Data Augmentation For Object Detection

This notebook serves as general manual to using this codebase. We cover all the major augmentations, as well as ways to combine them.

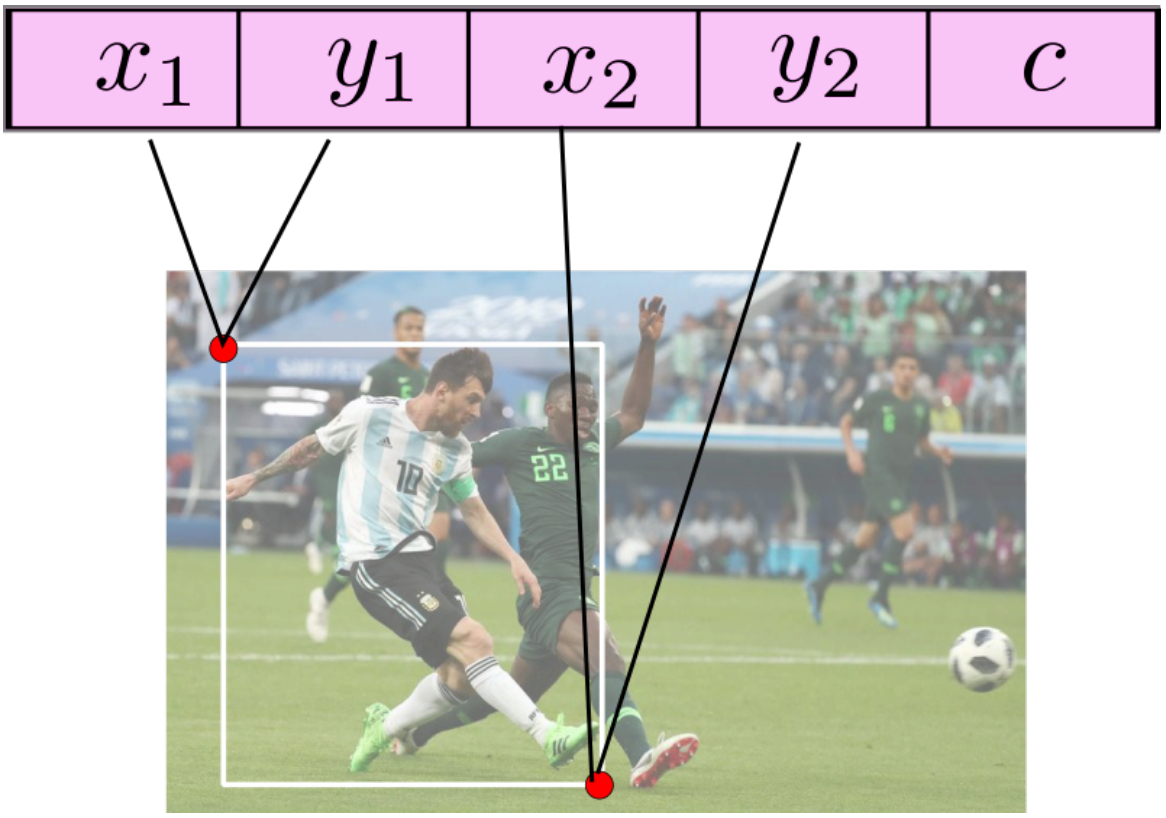
In [ ]:
```
from data_aug.data_aug import *
from data_aug.bbox_util import *
import numpy as np
import cv2
import matplotlib.pyplot as plt
import pickle as pkl
%matplotlib inline
```

Storage Format

First things first, we define how the storage formats required for images to work.

1. **The Image**: A OpenCV numpy array, of shape *(H x W x C)*.
2. **Annotations**: A numpy array of shape *N x 5* where *N* is the number of objects, one represented by each row. 5 columns represent the top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-right y-coordinate, and the class of the object.

Here is an image to aid your imagination.



Whatever format your annotations are present in make sure, you convert them to this format.

For demonstration purposes, we will be using the image above to show the transformations.

The image as well as it's annotation has been provided. The annotation is a numpy array in a

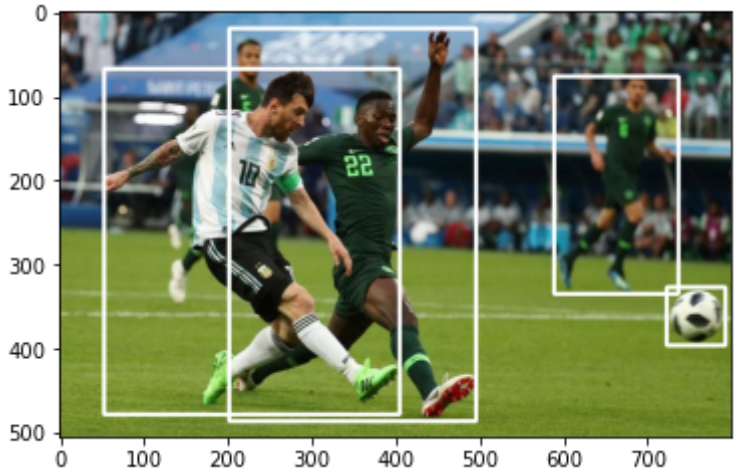
In [2]:
```
img = cv2.imread("messi.jpg")[:,:,::-1]   #opencv loads images in bgr. the
bboxes = pkl.load(open("messi_ann.pkl", "rb"))

#inspect the bounding boxes
print(bboxes)
```

```
[[ 53.          68.0000175 405.        478.9998225   0.        ]
 [202.          20.99992   496.        486.99978     0.        ]
 [589.          77.0001275 737.        335.9999825   0.        ]
 [723.         327.000125  793.        396.000295    1.        ]]
```

