Netwatch Object Recognition

Technical Manual

by

Benjamin Tremblay

Institute of Technology Carlow

Dr. Oisin Cawley

April 29, 2022

Table of contents

Faster R-CNN Model Fine-Tuning

datasets.py

```python
##

# @file datasets.py

# Python file containing the custom dataset class structure.

#


import os

import torch

from torch.utils.data import Dataset

from PIL import Image

from pycocotools.coco import COCO

from transforms import get_transform


## Custom dataset class for loading COCO formatted data.

class NetwatchDataset(Dataset):


    ## Class constructor.

    # @param self The object pointer.

    # @param root The path to the image data directory.

    # @param annotation The path to the image JSON annotation file.

    # @param transforms The composed transformations to use on the image data.

    # @return An initialised NetwatchDataset object.

    def __init__(self, root, annotation, transforms=None):

        self.root = root

        self.transforms = transforms

        self.coco = COCO(annotation)

        self.imgs = list(sorted(self.coco.imgs.keys()))


    ## Method that gets an item with the specified index from the items list.

    # @param self The object pointer.
    # @param index The index of the item to get from the item list.

       # @return tuple(img, target) The image object and its corresponding target
annotation data.

    def __getitem__(self, index):

        coco = self.coco

        # Image ID

        img_id = self.imgs[index]

        # Loads all the annotation IDs in an image

        ann_ids = coco.getAnnIds(imgIds=img_id)

        # Loads all the annotation data for all unique annotations in an image

        coco_annotation = coco.loadAnns(ann_ids)

        # Load image

        path = coco.loadImgs(img_id)[0]['file_name']

        img = Image.open(os.path.join(self.root, path))


        # Number of objects in the image

        num_objs = len(coco_annotation)


        boxes = []

        labels = []

        areas = []

        # In COCO format, bbox = [xmin, ymin, width, height]

        # In PyTorch, the input should be [xmin, ymin, xmax, ymax]

        for i in range(num_objs):

            xmin = coco_annotation[i]['bbox'][0]

            ymin = coco_annotation[i]['bbox'][1]

            xmax = xmin + coco_annotation[i]['bbox'][2]

            ymax = ymin + coco_annotation[i]['bbox'][3]

            boxes.append([xmin, ymin, xmax, ymax])

            labels.append(coco_annotation[i]['category_id'])

            areas.append(coco_annotation[i]['area'])


        # Convert everything into a torch.Tensor

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.as_tensor(labels, dtype=torch.int64)

        image_id = torch.tensor([img_id])

        areas = torch.as_tensor(areas, dtype=torch.float32)

        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)


        target = {}

        target["boxes"] = boxes

        target["labels"] = labels

        target["image_id"] = image_id

        target["area"] = areas

        target["iscrowd"] = iscrowd


        if self.transforms is not None:

            img = self.transforms(img)


        return img, target


    ## Method that gets the size of the dataset.

    # @param self The object pointer.

    # @return The length of the dataset object's item list.

    def __len__(self):

        return len(self.imgs)
```

