the filter size [3/25]

Solution 4A

‘$’ is for using the variable in the Linux

1.Using For loop to test:

Test Folder

```
for file in os.listdir('/content/DSS_TEST'):
  !./darknet detector test /content/DSS_License_plate/obj.data  /content/DSS_License_plate/yolov3-tiny_obj.cfg  "/content/DSS_License_plate/yolov3-tiny_obj_best2.weights" /content/DSS_TEST/$file -ext_output -thresh 0.05
  imShow('predictions.jpg')
```

2.Test one image for each time:

```
  !./darknet detector test /content/DSS_License_plate/obj.data  /content/DSS_License_plate/yolov3-tiny_obj.cfg  "/content/DSS_License_plate/yolov3-tiny_obj_best2.weights" /content/DSS_TEST/857.jpg -ext_output -thresh 0.05
  imShow('predictions.jpg')
```

Confidence Threshold

Absolute path for the test image

Refer to Prob4B
in the SAMPLE PROBLEMS

Solution 4A


License_plate: 0.98
B51 2019/01/19 22:34:02


License_plate: 0.16
License_plate: 0.99


License_plate: 1.00
353-MYT


License_plate: 0.11
B51 2019/04/14 10:31:22

Solution 4B

```
[21] ###IOU  function#####
     def IOU(rec1, rec2):

         S_rec1 = (rec1[2] - rec1[0]) * (rec1[3] - rec1[1])
         S_rec2 = (rec2[2] - rec2[0]) * (rec2[3] - rec2[1])

         sum_area = S_rec1 + S_rec2

         left_line = max(rec1[0], rec2[0])
         right_line = min(rec1[2], rec2[2])
         top_line = max(rec1[1], rec2[1])
         bottom_line = min(rec1[3], rec2[3])

         if left_line >= right_line or top_line >= bottom_line:
                 return 0
         else:
                 intersect = (right_line - left_line) * (bottom_line - top_line)
                 return (intersect / (sum_area - intersect))*1.0
```

Refer to Prob4C
in the SAMPLE PROBLEMS

Solution 4B

```
GT1  =  torch.FloatTensor([[618,854,744,935],[1023,598,1084,627]])
GT2  =  torch.FloatTensor([[902,435,1031,522]])
GT3  =  torch.FloatTensor([[902,435,1031,522]])
GT4  =  torch.FloatTensor([[732,499,777,522],[1129,453,1199,493]])
GT5  =  torch.FloatTensor([[648,722,742,775],[994,644,1038,689],[1781,566,1823,596]])

Prediction1  =  torch.FloatTensor([[618,852,618+114,852+83],[0,0,0,0]])
Prediction2  =  torch.FloatTensor([[6,464,6+47,464+92],[915,431,915+105,431+96]])
Prediction3  =  torch.FloatTensor([[1460,570,1460+112,570+63]])
Prediction4  =  torch.FloatTensor([[1133,448,1133+71,448+53],[0,0,0,0]])
Prediction5  =  torch.FloatTensor([[652,720,652+85,720+58],[0,0,0,0],[0,0,0,0]])


iou1_0  =  IOU(GT1[0],  Prediction1[0])
iou1_1  =  IOU(GT1[1],  Prediction1[1])

iou2_0  =  IOU(GT2[0],  Prediction2[1])

iou3_0  =  IOU(GT3[0],  Prediction3[0])

iou4_0  =  IOU(GT4[0],  Prediction4[1])
iou4_1  =  IOU(GT4[1],  Prediction4[0])

iou5_0  =  IOU(GT5[0],  Prediction5[0])
iou5_1  =  IOU(GT5[1],  Prediction5[1])
iou5_2  =  IOU(GT5[2],  Prediction5[2])

m_IOU=(iou1_0+iou1_1+iou2_0+iou3_0+iou4_0+iou4_1+iou5_0+iou5_1+iou5_2)/8
print(m_IOU)
```

Ground Truth and Prediction coordinate. (x1,y1,x2,y2) (option)

For computing the MSE loss in 4C using the PyTorch Function ‘MSELoss’, it need to be torch.FloatTensor. You can use your function to compute.

