Anil Poonai

Github link for files and code: https://github.com/DevonARP/DeepLearning_A2

For question 1 I wrote out all my notes and work before writing them here, I'll add my work/notes at the end of each part for this question

Problem 1: Part A

$L(w_1,w_2) = .5(aw_1^2 + bw_2^2)$

$\nabla L(w_1,w_2) = aw_1 + bw_2$

$\nabla^2 L(w_1,w_2) = a + b$

I'm making a = 1 and b = 2 to keep the double derivate positive and keep the graph concave up so it has a minimum value.

$\nabla^2 L(w_1,w_2) = a + b = 1 + 2 = 3$

Now, we go back to the first derivative

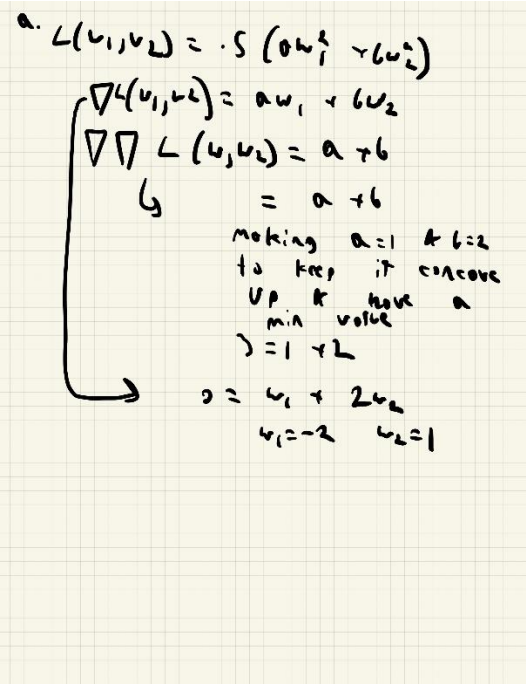
$\nabla L(w_1,w_2) = aw_1 + bw_2 = w_1 + 2w_2$

I can use $w_1 = -2$ and $w_2 = 1$, this gives the minimum value of 0.

References:

https://study.com/learn/lesson/how-to-find-the-maximum-value-of-a-function.html#:~:text=We%20will%20set%20the%20first,will%20be%20a%20minimum%20value

https://www.ocf.berkeley.edu/~reinholz/ed/07fa_m155/lectures/second_derivative.pdf



Problem 1: Part B

P_i is a dropout layer, I'll be using N for the dropout rate

Have to grab the derivative of L with respect to the corresponding weight

$W_i(t+1) = w_i(t) – N(\partial L/\partial w_i)$ This is being rearranged to follow the dropout rate argument formula

$W_1 = w_1(t) – Naw_1(t) = w_1(t)(1-Na) = p_1w_1(t)$

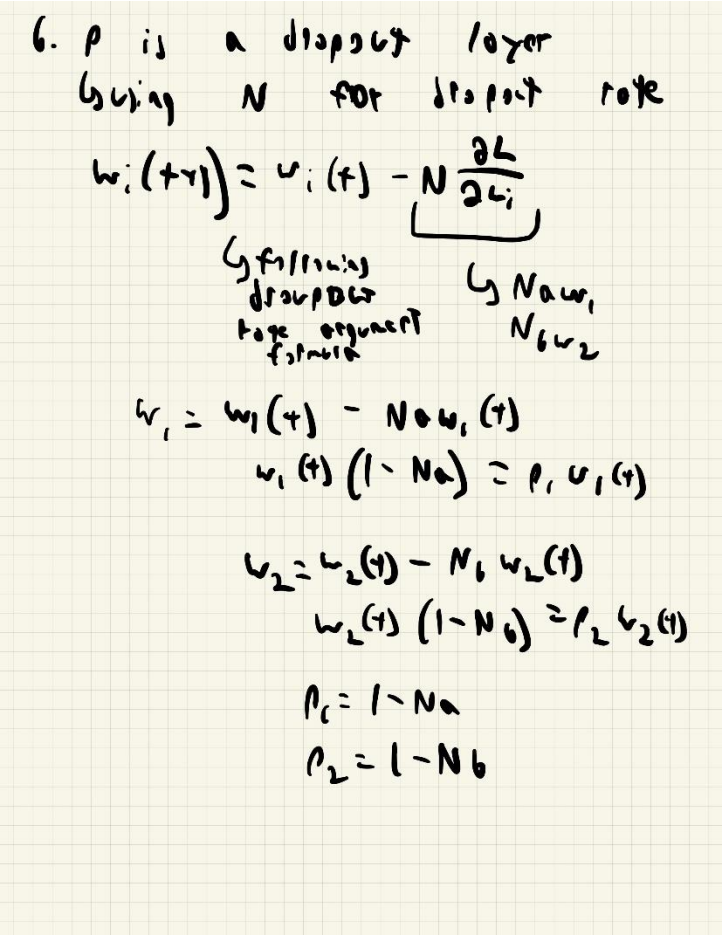
$W_2 = w_2(t) – Nbw_2(t) = w_2(t)(1-Nb) = p_2w_2(t)$

