Module 6 Assignment

In this final assignment of the course, you'll use deep learning to count people in a dataset of images.

You'll use skills learned throughout the course to select, use and interpret the outputs of a pre-trained model that's suited to the task.

0) Setup

In [1]:
```
import mxnet as mx
from mxnet import image
from mxnet.gluon.data.vision import transforms
import gluoncv as gcv
import hashlib
from pylab import rcParams
from matplotlib import pyplot as plt
from gluoncv import model_zoo, data, utils
import numpy as np
import os
from pathlib import Path

rcParams['figure.figsize'] = 5, 10
```

In [2]:
```
M6_DATA = Path(os.getenv('DATA_DIR', '../../data'), 'module_6')
M6_IMAGES = Path(M6_DATA, 'images')
M6_MODELS = Path(M6_DATA, 'models')
```

1) Selecting an appropriate model

GluonCV has a wide range pre-trained models so selecting the correct model for the task at hand can be tricky. As you can see below there are close to 200 different pre-trained model variants. You first task will be to select a good model given a few constraints.

In [3]:
```
for model in gcv.model_zoo.get_model_list():
    print(model)
```

```
resnet18_v1
resnet34_v1
resnet50_v1
resnet101_v1
resnet152_v1
resnet18_v2
resnet34_v2
resnet50_v2
resnet101_v2
resnet152_v2
se_resnet18_v1
se_resnet34_v1
se_resnet50_v1
se_resnet101_v1
se_resnet152_v1
se_resnet18_v2
se_resnet34_v2
se_resnet50_v2
se_resnet101_v2
```

Our objective is to count people so certain types of model are better suited for this task than others. We also want a resource efficient model so let's say that the selected model must consume <1GB of memory. And lastly our model must be fast and be able to process >100 images per second on the [GluonCV website (https://gluon-cv.mxnet.io/model_zoo)](https://gluon-cv.mxnet.io/model_zoo) benchmarks. Given these criteria, you first task is to select the most appropriate model for the task from the following list.

In [4]:
```
model_options = ['senet_154',
                 'mobilenetv3_large',
                 'faster_rcnn_fpn_resnet101_v1d_coco',
                 'yolo3_darknet53_coco',
                 'fcn_resnet101_coco',
                 'deeplab_resnet101_coco']

# set the variable `selected_model`
# YOUR CODE HERE
#raise NotImplementedError()

# choices I've tried
#selected_model = 'yolo3_darknet53_coco'
#selected_model = 'fcn_resnet101_coco'
#selected_model = 'deeplab_resnet101_coco'
#selected_model = 'faster_rcnn_fpn_resnet101_v1d_coco'
#selected_model = 'mobilenetv3_large'
#selected_model = 'senet_154'

selected_model = 'yolo3_darknet53_coco'
```

In [5]:
```
assert selected_model in set(model_options), 'You must choose from mode
assert hashlib.sha1(selected_model.encode('utf-8')).hexdigest()[:8] in
assert hashlib.sha1(selected_model.encode('utf-8')).hexdigest()[:8] ==
```

In [6]:
```
network = gcv.model_zoo.get_model(selected_model, pretrained=True, root=
```

2) Load Image

Your first task is to implement a function that loads an image from disk given a filepath.

It should return an 8-bit image array, that's in MXNet's NDArray format and in HWC layout (i.e. height, width then channel).

Hint: you can reference your solution from Assignment 3.

In [7]:
```
def load_image(filepath):
    """
    Should load image from disk.

    :param filepath: relative or absolute filepath to RGB image file in
    :type filepath: str

    :return: an array with pixel intensities (in HWC layout).
    :rtype: mx.nd.NDArray
    """
    # YOUR CODE HERE
    #raise NotImplementedError()

    image = mx.image.image.imread(filepath)
    return image
```

In [8]:
```
test_filepath = Path(M6_IMAGES, '32742378405_3ecc8cc958_b.jpg')
test_output = load_image(test_filepath)
assert test_output.shape[2] == 3  # RGB
assert test_output.dtype == np.uint8  # 0 - 255
assert isinstance(test_output, mx.nd.NDArray)  # MXNet NDArray, not Numl
```

In [9]:
```
plt.imshow(test_output.asnumpy())
fig = plt.gcf()
fig.set_size_inches(14, 14)
plt.show()
```

3) Transform Image

Up next, you should transform the image so it can be used as input to the pre-trained network.

