module_3_assignment

April 12, 2020

1 Module 3 Assignment

Your objective in this assignment is to implement a tennis ball detector using a pre-trained image classification network from GluonCV. We'll step through the pipeline, from loading and transforming an input image, to loading and using a pre-trained model. Since we're only interested in detecting tennis balls, this is a binary classification problem, which is slightly different to the multi-class classification setup we've seen so far.

1.0.1 0) Setup

We start with some initial setup: importing packages and setting the path to the data.

[1]:
```
import mxnet as mx
import gluoncv as gcv
import matplotlib.pyplot as plt
import numpy as np
import os
from pathlib import Path
```

[2]:
```
M3_DATA = Path(os.getenv('DATA_DIR', '../../data'), 'module_3')
M3_IMAGES = Path(M3_DATA, 'images')
M3_MODELS = Path(M3_DATA, 'models')
```

1.0.2 1) Loading an 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).

[3]:
```
def load_image(filepath):
    """
    Should load image from disk.

    :param filepath: relative or absolute filepath to RGB image file in JPG␣
 ↪format.
```

```
    :type filepath: str

    :return: an array with pixel intensities (in HWC layout).
    :rtype: mx.nd.NDArray
    """
    image = mx.image.imread(filepath)
    return image
```

[4]:
```
test_filepath = Path(M3_IMAGES, 'ben-hershey-VEW78A1YZ6I-unsplash.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 NumPy Array.
```

1.0.3 2) Transforming an image

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

Since we're going to use an ImageNet pre-trained network, we need to follow the same steps used for ImageNet pre-training.

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 preset transforms.

[5]:
```
def transform_image(array):
    """
    Should transform image by:

    1) Resizing the shortest dimension to 224. e.g (448, 1792) -> (224, 896).
    2) Cropping to a center square of dimension (224, 224).
    3) Converting the image from HWC layout to CHW layout.
    4) Normalizing the image using ImageNet statistics (i.e. per colour channel␣
↪mean and variance).
    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)
    :rtype: mx.nd.NDArray
    """
    image = gcv.data.transforms.presets.imagenet.transform_eval(array,␣
↪resize_short=224, crop_size=224)
    return image
```

[6]:
```
transformed_test_output = transform_image(test_output)
assert transformed_test_output.shape == (1, 3, 224, 224)
assert transformed_test_output.dtype == np.float32
```

1.0.4 3) Loading a model

With the image loaded and transformed, you now need to load a pre-trained classification model.

Choose a MobileNet 1.0 image classification model that's been pre-trained on ImageNet.

CAUTION!: Although the notebook interface has internet connectivity, the **autograders are not permitted to access the internet**. We have already downloaded the correct models and data for you to use so you don't need access to the internet. However, you do need to specify the correct path to the models when loading a model from the Gluon CV Model Zoo using `get_model` or otherwise. Set the `root` parameter to `M3_MODELS`. As an example, you should have something similar to `gcv.model_zoo.get_model(..., root=M3_MODELS)`. Usually, in the real world, you have internet access, so setting the `root` parameter isn't required (and it's set to `~/.mxnet` by default).

```
[7]: def load_pretrained_classification_network():
         """
         Loads a MobileNet 1.0 network that's been pre-trained on ImageNet.

         :return: a pre-trained network
         :rtype: mx.gluon.Block
         """
         network = gcv.model_zoo.get_model('mobilenet1.0', pretrained=True,␣
      ↪root=M3_MODELS)
         return network
```

```
[8]: network = load_pretrained_classification_network()
     assert isinstance(network, mx.gluon.Block), 'Model should be a Gluon Block'
     assert network.name.startswith('mobilenet'), 'Select MobileNet'
     params = network.collect_params(select=network.name + '_conv0_weight')
     assert list(params.items())[0][1].shape[0] == 32, 'Select MobileNet1.0'
```

1.0.5 4) Using a model

Your next task is to pass your transformed image through the network to obtain predicted probabilities for all ImageNet classes.

We'll ignore the requirement of creating just a tennis ball classifier for now.

Hint #1: Don't forget that you're typically working with a batch of images, even when you only have one image.

