Problem 1: Slow rate of descent (20 pts)
Consider a simple function having two weight variables: L(w1; w2) = 0:5(aw2 1 + bw2 2):
(a) Write down the gradient rL(w), and derive the weights w that achieve the minimum value of L.
So that ∇L(w)=[aw1,bw2]T for minimum value of L, we have **w1=0,w2=0**

(b) Instead of simply writing down the optimal weights, let's now try to optimize L using gradient descent. Starting from some arbitrary initialization point w1(0); w2(0), write down the gradient descent updates. Show that the updates have the form: w1(t + 1) = 1w1(t); w2(t + 1) = 2w2(t) where wi(t) represent the weights at the t th iteration. Derive the expressions for 1 and 2 in terms of a, b, and the learning rate.
First we know that grad_L = [a*w1, b*w2]
Because of the gradient update equation we have: w1(t + 1) = w1(t) - η a * w1(t) w2(t + 1) = w2(t) - η b * w2(t)
So that
λ1=1−ηa
λ2=1−ηb

(c) Under what values of the learning rate does gradient descent converge?
We have that |λ1|<1 and |λ2|<1 so that **0<η<2/max(a,b)**

(d) Provide a scenario under which the convergence rate of gradient descent is very slow.
(*Hint: consider the case where a=b is a very large ratio.*)
The case of the convergence rate of gradient descent is very slow can be true if a/b or b/a is a very large ratio, which means that the function L has a stretched shape. In this case gradient descent will zig-zag slowly towards the minimum, taking tiny steps. This slow progress is worsened if both a and b are very large, as it makes the function L even more stretched, and the steps of gradient descent even tinier.

Problem 2: Designing Convolution Filters by hand (20 pts)
Consider an input 2D input image and a 3x3 lter (say w) applied to this image. The goal is to guess good lters which implement each of the following image processing operations. Write both the weights of the lters and an explanation on how/why they work.
(a) Feature Extraction: The lter should output a high value when the input image has an edge, and a low value otherwise.
[0,1,0; 1,− 4,1; 0,1,0]

This is called the Laplacian filter,output a high value if the input image has an edge, and a low value otherwise.
(b) Blurring Filter: The lter should blur the image.
[1/9,1/9,1/9;1/9,1/9,1/9;1/9,1/9,1/9] this is a averaging filter
(c) Sharpening Filter Horizontal: The lter should sharpen the image Horizontally.
[-1, -1, -1; 2, 2, 2; -1, -1, -1] enhance horizontal edges. The positive weights in the middle row amplify horizontal features

(d) Noise Reduction: The lter should reduce noise in the image.
[1/16, 2/16, 1/16; 2/16, 4/16, 2/16; 1/16, 2/16, 1/16]This is a Gaussian blur filter, weighted by their spatial proximity to the target pixel, which can help to reduce noise.

Problem 3: The IOU Metric(10 pts) In class we discussed the IOU metric (or Jaccard similarity) for evaluating object detection.
(a) Using elementary properties of sets, prove that the IOU metric between any two pair of bouding boxes is always between 0 and 1.

IOU's function is Area of Intersection/Area of union. Since the Area of Intersection is either 0 in the worst case and area of intersection=area of union in the best case. The range of IOU will always remain between 0-1.

(b) If we represent each bounding box as a function of the top-left and bottom-right coordinates, then argue that the IOU metric is non-differentiable and hence cannot be used as a loss function for training a neural network

The IoU metric measures the overlap between two bounding boxes, but can act unpredictably in some situations like when there's no overlap, making the math tricky. When training a neural network, we need a smooth, easy-to-follow rule (a loss function) to guide the learning process. Since the IoU metric isn't smooth and can be tricky, it's not suitable to use as this rule for training the network.

Problem 5: Object Detection (25 pts) We will be finetuning a pretrained Torchvision object detection model in this problem. It will be helpful to go through the excellent tutorial with sample code provided by PyTorch here. You can reuse whatever code you like from the tutorial in your solution with proper attribution.
(a) The tutorial shows how to finetune a pretrained model (what they call option 1). Now follow their approach to add a different backbone (option 2). You can use the same dataset (and code) as the tutorial.
Using Resnet as backbone:

```
backbone = torchvision.models.resnet50(pretrained=True).features
backbone.out_channels = 2048

anchor_generator = AnchorGenerator(
    sizes=((32, 64, 128, 256, 512),),
    aspect_ratios=((0.5, 1.0, 2.0),) * 5
)

roi_pooler = torchvision.ops.MultiScaleRoIAlign(
    featmap_names=['0'],
    output_size=7,
    sampling_ratio=2,
)

model_resnet = FasterRCNN(
    backbone,
    num_classes=2,
    rpn_anchor_generator=anchor_generator,
    box_roi_pool=roi_pooler,
)
```

(b) How does the performance of the two models compare on the training data after 10 epochs?
Total model time is 0:00:14 in optional2 and for Resnet as backbone total model time is worse. Details are listed in q5.py in the github repo.
(c) Test both models on this image of the Beatles. Be careful to make sure that the dimensions align and the scaling is similar. How do the two models compare in terms of performance on this image?
For Beatles,Resnet is better.The tarinset is 66%,test set is 15%. Resnet IOU is 0.63 overall and pretrained one is 0.57