**Object Detection from Scratch**

[[Sharath S Hebbar](https://medium.com/@sharathhebbar24?source=post_page-----5ec93520adda--------------------------------)](https://medium.com/@sharathhebbar24?source=post_page-----5ec93520adda--------------------------------)

[Sharath S Hebbar](https://medium.com/@sharathhebbar24?source=post_page-----5ec93520adda--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fa5947dc8a8ee&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40sharathhebbar24%2Fobject-detection-from-scratch-5ec93520adda&user=Sharath+S+Hebbar&userId=a5947dc8a8ee&source=post_page-a5947dc8a8ee----5ec93520adda---------------------post_header-----------)

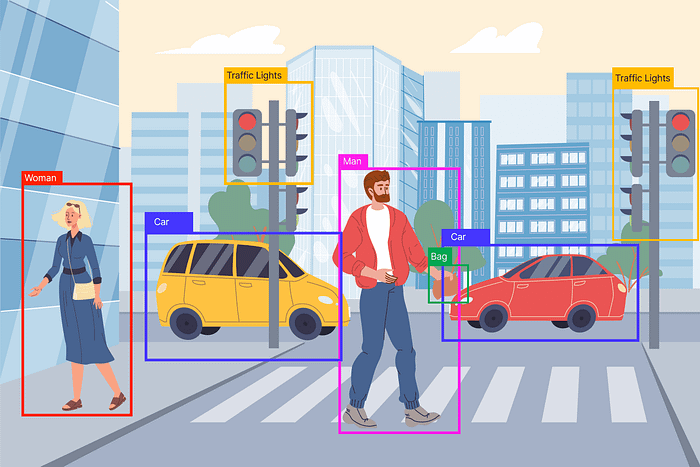
4 min read

·

3 days ago

1

[**GitHub**](https://github.com/SharathHebbar)[**LinkedIn**](https://www.linkedin.com/in/sharath-s-hebbar-14b541172/)[**Medium**](https://medium.com/@sharathhebbar24)[**Portfolio**](https://www.notion.so/ssh-blogs/Sharath-S-Hebbar-e7a58e16a1154467a74c1f1db1db0569)[**Substack**](https://sharathshebbar.substack.com/)



Object Detection.

Object detection is a pivotal task in computer vision, empowering machines to identify and locate objects within an image or video. In this article, we’ll embark on a journey to understand and implement object detection from scratch, demystifying the complex process behind this fascinating field.

The Object Detection takes into account the object in the image and its class (Classification) and its bounding box(Regression).

Several factors are involved in building Object Detection Like classifying Objects, finding bounding boxes, Region proposals, and anchor boxes, but here I have only considered the Regression and Classification part to build a simple Object Detection Model.

**Importing Libraries**

import os  
import shutil  
from glob import glob  
from pathlib import Path, PurePath  
  
import pandas as pd  
import numpy as np  
  
import matplotlib.pyplot as plt  
import matplotlib.patches as mpatches  
import seaborn as sns  
  
from PIL import Image  
import cv2  
  
import xml.dom.minidom as minidom  
import xmltodict  
  
from collections import Counter  
  
from tqdm import tqdm  
  
from sklearn.model\_selection import train\_test\_split  
  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
from torch.utils.data import Dataset, DataLoader  
import torchvision  
from torchvision.transforms import transforms  
from torchvision import datasets, models, transforms

**Previewing Images**

ANNOTATIONS = "/kaggle/input/face-mask-detection/annotations"  
IMAGES = "/kaggle/input/face-mask-detection/images"  
  
images = glob(IMAGES + "/\*")  
images = sorted(images)  
len(images)  
  
annotations = glob(f"{ANNOTATIONS}/\*")  
annotations = sorted(annotations)  
len(annotations)  
  
first\_nine\_images\_shape = []  
rows, cols = 3, 3  
plt.figure(figsize=(12, 12))  
  
for ind, img in enumerate(images):  
 if ind==9:  
 break  
 im = Image.open(img)  
 first\_nine\_images\_shape.append(im.size)  
 plt.subplot(rows, cols, ind+1)  
 plt.imshow(im)  
 im.close()