Hint #2: Remember that the direct outputs of our network aren't probabilities.

```
[9]: def predict_probabilities(network, data):
         """
```

```
    Should return the predicted probabilities of ImageNet classes for the given␣
↪image.

    :param network: pre-trained image classification model
    :type network: mx.gluon.Block
    :param data: batch of transformed images of shape (1, 3, 224, 224)
    :type data: mx.nd.NDArray

    :return: array of probabilities of shape (1000,)
    :rtype: mx.nd.NDArray
    """
    result = mx.nd.softmax(network(data)[0])
    return result
```

[10]:
```
pred_probas = predict_probabilities(network, transformed_test_output)
assert pred_probas.shape == (1000,)
np.testing.assert_almost_equal(pred_probas.sum().asscalar(), 1, decimal=5)
assert pred_probas.dtype == np.float32
```

1.0.6 5) Finding Class Label

Since we're only interested in tennis ball classification for now, we need a method of finding the probability associated with tennis ball out of the 1000 classes.

You should implement a function that returns the index of a given class label (e.g. `admiral` is index 321)

Hint: you're allowed to use variables that are defined globally on this occasion. You should think about which objects that have been previously defined has a list of class labels.

[11]:
```
def find_class_idx(label):
    """
    Should return the class index of a particular label.

    :param label: label of class
    :type label: str

    :return: class index
    :rtype: int
    """

    return network.classes.index(label)
```

[12]:
```
assert find_class_idx('tennis ball') == 852
assert find_class_idx('spiny lobster') == 123
assert find_class_idx('admiral') == 321
```

1.0.7 6) Slice Tennis Ball Class

Using the above function to find the correct index for tennis ball, you should implement a function to slice the calculated probability for tennis ball from the 1000 class probabilities calculated by the network. It should also convert the probability from MXNet `NDArray` to a NumPy `float32`.

We'll use this for our confidence score that the image is a tennis ball.

[13]:
```
def slice_tennis_ball_class(pred_probas):
    """
    Extracts the probability associated with tennis ball.

    :param pred_probas: array of ImageNet probabilities of shape (1000,)
    :type pred_probas: mx.nd.NDArray

    :return: probability of tennis ball
    :rtype: np.float32

    """
    return pred_probas[find_class_idx('tennis ball')].asscalar()
```

[14]:
```
pred_proba_tennis_ball = slice_tennis_ball_class(pred_probas)
assert isinstance(pred_proba_tennis_ball, np.float32)
np.testing.assert_almost_equal(pred_proba_tennis_ball, 0.9987876, decimal=3)
```

1.0.8 7) Classify Tennis Ball Images

We'll finish this assignment by bringing all of the components together and creating a `TennisBallClassifier` to classify images. You should implement the entire classification pipeline inside the `classify` function using the functions defined earlier on in the assignment. You should notice that the pre-trained model is loaded once during initialization, and then it should be used inside the `classify` method.

[19]:
```
class TennisBallClassifier():
    def __init__(self):
        self._network = load_pretrained_classification_network()

    def classify(self, filepath):

        self._image = load_image(filepath)
        self._image = transform_image(self._image)
        self._visualize(self._image)
        prediction = predict_probabilities(self._network, self._image)
        prediction = slice_tennis_ball_class(prediction)

        print('{0:.2%} confidence that image is a tennis ball.'.
 ↪format(prediction))
```

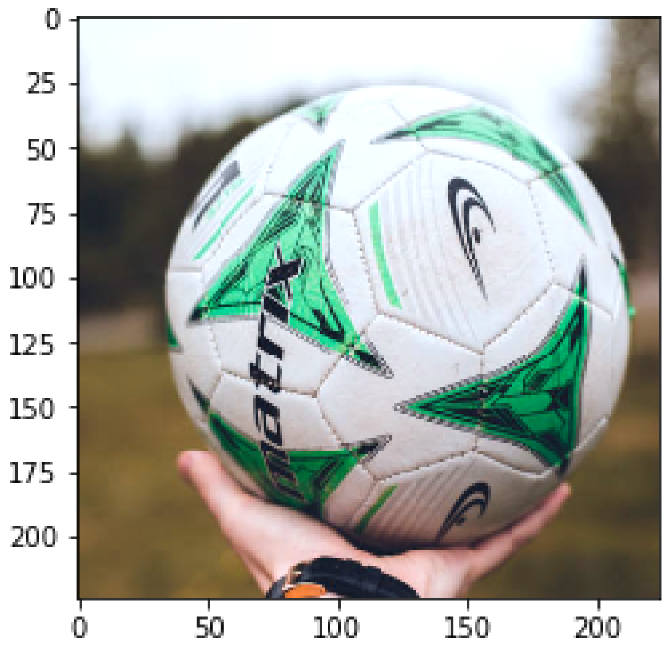
```
        return prediction

    def _visualize(self, transformed_image):
        """
        Since the transformed_image is in NCHW layout and the values are␣
↪normalized,
        this method slices and transposes to give CHW as required by matplotlib,
        and scales (-2, +2) to (0, 255) linearly.
        """
        chw_image = transformed_image[0].transpose((1,2,0))
        chw_image = ((chw_image * 64) + 128).clip(0, 255).astype('uint8')
        plt.imshow(chw_image.asnumpy())
```

