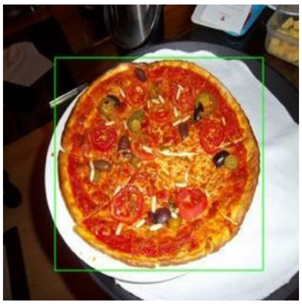
HW5 Report

Cheng-Yun Yang

◆ Creating Your Own Object Localization Dataset

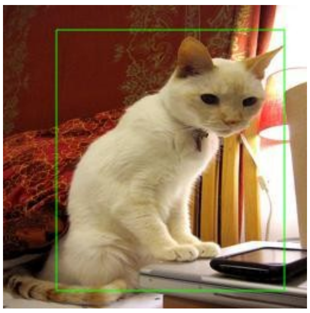
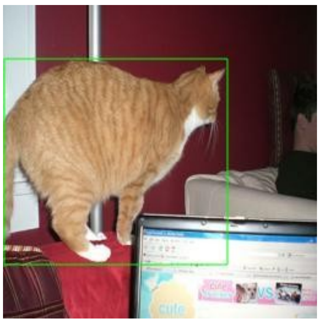
■ Pizza



■ Bus



■ Cat


◆ Building your deep neural network

ResBlock

```python
class ResBlock(nn.Module):
    def __init__(self, input_nc, output_nc, stride=1) -> None:
        super(ResBlock, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(input_nc, output_nc, kernel_size=3, stride=stride, padding=1),
                                   nn.BatchNorm2d(output_nc),
                                   nn.ReLU())
        self.conv2 = nn.Sequential(nn.Conv2d(output_nc, output_nc, kernel_size=3, stride=1, padding=1),
                                   nn.BatchNorm2d(output_nc))
        self.relu = nn.ReLU()
        # Add a 1x1 conv to match the feature map sizes
        self.shortcut = nn.Sequential(nn.Conv2d(input_nc, output_nc, kernel_size=1, stride=stride),
                                      nn.BatchNorm2d(output_nc))
    def forward(self, x):
        residual = self.shortcut(x)
        out = self.conv1(x)
        out = self.conv2(out)
        out = out.clone() + residual
        out = self.relu(out)
        return out
```

HW5Net

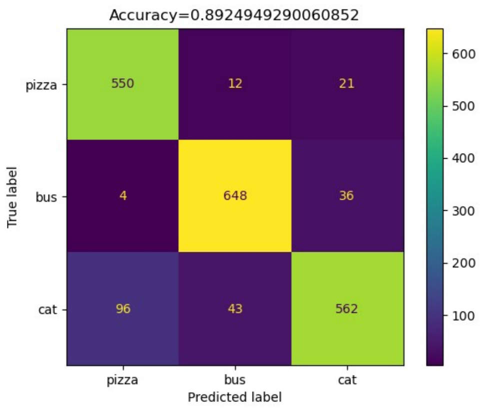
```python
class HW5Net(nn.Module):
    def __init__(self, input_nc, ngf=8, n_blocks=4) -> None:
        assert(n_blocks>=0)
        super(HW5Net, self).__init__()
        # The first conv layer
        model = [nn.ReflectionPad2d(3),
                 nn.Conv2d(input_nc, ngf, kernel_size=7, padding=0),
                 nn.BatchNorm2d(ngf),
                 nn.ReLU(True)]
        # Add downsampling layers
        n_downsampling = 4
        for i in range(n_downsampling):
            mult = 2**i # 1, 2, 4, 8
            model += [nn.Conv2d(ngf*mult, ngf*mult*2, kernel_size=3, stride=2, padding=1),
                      nn.BatchNorm2d(ngf*mult*2),
                      nn.ReLU(True)]
        # Add ResNet blocks
        mult = 2 ** n_downsampling # 16
        for i in range(n_blocks):
            model += [ResBlock(ngf*mult, ngf*mult, stride=2)]
        self.model = nn.Sequential(*model)
        # Classification head
        class_head = [nn.Dropout(0.5),
                      nn.Linear(512,3)]
        self.class_head = nn.Sequential(*class_head)
        # Regression head
        bbox_head = [nn.Dropout(0.5),
                     nn.Linear(512,64),
                     nn.ReLU(),
                     nn.BatchNorm1d(64),
                     nn.Dropout(0.5),
                     nn.Linear(64,4)]
        self.bbx_head = nn.Sequential(*bbox_head)
    def forward(self, x):
        ft = self.model(x)
        ft = ft.view(ft.shape[0], -1) # Change dimension of feature maps to fit linear layers
        cls = self.class_head(ft)
        bbox = self.bbx_head(ft)
        bbox = nn.Sigmoid()(bbox).clone()
        return cls, bbox
```