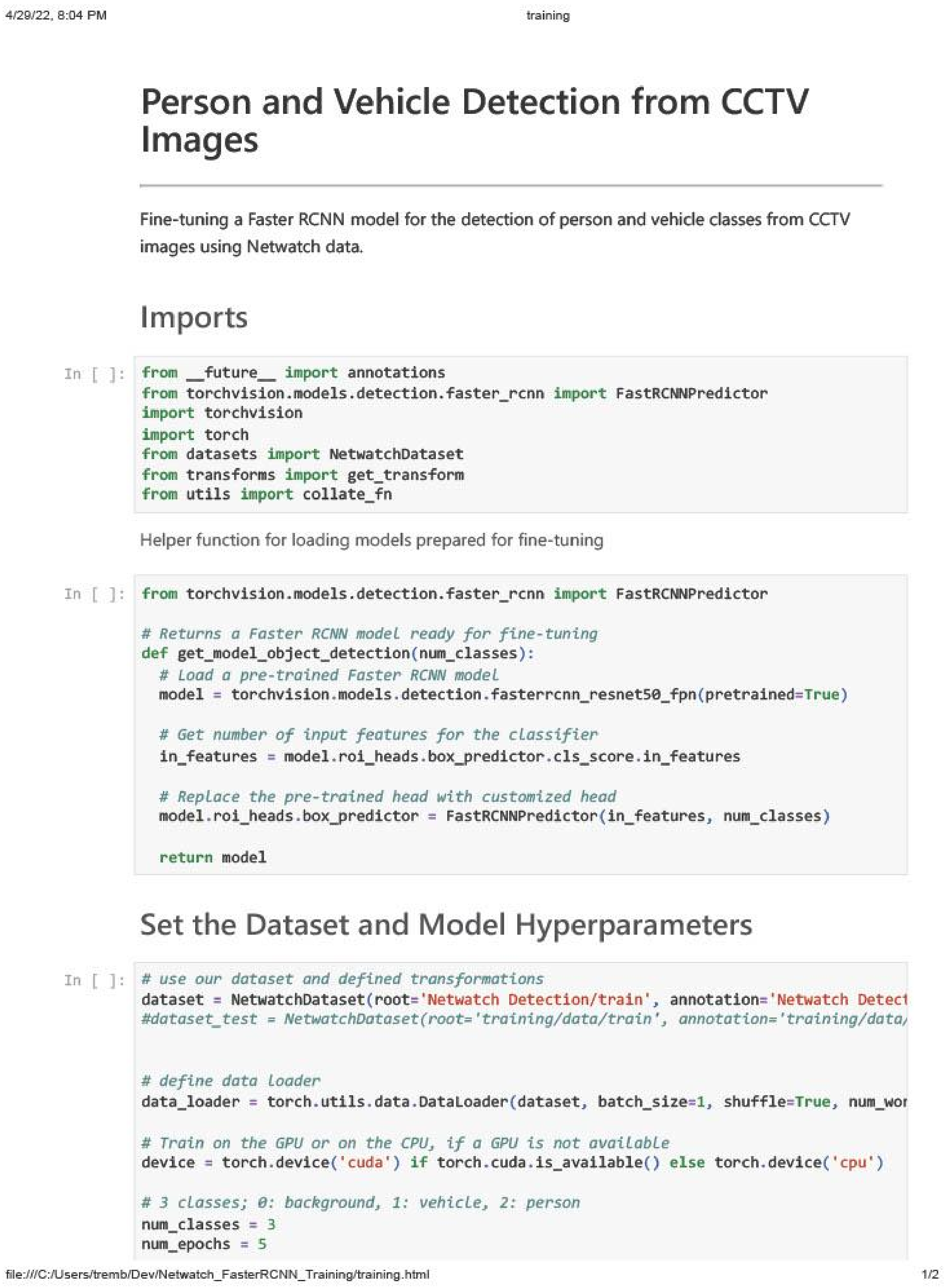
transforms.py

```
##

# @file transforms.py

# Python file containing data transform related code.

#

import torchvision.transforms as T



## Method for composing multiple transforms.

# @return transforms A composed list of transforms.

def get_transform():

    transforms = []

    transforms.append(T.ToTensor())

    return T.Compose(transforms)
```

utils.py

```
##

# @file utils.py

# Python file containing utility functions.

#

## Custom collate_fn method for padding batch data.

# @param batch The batched data being loaded.

def collate_fn(batch):