$P_1 = 1 – Na$

$P_2 = 1 – Nb$

References:

https://towardsdatascience.com/simplified-math-behind-dropout-in-deep-learning-6d50f3f47275#:~:text=In%20Keras%2C%20the%20dropout%20rate,can%20adversely%20affect%20the%20training.



Problem 1: Part C

It converges when the gradient of the cost function becomes 0.

In this case both p_1 and p_2 need to be 0

We can rearrange that to be 1 – Na and 1 – Nb equal 0

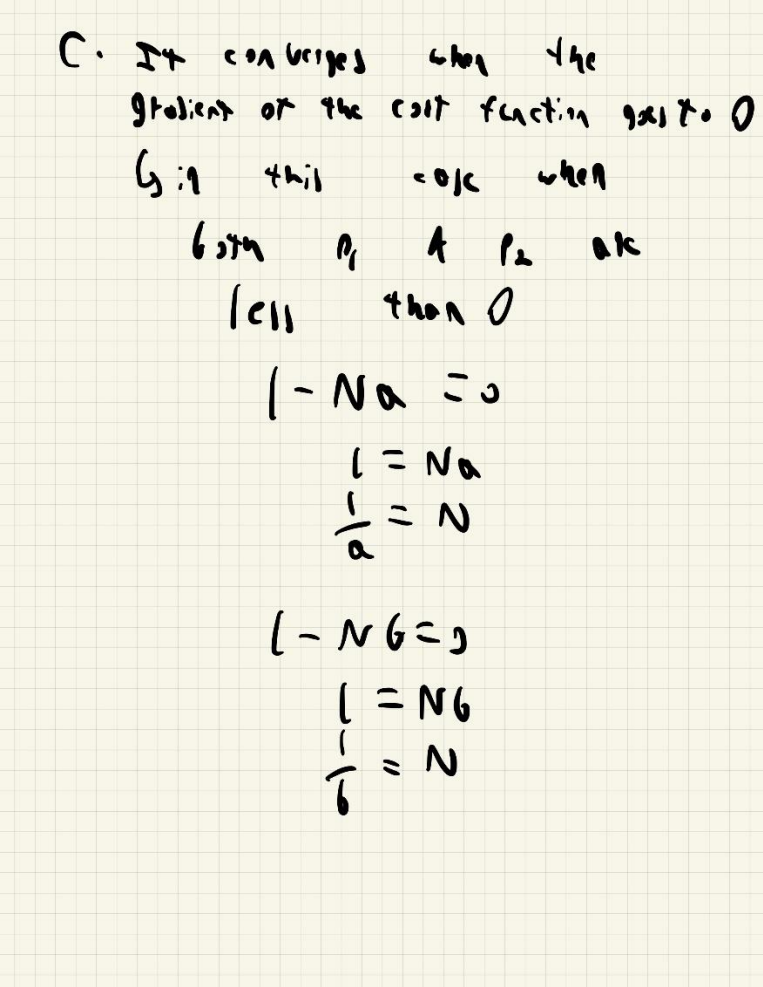
Then we can make it so that Na and Nb both equal 1

And we can end off with N equals 1/a and 1/b.