[20]:
```
classifier = TennisBallClassifier()
```

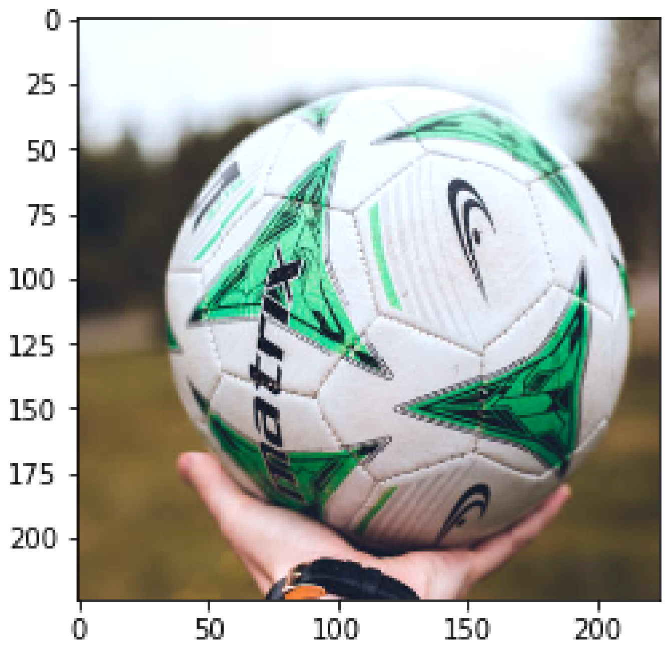
[21]:
```
filepath = Path(M3_IMAGES, 'erik-mclean-D23_XPbsx-8-unsplash.jpg')
pred_proba = classifier.classify(filepath)
```

```
0.00% confidence that image is a tennis ball.
```



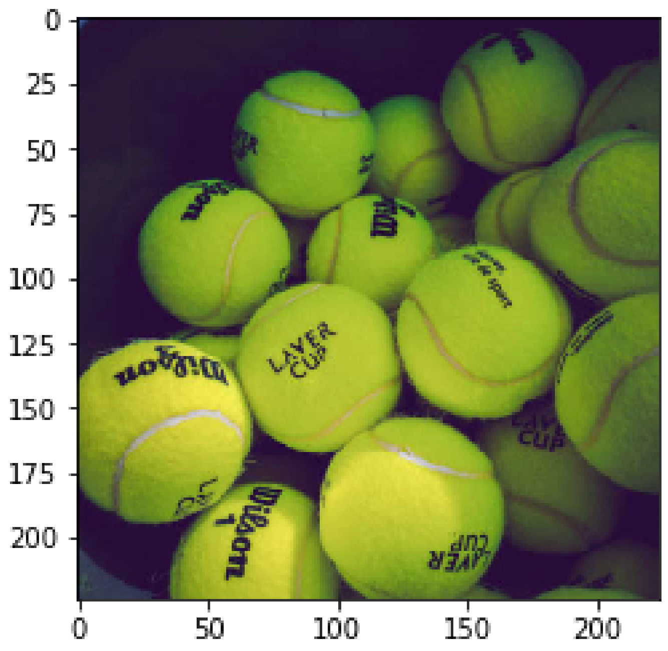
[22]:
```
filepath = Path(M3_IMAGES, 'erik-mclean-D23_XPbsx-8-unsplash.jpg')
pred_proba = classifier.classify(filepath)
np.testing.assert_almost_equal(pred_proba, 2.0355723e-05, decimal=3)
```

```
0.00% confidence that image is a tennis ball.
```



```
[23]: filepath = Path(M3_IMAGES, 'marvin-ronsdorf-CA998Anw2Lg-unsplash.jpg')
      pred_proba = classifier.classify(filepath)
      np.testing.assert_almost_equal(pred_proba, 0.9988895, decimal=3)
```

```
99.89% confidence that image is a tennis ball.
```



```
[ ]:
```