    return tuple(zip(*batch))
```

training.ipynb



Person and Vehicle Detection from CCTV Images

Fine-tuning a Faster RCNN model for the detection of person and vehicle classes from CCTV images using Netwatch data.

Imports

In [ ]:

```
from __future__ import annotations
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
import torchvision
import torch
from datasets import NetwatchDataset
from transforms import get_transform
from utils import collate_fn
```

Helper function for loading models prepared for fine-tuning

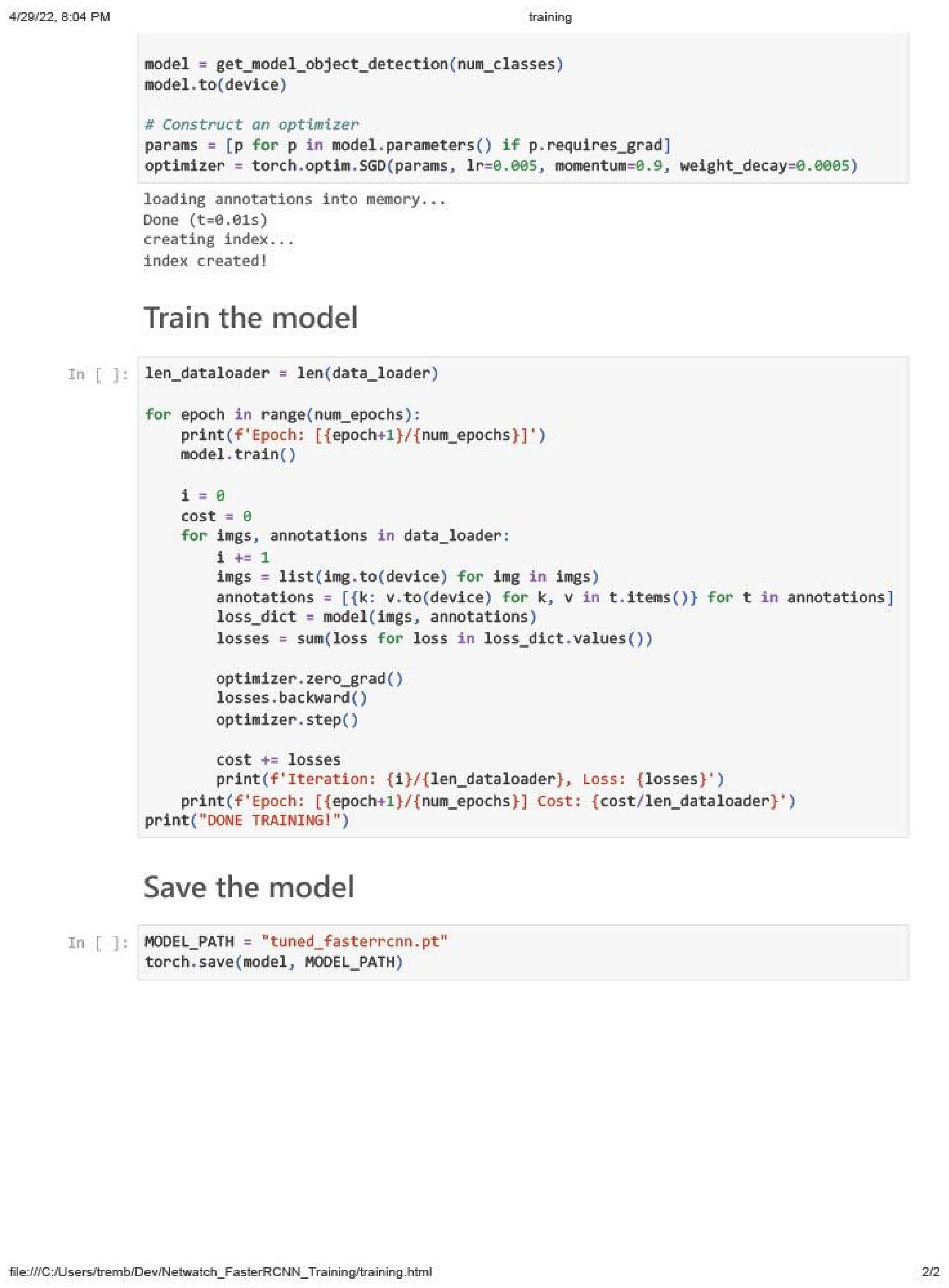
In [ ]:

