Deep Learning Assignment 2

Jagennath Hari

jh7454

1 Designing convolution filters by hand

Input : 2D Image (Assuming $Image \in R^{n \times n}$)
Kernel : $w \in R^{3 \times 3}$

1.1 Blurring filter

Applying **Box Blur**.

$$w = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Applying **Gaussian Blur**.
The Gaussian function can be defined as :-

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma}}$$

Where x, y are the pixel values at those locations and σ is the standard deviation of the Gaussian distribution.
Assuming $\sigma = 0.85$

$$w = \begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

1.2 Image sharpening (horizontal)

By using **Top Sobel**

$$w = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

By using **Top Prewitt**

$$w = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

1.3 Image sharpening (vertical)

By using **Left Sobel**

$$w = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

By using **Left Prewitt**

$$w = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

1.4 Image sharpening (diagonal)

By using **Emboss**

$w = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$

2 Weight decay

Dataset : $\{(x_i, y_i)\}^n{}_{i=1}$

Loss function : L(w)

2.1 L2-regularized loss

$Cost = \sum_{i=0}^{n} (y_i - \sum_{i=0}^{n} w_i)^2 + \lambda \sum_{i=0}^{n} {w_i}^2$

$L_2(w_i) = \sum_{i=0}^{n} L(w_i) + \lambda \sum_{i=0}^{n} {w_i}^2$

2.2 Gradient descent for L2-regularized loss

$w_{t+1} = w_t - \eta \nabla L_2(w_t)$

$\nabla L_2(w_t) = L(w_t) + 2\lambda W_t$

${}_{t+1} = w_t - \eta(L(w_t) + 2\lambda w_t)$

$w_{t+1} = (1 - 2\eta\lambda)w_t - \eta L(w_t)$

2.3 Weight decay conclusion

The weights are ”shrunk” or ”decayed” by $(1 - 2\eta\lambda)$ where η is the learning rate and λ is the weight decay.

2.4 λ vs η

Increasing λ causes more penalty to be applied on the cost function, it can also help in over-fitting. Initially the learning rate η should be a high value for a λ. Once it has converged enough, η should be decreased to allow the weights to stabilize and reach the local minima.

3 The IoU metric

The IoU metric can be defined as:-

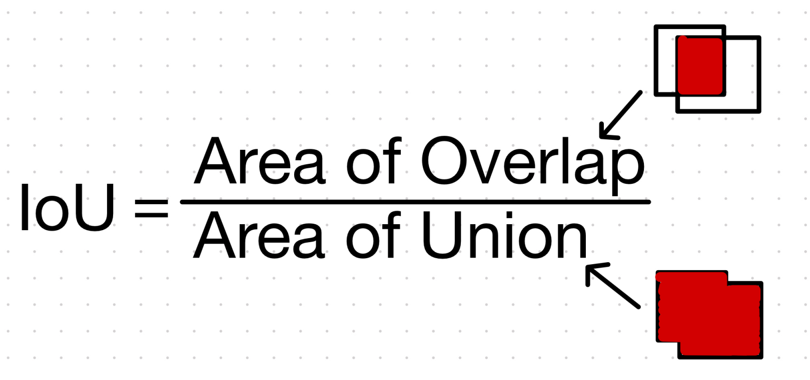


Figure 1:- IoU Metric

3.1 IoU $\in [0, 1]$

Let us assume a pair of bounding boxes A and B. The we can define the IoU metric for A and B as:-

$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$

$|A \cup B|$ and $|A \cap B|$ are non-negative as the intersection and union of two boxes cannot be negative. At a lower level we can also say that since, **area** cannot be negative.

$IoU \geq 0$

According to set theory, when two sets are overlapping:-

$|A \cup B| = |A \cap B|$

Then,

$IoU = 1$

Since, $|A \cap B|$ is always a subset of $|A \cup B|$, the denominator can only be greater than or equal to the numerator so we can assume,

$IoU \leq 1$

Therefore, $IoU(A, B) \in [0, 1]$

3.2 Showing that IoU metric is non-differentiable

Let us assume two bounding boxes. The IoU meteric initally is 0 as the boxes do not overlap, as it starts sliding over the other box the IoU metric starts to increase. Once, the box is fully overlapped with each other($IoU = 1$), it then starts to decrease until it finally becomes 0.

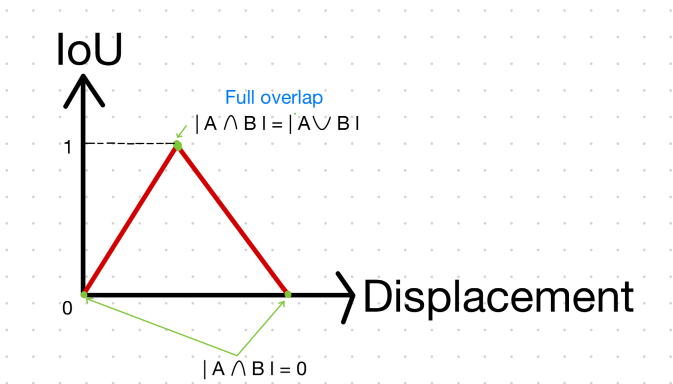


Figure 2:- Displacement Vs IoU

This graph is not **smooth**, so we can say it is **non-differentiable** and cannot be optimized using **Gradient Descent**.

4 Training AlexNet

In [42]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import _LRScheduler
import torch.utils.data as data

import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader

from sklearn import decomposition
from sklearn import manifold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np

import copy
import random
import time
```

In [43]:
```
if torch.backends.mps.is_built and torch.backends.mps.is_available
and torch.has_mps:
    device = "mps:0"
elif torch.has_cuda and torch.backends.cuda.is_built and torch.cuda
.is_available:
    device = "cuda:0"
else:
    device = "cpu"
print("Device set as " + device)
```

```
Device set as mps:0
```

In [44]:
```
SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
```

Out[44]:
```
<torch._C.Generator at 0x108ca4030>
```

In [45]:
```
ROOT = 'CIFAR10'
train_data = datasets.CIFAR10(root = ROOT,
                              train = True,
                              download = True)
```

```
Files already downloaded and verified
```

In [46]:
```
means = train_data.data.mean(axis = (0,1,2)) / 255
stds = train_data.data.std(axis = (0,1,2)) / 255
```

In [47]:
```
train_transforms = transforms.Compose([
                           transforms.RandomRotation(5),
                           transforms.RandomHorizontalFlip(0.5),
                           transforms.RandomCrop(32, padding = 2),
                           transforms.ToTensor(),
                           transforms.Normalize(mean = means,
                                                std = stds)
                       ])

test_transforms = transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize(mean = means,
                                                std = stds)
                       ])
```

In [48]:
```
train_data = datasets.CIFAR10(ROOT,
                              train = True,
                              download = True,
                              transform = train_transforms)

test_data = datasets.CIFAR10(ROOT,
                             train = False,
                             download = True,
                             transform = test_transforms)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