References:

https://www.cs.umd.edu/~djacobs/CMSC426/GradientDescent.pdf

https://www.cs.ubc.ca/~schmidtm/Courses/540-W18/L4.pdf



Problem 1: Part D

When either a/b or b/a is very large, in the first case, a/b, then a would be the larger number and N for w_1 would be comparatively smaller than w_2, and therefore w_1 would have to have more updates than w_2. The opposite is true for the second case, b would be the larger number and N for w_2 would be comparatively smaller than w_1, leaving w_2 to have more updates. The cases of a/b and b/a being very small is just the same case as the b/a and a/b being very large respectively.

Problem 2: Part A

I would use the Sobel filter, which uses two kernels, one for the horizontal edges and one for the vertical edges.

Horizontal edges

-1	0	1
-2	0	2
1	0	1

Vertical edges

-1	-2	-1
0	0	0
1	2	1

This works by looking for string changes in the image. The higher the sum of the numbers after the convolution, the more likely there is an edge there and the positive or negative sign indicates the direction of the edge. The output form both filters are then combined to see all the edges detected.

References:

https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection

https://automaticaddison.com/how-the-sobel-operator-works/

https://www.cs.auckland.ac.nz/compsci373s1c/PatricesLectures/Edge%20detection-Sobel_2up.pdf

Problem 2: Part B

I'm going to use a Box Blur Kernel because I end up mentioning the Gaussian Blur Kernel in Part D.

1	1	1
1	1	1
1	1	1

This works by giving each pixel the same weight and adding them all up then dividing by 9 in this case at the end, this makes it so that the output is a relative value to the other output points after the kernel as it adds up all of the points in a region and averages it out for every region.

References:

https://medium.com/hackernoon/cv-for-busy-developers-convolutions-5c984f216e8c#:~:text=The%20convolution%20of%20a%20Gaussian,the%20kernel%20values%20is%2016.

Problem 2: Part C

I'm combining some concepts from regular sharpening and edge detection for this.

0	-1	0
0	2	0
0	-1	0

We're focusing on the horizontal sharpening by grabbing the middle row of the region, this focuses on the center primarily and the middle row right after, with no focus on anything else as those values on the filter are zero.

References:

https://medium.com/@boelsmaxence/introduction-to-image-processing-filters-179607f9824a

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Problem 2: Part D

I'm going to use a Gaussian Blur Kernel, this can also be used to blue an image.

1	2	1
2	4	2
1	2	1

This works by giving the pixel near the center of the kernel more weight than the ones on the edges, this helps mute noise as the box blur would treat all the points with the same weight.

References:

https://medium.com/hackernoon/cv-for-busy-developers-convolutions-5c984f216e8c#:~:text=The%20convolution%20of%20a%20Gaussian,the%20kernel%20values%20is%2016.

Problem 3: Part A

Jaccard similarity is supposed to find how similar 2 sets of data are, so it has to range between 0, no correlation at all, and 1, they are the exact same set. It can't be out of that range since the extremes would be either 0 and 1 for no relation and 100% relation, any number between 0 and 1 would represent a partial match with the higher the value being the higher match percentage. This can also be described as the IOU being the division of the overlap of the predicted and ground truth, so it can only be between the ranges 0 and 1.

IOU = $|A \cap B| / |A \cup B|$ #A is the predicted and B is the ground truth

B counts for all true positives, false negatives, and false positives while A counts for true positives

Problem 3: Part B

IOU isn't differentiable inherently as it ranges from 0 to 1, which means it can't differentiate between distances from how similar sets are as you would need all real numbers available. It also isn't differentiable in the case mentioned because the parameters have no influence on it's gradient, so in the case of the top left to bottom right corners, the gradients are just going to be 0. Also, using the equation mentioned above indicates that the gradients would just be 0 everywhere as well, the parameters don't play a role in IOU calculation, it also is because the IOU metric itself isn't continuous natively.