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

# Returns a Faster RCNN model ready for fine-tuning
def get_model_object_detection(num_classes):
  # Load a pre-trained Faster RCNN model
  model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

  # 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 customized head
  model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

  return model
```

Set the Dataset and Model Hyperparameters

In [ ]:

```
# use our dataset and defined transformations
dataset = NetwatchDataset(root='Netwatch Detection/train', annotation='Netwatch Detect
#dataset_test = NetwatchDataset(root='training/data/train', annotation='training/data/


# define data loader
data_loader = torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=True, num_wor

# Train on the GPU or on the CPU, if a GPU is not available
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# 3 classes; 0: background, 1: vehicle, 2: person
num_classes = 3
num_epochs = 5
```



4/29/22, 8:04 PM training

```python
model = get_model_object_detection(num_classes)
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)
```

```
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
```

Train the model

In [ ]:

```python
len_dataloader = len(data_loader)

for epoch in range(num_epochs):
    print(f'Epoch: [{epoch+1}/{num_epochs}]')
    model.train()

    i = 0
    cost = 0
    for imgs, annotations in data_loader:
        i += 1
        imgs = list(img.to(device) for img in imgs)
        annotations = [{k: v.to(device) for k, v in t.items()} for t in annotations]
        loss_dict = model(imgs, annotations)
        losses = sum(loss for loss in loss_dict.values())

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

        cost += losses
        print(f'Iteration: {i}/{len_dataloader}, Loss: {losses}')
    print(f'Epoch: [{epoch+1}/{num_epochs}] Cost: {cost/len_dataloader}')
print("DONE TRAINING!")
```

Save the model

In [ ]:

```python
MODEL_PATH = "tuned_fasterrcnn.pt"
torch.save(model, MODEL_PATH)
```

file:///C:/Users/tremb/Dev/Netwatch_FasterRCNN_Training/training.html 2/2

Flask Web Application

custom_utils.py

```
# File containing all utility functions

#Imports
import cv2, os
from PIL import Image

# Function that loads the image from the file into a ndarray of shape
(H x W x C)
def loadImage(image_path):
    return cv2.imread(image_path)

# Function that removes every file from the given directory
def clearDirectory(directory_path):
    for fileName in os.listdir(directory_path):
        os.remove(os.path.join(directory_path, fileName))

# Function that converts images from a specified directory into a GIF
def createGif(imagePath):
    imgs = []
    for f in os.listdir(imagePath):
        img = Image.open(os.path.join(imagePath, f))
        imgs.append(img)

    image = imgs[0]

image.save(fp=r"C:\Users\tremb\Dev\PyTorch\app\static\outputs\sequence.
gif",
        format='GIF', append_images=imgs, save_all=True, duration=400,
loop=0)
```

detector.py

```
# Detector class file

# Imports
import torch
from torchvision import transforms as T
import matplotlib.pyplot as plt
import cv2
from custom_utils import loadImage

# Class for the object detection model instantiation
class Detector:
    # Class names that the fine-tuned model is trained on
    MY_CATEGORY_NAMES = [
    '__background__', 'vehicle', 'person']

    # Loads pre-trained model and sets it to evaluation mode
    def __init__(self):
        # Sets the device to gpu if available, else it uses cpu
        self.device = torch.device("cuda:0" if
torch.cuda.is_available() else "cpu")
        # Loads the model
        self.model = torch.load("app/model/tuned_fasterrcnn.pt")
        self.model.to(self.device).eval()