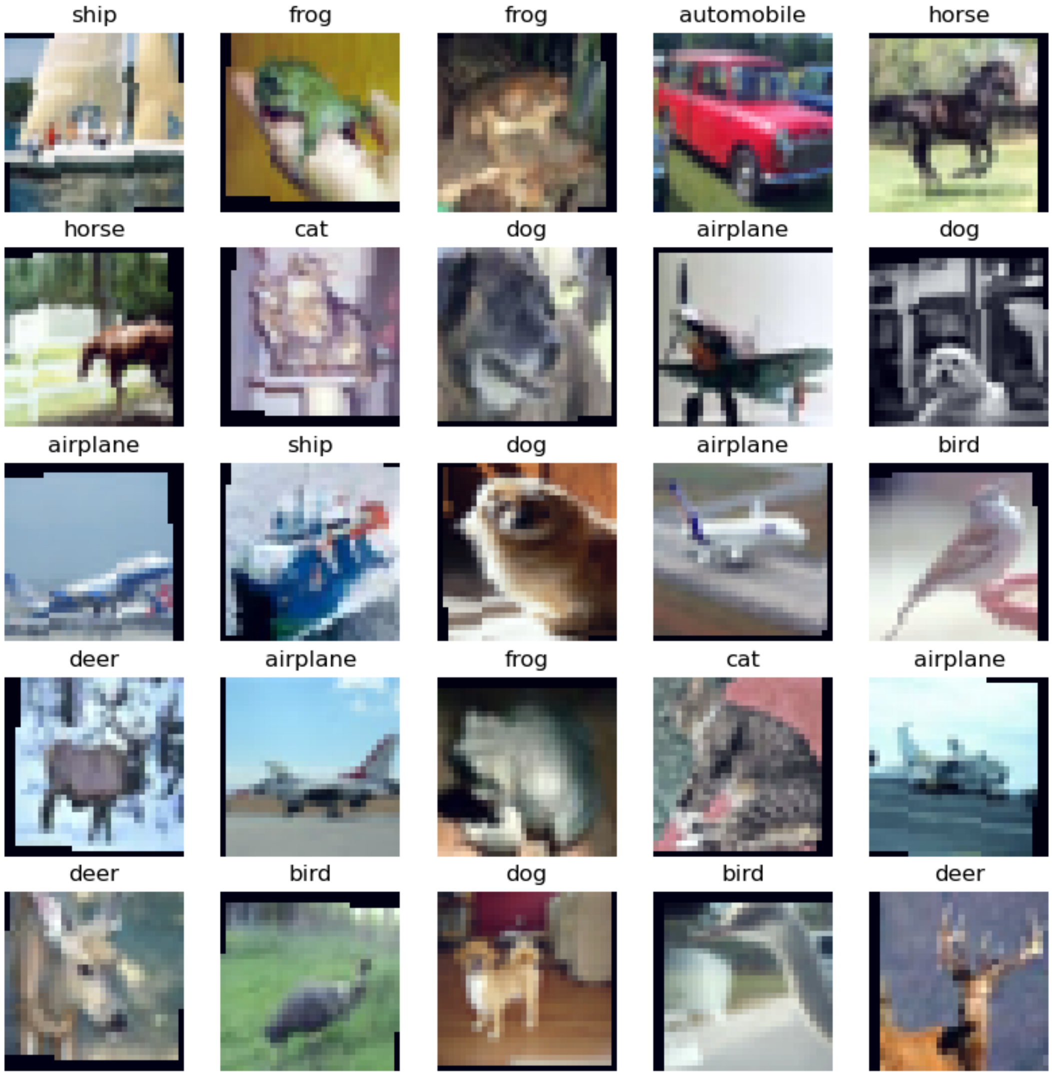
In [49]:
```
VALID_RATIO = 0.9

n_train_examples = int(len(train_data) * VALID_RATIO)
n_valid_examples = len(train_data) - n_train_examples

train_data, valid_data = data.random_split(train_data,
                                           [n_train_examples, n_val
id_examples])
```

In [50]:
```
valid_data = copy.deepcopy(valid_data)
valid_data.dataset.transform = test_transforms
```

In [51]:
```
def plot_images(images, labels, classes, normalize = False):

    n_images = len(images)

    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))

    fig = plt.figure(figsize = (10, 10))

    for i in range(rows*cols):

        ax = fig.add_subplot(rows, cols, i+1)

        image = images[i]

        if normalize:
            image_min = image.min()
            image_max = image.max()
            image.clamp_(min = image_min, max = image_max)
            image.add_(-image_min).div_(image_max - image_min + 1e-
5)

        ax.imshow(image.permute(1, 2, 0).cpu().numpy())
        ax.set_title(classes[labels[i]])
        ax.axis('off')
```

In [52]:
```
N_IMAGES = 25

images, labels = zip(*[(image, label) for image, label in
                           [train_data[i] for i in range(N_IMAGES)]
])

classes = test_data.classes
```

In [53]:
```python
plot_images(images, labels, classes, normalize = True)
```



In [54]:
```python
def normalize_image(image):
    image_min = image.min()
    image_max = image.max()
    image.clamp_(min = image_min, max = image_max)
    image.add_(-image_min).div_(image_max - image_min + 1e-5)
    return image
```

In [55]:
```
# Q1: Create data loaders for train_data, valid_data, test_data
# Use batch size 256


BATCH_SIZE = 256

train_iterator = DataLoader(train_data, batch_size = BATCH_SIZE, sh
uffle = True, num_workers = 0)

valid_iterator = DataLoader(valid_data, batch_size = BATCH_SIZE, sh
uffle = False, num_workers = 0)

test_iterator = DataLoader(test_data, batch_size = BATCH_SIZE, shuf
fle = False, num_workers = 0)
```

```
In [56]: class AlexNet(nn.Module):
             def __init__(self, output_dim):
                 super().__init__()

                 self.features = nn.Sequential(
                     # Define according to the steps described above
                     nn.Conv2d(3, 64, kernel_size = (3, 3), stride = 2, padd
         ing = 1, dtype = torch.float32),
                     nn.MaxPool2d(kernel_size = 2),
                     nn.ReLU(inplace = True),
                     nn.Conv2d(64, 192, kernel_size = (3, 3), stride = 1, pa
         dding = 1, dtype = torch.float32),
                     nn.MaxPool2d(kernel_size = 2),
                     nn.ReLU(inplace = True),
                     nn.Conv2d(192, 384, kernel_size = (3, 3), stride = 1, p
         adding = 1, dtype = torch.float32),
                     nn.ReLU(inplace = True),
                     nn.Conv2d(384, 256, kernel_size = (3, 3), stride = 1, p
         adding = 1, dtype = torch.float32),
                     nn.ReLU(inplace = True),
                     nn.Conv2d(256, 256, kernel_size = (3, 3), stride = 1, p
         adding = 1, dtype = torch.float32),
                     nn.MaxPool2d(kernel_size = 2),
                     nn.ReLU(inplace = True)
                 )

                 self.classifier = nn.Sequential(
                     # define according to the steps described above
                     nn.Dropout(p = 0.5),
                     nn.Linear(1024, 4096, dtype = torch.float32),
                     nn.ReLU(inplace = True),
                     nn.Dropout(p = 0.5),
                     nn.Linear(4096, 4096, dtype = torch.float32),
                     nn.ReLU(inplace = True),
                     nn.Linear(4096, output_dim, dtype = torch.float32)
                 )

             def forward(self, x):
                 x = self.features(x)
                 h = x.view(x.shape[0], -1)
                 x = self.classifier(h)
                 return x, h
```

```
In [57]: OUTPUT_DIM = 10
         model = AlexNet(OUTPUT_DIM)
```

```
In [58]: def initialize_parameters(m):
             if isinstance(m, nn.Conv2d):
                 nn.init.kaiming_normal_(m.weight.data, nonlinearity = 'relu
         ')
                 nn.init.constant_(m.bias.data, 0)
             elif isinstance(m, nn.Linear):
                 nn.init.xavier_normal_(m.weight.data, gain = nn.init.calcul
         ate_gain('relu'))
                 nn.init.constant_(m.bias.data, 0)
```

In [59]:
```
model.apply(initialize_parameters)
```

Out[59]:
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1))
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): ReLU(inplace=True)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padd
ing=(1, 1))
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1
, ceil_mode=False)
    (12): ReLU(inplace=True)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=1024, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

In [60]:
```
optimizer = optim.Adam(model.parameters(), lr = 1e-3)
criterion = nn.CrossEntropyLoss()

model = model.to(device)
criterion = criterion.to(device)
```

In [61]:
```
def calculate_accuracy(y_pred, y):
    top_pred = y_pred.argmax(1, keepdim = True)
    correct = top_pred.eq(y.view_as(top_pred)).sum()
    acc = correct.float() / y.shape[0]
    return acc
```

In [62]:
```
def train(model, iterator, optimizer, criterion, device):

    epoch_loss = 0
    epoch_acc = 0

    model.train()

    for (x, y) in iterator:

        x = x.to(device)
        y = y.to(device)

        optimizer.zero_grad()

        y_pred, _ = model(x)

        loss = criterion(y_pred, y)

        acc = calculate_accuracy(y_pred, y)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

In [63]:
```
def evaluate(model, iterator, criterion, device):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

        for (x, y) in iterator:

            x = x.to(device)
            y = y.to(device)

            y_pred, _ = model(x)

            loss = criterion(y_pred, y)

            acc = calculate_accuracy(y_pred, y)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

In [64]:
```
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

In [65]:
```
# Q3: train your model here for 25 epochs.
# Print out training and validation loss/accuracy of the model afte
r each epoch
# Keep track of the model that achieved best validation loss thus f
ar.

EPOCHS = 25

# Fill training code here
valid_loss_min = np.Inf
for i in range(EPOCHS):
    start_time = time.time()
    train_loss, train_acc = train(model = model, iterator = train_i
terator,optimizer = optimizer, criterion = criterion, device = devi
ce)
    valid_loss, valid_acc = evaluate(model = model, iterator = vali
d_iterator, criterion = criterion, device = device)
    end_time = time.time()
    elapsed_mins, elapsed_secs = epoch_time(start_time, end_time)
    print("Time elapsed: " + str(elapsed_mins) + " mins and " + str
(elapsed_secs) + " secs")
    print("Epoch: " + str(i) + " Train Loss: " + str(round(train_lo
ss, 3)) + " Train Accuracy: " + str(round(train_acc, 3)) + " Valida
tion Loss: " + str(round(valid_loss, 3)) + " Validation Accuracy: "
+ str(round(valid_acc, 3)))
    if valid_loss < valid_loss_min:
        print("Previous Validation loss: " + str(round(valid_loss_m
in, 3)) + " Current Validation Loss: " + str(round(valid_loss, 3)))
        print("Saving Model")
        torch.save(model.state_dict(), "AlexNet.pt")
        valid_loss_min = valid_loss
```

```
Time elapsed: 0 mins and 29 secs
Epoch: 0 Train Loss: inf Train Accuracy: 0.222 Validation Loss: 1.
561 Validation Accuracy: 0.392
Previous Validation loss: inf Current Validation Loss: 1.561
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 1 Train Loss: 1.519 Train Accuracy: 0.435 Validation Loss:
1.343 Validation Accuracy: 0.51
Previous Validation loss: 1.561 Current Validation Loss: 1.343
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 2 Train Loss: 1.351 Train Accuracy: 0.509 Validation Loss:
1.213 Validation Accuracy: 0.556
Previous Validation loss: 1.343 Current Validation Loss: 1.213
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 3 Train Loss: 1.262 Train Accuracy: 0.546 Validation Loss:
1.139 Validation Accuracy: 0.598
```

```
Previous Validation loss: 1.213 Current Validation Loss: 1.139
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 4 Train Loss: 1.17 Train Accuracy: 0.583 Validation Loss: 1
.072 Validation Accuracy: 0.629
Previous Validation loss: 1.139 Current Validation Loss: 1.072
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 5 Train Loss: 1.107 Train Accuracy: 0.607 Validation Loss:
1.016 Validation Accuracy: 0.638
Previous Validation loss: 1.072 Current Validation Loss: 1.016
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 6 Train Loss: 1.06 Train Accuracy: 0.624 Validation Loss: 0
.99 Validation Accuracy: 0.657
Previous Validation loss: 1.016 Current Validation Loss: 0.99
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 7 Train Loss: 1.001 Train Accuracy: 0.647 Validation Loss:
0.952 Validation Accuracy: 0.668
Previous Validation loss: 0.99 Current Validation Loss: 0.952
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 8 Train Loss: 0.973 Train Accuracy: 0.658 Validation Loss:
0.905 Validation Accuracy: 0.687
Previous Validation loss: 0.952 Current Validation Loss: 0.905
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 9 Train Loss: 0.923 Train Accuracy: 0.677 Validation Loss:
0.862 Validation Accuracy: 0.703
Previous Validation loss: 0.905 Current Validation Loss: 0.862
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 10 Train Loss: 0.899 Train Accuracy: 0.684 Validation Loss:
0.855 Validation Accuracy: 0.714
Previous Validation loss: 0.862 Current Validation Loss: 0.855
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 11 Train Loss: 0.87 Train Accuracy: 0.696 Validation Loss:
0.858 Validation Accuracy: 0.702
Time elapsed: 0 mins and 29 secs
Epoch: 12 Train Loss: 0.845 Train Accuracy: 0.706 Validation Loss:
0.814 Validation Accuracy: 0.722
Previous Validation loss: 0.855 Current Validation Loss: 0.814
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 13 Train Loss: 0.823 Train Accuracy: 0.713 Validation Loss:
0.789 Validation Accuracy: 0.735
Previous Validation loss: 0.814 Current Validation Loss: 0.789
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 14 Train Loss: 0.801 Train Accuracy: 0.722 Validation Loss:
0.79 Validation Accuracy: 0.727
Time elapsed: 0 mins and 29 secs
Epoch: 15 Train Loss: 0.78 Train Accuracy: 0.731 Validation Loss:
0.797 Validation Accuracy: 0.738
Time elapsed: 0 mins and 29 secs
```

```
Epoch: 16 Train Loss: 0.76 Train Accuracy: 0.739 Validation Loss:
0.775 Validation Accuracy: 0.735
Previous Validation loss: 0.789 Current Validation Loss: 0.775
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 17 Train Loss: 0.741 Train Accuracy: 0.742 Validation Loss:
0.759 Validation Accuracy: 0.74
Previous Validation loss: 0.775 Current Validation Loss: 0.759
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 18 Train Loss: 0.729 Train Accuracy: 0.746 Validation Loss:
0.759 Validation Accuracy: 0.75
Time elapsed: 0 mins and 29 secs
Epoch: 19 Train Loss: 0.711 Train Accuracy: 0.755 Validation Loss:
0.749 Validation Accuracy: 0.745
Previous Validation loss: 0.759 Current Validation Loss: 0.749
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 20 Train Loss: 0.698 Train Accuracy: 0.76 Validation Loss:
0.737 Validation Accuracy: 0.751
Previous Validation loss: 0.749 Current Validation Loss: 0.737
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 21 Train Loss: 0.68 Train Accuracy: 0.765 Validation Loss:
0.71 Validation Accuracy: 0.757
Previous Validation loss: 0.737 Current Validation Loss: 0.71
Saving Model
Time elapsed: 0 mins and 29 secs
Epoch: 22 Train Loss: 0.671 Train Accuracy: 0.766 Validation Loss:
0.721 Validation Accuracy: 0.757
Time elapsed: 0 mins and 29 secs
Epoch: 23 Train Loss: 0.659 Train Accuracy: 0.772 Validation Loss:
0.724 Validation Accuracy: 0.753
Time elapsed: 0 mins and 29 secs
Epoch: 24 Train Loss: 0.644 Train Accuracy: 0.777 Validation Loss:
0.716 Validation Accuracy: 0.764
```

In [66]:
```
model.load_state_dict(torch.load("AlexNet.pt"))
```

Out[66]:
```
<All keys matched successfully>
```

In [67]:
```
test_loss, test_acc = evaluate(model = model, iterator = test_itera
tor, criterion = criterion, device = device)
print("Test loss: " + str(round(test_loss, 3)) + " Test accuracy: "
+ str(round(test_acc, 3)))
```

```
Test loss: 0.718 Test accuracy: 0.755
```

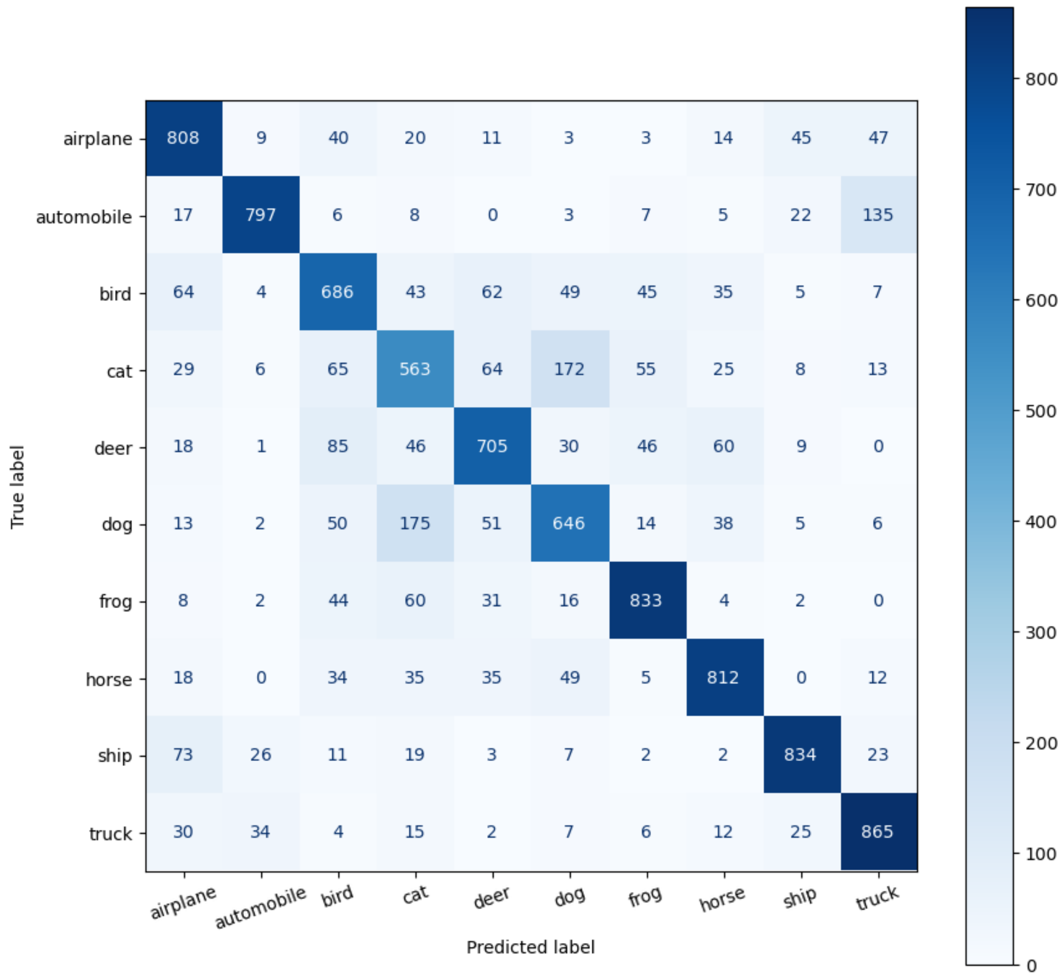
In [68]:
```
def get_predictions(model, iterator, device):

    model.eval()

    labels = []
    probs = []

    # Q4: Fill code here.
    with torch.no_grad():
        model = model.to(device)
        for (x, y) in iterator:
            x = x.to(device)
            y = y.to(device)
            y_pred, _ = model(x)
            y_prob = F.softmax(y_pred, dim = -1)
            top_pred = y_prob.argmax(1, keepdim = True)
            labels.append(y.cpu())
            probs.append(y_prob.cpu())

    labels = torch.cat(labels, dim = 0)
    probs = torch.cat(probs, dim = 0)

    return labels, probs
```

In [69]:
```
labels, probs = get_predictions(model, test_iterator, device)
```

In [70]:
```
pred_labels = torch.argmax(probs, 1)
```

In [71]:
```
def plot_confusion_matrix(labels, pred_labels, classes):

    fig = plt.figure(figsize = (10, 10));
    ax = fig.add_subplot(1, 1, 1);
    cm = confusion_matrix(labels, pred_labels);
    cm = ConfusionMatrixDisplay(cm, display_labels = classes);
    cm.plot(values_format = 'd', cmap = 'Blues', ax = ax)
    plt.xticks(rotation = 20)
```

```
In [72]: plot_confusion_matrix(labels, pred_labels, classes)
```



TorchVision Instance Segmentation Finetuning Tutorial

For this tutorial, we will be finetuning a pre-trained Mask R-CNN model in the *Penn-Fudan Database for Pedestrian Detection and Segmentation*. It contains 170 images with 345 instances of pedestrians, and we will use it to illustrate how to use the new features in torchvision in order to train an instance segmentation model on a custom dataset.

First, we need to install `pycocotools`. This library will be used for computing the evaluation metrics following the COCO metric for intersection over union.

```
%%shell

pip install cython
# Install pycocotools, the version by default in Colab
# has a bug fixed in https://github.com/cocodataset/cocoapi/pull/354
pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=Pytho
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Requirement already satisfied: cython in /usr/local/lib/python3.7/dist-package
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting git+https://github.com/cocodataset/cocoapi.git#subdirectory=Python/
  Cloning https://github.com/cocodataset/cocoapi.git to /tmp/pip-req-build-npv
  Running command git clone -q https://github.com/cocodataset/cocoapi.git /tmp
Requirement already satisfied: setuptools>=18.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: cython>=0.27.3 in /usr/local/lib/python3.7/dist
Requirement already satisfied: matplotlib>=2.1.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/d
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /us
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/d
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packa
Building wheels for collected packages: pycocotools
  Building wheel for pycocotools (setup.py) ... done
  Created wheel for pycocotools: filename=pycocotools-2.0-cp37-cp37m-linux_x86
  Stored in directory: /tmp/pip-ephem-wheel-cache-44c08a21/wheels/e2/6b/1d/344
Successfully built pycocotools
Installing collected packages: pycocotools
  Attempting uninstall: pycocotools
    Found existing installation: pycocotools 2.0.5
    Uninstalling pycocotools-2.0.5:
      Successfully uninstalled pycocotools-2.0.5
Successfully installed pycocotools-2.0
```

Defining the Dataset

Defining the Dataset

The torchvision reference scripts for training object detection, instance segmentation and person keypoint detection allows for easily supporting adding new custom datasets. The dataset should inherit from the standard `torch.utils.data.Dataset` class, and implement `__len__` and `__getitem__`.

The only specificity that we require is that the dataset `__getitem__` should return:

- image: a PIL Image of size (H, W)
- target: a dict containing the following fields
    - `boxes` (`FloatTensor[N, 4]`): the coordinates of the `N` bounding boxes in `[x0, y0, x1, y1]` format, ranging from `0` to `W` and `0` to `H`
    - `labels` (`Int64Tensor[N]`): the label for each bounding box
    - `image_id` (`Int64Tensor[1]`): an image identifier. It should be unique between all the images in the dataset, and is used during evaluation
    - `area` (`Tensor[N]`): The area of the bounding box. This is used during evaluation with the COCO metric, to separate the metric scores between small, medium and large boxes.
    - `iscrowd` (`UInt8Tensor[N]`): instances with `iscrowd=True` will be ignored during evaluation.
    - (optionally) `masks` (`UInt8Tensor[N, H, W]`): The segmentation masks for each one of the objects
    - (optionally) `keypoints` (`FloatTensor[N, K, 3]`): For each one of the `N` objects, it contains the `K` keypoints in `[x, y, visibility]` format, defining the object. `visibility=0` means that the keypoint is not visible. Note that for data augmentation, the notion of flipping a keypoint is dependent on the data representation, and you should probably adapt `references/detection/transforms.py` for your new keypoint representation

If your model returns the above methods, they will make it work for both training and evaluation, and will use the evaluation scripts from pycocotools.

One note on the labels. The model considers class 0 as background. If your dataset does not contain the background class, you should not have 0 in your labels. For example, assuming you have just two classes, cat and dog, you can define 1 (not 0) to represent cats and 2 to represent dogs. So, for instance, if one of the images has both classes, your labels tensor should look like [1,2].

Additionally, if you want to use aspect ratio grouping during training (so that each batch only contains images with similar aspect ratio), then it is recommended to also implement a `get_height_and_width` method, which returns the height and the width of the image. If this method is not provided, we query all elements of the dataset via `__getitem__` , which loads the image in memory and is slower than if a custom method is provided.

Writing a custom dataset for Penn-Fudan

Let's write a dataset for the Penn-Fudan dataset.

First, let's download and extract the data, present in a zip file at https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip

```
%%shell

# download the Penn-Fudan dataset
wget https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip .
# extract it in the current folder
unzip PennFudanPed.zip
```

```
  inflating: PennFudanPed/PNGImages/PennPed00039.png
  inflating: PennFudanPed/PNGImages/PennPed00040.png
  inflating: PennFudanPed/PNGImages/PennPed00041.png
  inflating: PennFudanPed/PNGImages/PennPed00042.png
  inflating: PennFudanPed/PNGImages/PennPed00043.png
  inflating: PennFudanPed/PNGImages/PennPed00044.png
  inflating: PennFudanPed/PNGImages/PennPed00045.png
  inflating: PennFudanPed/PNGImages/PennPed00046.png
  inflating: PennFudanPed/PNGImages/PennPed00047.png
  inflating: PennFudanPed/PNGImages/PennPed00048.png
  inflating: PennFudanPed/PNGImages/PennPed00049.png
  inflating: PennFudanPed/PNGImages/PennPed00050.png
  inflating: PennFudanPed/PNGImages/PennPed00051.png
  inflating: PennFudanPed/PNGImages/PennPed00052.png
  inflating: PennFudanPed/PNGImages/PennPed00053.png
  inflating: PennFudanPed/PNGImages/PennPed00054.png
  inflating: PennFudanPed/PNGImages/PennPed00055.png
  inflating: PennFudanPed/PNGImages/PennPed00056.png
  inflating: PennFudanPed/PNGImages/PennPed00057.png
  inflating: PennFudanPed/PNGImages/PennPed00058.png
  inflating: PennFudanPed/PNGImages/PennPed00059.png
  inflating: PennFudanPed/PNGImages/PennPed00060.png
  inflating: PennFudanPed/PNGImages/PennPed00061.png
  inflating: PennFudanPed/PNGImages/PennPed00062.png
  inflating: PennFudanPed/PNGImages/PennPed00063.png
  inflating: PennFudanPed/PNGImages/PennPed00064.png
  inflating: PennFudanPed/PNGImages/PennPed00065.png
  inflating: PennFudanPed/PNGImages/PennPed00066.png
  inflating: PennFudanPed/PNGImages/PennPed00067.png
```

```
inflating: PennFudanPed/PNGImages/PennPed00068.png
inflating: PennFudanPed/PNGImages/PennPed00069.png
inflating: PennFudanPed/PNGImages/PennPed00070.png
inflating: PennFudanPed/PNGImages/PennPed00071.png
inflating: PennFudanPed/PNGImages/PennPed00072.png
inflating: PennFudanPed/PNGImages/PennPed00073.png
inflating: PennFudanPed/PNGImages/PennPed00074.png
inflating: PennFudanPed/PNGImages/PennPed00075.png
inflating: PennFudanPed/PNGImages/PennPed00076.png
inflating: PennFudanPed/PNGImages/PennPed00077.png
inflating: PennFudanPed/PNGImages/PennPed00078.png
inflating: PennFudanPed/PNGImages/PennPed00079.png
inflating: PennFudanPed/PNGImages/PennPed00080.png
inflating: PennFudanPed/PNGImages/PennPed00081.png
inflating: PennFudanPed/PNGImages/PennPed00082.png
inflating: PennFudanPed/PNGImages/PennPed00083.png
inflating: PennFudanPed/PNGImages/PennPed00084.png
inflating: PennFudanPed/PNGImages/PennPed00085.png
inflating: PennFudanPed/PNGImages/PennPed00086.png
inflating: PennFudanPed/PNGImages/PennPed00087.png
inflating: PennFudanPed/PNGImages/PennPed00088.png
inflating: PennFudanPed/PNGImages/PennPed00089.png
inflating: PennFudanPed/PNGImages/PennPed00090.png
inflating: PennFudanPed/PNGImages/PennPed00091.png
inflating: PennFudanPed/PNGImages/PennPed00092.png
inflating: PennFudanPed/PNGImages/PennPed00093.png
inflating: PennFudanPed/PNGImages/PennPed00094.png
inflating: PennFudanPed/PNGImages/PennPed00095.png
inflating: PennFudanPed/PNGImages/PennPed00096.png
inflating: PennFudanPed/readme.txt
```

Let's have a look at the dataset and how it is layed down.

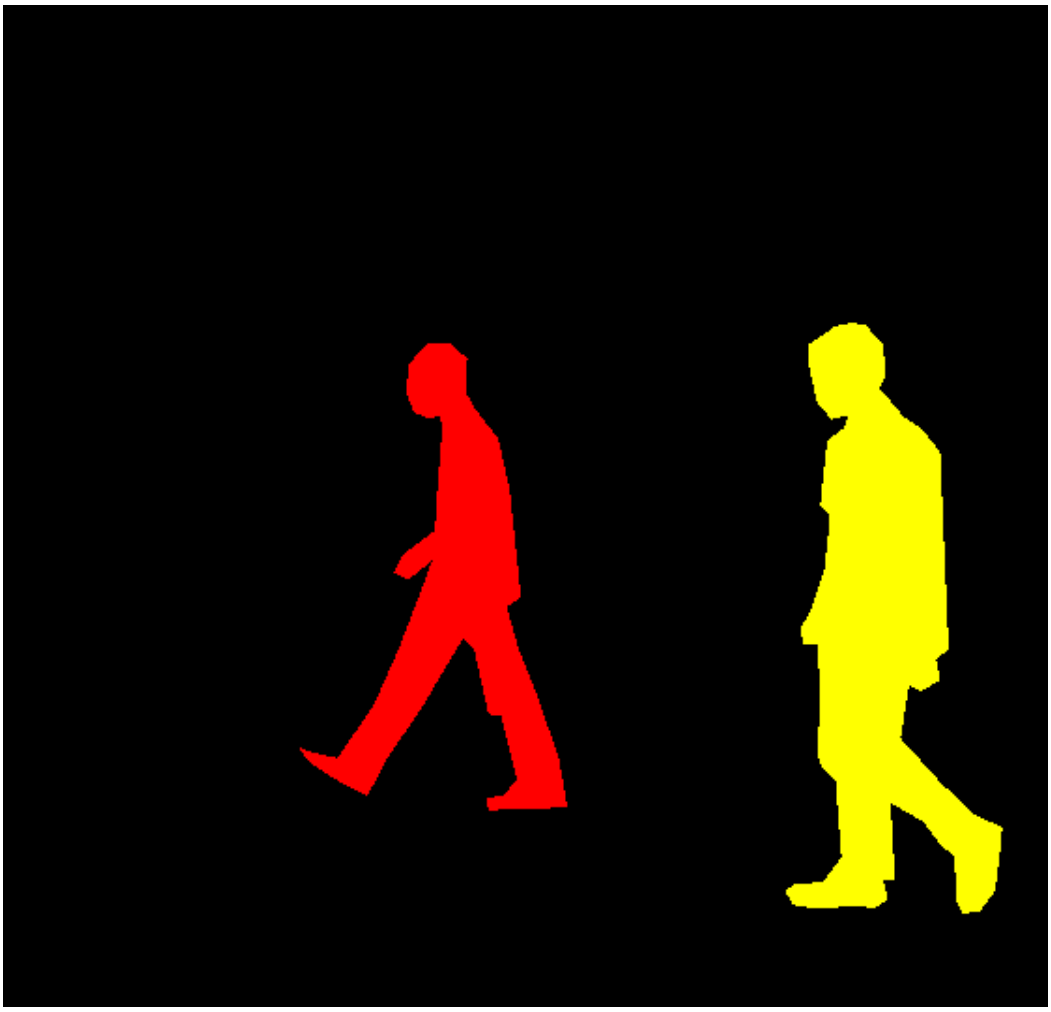
The data is structured as follows

```
PennFudanPed/
  PedMasks/
    FudanPed00001_mask.png
    FudanPed00002_mask.png
    FudanPed00003_mask.png
    FudanPed00004_mask.png
    ...
  PNGImages/
    FudanPed00001.png
    FudanPed00002.png
    FudanPed00003.png
    FudanPed00004.png
```

Here is one example of an image in the dataset, with its corresponding instance segmentation mask

```
from PIL import Image
Image.open('PennFudanPed/PNGImages/FudanPed00001.png')
```

```
mask = Image.open('PennFudanPed/PedMasks/FudanPed00001_mask.png')
# each mask instance has a different color, from zero to N, where
# N is the number of instances. In order to make visualization easier,
# let's adda color palette to the mask.
mask.putpalette([
    0, 0, 0, # black background
    255, 0, 0, # index 1 is red
    255, 255, 0, # index 2 is yellow
    255, 153, 0, # index 3 is orange
])
mask
```



So each image has a corresponding segmentation mask, where each color correspond to a different instance. Let's write a `torch.utils.data.Dataset` class for this dataset.