Problem 4:

Code and answers will be below in an attached pdf to this document

Problem 5: Part A

Code and answers will be below in an attached pdf to this document

Problem 5: Part B

Code will be below in an attached pdf to this document

The backbone model is slightly better regarding loss after training as it has a lower loss and loss classifier but when looking at the example image it actually picks up 1 less person than the finetuning model but it is more confident in it's predictions. This also leads to the IOU metric being an indicator, with the higher values being given towards the finetuning model, meaning it actually ends up matching more objects with that model. I'm not saying which is better as that's depends on what outcome is wanted but I do like the results from the finetuning a model a bit more, the IOU metric helps a lot.

Finetuning model

Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.1793 (0.1891) loss_classifier: 0.0253 (0.0269) loss_box_reg: 0.0378 (0.0417) loss_mask: 0.1127 (0.1169) loss_objectness: 0.0004 (0.0006) loss_rpn_box_reg: 0.0029 (0.0030) time: 0.5977 data: 0.0096 max mem: 3778

IoU metric: bbox

Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.839

Average Precision (AP) @[IoU=0.50 | area= all | maxDets=100] = 0.979

Average Precision (AP) @[IoU=0.75 | area= all | maxDets=100] = 0.931

Average Precision (AP) @[IoU=0.50:0.95 | area= small | maxDets=100] = 0.355

Average Precision (AP) @[IoU=0.50:0.95 | area=medium | maxDets=100] = 0.659

Average Precision (AP) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.860

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 1] = 0.424

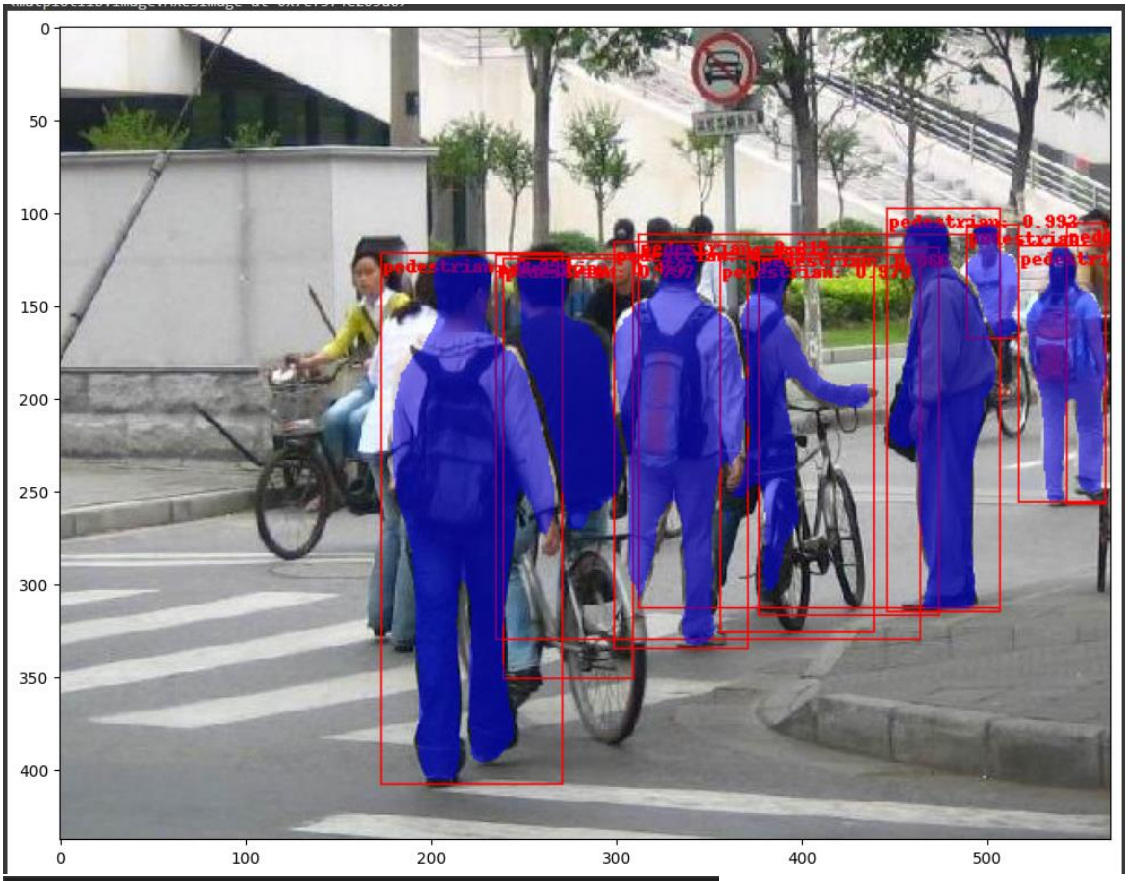
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 10] = 0.877