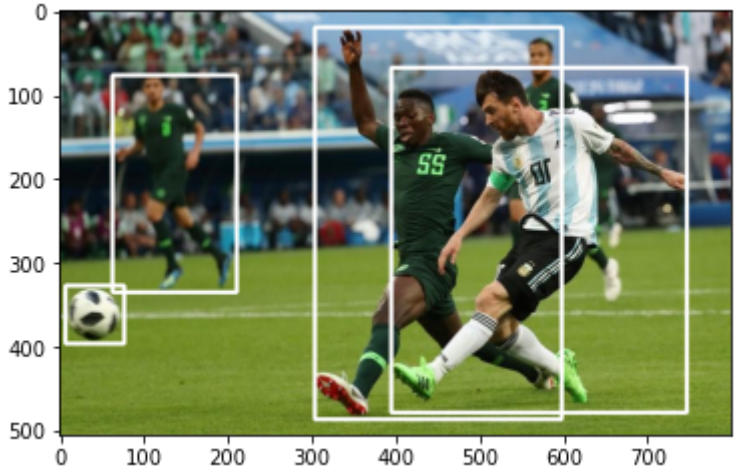
You can use the function `draw_rect` to plot the bounding boxes on an image.

In [3]:
```
plotted_img = draw_rect(img, bboxes)
plt.imshow(plotted_img)
plt.show()
```



Now, we can get started with our image augmentations. The first one is **Horizontal Flipping**. The function takes one arguement, *p* which is the probability that the image will be flipped.

In [5]:
```
img_, bboxes_ = RandomHorizontalFlip(1)(img.copy(), bboxes.copy())
plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```

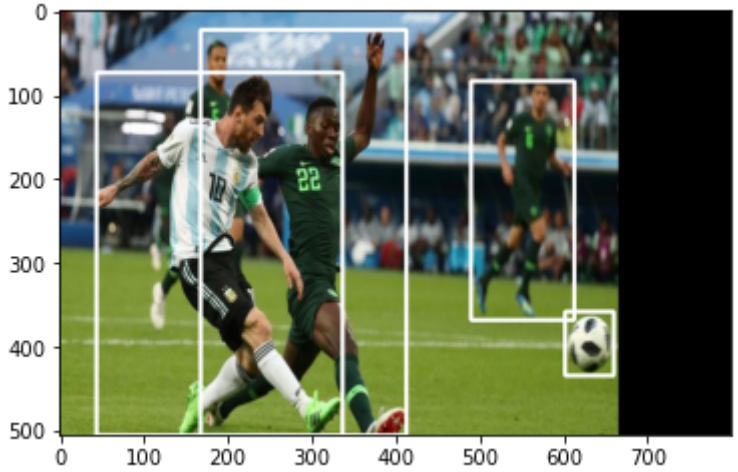


Scaling. Scales the image. If the argument *diff* is True, then the image is scaled with different values in the vertical and the horizontal directions, i.e. aspect ratio is not maintained.

If the first argument is a float, then the scaling factors for both x and y directions are

randomly sampled from *(- arg, arg)*. Otherwise, you can specify a tuple for this range.

In [6]:
```
img_, bboxes_ = RandomScale(0.3, diff = True)(img.copy(), bboxes.copy())
plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```



Translation. Translates the image. If the argument *diff* is True, then the image is translated with different values in the vertical and the horizontal directions.

If the first argument is a float, then the translating factors for both x and y directions are randomly sampled from *(- arg, arg)*. Otherwise, you can specify a tuple for this range.

In [7]:
```
img_, bboxes_ = RandomTranslate(0.3, diff = True)(img.copy(), bboxes.copy()
plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```

Rotation. Rotates the image.

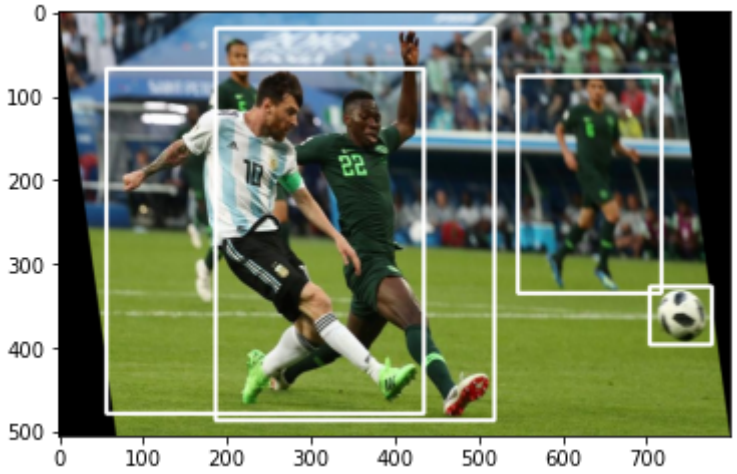
If the first argument is a int, then the rotating angle, in degrees, is sampled from *(- arg, arg)*. Otherwise, you can specify a tuple for this range.

In [8]:
```
img_, bboxes_ = RandomRotate(20)(img.copy(), bboxes.copy())
plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```