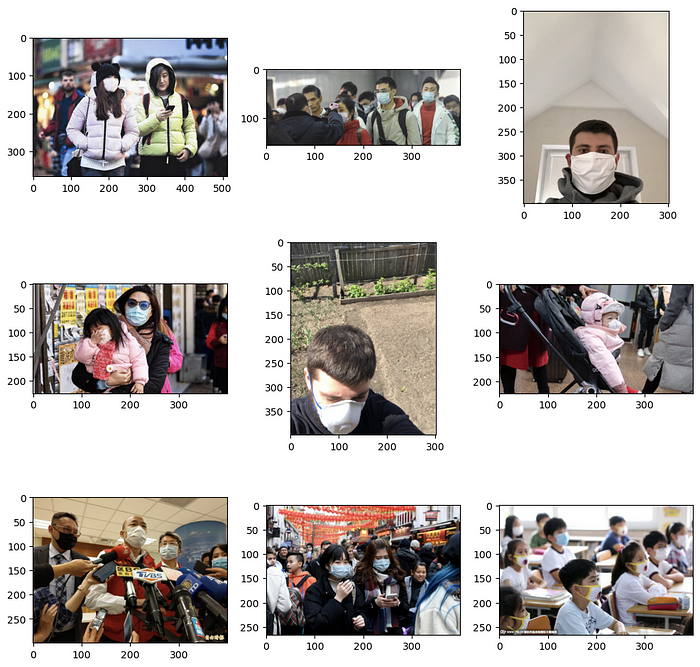


Image Preview

def annotate\_image(img, xml\_file):  
 with open(xml\_file) as f:  
 doc = xmltodict.parse(f.read())  
   
 image = plt.imread(img)  
   
 fig, ax = plt.subplots(1)  
 ax.axis("off")  
 fig.set\_size\_inches(10, 5)  
   
 temp = doc['annotation']['object']  
   
 if isinstance(temp, list):  
 for i in range(len(temp)):  
 if temp[i]["name"] == "with\_mask":  
 plot\_coords(temp[i]['bndbox'], 'green', temp[i]["name"],ax)  
   
 if temp[i]['name'] == "mask\_weared\_incorrect":  
 plot\_coords(temp[i]['bndbox'], 'yellow', temp[i]["name"], ax)  
   
 if temp[i]['name'] == "without\_mask":  
 plot\_coords(temp[i]['bndbox'], 'red', temp[i]["name"], ax)  
 else:  
 x, y, w, h = list(map(int, temp["bndbox"].values()))  
 edgecolor = {"with\_mask": "g", "without\_mask": "r", "mask\_weared\_incorrect": "y"}  
 mpatch = mpatches.Rectangle(  
 (x, y),  
 w-x, h-y,  
 linewidth=1,  
 edgecolor=edgecolor[temp["name"]],  
 facecolor="none")  
 ax.add\_patch(mpatch)  
 rx, ry = mpatch.get\_xy()  
 ax.annotate(temp["name"], (rx, ry),  
 color=edgecolor[temp["name"]], weight='bold',  
 fontsize=10, ha='left', va='baseline')  
 ax.imshow(image)



Face Mask

**Create Image Data Loader and Dataset**

train = datasets.ImageFolder(annotated\_dataset\_folder,  
 transform = transforms.Compose([  
 transforms.Resize(IMG\_WIDTH),  
 transforms.RandomCrop(IMG\_WIDTH),  
 transforms.ToTensor()  
 ]))  
train\_data\_loader = torch.utils.data.DataLoader(train, 32, shuffle=True, num\_workers=3, pin\_memory=True)

def get\_mean\_std(data\_loader):  
 sum\_, squared\_sum, batches = 0, 0, 0  
 for data, \_ in data\_loader:  
 sum\_ += torch.mean(data, dim=([0, 2, 3]))  
 squared\_sum += torch.mean(data \*\* 2, dim=([0, 2, 3]))  
 batches += 1  
   
 mean = sum\_ / batches  
 std = (squared\_sum / batches - mean \*\* 2) \*\* 0.5  
 return mean, std  
  
mean, std = get\_mean\_std(train\_data\_loader)  
mean, std

train\_transform = transforms.Compose([  
 transforms.ToPILImage(),  
 transforms.Resize(IMG\_HEIGHT),  
 transforms.RandomCrop(IMG\_WIDTH),  
 transforms.RandomHorizontalFlip(),  
 transforms.ToTensor(),  
 transforms.Normalize(mean, std)  
])  
  
test\_transform = transforms.Compose([  
 transforms.ToPILImage(),  
 transforms.Resize(IMG\_HEIGHT),  
 transforms.CenterCrop(IMG\_HEIGHT),  
 transforms.ToTensor(),  
 transforms.Normalize(mean, std)])

class CustomTensorDataset(Dataset):  
   
 def \_\_init\_\_(self, tensors, transforms=None):  
 self.tensors = tensors  
 self.transforms = transforms  
   
 def \_\_getitem\_\_(self, index):  
 label = self.tensors[1][index]  
 bbox = self.tensors[2][index]  
   
 image = cv2.imread(self.tensors[0][index])  
 image = torch.tensor(image, dtype=torch.float32).permute(2, 0, 1)  
   
 if self.transforms:  
 image = self.transforms(image)  
   
 return (image, label, bbox)  
   
 def \_\_len\_\_(self):  
 return self.tensors[0].shape[0]

trainset = CustomTensorDataset((train\_images, train\_labels, train\_bbox),  
 transforms=train\_transform)  
testset = CustomTensorDataset((test\_images, test\_labels, test\_bbox),  
 transforms=test\_transform)  
  
train\_data\_loader = torch.utils.data.DataLoader(trainset, BATCH\_SIZE, shuffle=True)  
test\_data\_loader = torch.utils.data.DataLoader(testset, BATCH\_SIZE, shuffle=True)

**Model Architecture of a Simple Object Detection**

For simplicity's sake, I took the image size as 224, 224, 3

class ObjectDetection(nn.Module):  
   
 def \_\_init\_\_(self, in\_channels=3, out\_channels\_cnn=3, bboxes=4):  
 super().\_\_init\_\_()  
 hidden\_channel1 = 32  
 hidden\_channel2 = 64  
 hidden\_channel3 = 128  
 kernel\_size = 3  
 stride = 1  
 padding = 1  
 self.conv1 = nn.Conv2d(in\_channels=in\_channels,  
 out\_channels=hidden\_channel1,  
 kernel\_size=kernel\_size,   
 stride=stride,   
 padding=padding)  
 self.conv2 = nn.Conv2d(in\_channels=hidden\_channel1,  
 out\_channels=hidden\_channel2,  
 kernel\_size=kernel\_size,   
 stride=stride,   
 padding=padding)  
 self.conv3 = nn.Conv2d(in\_channels=hidden\_channel2,  
 out\_channels=hidden\_channel2,  
 kernel\_size=kernel\_size,   
 stride=stride,   
 padding=padding)  
 self.conv4 = nn.Conv2d(in\_channels=hidden\_channel2,  
 out\_channels=hidden\_channel3,  
 kernel\_size=kernel\_size,   
 stride=stride,   
 padding=padding)  
 self.conv5 = nn.Conv2d(in\_channels=hidden\_channel3,  
 out\_channels=hidden\_channel3,  
 kernel\_size=kernel\_size,   
 stride=stride,   
 padding=padding)  
   
 self.batchnorm1 = nn.BatchNorm2d(hidden\_channel1)  
 self.batchnorm2 = nn.BatchNorm2d(hidden\_channel2)  
 self.batchnorm3 = nn.BatchNorm2d(hidden\_channel3)  
 self.maxpool = nn.MaxPool2d(kernel\_size=2)  
 self.relu = nn.ReLU()  
 self.fc = nn.Flatten()  
 self.cnn\_layer = nn.Linear(7\*7\*128, out\_channels\_cnn)  
 self.regressor = nn.Linear(7\*7\*128, bboxes)  
   
 def cnn\_layers(self, x):  
 x = self.relu(x)  
 x = self.maxpool(x)  
 return x  
   
 def feature\_extractor(self, x):  
  
 x = self.conv1(x)  
 x = self.batchnorm1(x)  
 x = self.cnn\_layers(x)  
   
 x = self.conv2(x)  
 x = self.batchnorm2(x)  
 x = self.cnn\_layers(x)  
   
 x = self.conv3(x)  
 x = self.batchnorm2(x)  
 x = self.cnn\_layers(x)  
   
 x = self.conv4(x)  
 x = self.batchnorm3(x)  
 x = self.cnn\_layers(x)  
   
 x = self.conv5(x)  
 x = self.batchnorm3(x)  
 x = self.cnn\_layers(x)  
   
 x = self.fc(x)  
 return x  
   
 def forward(self, x):  
 x = self.feature\_extractor(x)  
 classifier\_op = self.cnn\_layer(x)  
 regressor\_op = self.regressor(x)  
 return (regressor\_op, classifier\_op)

model = ObjectDetection(in\_channels=3, out\_channels\_cnn=3, bboxes=4)  
model = model.to(device)

**Training Model**

EPOCHS = 10  
train\_loss = []  
train\_accuracy = []  
test\_loss = []  
test\_accuracy = []  
  
for epoch in tqdm(range(EPOCHS)):  
 correct = 0  
 iterations = 0  
 iter\_loss = 0  
 model.train()  
 for i, (images, labels, bbox) in enumerate(train\_data\_loader):  
 images = images.to(device)  
 labels = labels.to(device)  
 bbox = bbox.to(device)  
  
 regressor, classifier = model(images)  
  
 \_, predicted = torch.max(classifier, 1) ## To get the labels of predicted   
 predicted\_bbox = bbox + regressor ## to get the bbox of the predicted (add the regression offset with the original bbox)  
  
 clf\_loss = classification\_loss\_fn(classifier, labels)  
 reg\_loss = bbox\_loss\_fn(predicted\_bbox, bbox)  
   
 total\_loss = (clf\_loss + reg\_loss).clone().detach().requires\_grad\_(True)  
  
 opt.zero\_grad()  
 total\_loss.backward()  
 opt.step()  
   
 iter\_loss += total\_loss.item()  
 correct += (predicted == labels).sum().item()  
 iterations += 1  
   
 train\_loss.append(iter\_loss / iterations)  
 train\_accuracy.append((100 \* correct / len(trainset)))  
 print(f"Epoch [{epoch + 1} / {EPOCHS}], Training Loss: {train\_loss[-1]:.3f}, Training Accuracy: {train\_accuracy[-1]:.3f}")

**Note: This is just a simple model that needs to be added in so many things.**

**References**

**[ML-Project-list/computer-vision/object-detection/face\_mask\_detection/object-detection-using-pytorch…](https://github.com/SharathHebbar/ML-Project-list/blob/master/computer-vision/object-detection/face_mask_detection/object-detection-using-pytorch.ipynb?source=post_page-----5ec93520adda--------------------------------" \t "_blank)**

[List of all ML projects. Contribute to SharathHebbar/ML-Project-list development by creating an account on GitHub.](https://github.com/SharathHebbar/ML-Project-list/blob/master/computer-vision/object-detection/face_mask_detection/object-detection-using-pytorch.ipynb?source=post_page-----5ec93520adda--------------------------------" \t "_blank)

[github.com](https://github.com/SharathHebbar/ML-Project-list/blob/master/computer-vision/object-detection/face_mask_detection/object-detection-using-pytorch.ipynb?source=post_page-----5ec93520adda--------------------------------" \t "_blank)

**[Object-Detection using PyTorch](https://www.kaggle.com/code/sharathshebbar/object-detection-using-pytorch?source=post_page-----5ec93520adda--------------------------------" \t "_blank)**

[Explore and run machine learning code with Kaggle Notebooks | Using data from Face Mask Detection](https://www.kaggle.com/code/sharathshebbar/object-detection-using-pytorch?source=post_page-----5ec93520adda--------------------------------" \t "_blank)

[www.kaggle.com](https://www.kaggle.com/code/sharathshebbar/object-detection-using-pytorch?source=post_page-----5ec93520adda--------------------------------" \t "_blank)