(x, y, x + width, y + height)

```
tensor(0.3927)
```

Refer to Prob4D in the SAMPLE PROBLEMS

Correct answer rate:7/16

Solution 4C

```
[34] import torch
     mse = torch.nn.MSELoss()
     loss1 = mse(Prediction1,GT1)
     loss2 = mse(Prediction2[1],GT2[0])
     loss4_0 = mse(Prediction4[0],GT4[1])
     loss4_1 = mse(Prediction4[1],GT4[0])
     loss5 = mse(Prediction5,GT5)
     total_loss = (loss1+loss2+loss4_0+loss4_1+loss5)/(2+1+1+1+3)
     print(total_loss)
```

MSELoss in PyTorch function

```
tensor(203844.2500)
```

Correct answer rate:3/16

Solution 4D


```
License_plate: 98%     (left_x:   652   top_y:   720   width:    85   height:    58)
```

Score:0.98
Score:0
Score:0

Threshold=0.1 TP=1, FP=0, FN=2
Threshold=0.3 TP=1, FP=0, FN=2
Threshold=0.5 TP=1, FP=0, FN=2
Threshold=0.7 TP=1, FP=0, FN=2
Threshold=0.8 TP=1, FP=0, FN=2

Solution 4D

License_plate: 16%	(left_x:	6	top_y:	464	width:	47	height:	92)
License_plate: 99%	(left_x:	915	top_y:	431	width:	105	height:	96)

Score:0.99
Score:0.16

Threshold=0.1 TP=1, FP=1, FN=0
Threshold=0.3 TP=1, FP=0, FN=0
Threshold=0.5 TP=1, FP=0, FN=0
Threshold=0.7 TP=1, FP=0, FN=0
Threshold=0.8 TP=1, FP=0, FN=0

Solution 4D

`License_plate: 100%      (left_x:   618   top_y:   852   width:   114   height:    83)`

Score:0

Score:1

Threshold=0.1 TP=1, FP=0, FN=1

Threshold=0.3 TP=1, FP=0, FN=1

Threshold=0.5 TP=1, FP=0, FN=1

Threshold=0.7 TP=1, FP=0, FN=1

Threshold=0.8 TP=1, FP=0, FN=1

Solution 4D

License_plate: 11%   (left_x: 1133   top_y: 448   width: 71   height: 53)

Score:0

Score:0.11

Threshold=0.1 TP=1, FP=0, FN=1

Threshold=0.3 TP=0, FP=0, FN=2

Threshold=0.5 TP=0, FP=0, FN=2

Threshold=0.7 TP=0, FP=0, FN=2

Threshold=0.8 TP=0, FP=0, FN=2

Solution 4D

$$Precision = \frac{TP}{(TP + FP)}$$

$$Recall = \frac{TP}{(TP + FN)}$$

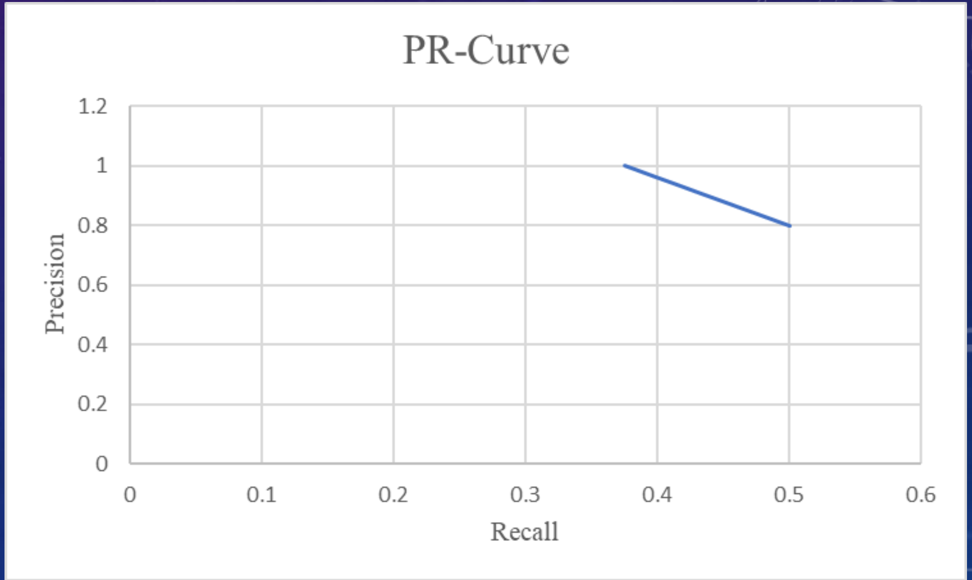
Threshold = 0.1, TP =4, FP=1, FN=4, Precision=0.8, Recall=0.5

Threshold = 0.3, TP =3, FP=0, FN=5, Precision=1, Recall=0.375

Threshold = 0.5, TP =3, FP=0, FN=5, Precision=1, Recall=0.375

Threshold = 0.7, TP =3, FP=0, FN=5, Precision=1, Recall=0.375

Threshold = 0.8, TP =3, FP=0, FN=5, Precision=1, Recall=0.375



Refer to Prob6
in the SAMPLE PROBLEMS

Correct answer rate:3/16

Solution 4E

The number of YOLO filters can be computed as following:

- (num of classes + 5) * 3
- (5+5)*3 = 30

Refer to Prob4A
in the SAMPLE PROBLEMS

Correct answer rate:6/16

Problem 5 [25/120]

5. Please modify the Prob5.ipynb :

A. Train from scratch (i.e. without using the pretrain model). [3/25]
B. Consider the five images shown in DSS_TEST folder, please compare the average IOU obtained by using the pretrained model (given in the code) and your train-from-scratch model (from A.). [6/25]
C. What is Average Precision (AP)? [3/25]
D. Given the five images in B. Please using the pertained model to compute the AP by threshold=[0.1, 0.3, 0.5, 0.7]. [8/25]
E. Write down your observation for the estimation from D. [5/25]

Solution 5A

Deleted the pretrained weight

```
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
import os
cfg = get_cfg()
cfg.merge_from_file("./detectron2_repo/configs/COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
cfg.DATASETS.TRAIN = ("plate_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 5
#cfg.MODEL.WEIGHTS = "detectron2://COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x/137849600/model_final_f10217.pkl"  # initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 1500
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1  # only has one class

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Solution 5B

Compare the pretrained model (given in the code) and your train-from-scratch model

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "pretrain.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set the testing threshold for this model
cfg.DATASETS.TEST = ("plate_val", )
predictor = DefaultPredictor(cfg)
```

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set the testing threshold for this model
cfg.DATASETS.TEST = ("plate_val", )
predictor = DefaultPredictor(cfg)
```

Solution 5B (pretrain result) Refer to Prob5D in the sample problems.

```
from detectron2.utils.visualizer import ColorMode
dataset_dicts = get_plate_dicts("DSS_TEST")
for d in dataset_dicts:
    if d["file_name"] == 'DSS_TEST/1037.jpg':
        annos=d.get("annotations", None)
        boxes = [BoxMode.convert(x["bbox"], x["bbox_mode"], BoxMode.XYXY_ABS) for x in annos]
        print(boxes)

        im = cv2.imread('./DSS_TEST/1037.jpg')
        outputs = predictor(im)

        v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
        out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
        bbox = outputs["instances"].pred_boxes.to("cpu")
        print(bbox)

    elif d["file_name"] == 'DSS_TEST/15.jpg':
        annos=d.get("annotations", None)
        boxes = [BoxMode.convert(x["bbox"], x["bbox_mode"], BoxMode.XYXY_ABS) for x in annos]
        print(boxes)

        im = cv2.imread('./DSS_TEST//15.jpg')
        outputs = predictor(im)

        v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
        out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
        bbox = outputs["instances"].pred_boxes.to("cpu")
        print(bbox)
```

```
[[616, 859, 747, 929]]
```

G.T.

```
Boxes(tensor([[ 610.3941,  846.3649,  763.7913,  940.7380],
        [1023.8463,  594.4886, 1081.0227,  624.0331],
        [ 589.4059,  684.5179,  639.3547,  714.4788]]))
```

Predict

IOU: 0.633437371308994

```
[[898, 437, 1030, 518]]
```

G.T.

```
Boxes(tensor([[ 899.3004,  432.6227, 1052.1821,  518.4718],
        [1682.3522,  925.6249, 1750.5143,  971.2185]]))
```

Predict

IOU: 0.8001962646564795

Solution 5B (pretrain result)

```
elif  d["file_name"]  ==  'DSS_TEST/174.jpg':
    annos=d.get("annotations",  None)
    boxes  =  [BoxMode.convert(x["bbox"],  x["bbox_mode"],  BoxMode.XYXY_ABS)  for  x  in  annos]
    print(boxes)

    im  =  cv2.imread('./DSS_TEST//174.jpg')
    outputs  =  predictor(im)

    v  =  Visualizer(im[:, :, ::-1],  MetadataCatalog.get(cfg.DATASETS.TRAIN[0]),  scale=1.2)
    out  =  v.draw_instance_predictions(outputs["instances"].to("cpu"))
    bbox  =  outputs["instances"].pred_boxes.to("cpu")
    print(bbox)
elif  d["file_name"]  ==  'DSS_TEST/285.jpg':
    annos=d.get("annotations",  None)
    boxes  =  [BoxMode.convert(x["bbox"],  x["bbox_mode"],  BoxMode.XYXY_ABS)  for  x  in  annos]
    print(boxes)

    im  =  cv2.imread('./DSS_TEST//285.jpg')
    outputs  =  predictor(im)

    v  =  Visualizer(im[:, :, ::-1],  MetadataCatalog.get(cfg.DATASETS.TRAIN[0]),  scale=1.2)
    out  =  v.draw_instance_predictions(outputs["instances"].to("cpu"))
    bbox  =  outputs["instances"].pred_boxes.to("cpu")
    print(bbox)
elif  d["file_name"]  ==  'DSS_TEST/887.jpg':
    annos=d.get("annotations",  None)
    boxes  =  [BoxMode.convert(x["bbox"],  x["bbox_mode"],  BoxMode.XYXY_ABS)  for  x  in  annos]
    print(boxes)

    im  =  cv2.imread('./DSS_TEST/887.jpg')
    outputs  =  predictor(im)

    v  =  Visualizer(im[:, :, ::-1],  MetadataCatalog.get(cfg.DATASETS.TRAIN[0]),  scale=1.2)
    out  =  v.draw_instance_predictions(outputs["instances"].to("cpu"))
    bbox  =  outputs["instances"].pred_boxes.to("cpu")
    print(bbox)
```

```
[[1445, 570, 1562, 640]]
```

G.T.

```
Boxes(tensor([[1456.3356,  567.6526, 1576.4139,  640.3604]]))
```

Predict

IOU:

```
0.7766079393461236
```

```
[[1129, 453, 1199, 492]]
```

G.T.

```
Boxes(tensor([[1131.6628,  451.7733, 1196.1841,  486.2426],
        [ 728.9242,  497.3344,  777.7681,  523.4725]]))
```

Predict

IOU:

```
0.763525294707819
```

```
[[653, 726, 740, 779]]
```

G.T.

```
Boxes(tensor([[647.6771, 726.8718, 737.7940, 778.1450],
        [ 38.7002, 210.5602, 129.2993, 264.2011]]))
```

Predict

IOU:

```
0.8901984137518422
```

Solution 5B (train-from-scratch result)

```python
from detectron2.utils.visualizer import ColorMode
dataset_dicts = get_plate_dicts("DSS_TEST")
for d in dataset_dicts:
    if d["file_name"] == 'DSS_TEST/1037.jpg':
        annos=d.get("annotations", None)
        boxes = [BoxMode.convert(x["bbox"], x["bbox_mode"], BoxMode.XYXY_ABS) for x in annos]
        print(boxes)

        im = cv2.imread('./DSS_TEST/1037.jpg')
        outputs = predictor(im)

        v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
        out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
        bbox = outputs["instances"].pred_boxes.to("cpu")
        print(bbox)

    elif d["file_name"] == 'DSS_TEST/15.jpg':
        annos=d.get("annotations", None)
        boxes = [BoxMode.convert(x["bbox"], x["bbox_mode"], BoxMode.XYXY_ABS) for x in annos]
        print(boxes)

        im = cv2.imread('./DSS_TEST//15.jpg')
        outputs = predictor(im)

        v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
        out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
        bbox = outputs["instances"].pred_boxes.to("cpu")
        print(bbox)
```

```
[[616, 859, 747, 929]]
```

G.T.

```
Boxes(tensor([[ 617.9656,  854.7482,  749.3472,  932.1384],
        [1029.3499,  596.4106, 1079.4072,  621.1848],
        [ 591.4240,  687.7878,  637.0215,  713.3369]]))
```

Predict

IOU:

```
0.8764868879107567
```

```
[[898, 437, 1030, 518]]
```

G.T.

```
Boxes(tensor([[ 906.5258,  433.1608, 1033.8909,  516.6366],
        [1681.0577,  914.6115, 1751.5081,  978.3536]]))
```

Predict

IOU:

```
0.8557309185712306
```

Solution 5B (train-from-scratch result)

```
elif d["file_name"] == 'DSS_TEST/174.jpg':
    annos=d.get("annotations", None)
    boxes = [BoxMode.convert(x["bbox"], x["bbox_mode"], BoxMode.XYXY_ABS) for x in annos]
    print(boxes)

    im = cv2.imread('./DSS_TEST//174.jpg')
    outputs = predictor(im)

    v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    bbox = outputs["instances"].pred_boxes.to("cpu")
    print(bbox)
elif d["file_name"] == 'DSS_TEST/285.jpg':
    annos=d.get("annotations", None)
    boxes = [BoxMode.convert(x["bbox"], x["bbox_mode"], BoxMode.XYXY_ABS) for x in annos]
    print(boxes)

    im = cv2.imread('./DSS_TEST//285.jpg')
    outputs = predictor(im)

    v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    bbox = outputs["instances"].pred_boxes.to("cpu")
    print(bbox)
elif d["file_name"] == 'DSS_TEST/887.jpg':
    annos=d.get("annotations", None)
    boxes = [BoxMode.convert(x["bbox"], x["bbox_mode"], BoxMode.XYXY_ABS) for x in annos]
    print(boxes)

    im = cv2.imread('./DSS_TEST/887.jpg')
    outputs = predictor(im)

    v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    bbox = outputs["instances"].pred_boxes.to("cpu")
    print(bbox)
```

[[1445, 570, 1562, 640]] G.T.

Boxes(tensor([[1457.2549, 568.6931, 1567.3303, 636.7687]])) Predict

IOU: 0.8048206613556255

[[1129, 453, 1199, 492]] G.T.

Boxes(tensor([[1130.3367, 451.3373, 1202.6555, 491.5055],
[729.2797, 498.7573, 773.1477, 520.5765]])) Predict

IOU: 0.8839563783961586

[[653, 726, 740, 779]] G.T.

Boxes(tensor([[653.4498, 724.0403, 735.4070, 779.3485],
[32.9130, 193.5313, 137.9621, 276.0954]])) Predict

IOU: 0.904911354001782

Correct answer rate:2/16

Solution 5C

Average precision is a measure that combines recall and precision for ranked retrieval results. For one information need, the average precision is the mean of the precision scores after each relevant document is retrieved.

$$Average\ Precision = \frac{\sum_r P@r}{R}$$

where r is the rank of each relevant document, R is the total number of relevant documents, and $P@r$ is the precision of the top-r retrieved documents.

Correct answer rate:7/16

Solution 5D

G.T.

Predict

Threshold 0.1

G.T.

```
[[616, 859, 747, 929]]
```

Predict

```
Boxes(tensor([[ 610.3941,  846.3649,  763.7913,  940.7380],
        [1023.8463,  594.4886, 1081.0227,  624.0331],
        [ 589.4059,  684.5179,  639.3547,  714.4788],
        [ 614.0393,  795.1594,  827.4500,  994.3911],
        [1442.4265,  500.7701, 1501.3544,  529.8929],
        [ 596.9515,  684.5768,  633.8301,  704.8317],
        [1436.3459,  489.6292, 1499.8990,  545.6685],
        [1040.9517,  595.4321, 1085.1411,  617.5831],
        [ 571.7222,  680.3185,  644.3244,  724.0168]]))
```

Threshold 0.3

G.T.

```
[[616, 859, 747, 929]]
```

Predict

```
Boxes(tensor([[ 610.3941,  846.3649,  763.7913,  940.7380],
        [1023.8463,  594.4886, 1081.0227,  624.0331],
        [ 589.4059,  684.5179,  639.3547,  714.4788],
        [ 614.0393,  795.1594,  827.4500,  994.3911]]))
```

Threshold 0.5

G.T.

```
[[616, 859, 747, 929]]
```

Predict

```
Boxes(tensor([[ 610.3941,  846.3649,  763.7913,  940.7380],
        [1023.8463,  594.4886, 1081.0227,  624.0331],
        [ 589.4059,  684.5179,  639.3547,  714.4788],
        [ 614.0393,  795.1594,  827.4500,  994.3911]]))
```

Threshold 0.7

G.T.

```
[[616, 859, 747, 929]]
```

Predict

```
Boxes(tensor([[ 610.3941,  846.3649,  763.7913,  940.7380],
        [1023.8463,  594.4886, 1081.0227,  624.0331],
        [ 589.4059,  684.5179,  639.3547,  714.4788]]))
```

Precision

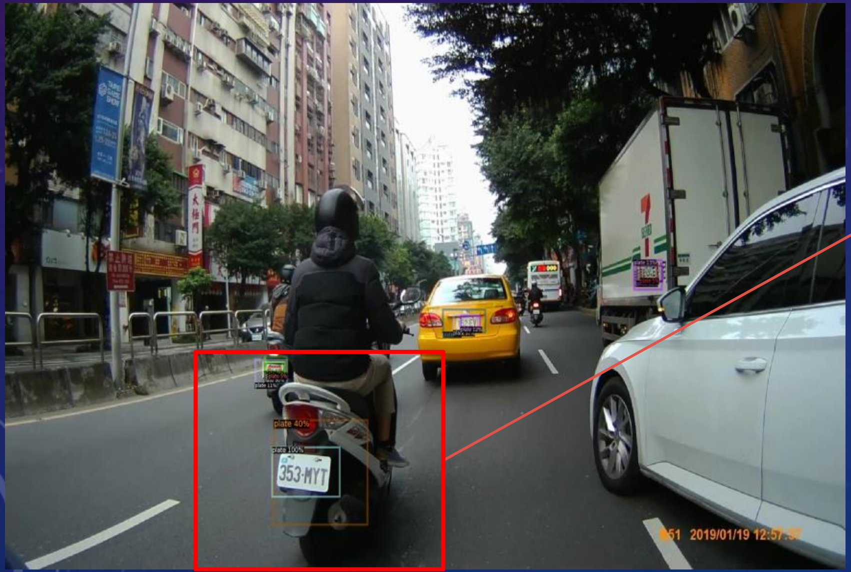
Threshold 0.1 :2/(2+7)=0.22, Threshold 0.3 :2/(2+2)=0.5

Threshold 0.5 :1/(1+3)=0.25, Threshold 0.7 :1/(1+2)=0.33

Solution 5D

G.T.

Predict



Threshold 0.05

Solution 5D

G.T.

Predict

G.T.