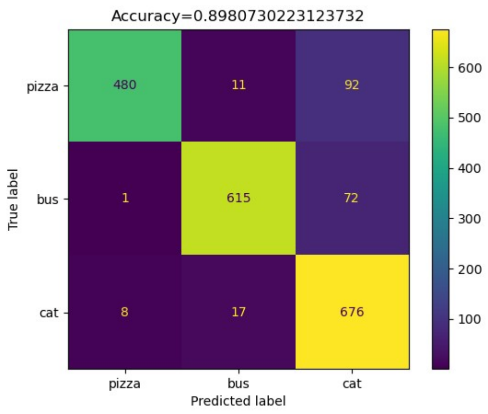
- **64 learnable layers in total**

- ### Training and Evaluating Your Trained Network

  - #### Confusion matrix

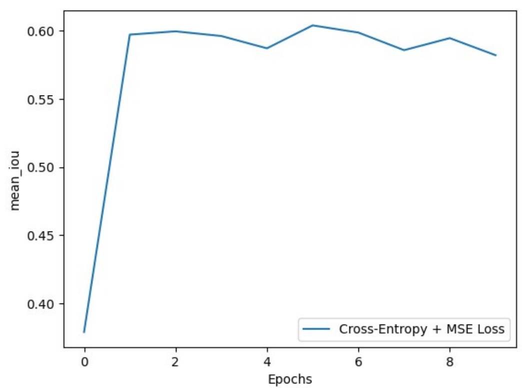
MSE-based loss **CIoU loss**

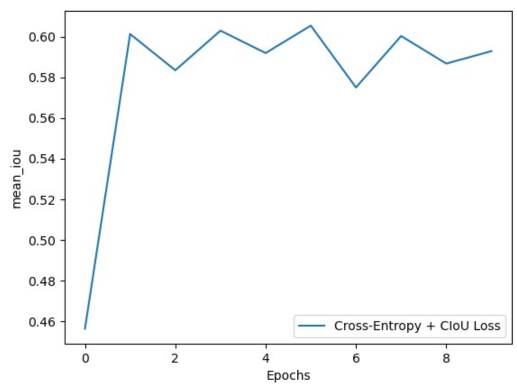




Since they both use cross-entropy loss as classification loss, the accuracy is roughly the same.

  - #### Mean IoU

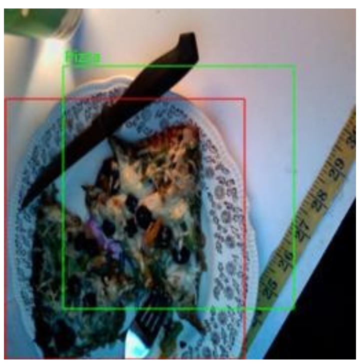




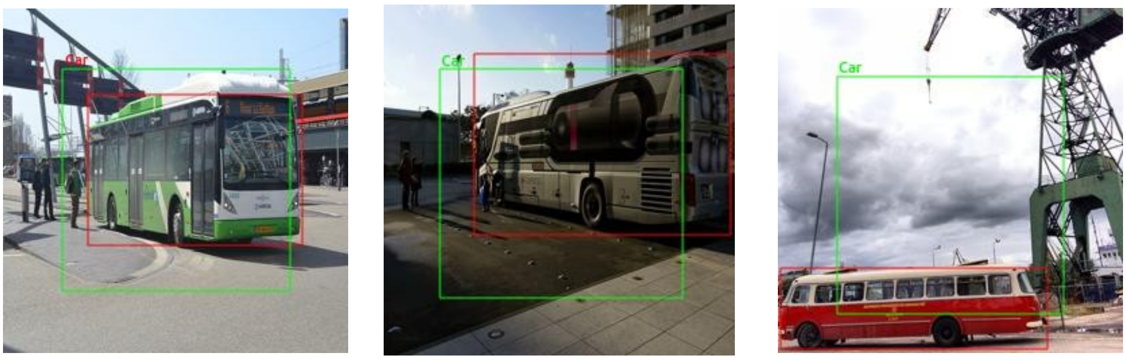
The highest mean IoU over epochs of using CIoU loss is a little bit better than that of MSE loss. Another difference is that training with CIoU is more unstable since it brings larger loss value compared to that of MSE loss.

  - Visualization

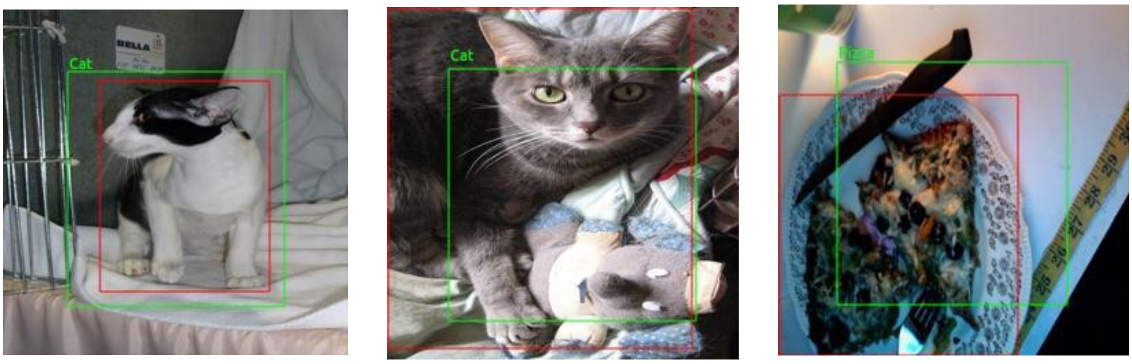
Pizza



Bus



Cat



We can observe that the prediction box tends to locate at the center of the images. The possible reason is that most training samples locate at the the center so the detector also learns this feature. For the right one of the bus sample, we can see the IoU is relatively low because the bus is in the peripheral region of image. This also gives us an intuition of anchor box setting, which we might work in the next assignment. If we assign some pre-defined anchors and learn the offsets to those anchors, it can be expected that the performance will be better. At least we can somewhat limit every bounding box to learn to be a middle normal box. Besides, I observe that the training tends to overfit the training dataset, so if we can do something like early-stopping or add some dropout layers or tune the regularization terms, the problem of overfitting might be alleviated to make the overall performance better.