```
import os
```

```
import numpy as np
import torch
import torch.utils.data
from PIL import Image


class PennFudanDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        # load all image files, sorting them to
        # ensure that they are aligned
        self.imgs = list(sorted(os.listdir(os.path.join(root, "PNGImages"))))
        self.masks = list(sorted(os.listdir(os.path.join(root, "PedMasks"))))

    def __getitem__(self, idx):
        # load images ad masks
        img_path = os.path.join(self.root, "PNGImages", self.imgs[idx])
        mask_path = os.path.join(self.root, "PedMasks", self.masks[idx])
        img = Image.open(img_path).convert("RGB")
        # note that we haven't converted the mask to RGB,
        # because each color corresponds to a different instance
        # with 0 being background
        mask = Image.open(mask_path)

        mask = np.array(mask)
        # instances are encoded as different colors
        obj_ids = np.unique(mask)
        # first id is the background, so remove it
        obj_ids = obj_ids[1:]

        # split the color-encoded mask into a set
        # of binary masks
        masks = mask == obj_ids[:, None, None]

        # get bounding box coordinates for each mask
        num_objs = len(obj_ids)
        boxes = []
        for i in range(num_objs):
            pos = np.where(masks[i])
            xmin = np.min(pos[1])
            xmax = np.max(pos[1])
            ymin = np.min(pos[0])
            ymax = np.max(pos[0])
            boxes.append([xmin, ymin, xmax, ymax])

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        # there is only one class
```

```
        labels = torch.ones((num_objs,), dtype=torch.int64)
        masks = torch.as_tensor(masks, dtype=torch.uint8)

        image_id = torch.tensor([idx])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
        # suppose all instances are not crowd
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target

    def __len__(self):
        return len(self.imgs)
```

That's all for the dataset. Let's see how the outputs are structured for this dataset

```
dataset = PennFudanDataset('PennFudanPed/')
dataset[0]
```

```
(<PIL.Image.Image image mode=RGB size=559x536 at 0x7F63EE35AA50>,
 {'boxes': tensor([[159., 181., 301., 430.],
          [419., 170., 534., 485.]]),
  'labels': tensor([1, 1]),
  'masks': tensor([[[0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           ...,
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0]],

          [[0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           ...,
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0]]], dtype=torch.uint8),
  'image_id': tensor([0]),
  'area': tensor([35358., 36225.]),
  'iscrowd': tensor([0, 0])})
```

So we can see that by default, the dataset returns a `PIL.Image` and a dictionary containing several fields, including `boxes`, `labels` and `masks`.

Defining your model

In this tutorial, we will be using Mask R-CNN, which is based on top of Faster R-CNN. Faster R-CNN is a model that predicts both bounding boxes and class scores for potential objects in the image.

Mask R-CNN adds an extra branch into Faster R-CNN, which also predicts segmentation masks for each instance.

There are two common situations where one might want to modify one of the available models in torchvision modelzoo. The first is when we want to start from a pre-trained model, and just finetune the last layer. The other is when we want to replace the backbone of the model with a different one (for faster predictions, for example).

Let's go see how we would do one or another in the following sections.

1 - Finetuning from a pretrained model

Let's suppose that you want to start from a model pre-trained on COCO and want to finetune it for your particular classes. Here is a possible way of doing it:

```
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

# load a model pre-trained pre-trained on COCO
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

# replace the classifier with a new one, that has
# num_classes which is user-defined
num_classes = 2  # 1 class (person) + background
# get number of input features for the classifier
in_features = model.roi_heads.box_predictor.cls_score.in_features
# replace the pre-trained head with a new one
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

2 - Modifying the model to add a different backbone

Another common situation arises when the user wants to replace the backbone of a detection model with a different one. For example, the current default backbone (ResNet-50) might be too big for some applications, and smaller models might be necessary.

Here is how we would go into leveraging the functions provided by torchvision to modify a backbone.

```
import torchvision
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator

# load a pre-trained model for classification and return
# only the features
backbone = torchvision.models.mobilenet_v2(pretrained=True).features
# FasterRCNN needs to know the number of
# output channels in a backbone. For mobilenet_v2, it's 1280
```

```
# so we need to add it here
backbone.out_channels = 1280

# let's make the RPN generate 5 x 3 anchors per spatial
# location, with 5 different sizes and 3 different aspect
# ratios. We have a Tuple[Tuple[int]] because each feature
# map could potentially have different sizes and
# aspect ratios
anchor_generator = AnchorGenerator(sizes=((32, 64, 128, 256, 512),),
                                   aspect_ratios=((0.5, 1.0, 2.0),))

# let's define what are the feature maps that we will
# use to perform the region of interest cropping, as well as
# the size of the crop after rescaling.
# if your backbone returns a Tensor, featmap_names is expected to
# be [0]. More generally, the backbone should return an
# OrderedDict[Tensor], and in featmap_names you can choose which
# feature maps to use.
roi_pooler = torchvision.ops.MultiScaleRoIAlign(featmap_names=[0],
                                                output_size=7,
                                                sampling_ratio=2)

# put the pieces together inside a FasterRCNN model
model = FasterRCNN(backbone,
                   num_classes=2,
                   rpn_anchor_generator=anchor_generator,
                   box_roi_pool=roi_pooler)
```

An Instance segmentation model for PennFudan Dataset

In our case, we want to fine-tune from a pre-trained model, given that our dataset is very small. So we will be following approach number 1.

Here we want to also compute the instance segmentation masks, so we will be using Mask R-CNN:

```
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor


def get_instance_segmentation_model(num_classes):
    # load an instance segmentation model pre-trained on COCO
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)

    # get the number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    # now get the number of input features for the mask classifier
    in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256
    # and replace the mask predictor with a new one
    model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask,
                                                       hidden_layer,
                                                       num_classes)

    return model
```

That's it, this will make model be ready to be trained and evaluated on our custom dataset.

Training and evaluation functions

In `references/detection/`, we have a number of helper functions to simplify training and evaluating detection models. Here, we will use `references/detection/engine.py`, `references/detection/utils.py` and `references/detection/transforms.py`.