    # Method that inferences the model on the image input and returns
the predictions
    def get_prediction(self, imageFile, threshold):
        image = loadImage(imageFile)
        transform = T.Compose([T.ToTensor()])# Transforms the loaded
image (with normalisation) into a float tensor of shape (C x H x W)
        imgTensor = transform(image).to(self.device)

        # Perform the prediction on the transformed image (tensor)
        pred = self.model([imgTensor])
        # Adding each predicted class to the pred_class array
        pred_class = [self.MY_CATEGORY_NAMES[i] for i in
list(pred[0]['labels'].cpu().numpy())]
```

```
        # Adding each predicted bounding box to the pred_boxes array
        pred_boxes = [[(int(i[0]), int(i[1])), (int(i[2]), int(i[3]))]
for i in list(pred[0]['boxes'].detach().cpu().numpy())]
        # Adding each prediction score to the pred_score array
        pred_score = list(pred[0]['scores'].detach().cpu().numpy())

        # If there are no predictions, or there is no score that is
above the threshold
        if(len(pred_score) == 0 or max(pred_score) < threshold):
            pred_box = []
            pred_class = []

        # Else filter through pred_scores and get the last index where
the score > threshold
        # Assign the box and class arrays with index values until
pred_t index
        else:
            pred_t = [pred_score.index(x) for x in pred_score if x >
threshold][-1]
            pred_box = pred_boxes[:pred_t+1]
            pred_class = pred_class[:pred_t+1]

        # Returns a tuple containing the final arrays storing the
inference results
        return (pred_box, pred_class)

    # Method that saves an image with drawn bounding boxes and labels
on it
    def create_visualization(self, imageFile, fileName, boxes, classes,
rect_thickness, text_thickness, text_size):
        image = loadImage(imageFile)
        img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Assigning the boxes and classes from the inference
        boxes_ = boxes
        classes_ = classes

        # Each box means a separate detection
        for i in range(len(boxes)):
            # Only display data if the class == 'person'
            if(classes[i] == 'person'):
                #r,g,b,a = (255, 56, 56, 1) # red
                r,g,b,a = (255, 105, 180, 1) # pink
                # Draw box with the coordinates
                cv2.rectangle(img, boxes_[i][0], boxes_[i][1],
color=(r,g,b,a), thickness=rect_thickness)
                # Put the text that goes with the box
                cv2.putText(img, classes_[i], boxes_[i][0],
cv2.FONT_HERSHEY_SIMPLEX, text_size, (r,g,b,a),
thickness=text_thickness)
            elif(classes[i] == 'car' or classes[i] == 'vehicle'):
                #r,g,b,a = (56, 56, 255, 1) # blue
                r,g,b,a = (0, 255, 255, 1) # cyan
                # Draw box with the coordinates
                cv2.rectangle(img, boxes_[i][0], boxes_[i][1],
color=(r,g,b,a), thickness=rect_thickness)
                # Put the text that goes with the box
                cv2.putText(img, classes_[i], boxes_[i][0],
cv2.FONT_HERSHEY_SIMPLEX, text_size, (r,g,b,a),
thickness=text_thickness)

        # Create output image
        dpi = 96
        plt.figure(frameon=False, figsize=(352/dpi, 288/dpi), dpi=dpi)
# 352 x 288
        plt.imshow(img)
        plt.xticks([])
        plt.yticks([])
        plt.subplots_adjust(top=1, bottom=0, right=1, left=0, hspace=0,
wspace=0)
        plt.savefig("C:/Users/tremb/Dev/PyTorch/app/outputs/"+fileName,
dpi=dpi)

plt.savefig("C:/Users/tremb/Dev/PyTorch/app/static/outputs/"+fileName,
dpi=dpi)
```

app.py

```
# Main application file

# Imports
import os, secrets
from flask import Flask, request, render_template, flash,
get_flashed_messages
from detector import Detector
from custom_utils import clearDirectory, createGif

secret = secrets.token_urlsafe(32)

app = Flask(__name__)
app.config["TEMPLATES_AUTO_RELOAD"] = True
app.config["UPLOAD_FOLDER"] = r"C:\Users\tremb\Dev\PyTorch\app\uploads"
app.secret_key = secret

STATIC_OUTPUT_FOLDER = r"C:\Users\tremb\Dev\PyTorch\app\static\outputs"
OUTPUT_FOLDER = r"C:\Users\tremb\Dev\PyTorch\app\outputs"

detector = Detector()

# The home page route
@app.route('/')
def home():
    # Empty all the directories storing the temporary data
    clearDirectory(app.config["UPLOAD_FOLDER"])
    clearDirectory(STATIC_OUTPUT_FOLDER)
    clearDirectory(OUTPUT_FOLDER)
    return render_template("index.html")