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.877

Average Recall (AR) @[IoU=0.50:0.95 | area= small | maxDets=100] = 0.467

Average Recall (AR) @[IoU=0.50:0.95 | area=medium | maxDets=100] = 0.825

Average Recall (AR) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.894




```
pred_labels
```

```
['pedestrian: 0.995',
 'pedestrian: 0.994',
 'pedestrian: 0.992',
 'pedestrian: 0.988',
 'pedestrian: 0.979',
 'pedestrian: 0.797',
 'pedestrian: 0.453',
 'pedestrian: 0.215',
 'pedestrian: 0.089',
 'pedestrian: 0.066',
 'pedestrian: 0.055']
```

```
[14] pred_boxes
```

```
tensor([[299, 116, 371, 335],
        [173, 122, 271, 408],
        [446,  98, 507, 315],
        [517, 118, 564, 256],
        [356, 125, 439, 326],
        [239, 125, 309, 351],
        [235, 123, 464, 330],
        [312, 112, 507, 313],
        [489, 107, 517, 168],
        [377, 119, 474, 317],
        [543, 106, 564, 257]], device='cuda:0')
```

Backbone Model

Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.1621 (0.1808) loss_classifier: 0.0219 (0.0247) loss_box_reg: 0.0326 (0.0377) loss_mask: 0.1089 (0.1143) loss_objectness: 0.0002 (0.0011) loss_rpn_box_reg: 0.0033 (0.0029) time: 0.5792 data: 0.0088 max mem: 3780

IoU metric: bbox

Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.806

Average Precision (AP) @[IoU=0.50 | area= all | maxDets=100] = 0.975

Average Precision (AP) @[IoU=0.75 | area= all | maxDets=100] = 0.919

Average Precision (AP) @[IoU=0.50:0.95 | area= small | maxDets=100] = 0.348

Average Precision (AP) @[IoU=0.50:0.95 | area=medium | maxDets=100] = 0.700

Average Precision (AP) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.830

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 1] = 0.354

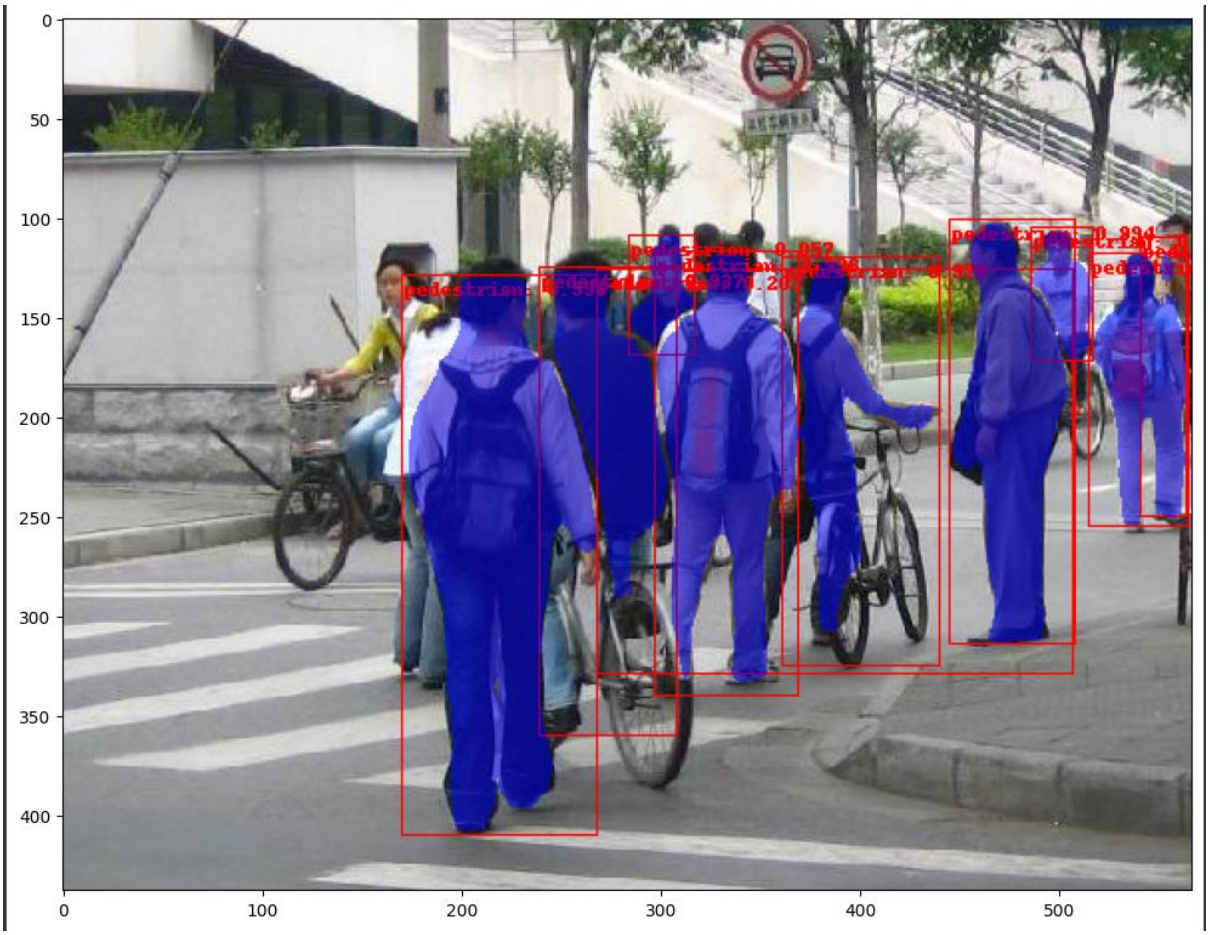
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 10] = 0.853

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.853

Average Recall (AR) @[IoU=0.50:0.95 | area= small | maxDets=100] = 0.467

Average Recall (AR) @[IoU=0.50:0.95 | area=medium | maxDets=100] = 0.775

Average Recall (AR) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.872



```
pred_labels
```

```
['pedestrian: 0.996',
 'pedestrian: 0.996',
 'pedestrian: 0.994',
 'pedestrian: 0.987',
 'pedestrian: 0.976',
 'pedestrian: 0.937',
 'pedestrian: 0.203',
 'pedestrian: 0.181',
 'pedestrian: 0.119',
 'pedestrian: 0.057']
```

```
[14] pred_boxes
```