Make sure you return an un-normalized version of the image that can be used for visualizations.

See the docstring for more details, but don't forget that GluonCV contains a number of utilities and helper functions to make your life easier! Check out the presets transforms.

In [10]:
```
def transform_image(array):
    """
    Should transform image by:

    1) Resizing the shortest dimension to 416. e.g (832, 3328) -> (416,
    2) Cropping to a center square of dimension (416, 416).
    3) Converting the image from HWC layout to CHW layout.
    4) Normalizing the image using COCO statistics (i.e. per colour char
    5) Creating a batch of 1 image.

    :param filepath: array (in HWC layout).
    :type filepath: mx.nd.NDArray

    :return: a batch of a single transformed images (in NCHW layout) and
    :rtype: tuple of (mx.nd.NDArray, numpy.ndarray)
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    #width, height, channel = array.shape

    #print('height is {}'.format(width))
    #print('width is {}'.format(height))
    #print('channel is {}'.format(channel))

    # width is shortest
    #re_w = 416.0
    #factor = 416.0 / width
    #re_h = width * factor

    # height is shortest
    #re_h = 416.0
    #factor = 416.0 / height
    #re_w = height * factor


    # Use Compose to form sequence of transformations
    # transform_fn = gluon.data.vision.transforms.Compose([
    #transform_fn = gluon.data.vision.transforms.Compose([
    # transform_fn = transforms.Compose([
    #transform_fn = mx.gluon.data.vision.transforms.Compose([
    #    mx.gluon.data.vision.transforms.Resize(re_h,re_w),
    #    mx.gluon.data.vision.transforms.CenterCrop((416,416)),
#        transforms.ToTensor(),
    #    mx.gluon.data.vision.transforms.Normalize(mean=0.13, std=0.31)
    #])

    #image_HWC = array.transform(transform_fn)
    #image_CHW = image_HWC.transpose((2,0,1))
    #image_NCHW = image_CHW.expand_dims(axis=0)



    # return  Tranform Image into mxnext array and NCHW format, Un-norma
    # This preset does everything you need.
    #       gcv.data.transforms.presets.imagenet.transform_eval(array)
    #image = gcv.data.transforms.presets.yolo.YOLO3DefaultValTransform(a
```

```
    # don't need to do any of this shit above, it will do all that for u
    image = gcv.data.transforms.presets.yolo.transform_test(array,short=

    #print("image transformed is {}" . format(image.shape))

    return image
```

In [11]:
```
assert test_output.shape[2] == 3  # RGB
foo, goo = transform_image(test_output)
foo[0].shape
```

Out[11]: (3, 416, 625)

In [12]:
```
norm_image, unnorm_image = transform_image(test_output)
assert norm_image.shape == (1, 3, 416, 625)
assert norm_image.dtype == np.float32
assert unnorm_image.shape == (416, 625, 3)
assert unnorm_image.dtype == np.uint8
```

In [13]:
```
plt.imshow(unnorm_image)
fig = plt.gcf()
fig.set_size_inches(14, 14)
plt.show()
```

4) Using a model

Your next task is to pass a transformed image through the network to obtain bounding box and class predictions from the network.

We'll ignore the requirement of creating just a people detector for now.

Hint: Don't forget that you're typically working with a batch of images, even when you only have one image.

In [14]:
```
def detect(network, data):
    """
    Should return the bounding boxes and class predictions from a given

    :param network: pre-trained object detection model
    :type network: mx.gluon.Block
    :param data: batch of transformed images of shape (1, 3, 416, 416)
    :type data: mx.nd.NDArray

    :return: tuple of class_ids, scores, bounding_boxes
    :rtype: tuple of mx.nd.NDArrays
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    class_ids, scores, bounding_boxes = network(data)

    return class_ids, scores, bounding_boxes
```

In [15]:
```
class_ids, scores, bounding_boxes = detect(network, norm_image)
```

In [16]:
```
print("class_ids.share {}".format(class_ids.shape))
```

```
class_ids.share (1, 100, 1)
```

In [17]:
```
class_ids, scores, bounding_boxes = detect(network, norm_image)
assert class_ids.shape == (1, 100, 1)
assert scores.shape == (1, 100, 1)
assert bounding_boxes.shape == (1, 100, 4)
```

In [18]:
```
ax = utils.viz.plot_bbox(unnorm_image, bounding_boxes[0], scores[0], cla
fig = plt.gcf()
fig.set_size_inches(14, 14)
plt.show()
```

We can see that the bounding boxes are placed around all the people in the image, including the goal keeper in the distance too.

Also spot that each detected object has a corresponding class label and a confidence score.

5) Counting Objects

In this section you should use the network outputs to determine the number of people in the

image.

You'll notice that the `count_object` function has a `threshold` parameter that can control the required confidence level. When the confidence exceeds this amount for a detected person, you should add one to the total person count.

Hint: it's possible to use MXNet NDArray functions to do this calculations without loops.

Hint: All 3 network outputs are avaliable in the function (i.e. `class_ids` , `scores` , `bounding_boxes`) but only two of them are required for this task.

Hint: Object labels can be found on the network for pre-trained object detection models.

Hint: Your function should return a Python float so conversion from MXNet NDArray to Python float is required.

In [19]:
```
def count_object(network, class_ids, scores, bounding_boxes, object_labe
    """
    Counts objects of a given type that are predicted by the network.

    :param network: object detection network
    :type network: mx.gluon.nn.Block
    :param class_ids: predicted object class indexes (e.g. 123)
    :type class_ids: mx.nd.NDArrays
    :param scores: predicted object confidence
    :type scores: mx.nd.NDArrays
    :param bounding_boxes: predicted object locations
    :type bounding_boxes: mx.nd.NDArrays
    :param object_label: object to be counted (e.g. "person")
    :type object_label: str
    :param threshold: required confidence for object to be counted
    :type threshold: float

    :return: number of objects that are predicted by the network.
    :rtype: int
    """
    # YOUR CODE HERE
    #raise NotImplementedError()
    high_certainity = mx.nd.logical_and(scores, scores > threshold)
    #print(high_certainity)
    match_id = network.classes.index(object_label)
    match = class_ids == match_id
    #print(match)
    #print(mx.nd.logical_and(high_certainity, match))
    objs_detected_which_match_lablel_with_high_certainity = mx.nd.logica
    #print(objs_detected_which_match_lablel_with_high_certainity)
    num_objs_detected_which_match_lablel_with_high_certainity = objs_de
    #print(num_objs_detected_which_match_lablel_with_high_certainity)


    return num_objs_detected_which_match_lablel_with_high_certainity
```

In [20]:
```
for object_label in ["person", "sports ball"]:
    count = count_object(network, class_ids, scores, bounding_boxes, obj
    print("{} objects of class '{}' detected".format(count, object_labe
```

```
6.0 objects of class 'person' detected
1.0 objects of class 'sports ball' detected
```

In [21]:
```
num_people = count_object(network, class_ids, scores, bounding_boxes, "p
assert num_people == 6
```

In [22]:
```
thresholds = [0, 0.5, 0.75, 0.9, 0.99, 0.999]
for threshold in thresholds:
    num_people = count_object(network, class_ids, scores, bounding_boxes
    print("{} people detected using a threshold of {}.".format(num_peop
```

```
6.0 people detected using a threshold of 0.
6.0 people detected using a threshold of 0.5.
6.0 people detected using a threshold of 0.75.
5.0 people detected using a threshold of 0.9.
3.0 people detected using a threshold of 0.99.
0.0 people detected using a threshold of 0.999.
```

You should experiment with different values for the threshold across more than a single image, but for now we'll use a threshold of 0.5.

6) Count People

In this section, you should stack all of the previously defined functions together and complete the implementation of the `count` method to return the total number of people in an image. Use the `_network` and `_threshold` from `__init__`.

In [23]:
```
class PersonCounter():
    def __init__(self, threshold):
        self._network = gcv.model_zoo.get_model(selected_model, pretrai
        self._threshold = threshold

    def set_threshold(self, threshold):
        self._threshold = threshold

    def count(self, filepath, visualize=False):
        # YOUR CODE HERE
        #raise NotImplementedError()
        test_output = load_image(filepath)
        norm_image, unnorm_image = transform_image(test_output)
        class_ids, scores, bounding_boxes = detect(self._network, norm_
        if visualize:
            self._visualize(unnorm_image, class_ids, scores, bounding_b
        # YOUR CODE HERE
        #raise NotImplementedError()
        num_people = count_object(self._network, class_ids, scores, bou
        if num_people == 1:
            print('{} person detected in {}.'.format(num_people, filepa
        else:
            print('{} people detected in {}.'.format(num_people, filepa
        return num_people

    def _visualize(self, unnorm_image, class_ids, scores, bounding_boxe
        """
        Since the transformed_image is in NCHW layout and the values ar
        this method slices and transposes to give CHW as required by ma
        and scales (-2, +2) to (0, 255) linearly.
        """
        ax = utils.viz.plot_bbox(unnorm_image,
                                 bounding_boxes[0],
                                 scores[0],
                                 class_ids[0],
                                 class_names=self._network.classes)
        fig = plt.gcf()
        fig.set_size_inches(14, 14)
        plt.show()
```

In [24]:
```
#counter = PersonCounter(threshold=0.9)
#counter.count(Path(M6_IMAGES, '31928213423_090ec29bcf_b.jpg'), visuali
#counter.count(Path(M6_IMAGES, '32701657536_8a0d9e157f_b.jpg'), visuali
#counter.count(Path(M6_IMAGES, '25751294956_fa3ee87fb8_b.jpg'), visuali
#counter.set_threshold(0.5)
#counter.count(Path(M6_IMAGES, '31928213423_090ec29bcf_b.jpg'), visuali
#counter.count(Path(M6_IMAGES, '32701657536_8a0d9e157f_b.jpg'), visuali
#counter.count(Path(M6_IMAGES, '25751294956_fa3ee87fb8_b.jpg'), visuali
```

In [25]:
```
counter = PersonCounter(threshold=0.9)
counter.count(Path(M6_IMAGES, '31928213423_090ec29bcf_b.jpg'), visualiz
```

```
2.0 people detected in ../../data/module_6/images/31928213423_090ec29
bcf_b.jpg.
```

Out[25]: True

In [26]:
```
counter = PersonCounter(threshold=0.9)
counter.count(Path(M6_IMAGES, '32701657536_8a0d9e157f_b.jpg'), visualiz
```

```
3.0 people detected in ../../data/module_6/images/32701657536_8a0d9e15
7f_b.jpg.
```

Out[26]: True

In [27]:
```
counter = PersonCounter(threshold=0.9)
counter.count(Path(M6_IMAGES, '25751294956_fa3ee87fb8_b.jpg'), visualiz
```

```
4.0 people detected in ../../data/module_6/images/25751294956_fa3ee87f
b8_b.jpg.
```