# The detection route that acts as API
@app.route("/detect", methods=["POST"])
def upload_files():
    if request.method == "POST":
        # Get the uploaded files
        files = request.files.getlist("file")
        # Check number of files
        if (len(files) != 3):
            flash("You must select a full image sequence (3 images)",
"error")
            # Re-render home page
            return render_template("index.html",
message=get_flashed_messages()[0])
        else:
            outputImages = []
            nbOfDetections = 0

            for file in files:
                # Save image
                imagePath = os.path.join(app.config["UPLOAD_FOLDER"],
file.filename)
                file.save(imagePath)
                outputImages.append("outputs/"+file.filename)
                # Get detections from image
                boxes, classes = detector.get_prediction(imagePath,
0.70) # Confidence score threshold set to .70 or 70%
                # Increment the number of detections
                nbOfDetections += classes.count("person") +
classes.count("vehicle")
                # Draw detections on image
                detector.create_visualization(imagePath, file.filename,
boxes, classes, 2, 2, .7)
            # Create and save a GIF of the image sequence in the upload
directory
            createGif(app.config["UPLOAD_FOLDER"])
            # Render the result page with the outputImages and
nbOfDetection route parameters
            return render_template("result.html",
outputImages=outputImages, nbOfDetections=nbOfDetections)
    return render_template("index.html")

# Launches local web app when file is ran
if __name__ == "__main__":
    app.run()
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <link rel="stylesheet"
href="//stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.cs
s"
integrity="sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZ
WKYbI706tWS" crossorigin="anonymous">
    <style>
      .bd-placeholder-img {
        font-size: 1.125rem;
        text-anchor: middle;
      }

      @media (min-width: 768px) {
        .bd-placeholder-img-lg {
          font-size: 3.5rem;
        }
      }
    </style>
    <link rel="stylesheet" href="/static/style.css">

    <title>Object Detection in PyTorch</title>
  </head>
  <body class="text-center">
    {% if message %}
      <script>alert("{{message}}");</script>
    {% endif %}

      <form class="form-index" action="http://localhost:5000/detect"
method="POST" enctype="multipart/form-data">
        <img class="mb-4" src="/static/pytorch-logo.png" alt=""
width="72">
```

```
        <h1 class="h3 mb-3 font-weight-normal">Upload a Netwatch Image
Sequence</h1>
        <br/><br/>
        <input type="file" name="file" accept="image/png, image/jpg,
image/jpeg" class="form-control-file" id="inputfile" multiple/>
        <br/>
        <button class="btn btn-lg btn-primary btn-block" type="submit"
onclick="showLoading()">Detect</button>
        <div id="load" class="loading"
style="visibility:hidden;"></div>
      </form>
    <script>
      function showLoading(){
        document.getElementById("load").style.visibility = "visible";
      }
    </script>
    <script src="//code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8ab
tTE1Pi6jizo" crossorigin="anonymous"></script>
    <script
src="//cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.j
s"
integrity="sha384-wHAiFfRlMFy6i5SRaxvfOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu
1hA6ZOblgut" crossorigin="anonymous"></script>
    <script
src="//stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"
integrity="sha384-B0UglyR+jN6CkvvICOB2joaf5I4l3gm9GU6Hc1og6Ls7i6U/mkkad
uKaBhlAXv9k" crossorigin="anonymous"></script>
    <script type="text/javascript">
      $('#inputfile').bind('change', function() {
          let fileSize = this.files[0].size/1024/1024; // this gives in
MB
          if (fileSize > 1) {
            $("#inputfile").val(null);
            alert('file is too big. images more than 1MB are not
allowed')
            return
          }
          let ext =
$('#inputfile').val().split('.').pop().toLowerCase();
```

```
          if($.inArray(ext, ['jpg','jpeg']) == -1) {
            $("#inputfile").val(null);
            alert('Only jpeg/jpg files are allowed.');
          }
        });
    </script>
  </body>
</html>
```

result.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <link rel="stylesheet"
href="//stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.cs
s"
integrity="sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZ
WKYbI706tWS" crossorigin="anonymous">
    <style>
      .bd-placeholder-img {
        font-size: 1.125rem;
        text-anchor: middle;
      }
      @media (min-width: 768px) {
        .bd-placeholder-img-lg {
          font-size: 3.5rem;
        }
      }
    </style>
    <link rel="stylesheet" href="/static/style.css">
    <title>Object Detection in PyTorch</title>
    <script>
      var isPopupDisplayed = false;
      var modal = document.getElementById("myModal");
      var btn = document.getElementById("myBtn");
      var span = document.getElementsByClassName("close")[0];

      span.onclick = function(){
        modal.style.display = "none";
      }

      function showPopup(){
        document.getElementById("myModal").style.display = "block";
      }

      function closePopup(){
        document.getElementById("myModal").style.display = "none";
      }

      window.onclick = function(event){
        document.getElementById("myModal").style.display = "none";
      }
    </script>
  </head>
  <form action="{{ url_for('home') }}">
    <button class="btn btn-primary back-btn" onclick="">Back</button>
  </form>
  <body class="text-center">
    <div class="form-results">
        <h1 class="h3 mb-3 font-weight-normal"><b>Detection
Results</b></h1>
        <div class="image-sequence">
            <div class="image">
                <img src="{{url_for('static',
filename=outputImages[0])}}" alt="" width="350">
            </div>
            <div class="image">
                <img src="{{url_for('static',
filename=outputImages[1])}}" alt="" width="350">
            </div>
            <div class="image">
                <img src="{{url_for('static',
filename=outputImages[2])}}" alt="" width="350">
            </div>
        </div>
        <button id="myBtn" class="btn btn-lg btn-primary"
onclick="showPopup()">Show Animated Sequence</button>
        <div id="myModal" class="modal">
          <div class="modal-window">
            <span onclick="closePopup()" class="close">&times;</span>
            <div class="modal-window-content">
              <img src="{{url_for('static',
filename='outputs/sequence.gif')}}" alt="">
            </div>
          </div>
        </div>
        <div class="detection-info">
            <h5 class="h4 mb-3 font-weight-normal">Number of
Detections: <b>{{ nbOfDetections }}</b></h5>
            {% if nbOfDetections > 0 %}
            <h5 class="h4 mb-3 font-weight-normal">Human Activity:
<b>True</b></h5>
            {% else %}
            <h5 class="h4 mb-3 font-weight-normal">Human Activity:
<b>False</b></h5>
            {% endif %}
        </div>
    </div>
    <script src="//code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8ab
tTE1Pi6jizo" crossorigin="anonymous"></script>
    <script
src="//cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.j
s"
integrity="sha384-wHAiFfRlMFy6i5SRaxvfOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu
1hA6ZOblgut" crossorigin="anonymous"></script>
    <script
src="//stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"
integrity="sha384-B0UglyR+jN6CkvvICOB2joaf5I4l3gm9GU6Hc1og6Ls7i6U/mkkad
uKaBhlAXv9k" crossorigin="anonymous"></script>
  </body>
</html>
```