Let's copy those files (and their dependencies) in here so that they are available in the notebook

```
%%shell

# Download TorchVision repo to use some files from
# references/detection
git clone https://github.com/pytorch/vision.git
cd vision
git checkout v0.8.2

cp references/detection/utils.py ../
cp references/detection/transforms.py ../
cp references/detection/coco_eval.py ../
cp references/detection/engine.py ../
cp references/detection/coco_utils.py ../
```

```
Cloning into 'vision'...
remote: Enumerating objects: 231049, done.
remote: Counting objects: 100% (4720/4720), done.
remote: Compressing objects: 100% (476/476), done.
remote: Total 231049 (delta 4339), reused 4557 (delta 4236), pack-reused 2263
Receiving objects: 100% (231049/231049), 468.40 MiB | 17.04 MiB/s, done.
Resolving deltas: 100% (209412/209412), done.
Note: checking out 'v0.8.2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 2f40a483d [v0.8.X] .circleci: Add Python 3.9 to CI (#3063)
```

Let's write some helper functions for data augmentation / transformation, which leverages the functions in `refereces/detection` that we have just copied:

```
from engine import train_one_epoch, evaluate
import utils
import transforms as T


def get_transform(train):
    transforms = []
    # converts the image, a PIL image, into a PyTorch Tensor
    transforms.append(T.ToTensor())
    if train:
        # during training, randomly flip the training images
        # and ground-truth for data augmentation
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
```

Testing forward() method

Before iterating over the dataset, it's good to see what the model expects during training and inference time on sample data.

```
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
dataset = PennFudanDataset('PennFudanPed', get_transform(train=True))
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=2, shuffle=True, num_workers=4,
    collate_fn=utils.collate_fn
)
# For Training
images,targets = next(iter(data_loader))
images = list(image for image in images)
targets = [{k: v for k, v in t.items()} for t in targets]
output = model(images,targets)   # Returns losses and detections
# For inference
model.eval()
x = [torch.rand(3, 300, 400), torch.rand(3, 500, 400)]
predictions = model(x)           # Returns predictions
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:209: UserW
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and will be re
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserW
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_cocc
```

100% 160M/160M [00:00<00:00, 211MB/s]

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:566: Use
  cpuset_checked))
```

Note that we do not need to add a mean/std normalization nor image rescaling in the data transforms, as those are handled internally by the Mask R-CNN model.

Putting everything together

We now have the dataset class, the models and the data transforms. Let's instantiate them

```
# use our dataset and defined transformations
dataset = PennFudanDataset('PennFudanPed', get_transform(train=True))
dataset_test = PennFudanDataset('PennFudanPed', get_transform(train=False))

# split the dataset in train and test set
torch.manual_seed(1)
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-50])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-50:])

# define training and validation data loaders
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=1, shuffle=True, num_workers=4,
    collate_fn=utils.collate_fn)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test, batch_size=1, shuffle=False, num_workers=4,
    collate_fn=utils.collate_fn)
```

Now let's instantiate the model and the optimizer

```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# our dataset has two classes only - background and person
num_classes = 2

# get the model using our helper function
model = get_instance_segmentation_model(num_classes)
# move model to the right device
model.to(device)

# construct an optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

# and a learning rate scheduler which decreases the learning rate by
# 10x every 3 epochs
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

And now let's train the model for 10 epochs, evaluating at the end of every epoch.

```
# let's train it for 10 epochs
```

```
# let's train it for 10 epochs
from torch.optim.lr_scheduler import StepLR
num_epochs = 10

for epoch in range(num_epochs):
    # train for one epoch, printing every 10 iterations
    train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=10)
    # update the learning rate
    lr_scheduler.step()
    # evaluate on the test dataset
    evaluate(model, data_loader_test, device=device)
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.34
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.79
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.79
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.71
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.80
Epoch: [6]  [  0/120]  eta: 0:01:10  lr: 0.000050  loss: 0.0988 (0.0988)  loss
Epoch: [6]  [ 10/120]  eta: 0:00:34  lr: 0.000050  loss: 0.1636 (0.1725)  loss
Epoch: [6]  [ 20/120]  eta: 0:00:29  lr: 0.000050  loss: 0.1700 (0.1854)  loss
Epoch: [6]  [ 30/120]  eta: 0:00:26  lr: 0.000050  loss: 0.1559 (0.1770)  loss
Epoch: [6]  [ 40/120]  eta: 0:00:23  lr: 0.000050  loss: 0.1581 (0.1783)  loss
Epoch: [6]  [ 50/120]  eta: 0:00:19  lr: 0.000050  loss: 0.1596 (0.1759)  loss
Epoch: [6]  [ 60/120]  eta: 0:00:17  lr: 0.000050  loss: 0.1452 (0.1713)  loss
Epoch: [6]  [ 70/120]  eta: 0:00:14  lr: 0.000050  loss: 0.1451 (0.1693)  loss
Epoch: [6]  [ 80/120]  eta: 0:00:11  lr: 0.000050  loss: 0.1439 (0.1681)  loss
Epoch: [6]  [ 90/120]  eta: 0:00:08  lr: 0.000050  loss: 0.1349 (0.1657)  loss
Epoch: [6]  [100/120]  eta: 0:00:05  lr: 0.000050  loss: 0.1349 (0.1647)  loss
Epoch: [6]  [110/120]  eta: 0:00:02  lr: 0.000050  loss: 0.1374 (0.1659)  loss
Epoch: [6]  [119/120]  eta: 0:00:00  lr: 0.000050  loss: 0.1511 (0.1651)  loss
Epoch: [6] Total time: 0:00:34 (0.2840 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:23  model_time: 0.1538 (0.1538)  evaluator_time: 0.0
Test:  [49/50]  eta: 0:00:00  model_time: 0.1070 (0.1079)  evaluator_time: 0.0
Test: Total time: 0:00:06 (0.1287 s / it)
Averaged stats: model_time: 0.1070 (0.1079)  evaluator_time: 0.0034 (0.0055)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.82
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.98
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.94
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.56
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.83
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.38
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.87
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.87
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.77
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.87
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.75
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.98
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.89
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.48
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.76
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.34
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.80
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.80
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.72
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.80
Epoch: [7]  [  0/120]  eta: 0:01:20  lr: 0.000050  loss: 0.1976 (0.1976)  loss
Epoch: [7]  [ 10/120]  eta: 0:00:33  lr: 0.000050  loss: 0.1270 (0.1352)  loss
Epoch: [7]  [ 20/120]  eta: 0:00:29  lr: 0.000050  loss: 0.1273 (0.1466)  loss
Epoch: [7]  [ 30/120]  eta: 0:00:26  lr: 0.000050  loss: 0.1674 (0.1587)  loss
```

Now that training has finished, let's have a look at what it actually predicts in a test image

```
# pick one image from the test set
img, _ = dataset_test[0]
# put the model in evaluation mode
model.eval()
with torch.no_grad():
    prediction = model([img.to(device)])
```

Printing the prediction shows that we have a list of dictionaries. Each element of the list corresponds to a different image. As we have a single image, there is a single dictionary in the list. The dictionary contains the predictions for the image we passed. In this case, we can see that it contains `boxes`, `labels`, `masks` and `scores` as fields.

```
prediction
```

```
[{'boxes': tensor([[ 64.4480,  39.0130, 196.8688, 322.7589],
          [276.2072,  19.6565, 290.6538,  74.4100]], device='cuda:0'),
  'labels': tensor([1, 1], device='cuda:0'),
  'scores': tensor([0.9979, 0.0516], device='cuda:0'),
  'masks': tensor([[[[0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.],
            ...,
            [0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.]]],


          [[[0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.],
            ...,
            [0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.],
            [0., 0., 0.,  ..., 0., 0., 0.]]]], device='cuda:0')}]
```

Let's inspect the image and the predicted segmentation masks.

For that, we need to convert the image, which has been rescaled to 0-1 and had the channels flipped so that we have it in `[C, H, W]` format.

```
Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

And let's now visualize the top predicted segmentation mask. The masks are predicted as `[N, 1, H, W]`, where `N` is the number of predictions, and are probability maps between 0-1.