Out[27]: False

In [28]:
```
counter = PersonCounter(threshold=0.9)
assert counter.count(Path(M6_IMAGES, '31928213423_090ec29bcf_b.jpg')) ==
assert counter.count(Path(M6_IMAGES, '32701657536_8a0d9e157f_b.jpg')) ==
assert counter.count(Path(M6_IMAGES, '25751294956_fa3ee87fb8_b.jpg')) ==
counter.set_threshold(0.5)
assert counter.count(Path(M6_IMAGES, '31928213423_090ec29bcf_b.jpg'), v
assert counter.count(Path(M6_IMAGES, '32701657536_8a0d9e157f_b.jpg'), v
assert counter.count(Path(M6_IMAGES, '25751294956_fa3ee87fb8_b.jpg'), v
```

```
2.0 people detected in ../../data/module_6/images/31928213423_090ec29
bcf_b.jpg.
3.0 people detected in ../../data/module_6/images/32701657536_8a0d9e1
57f_b.jpg.
4.0 people detected in ../../data/module_6/images/25751294956_fa3ee87
fb8_b.jpg.
```

Although these results look good, there are still a few failure cases.

`18611133536_534285f26d_b.jpg` misses quite a few people and
`3354172257_a48ba3d1d8_b.jpg` two people become one.

See if you can find more failure cases.

In [29]:
```
counter.count(Path(M6_IMAGES, '18611133536_534285f26d_b.jpg'), visualiz
```

```
17.0 people detected in ../../data/module_6/images/18611133536_534285f
26d_b.jpg.
```

Out[29]: 17.0

7) Count all people in a collection of images

With our person counter implemented, let's run through the whole dataset and count the number

of people in our image directory.

In [*]:
```
total_count = 0
for filepath in M6_IMAGES.glob('**/*.jpg'):
    total_count += counter.count(filepath)
print("### Summary: {} people detected.".format(total_count))
```

```
0.0 people detected in ../../data/module_6/images/26795765640_2aac096e
05_b.jpg.
4.0 people detected in ../../data/module_6/images/3263304224_621f36f98
b.jpg.
4.0 people detected in ../../data/module_6/images/31928213423_090ec29b
cf_b.jpg.
1.0 person detected in ../../data/module_6/images/29756013612_0018b057
d8_b.jpg.
1.0 person detected in ../../data/module_6/images/29869572955_0b59d43c
55_z.jpg.
17.0 people detected in ../../data/module_6/images/18611133536_534285f
26d_b.jpg.
0.0 people detected in ../../data/module_6/images/14920793105_74c19b98
1f_z.jpg.
3.0 people detected in ../../data/module_6/images/32701657536_8a0d9e15
7f_b.jpg.
4.0 people detected in ../../data/module_6/images/3354172257_a48ba3d1d
8_b.jpg.
3.0 people detected in ../../data/module_6/images/47764429062_8c19a19c
d1_b.jpg.
1.0 person detected in ../../data/module_6/images/36949889492_c946a8bd
dc_z.jpg.
1.0 person detected in ../../data/module_6/images/14986165665_00d6b5ea
1b_z.jpg.
2.0 people detected in ../../data/module_6/images/2455110397_757dfec32
4_z.jpg.
3.0 people detected in ../../data/module_6/images/9480259864_3d7d610e1
1_b.jpg.
1.0 person detected in ../../data/module_6/images/29243394393_889afb2f
c7_b.jpg.
1.0 person detected in ../../data/module_6/images/34076077526_52424445
c7_b.jpg.
2.0 people detected in ../../data/module_6/images/31899210544_c8303dee
b8_b.jpg.
1.0 person detected in ../../data/module_6/images/29575993250_f5b6d20b
7e_b.jpg.
6.0 people detected in ../../data/module_6/images/32742378405_3ecc8cc9
58_b.jpg.
```

Summary & Congratulations!

In this final assignment of the course, you used a pre-trained object detection model to count people in a dataset of images.

Although we are not eliminating false positives, the model was able to identify over 70 people in the dataset.

You've now completed Computer Vision with GluonCV from AWS. We hope to see you in the next course.

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```