Blazor Web Application

IDetector.cs

```
namespace ObjectRecogntionWebApp.Model
{
    /// <summary>
    /// Interface for object detector classes.
    /// </summary>
    public interface IDetector
    {
        /// <summary>
        /// Method that performs detections on image input data.
        /// </summary>
        /// <param name="imageUrl">The file path to the
image.</param>
        /// <param name="labels">The HashSet containing class
labels for detection filtering.</param>
        /// <param name="threshold">The score threshold used for
detection filtering.</param>
        /// <returns>A list of detections.</returns>
        public List<Detection> DetectObjects(string imageUrl,
HashSet<string> labels, float threshold);
    }
}
```

FasterRCNN.cs

```
using Microsoft.ML.OnnxRuntime;
using Microsoft.ML.OnnxRuntime.Tensors;
using SixLabors.ImageSharp;
using SixLabors.ImageSharp.PixelFormats;
using SixLabors.ImageSharp.Processing;
using SixLabors.ImageSharp.Drawing.Processing;
using SixLabors.Fonts;
using SessionOptions = Microsoft.ML.OnnxRuntime.SessionOptions;
using System.Diagnostics;

namespace ObjectRecogntionWebApp.Model
{
    /// <summary>
    /// Class for instantiating a Faster RCNN detector.
    /// </summary>
    public class FasterRCNN : IDetector
    {
        InferenceSession Session;
        /// <summary>
        /// Class constructor that assigns an InferenceSession
using the model path.
        /// </summary>
        public FasterRCNN()
        {
            SessionOptions options = new SessionOptions();
            options.AppendExecutionProvider_CUDA(0);
            Session = new
InferenceSession("C:/Users/tremb/source/repos/ObjectRecogntionWebA
pp/ObjectRecogntionWebApp/wwwroot/onnx/FasterRCNN-10.onnx",
options);
        }

        /// <summary>
        /// Method that performs object detection on image input
data.
        /// </summary>
        /// <param name="imagePath">The file path to the
image.</param>
        /// <param name="classes">The HashSet containing class
labels for detection filtering.</param>
        /// <param name="threshold">The score threshold used for
detection filtering.</param>
        /// <returns>A list of detections.</returns>
        public List<Detection> DetectObjects(string imagePath,
HashSet<string> classes, float threshold)
        {
            // Load image from path
            Image<Bgr24> image = Image.Load<Bgr24>(imagePath);
            // Setup input
            var input = new List<NamedOnnxValue>
            {
                NamedOnnxValue.CreateFromTensor("image",
PreProcessImage(image))
            };

            // Run inference
            Stopwatch sw = new Stopwatch();
            sw.Start();
            var results = Session.Run(input);
            sw.Stop();
            Debug.WriteLine(sw.ElapsedMilliseconds);

            // Post process results
            var resultsArray = results.ToArray();
            float[] boxes =
resultsArray[0].AsEnumerable<float>().ToArray();
            long[] labels =
resultsArray[1].AsEnumerable<long>().ToArray();
            float[] scores =
resultsArray[2].AsEnumerable<float>().ToArray();
            var detections = new List<Detection>();
            var minScore = threshold;

            // Iterating by 4 because every box has 4 sequenced
float values (xmin, ymin, xmax, ymax)
            for (int i = 0; i < boxes.Length - 4; i += 4)
            {
                // Divide by 4 to get appropriate bounding box
index
                var index = i / 4;
                string label = LabelMap.Labels[labels[index]];

                if (classes.Contains(label) && (scores[index] >=
minScore))
                {
                    // Passed the accepted confidence threshold,
add to Prediction list
                    detections.Add(new Detection
                    {
                        Box = new Box(boxes[i], boxes[i + 1],
boxes[i + 2], boxes[i + 3]),
                        Label = label,
                        Score = scores[index]
                    });
                }
            }
            return detections;
        }

        /// <summary>
        /// Method that uses detection data to draw over an image
for output visualization.
        /// </summary>
        /// <param name="imagePath">The original image that went
through detection.</param>
        /// <param name="detections">The detections that were
inferred from the image.</param>
        /// <returns></returns>
        public Image DrawDetections(string imagePath,
List<Detection> detections)
        {
            // Load image from path
            Image<Bgr24> image = Image.Load<Bgr24>(imagePath);

            // Set the font for text to be drawn on image
            Font font = SystemFonts.CreateFont("Arial", 13);

            // Get the ratio for resizing boxes to original image
            float ratio = 800f / Math.Min(image.Width,
image.Height);

            foreach (var d in detections)
            {
                // Resize the bounding box
                d.Box.Xmin /= ratio;
                d.Box.Ymin /= ratio;
                d.Box.Ymax /= ratio;
                d.Box.Xmax /= ratio;

                // Apply transformations to image
                image.Mutate(x =>
                {
                    // Draw bounding box over the image
                    x.DrawLines(Color.Red, 2f, new PointF[]
                    {
                        new PointF(d.Box.Xmin, d.Box.Ymin),
                        new PointF(d.Box.Xmax, d.Box.Ymin),

                        new PointF(d.Box.Xmax, d.Box.Ymin),
                        new PointF(d.Box.Xmax, d.Box.Ymax),

                        new PointF(d.Box.Xmax, d.Box.Ymax),
                        new PointF(d.Box.Xmin, d.Box.Ymax),

                        new PointF(d.Box.Xmin, d.Box.Ymax),
                        new PointF(d.Box.Xmin, d.Box.Ymin)
                    });
                    // Draw the class label and accuracy score
over the image
                    x.DrawText($"{d.Label}, {d.Score:0.00}", font,
Color.White, new PointF(d.Box.Xmin, d.Box.Ymin - 15));
                });
            }
            return image;
        }

        /// <summary>
        /// Method that pre processes an image into the proper
format for inference with Faster RCNN models.
        /// </summary>
        /// <param name="image">Image object to pre
process.</param>
        /// <returns>Processed image tensor object.</returns>
        private Tensor<float> PreProcessImage(Image<Bgr24> image)
        {
            var clonedImage = image.Clone();

            // Resize image
            float ratio = 800f / Math.Min(clonedImage.Width,
clonedImage.Height);
            clonedImage.Mutate(x => x.Resize((int)(ratio *
clonedImage.Width), (int)(ratio * clonedImage.Height)));

            // Pad to be divisible by 32
            var paddedHeight =
(int)(Math.Ceiling(clonedImage.Height / 32f) * 32f);
            var paddedWidth = (int)(Math.Ceiling(clonedImage.Width
/ 32f) * 32f);

            var processedInput = new DenseTensor<float>(new[] { 3,
paddedHeight, paddedWidth });

            // Normalise image
            var mean = new[] { 102.9801f, 115.9465f, 122.7717f };
            clonedImage.ProcessPixelRows(accessor =>
            {
                for (int y = paddedHeight - clonedImage.Height; y
< clonedImage.Height; y++)
                {
                    var pixelRow = accessor.GetRowSpan(y);
                    for (int x = paddedWidth - clonedImage.Width;
x < clonedImage.Width; x++)
                    {
                        processedInput[0, y, x] = pixelRow[x].B -
mean[0];
                        processedInput[1, y, x] = pixelRow[x].G -
mean[1];
                        processedInput[2, y, x] = pixelRow[x].R -
mean[2];
                    }
                }
            });
            return processedInput;
        }
    }

    /// <summary>
    /// Stores the coordinates of a bounding box.
    /// </summary>
    public class Box
    {
        public float Xmin { get; set; }
        public float Ymin { get; set; }
        public float Xmax { get; set; }
        public float Ymax { get; set; }

        public Box(float xmin, float ymin, float xmax, float ymax)
        {
            Xmin = xmin;
            Ymin = ymin;
            Xmax = xmax;
            Ymax = ymax;

        }
    }

    /// <summary>
    /// Stores data of a detection.
    /// </summary>
    public class Detection
    {
        public Box Box { get; set; }
        public string Label { get; set; }
        public float Score { get; set; }
    }
}
```

AppState.cs

```
namespace ObjectRecogntionWebApp.Model
{
    /// <summary>
    /// Class for storing the configuration state of the
application.
    /// </summary>
    public class AppState
    {
        // Default properties of the app
        public FasterRCNN Detector { get; set; } = new
FasterRCNN();
        public float Threshold { get; set; } = 0.7f;
        public HashSet<string> Classes { get; set; } = new
HashSet<string>{"person", "car"};
    }
}
```