```
[[898, 437, 1030, 518]]
```

Predict

Threshold 0.1

```
Boxes(tensor([[ 899.3004,  432.6227, 1052.1821,  518.4718],
        [1682.3522,  925.6249, 1750.5143,  971.2185],
        [1644.0000,  859.3284, 1685.4377,  892.8536],
        [1679.0481,  908.3776, 1747.6565,  998.6457],
        [1633.3645,  851.0414, 1692.8724,  904.2740],
        [1692.0114,  934.5619, 1737.9836,  967.4363],
        [ 884.5193,  389.2024, 1119.4900,  570.8593]]))
```

G.T.

```
[[898, 437, 1030, 518]]
```

Predict

Threshold 0.3

```
Boxes(tensor([[ 899.3004,  432.6227, 1052.1821,  518.4718],
        [1682.3522,  925.6249, 1750.5143,  971.2185],
        [1644.0000,  859.3284, 1685.4377,  892.8536],
        [1679.0481,  908.3776, 1747.6565,  998.6457],
        [1633.3645,  851.0414, 1692.8724,  904.2740]]))
```

G.T.

```
[[898, 437, 1030, 518]]
```

Predict

Threshold 0.5

```
Boxes(tensor([[ 899.3004,  432.6227, 1052.1821,  518.4718],
        [1682.3522,  925.6249, 1750.5143,  971.2185],
        [1644.0000,  859.3284, 1685.4377,  892.8536]]))
```

G.T.

```
[[898, 437, 1030, 518]]
```

Predict

Threshold 0.7

```
Boxes(tensor([[ 899.3004,  432.6227, 1052.1821,  518.4718],
        [1682.3522,  925.6249, 1750.5143,  971.2185]]))
```

Precision

Threshold 0.1 :2/(2+5)=0.29, Threshold 0.3 :1/(1+4)=0.2

Threshold 0.5 :1/(1+2)=0.33, Threshold 0.7 :1/(1+1)=0.5

Solution 5D

G.T.

Predict

G.T.
[[1445, 570, 1562, 640]

Predict
Threshold 0.1
Boxes(tensor([[1456.3356, 567.6526, 1576.4139, 640.3604],
[1441.2900, 537.8801, 1620.4365, 675.8806]]))

G.T.
[[1445, 570, 1562, 640]]

Predict
Threshold 0.3
Boxes(tensor([[1456.3356, 567.6526, 1576.4139, 640.3604]]))

G.T.
[[1445, 570, 1562, 640]]

Predict
Threshold 0.5
Boxes(tensor([[1456.3356, 567.6526, 1576.4139, 640.3604]]))

G.T.
[[1445, 570, 1562, 640]]

Predict
Threshold 0.7
Boxes(tensor([[1456.3356, 567.6526, 1576.4139, 640.3604]]))

Precision

Threshold 0.1 :2/(2+0)=1, Threshold 0.3 :1/(1+0)=1

Threshold 0.5 :1/(1+0)=1, Threshold 0.7 :1/(1+0)=1

Solution 5D

G.T.

Predict

G.T.

Predict

Threshold 0.1

```
[[1129,  453, 1199,  492]]
Boxes(tensor([[1131.6628,  451.7733, 1196.1841,  486.2426],
          [ 728.9242,  497.3344,  777.7681,  523.4725],
          [1229.3875,  519.2971, 1279.4043,  545.2972],
          [ 722.2908,  491.9247,  779.2850,  537.5136],
          [1129.1610,  445.4526, 1209.6324,  501.2574],
          [ 246.2485,  346.6805,  285.1299,  370.5841],
          [1207.5085,  517.2770, 1288.6174,  554.7692],
          [1247.3260,  517.0416, 1280.3325,  538.0410],
          [1110.5310,  518.2087, 1173.2343,  545.2712]]))
```

G.T.

Predict

Threshold 0.3

```
[[1129,  453, 1199,  492]]
Boxes(tensor([[1131.6628,  451.7733, 1196.1841,  486.2426],
          [ 728.9242,  497.3344,  777.7681,  523.4725],
          [1229.3875,  519.2971, 1279.4043,  545.2972],
          [ 722.2908,  491.9247,  779.2850,  537.5136]]))
```

G.T.

Predict

Threshold 0.5

```
[[1129,  453, 1199,  492]]
Boxes(tensor([[1131.6628,  451.7733, 1196.1841,  486.2426],
          [ 728.9242,  497.3344,  777.7681,  523.4725],
          [1229.3875,  519.2971, 1279.4043,  545.2972]]))
```

G.T.

Predict

Threshold 0.7

```
[[1129,  453, 1199,  492]]
Boxes(tensor([[1131.6628,  451.7733, 1196.1841,  486.2426],
          [ 728.9242,  497.3344,  777.7681,  523.4725]]))
```

Precision

Threshold 0.1 :2/(2+7)=0.22, Threshold 0.3 :1/(1+3)=0.25

Threshold 0.5 :1/(1+2)=0.33, Threshold 0.7 :1/(1+1)=0.5

Solution 5D

G.T.

Predict

G.T. [[653, 726, 740, 779]]

Predict

Threshold 0.1

```
Boxes(tensor([[ 647.6771,   726.8718,   737.7940,   778.1450],
        [  38.7002,   210.5602,   129.2993,   264.2011],
        [ 151.1669,   209.3624,   222.6788,   239.9454],
        [   8.2231,   203.5128,   206.5420,   266.6965],
        [1493.5713,  1000.1161,  1528.9336,  1019.8029]]))
```

G.T. [[653, 726, 740, 779]]

Predict

Threshold 0.3

```
Boxes(tensor([[647.6771, 726.8718, 737.7940, 778.1450],
        [ 38.7002, 210.5602, 129.2993, 264.2011],
        [151.1669, 209.3624, 222.6788, 239.9454]]))
```

G.T. [[653, 726, 740, 779]]

Predict

Threshold 0.5

```
Boxes(tensor([[647.6771, 726.8718, 737.7940, 778.1450],
        [ 38.7002, 210.5602, 129.2993, 264.2011]]))
```

G.T. [[653, 726, 740, 779]]

Predict

Threshold 0.7

```
Boxes(tensor([[647.6771, 726.8718, 737.7940, 778.1450],
        [ 38.7002, 210.5602, 129.2993, 264.2011]]))
```

Precision

Threshold 0.1 :1/(1+4)=0.2 Threshold 0.3 :1/(1+2)=0.33

Threshold 0.5 :1/(1+1)=0.5 Threshold 0.7 :1/(1+1)=0.5

Correct answer rate:0/16

Solution 5D

Average Precision

Threshold 0.1 : (0.22+0.29+1+0.22+0.2)/5=0.386

Threshold 0.3 : (0.5+0.2+1+0.25+0.33)/5=0.456

Threshold 0.5 : (0.25+0.33+1+0.33+0.5)/5=0.482

Threshold 0.7 : (0.33+0.5+1+0.5+0.5)/5=0.566

Solution 5E

- It is found that when the threshold is higher, the average precision will be higher, too.

Correct answer rate: 3/16