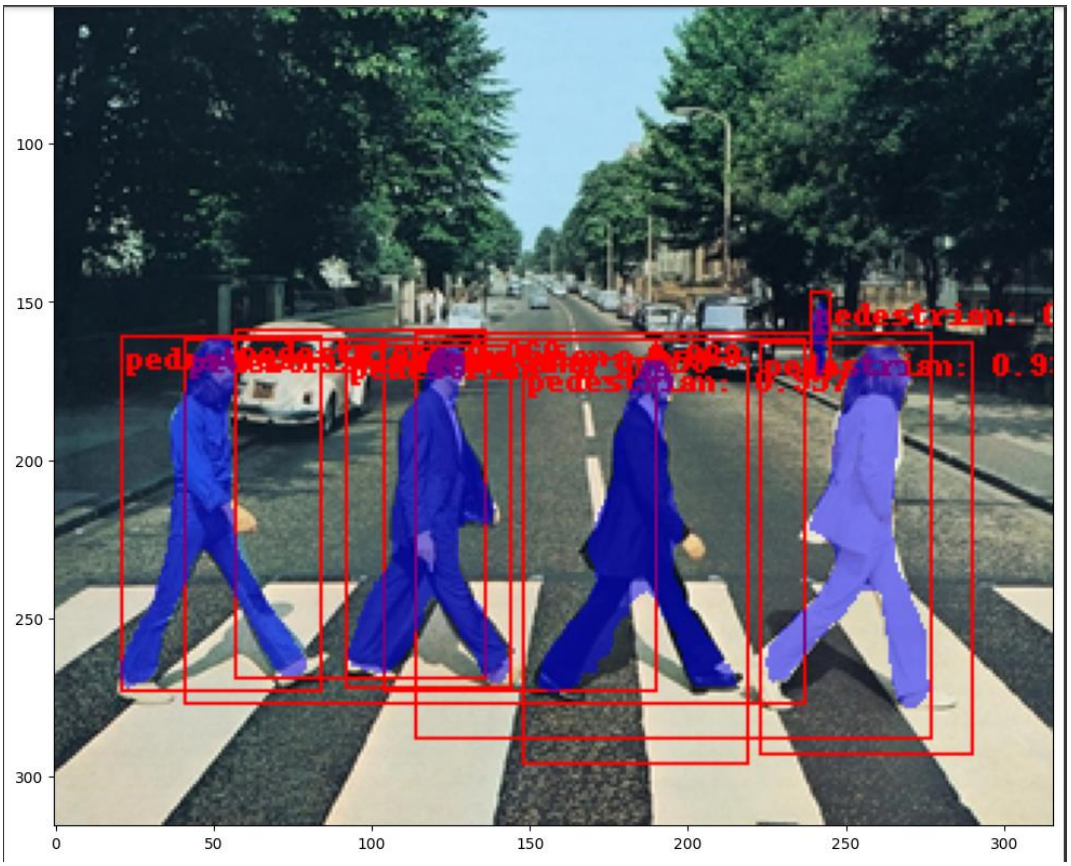
```
tensor([[170, 129, 268, 410],
        [297, 117, 369, 340],
        [445, 101, 508, 314],
        [515, 118, 565, 255],
        [361, 120, 440, 325],
        [239, 125, 308, 360],
        [268, 126, 507, 329],
        [486, 105, 517, 172],
        [541, 110, 564, 250],
        [284, 109, 317, 169]], device='cuda:0')
```

Problem 5: Part C

Code will be below in an attached pdf to this document

There are 5 people in the photograph, 4 are in the center and easy to spot but there's a fifth on the right side a bit in the back, the finetuning model picked up on that and with low confidence but the backbone model only recognized 4 people, being the 4 Beatles members in the middle of the photograph. The finetuning model also picked up some errors, it had bounding boxes around multiple Beatles members as possible objects but gave it a really low confidence score. So the finetuning model performed better here as it caught the 4 members with high confidence, a person in the back with fairly low confidence and had 4 other bounding boxes with multiple people in them but gave those boxes a really low confidence score, I would calculate those as False Positives. The backbone model performed worst not only because it missed a person entirely but because the confidence it had about the 4 Beatles members in the center of the image was low, it gave George Harrison a confidence score of less than .5 but the rest as above .9. All that being said is it important to point out that the finetuning model does have a higher error rate. Again, I'm not saying which is better but I still like the finetuning model a bit more, it might have more errors but it is more confident bout detecting objects and can pick up on hard to see objects.