Index.razor

```
@page "/"
@using System.Diagnostics
@inject IWebHostEnvironment Environment
@inject ILogger<Index> Logger
@inject IJSRuntime JsRuntime
@inject NavigationManager NavigationManager
@inject AppState State

<PageTitle>Home</PageTitle>
<body>
    <div class="container">
        <div class="centered">
            <img id="logo" src="mlnet_logo.png" alt=""
width="250">
            <h1 class="h3 mb-3 font-weight-normal
text-center">Upload a Netwatch Image Sequence</h1><br/>
            <div>
                <InputFile class="mb-3 form-control-file"
name="file" accept="image/png, image/jpg, image/jpeg"
id="inputfile" type="file" OnChange="@LoadFiles" multiple/>
            </div>
            <button class="btn btn-primary btn-lg detect-btn"
@onclick="HandleDetection">Detect</button>
        </div>
    </div>
</body>

@code{
    private List<string> loadedFilePaths = new();
    private string currPath = Directory.GetCurrentDirectory();
    // Function that validates and stores the user uploaded files
    private async Task LoadFiles(InputFileChangeEventArgs e)
    {
        if (e.FileCount != 3)
        {
            // Empty the loadedFilePaths List
            loadedFilePaths.Clear();
        }
        else
        {
            // Set the target upload directory path
            String uploadPath = @$"{currPath}\wwwroot\uploads";
            DirectoryInfo di = new DirectoryInfo(uploadPath);

            // Handle the upload directory
            if (di.Exists)
            {
                // Empty the directory
                var files = di.GetFiles();
                foreach(var file in files)
                {
                    file.Delete();
                }
                // Empty the loadedFilePaths List
                loadedFilePaths.Clear();
            }
            else
            {
                // Create the directory
                di.Create();
            }

            foreach (var file in e.GetMultipleFiles())
            {
                try
                {
                    var path =
Path.Combine(Environment.ContentRootPath, "wwwroot/uploads",
file.Name);
                    // Add the full image path to List
                    loadedFilePaths.Add(path);

                    // Save the file
                    await using FileStream fs = new(path,
FileMode.Create);
                    await file.OpenReadStream().CopyToAsync(fs);
                }
                catch (Exception ex)
                {
                    Logger.LogError("File: {Filename} Error:
{Error}",
                        file.Name, ex.Message);
                }
            }
        }
    }

    // Variable acting as a tracker to know when to display images
on the index page
    bool doneLoading;

    // Function for performing detections on input data
    private async void HandleDetection()
    {
        if (loadedFilePaths.Count == 3)
        {
            // Sets the target output directory path
            String outputPath = @$"{currPath}\wwwroot\outputs";
            DirectoryInfo di = new DirectoryInfo(outputPath);
            if (di.Exists)
            {
                // Empty the directory
                var files = di.GetFiles();
                foreach(var file in files)
                {
                    file.Delete();
                }
            }
            else
            {
                // Create the directory
                di.Create();
            }

            // Index counter to use in the output file names
            int index = 0;
            // Number of detections to display on results page
            int nbOfDetections = 0;
            //float sequenceInferenceTime = 0;

            // Perform detection on each of the 3 files that were
uploaded
            foreach (var filePath in loadedFilePaths)
            {
                index++;
                // Get detections
                var detections =
State.Detector.DetectObjects(filePath, State.Classes,
State.Threshold);
                // Increment detection count
                nbOfDetections += detections.Count();
                // Get image output
                Image imageOutput =
State.Detector.DrawDetections(filePath, detections);
                // Save image output to the output path

imageOutput.SaveAsJpeg(@$"{outputPath}\output_{index}.jpg");
            }
            // Clear the list containing the uploaded file paths
            loadedFilePaths.Clear();
            doneLoading = true;
            // Redirect user to the results page with the route
parameters
            NavigationManager.NavigateTo("results/" +
nbOfDetections, true);
        }
        else
        {
            // Alert message if the attempted number of files
uploaded is not equal to 3
            await JsRuntime.InvokeVoidAsync("alert", "You must
upload 3 images.");
```

```
        }
    }
}
```

Results.razor

```
@page "/results/{DetectionCount:int}"

<PageTitle>Results</PageTitle>
<body>
      <div class="container">
            <h3>Detection Results</h3>
            <span><img id="image-output" src="outputs\output_1.jpg"
/></span>
            <span><img id="image-output" src="outputs\output_2.jpg"
/></span>
            <span><img id="image-output" src="outputs\output_3.jpg"
/></span>
            <div class="results-info">
                  <div>
                        <label>Total Detections:
</label><b>@DetectionCount</b>
                  </div>
                  <div>
                        <label>Activity: </label><b>@IsActivity</b>
                  </div>
            </div>
      </div>
</body>

@code {

      // DetectionCount value that is received from the page route
      [Parameter]
      public int? DetectionCount { get; set; }

      // Variable for displaying human activity as true or false
      bool IsActivity { get; set; }

      // Sets the IsActivity to true or false on page
initialisation using the detection count parameter
      protected override void OnInitialized()
      {
            if(DetectionCount >= 1)
            {
                  IsActivity = true;
            }
            else
            {
                  IsActivity = false;
            }
      }
}
```