```
from torchvision.utils import draw_bounding_boxes
from torchvision.ops import nms
from torchvision.io import read_image
from torchvision import transforms
transform = transforms.Compose([
    transforms.PILToTensor()
])
IMG = Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
img_tensor = transform(IMG)
BB = prediction[0]['boxes']
scores = prediction[0]['scores']
for  i ,score in enumerate(scores):
  if score > 0.2:
    show = draw_bounding_boxes(img_tensor, BB[i][None, :],width=3)
show = torchvision.transforms.ToPILImage()(show)
show
```

```
beatles = torchvision.io.read_image("/content/Beatles_-_Abbey_Road.jpg")
beatles = beatles.div(255)
model.eval()
with torch.no_grad():
    predictionBeatles = model([beatles.to(device)])
```

```
beatles = torchvision.io.read_image("/content/Beatles_-_Abbey_Road.jpg")
BBbeatles = predictionBeatles[0]['boxes']
scores = predictionBeatles[0]['scores']
for  i ,score in enumerate(scores):
  if score > 0.5:
    beatles = draw_bounding_boxes(beatles, BBbeatles[i][None, :],width=3)
show1 = torchvision.transforms.ToPILImage()(beatles)
show1
```

```
import torchvision
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator

# load a pre-trained model for classification and return
# only the features
backbone = torchvision.models.mobilenet_v2(pretrained=True).features
#we need to specify an outchannel of this backone specifically because this outchar
#used as an inchannel for the RPNHEAD which is producing the out of RegionProposalN
#we can know the number of outchannels by looking into the backbone "backbone??"
backbone.out_channels = 1280
#by default the achor generator FasterRcnn assign will be for a FPN backone, so
#we need to specify a  different anchor generator
anchor_generator = AnchorGenerator(sizes=((128, 256, 512),),
                                   aspect_ratios=((0.5, 1.0, 2.0),))
#here at each position in the grid there will be 3x3=9 anchors
#and if our backbone is not FPN then the forward method will assign the name '0' to
#so we need to specify '0 as feature map name'
roi_pooler = torchvision.ops.MultiScaleRoIAlign(featmap_names=['0'],
                                                output_size=9,
                                           sampling_ratio=2)
#the output size is the output shape of the roi pooled features which will be used
model = FasterRCNN(backbone,num_classes=2,rpn_anchor_generator=anchor_generator)
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:209: UserW
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and will be re
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserW
  warnings.warn(msg)
```

```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# our dataset has two classes only - background and person
num_classes = 2

# get the model using our helper function
# move model to the right device
model2 = model.to(device)

# construct an optimizer
params2 = [p for p in model2.parameters() if p.requires_grad]
optimizer2 = torch.optim.Adam(params2, lr = 0.0001, weight_decay=0.00005)

# and a learning rate scheduler which decreases the learning rate by
# 10x every 3 epochs
lr_scheduler2 = torch.optim.lr_scheduler.StepLR(optimizer2,
                                               step_size=3,
                                               gamma=0.1)


from torch.optim.lr_scheduler import StepLR
num_epochs = 10

for epoch in range(num_epochs):
    # train for one epoch, printing every 10 iterations
    train_one_epoch(model2, optimizer2, data_loader, device, epoch, print_freq=10
    # update the learning rate
    lr_scheduler2.step()
    # evaluate on the test dataset
    evaluate(model2, data_loader_test, device=device)
```

```
Epoch: [8]  [110/120]  eta: 0:00:02  lr: 0.000000  loss: 0.0848 (0.1096)  los
Epoch: [8]  [119/120]  eta: 0:00:00  lr: 0.000000  loss: 0.1135 (0.1112)  los
Epoch: [8] Total time: 0:00:24 (0.2074 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:22  model_time: 0.0919 (0.0919)  evaluator_time: 0.
Test:  [49/50]  eta: 0:00:00  model_time: 0.0469 (0.0476)  evaluator_time: 0.
Test: Total time: 0:00:03 (0.0653 s / it)
Averaged stats: model_time: 0.0469 (0.0476)  evaluator_time: 0.0010 (0.0013)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.5
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.9
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.5
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.5
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.2
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.58
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.58
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.12
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.61
Epoch: [9]  [  0/120]  eta: 0:01:15  lr: 0.000000  loss: 0.1117 (0.1117)  loss
Epoch: [9]  [ 10/120]  eta: 0:00:26  lr: 0.000000  loss: 0.0691 (0.0763)  loss
Epoch: [9]  [ 20/120]  eta: 0:00:22  lr: 0.000000  loss: 0.0790 (0.0982)  loss
Epoch: [9]  [ 30/120]  eta: 0:00:19  lr: 0.000000  loss: 0.0941 (0.1041)  loss
Epoch: [9]  [ 40/120]  eta: 0:00:17  lr: 0.000000  loss: 0.0779 (0.0992)  loss
Epoch: [9]  [ 50/120]  eta: 0:00:14  lr: 0.000000  loss: 0.0779 (0.1007)  loss
Epoch: [9]  [ 60/120]  eta: 0:00:12  lr: 0.000000  loss: 0.1129 (0.1063)  loss
Epoch: [9]  [ 70/120]  eta: 0:00:10  lr: 0.000000  loss: 0.1066 (0.1097)  loss
Epoch: [9]  [ 80/120]  eta: 0:00:08  lr: 0.000000  loss: 0.0865 (0.1069)  loss
Epoch: [9]  [ 90/120]  eta: 0:00:06  lr: 0.000000  loss: 0.0662 (0.1055)  loss
Epoch: [9]  [100/120]  eta: 0:00:04  lr: 0.000000  loss: 0.0872 (0.1076)  loss
Epoch: [9]  [110/120]  eta: 0:00:02  lr: 0.000000  loss: 0.0872 (0.1085)  loss
Epoch: [9]  [119/120]  eta: 0:00:00  lr: 0.000000  loss: 0.0753 (0.1087)  loss
Epoch: [9] Total time: 0:00:25 (0.2095 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:23  model_time: 0.1162 (0.1162)  evaluator_time: 0.0
Test:  [49/50]  eta: 0:00:00  model_time: 0.0462 (0.0486)  evaluator_time: 0.0
Test: Total time: 0:00:03 (0.0669 s / it)
Averaged stats: model_time: 0.0462 (0.0486)  evaluator_time: 0.0011 (0.0013)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.52
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.94
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.56
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.11
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.55
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.28
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.60
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.60
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.0
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.17
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.63
```

```
# pick one image from the test set
img, _ = dataset_test[0]
# put the model in evaluation mode
model2.eval()
with torch.no_grad():
    prediction2 = model2([img.to(device)])
```

```
from torchvision.utils import draw_bounding_boxes
from torchvision.io import read_image
from torchvision import transforms
transform = transforms.Compose([
    transforms.PILToTensor()
])
IMG = Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
img_tensor = transform(IMG)
BB = prediction[0]['boxes']
scores = prediction[0]['scores']
for  i ,score in enumerate(scores):
  if score > 0.2:
    show = draw_bounding_boxes(img_tensor, BB[i][None, :],width=3)
show = draw_bounding_boxes(img_tensor, BB,width=3)
show = torchvision.transforms.ToPILImage()(show)
show
```

```
beatles = torchvision.io.read_image("/content/Beatles_-_Abbey_Road.jpg")
beatles = beatles.div(255)
model2.eval()
with torch.no_grad():
    predictionBeatles = model2([beatles.to(device)])
```

```
beatles = torchvision.io.read_image("/content/Beatles_-_Abbey_Road.jpg")
BBbeatles = predictionBeatles[0]['boxes']
scores = predictionBeatles[0]['scores']
for  i ,score in enumerate(scores):
  if score > 0.15:
    beatles = draw_bounding_boxes(beatles, BBbeatles[i][None, :],width=3)
#show1 = draw_bounding_boxes(beatles, BBbeatles, width=3)
show1 = torchvision.transforms.ToPILImage()(beatles)
show1
```

Looks pretty good!

Wrapping up

In this tutorial, you have learned how to create your own training pipeline for instance segmentation models, on a custom dataset. For that, you wrote a `torch.utils.data.Dataset` class that returns the images and the ground truth boxes and segmentation masks. You also leveraged a Mask R-CNN model pre-trained on COCO train2017 in order to perform transfer learning on this new dataset.

For a more complete example, which includes multi-machine / multi-gpu training, check `references/detection/train.py`, which is present in the [torchvision GitHub repo](https://github.com/pytorch/vision).

My model is performing relatively worse than the original model in the tutorial Resnet50. I have also tuned the output according to the scores, this score has been tuned for each model to remove inaccurate boxes/detection.