Finetuning Model



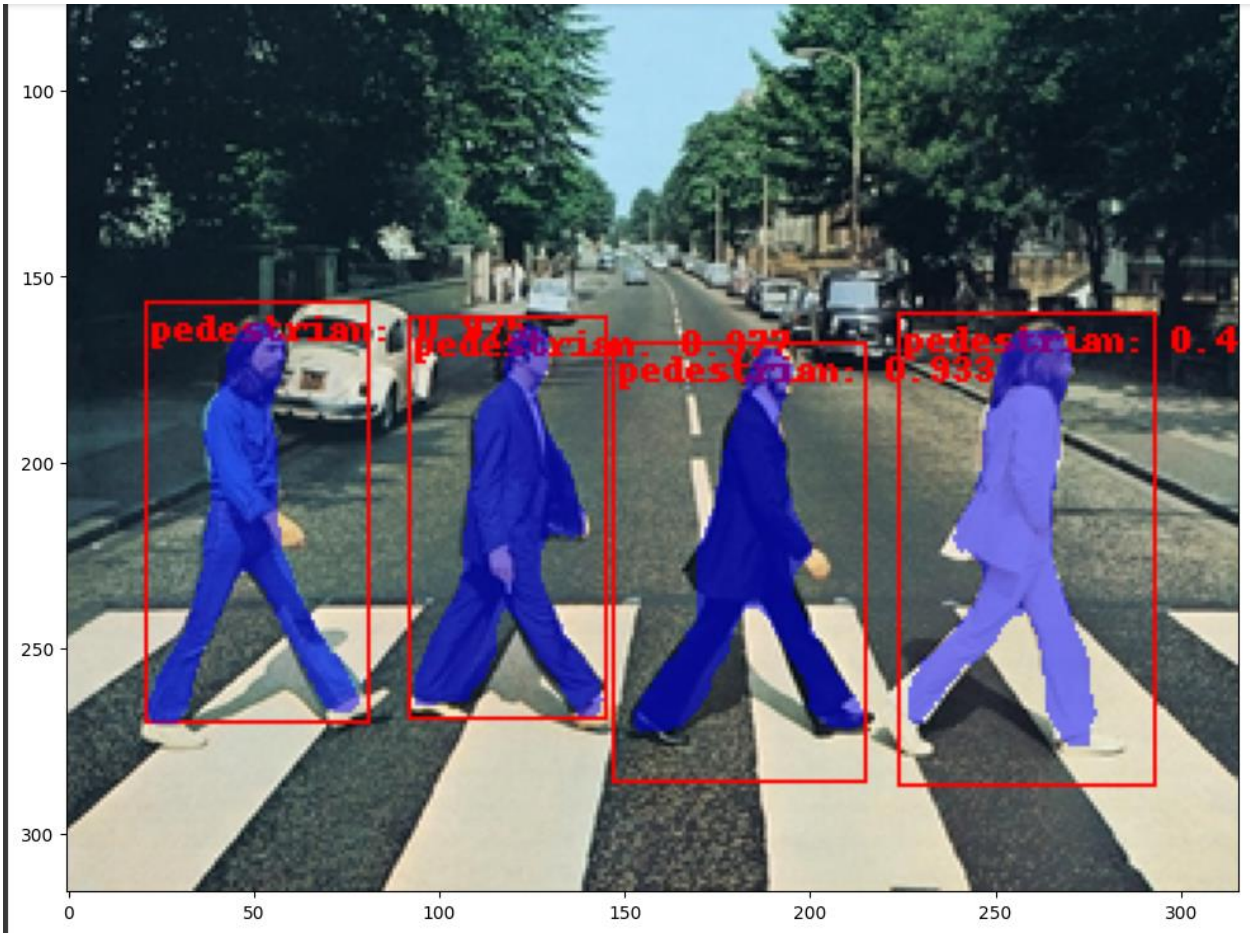
```
[10] pred_labels
```

```
['pedestrian: 0.987',
 'pedestrian: 0.985',
 'pedestrian: 0.957',
 'pedestrian: 0.939',
 'pedestrian: 0.103',
 'pedestrian: 0.093',
 'pedestrian: 0.089',
 'pedestrian: 0.060',
 'pedestrian: 0.059']
```

```
[11] pred_boxes
```

```
tensor([[ 21, 161,  84, 273],
        [ 92, 164, 144, 272],
        [148, 168, 219, 296],
        [223, 163, 290, 293],
        [ 41, 162, 237, 277],
        [239, 147, 245, 174],
        [114, 160, 277, 288],
        [ 57, 159, 136, 269],
        [104, 162, 190, 273]], device='cuda:0')
```

Backbone Model




```
pred_labels
```

```
['pedestrian: 0.977',
 'pedestrian: 0.975',
 'pedestrian: 0.933',
 'pedestrian: 0.496']
```

```
[11] pred_boxes
```

```
tensor([[ 92, 161, 145, 269],
        [ 21, 157,  81, 270],
        [147, 168, 215, 286],
        [224, 160, 293, 287]], device='cuda:0')
```