Shearing. Sheares the image horizontally

If the first argument is a float, then the shearing factor is sampled from *(- arg, arg)*. Otherwise, you can specify a tuple for this range.

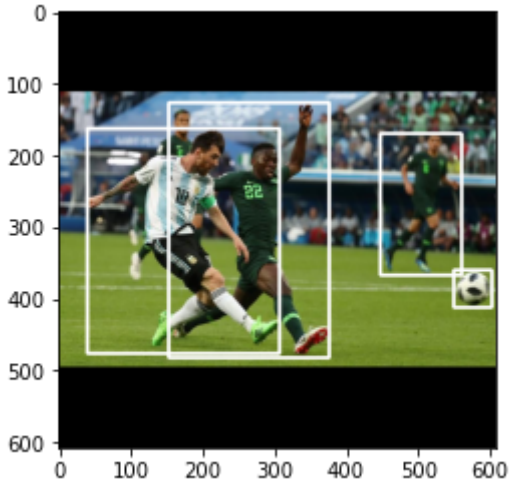
In [9]:
```
img_, bboxes_ = RandomShear(0.2)(img.copy(), bboxes.copy())
plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```



Resizing. Resizes the image to square dimensions while keeping the aspect ratio constant.

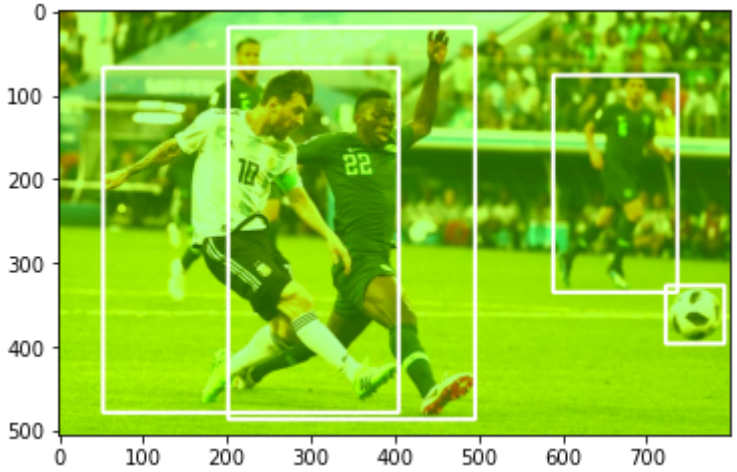
The argument to this augmentation is the side of the square.

In [10]:
```
img_, bboxes_ = Resize(608)(img.copy(), bboxes.copy())
plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```



HSV transforms are supported as well.

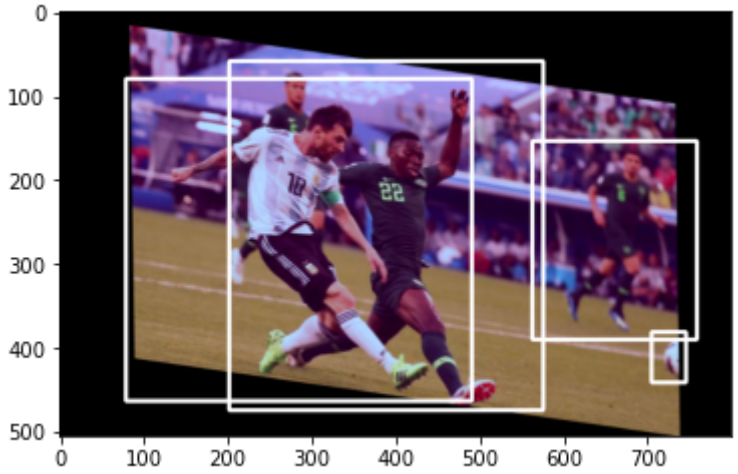
In [11]:
```
img_, bboxes_ = RandomHSV(100, 100, 100)(img.copy(), bboxes.copy())
plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```



You can combine multiple transforms together by using the Sequence class as follows.

In [12]:
```
seq = Sequence([RandomHSV(40, 40, 30),RandomHorizontalFlip(), RandomScale()])
img_, bboxes_ = seq(img.copy(), bboxes.copy())

plotted_img = draw_rect(img_, bboxes_)
plt.imshow(plotted_img)
plt.show()
```



A list of all possible transforms can be found in the `docs` folder.

In [ ]: