In [1]:
```
import tensorflow as tf
import json
import numpy as np
from matplotlib import pyplot as plt
```

In [2]:
```
images = tf.data.Dataset.list_files('D:\\data\\images\\*.jpg')
```

In [3]:
```
images.as_numpy_iterator().next()
```

Out[3]:
```
b'D:\\data\\images\\30.jpg'
```

In [4]:
```
def load_image(x):
    byte_img = tf.io.read_file(x)
    img = tf.io.decode_jpeg(byte_img)
    return img
```

In [5]:
```
images = images.map(load_image)
```

In [6]:
```
images.as_numpy_iterator().next()
```

Out[6]:
```
array([[[194,  44,  55],
        [191,  41,  52],
        [187,  35,  47],
        ...,
        [ 19,   1,   0],
        [ 18,   3,   0],
        [ 19,   4,   1]],

       [[191,  41,  52],
        [188,  38,  49],
        [185,  33,  45],
        ...,
        [ 19,   1,   0],
        [ 18,   3,   0],
        [ 19,   4,   1]],

       [[186,  36,  47],
        [184,  34,  45],
        [183,  31,  44],
        ...,
        [ 20,   2,   0],
        [ 18,   3,   0],
        [ 19,   4,   1]],

       ...,

       [[ 26,  46,  71],
        [ 25,  45,  70],
        [ 24,  44,  69],
        ...,
        [130, 130, 122],
        [128, 128, 120],
        [123, 123, 115]],

       [[ 26,  46,  71],
        [ 25,  45,  70],
        [ 24,  44,  69],
        ...,
        [129, 129, 121],
        [130, 130, 122],
        [126, 126, 118]],

       [[ 26,  46,  71],
        [ 25,  45,  70],
        [ 24,  44,  69],
        ...,
        [127, 127, 119],
        [129, 129, 121],
        [127, 127, 119]]], dtype=uint8)
```

In [7]:
```
type(images)
```

Out[7]:
```
tensorflow.python.data.ops.map_op._MapDataset
```

In [8]:

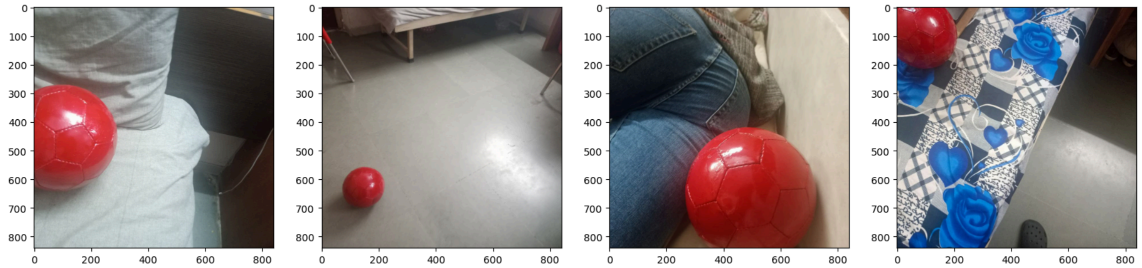
```
image_generator = images.batch(4).as_numpy_iterator()
```

In [9]:

```
plot_images = image_generator.next()
```

In [10]:

```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, image in enumerate(plot_images):
    ax[idx].imshow(image)
plt.show()
```



In [11]:

```
import os
for folder in ['train','test','val']:
    for file in os.listdir(os.path.join('D:\\data', folder, 'images')):

        filename = file.split('.')[0]+'.json'
        existing_filepath = os.path.join('D:\\data','labels', filename)
        if os.path.exists(existing_filepath):
            new_filepath = os.path.join('D:\\data',folder,'labels',filename
            os.replace(existing_filepath, new_filepath)
```

In [ ]:

In [12]:

```
import albumentations as alb
```

```
C:\Users\sanke\anaconda3\Lib\site-packages\paramiko\transport.py:219: Cryp
tographyDeprecationWarning: Blowfish has been deprecated
  "class": algorithms.Blowfish,
```

In [13]:

```
augmentor = alb.Compose([alb.RandomCrop(width=800, height=800),
                         alb.HorizontalFlip(p=0.5),
                         alb.RandomBrightnessContrast(p=0.2),
                         alb.RandomGamma(p=0.2),
                         alb.RGBShift(p=0.2),
                         alb.VerticalFlip(p=0.5)],
                       bbox_params=alb.BboxParams(format='albumentations',
                                                  label_fields=['class_labe
```

In [14]:

```
import cv2
img = cv2.imread(os.path.join('D:\\data','train', 'images','11.jpg'))
```

In [15]:
```
print(img)
```

```
[[[156 161 152]
  [158 163 154]
  [160 165 156]
  ...
  [162 159 155]
  [162 159 155]
  [163 160 156]]

 [[156 161 152]
  [157 162 153]
  [159 164 155]
  ...
  [162 159 155]
  [163 160 156]
  [163 160 156]]

 [[155 160 151]
  [156 161 152]
  [158 163 154]
  ...
  [162 159 155]
  [162 159 155]
  [163 160 156]]

 ...

 [[179 178 168]
  [182 181 171]
  [187 186 176]
  ...
  [ 15  22  39]
  [ 15  22  39]
  [ 15  22  39]]

 [[182 181 171]
  [184 183 173]
  [187 186 176]
  ...
  [ 15  22  39]
  [ 15  22  39]
  [ 15  22  39]]

 [[183 182 172]
  [183 182 172]
  [184 183 173]
  ...
  [ 15  22  39]
  [ 15  22  39]
  [ 15  22  39]]]
```

In [16]:
```
with open(os.path.join('D:\\data', 'train', 'labels', '11.json'), 'r') as f
    label = json.load(f)
```

In [17]:
```
label
```

Out[17]:
```
{'version': '5.4.1',
 'flags': {},
 'shapes': [{'label': 'BALL',
   'points': [[3.181818181818187, 270.0],
    [295.9090909090909, 644.5454545454545]],
   'group_id': None,
   'description': '',
   'shape_type': 'rectangle',
   'flags': {},
   'mask': None}],
 'imagePath': '..\\images\\11.jpg',
 'imageData': '/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQN
DAsLDBkSEw8UHRofHh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2wBDAQg
JCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMj
IyMjIyMjIyMjL/wAARCANIA0gDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECA
wQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEI
I0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZ
naGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8
jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAA
```

In [18]:
```
type(label)
```

Out[18]:
```
dict
```

In [19]:
```
label['shapes'][0]['points']
```

Out[19]:
```
[[3.181818181818187, 270.0], [295.9090909090909, 644.5454545454545]]
```

In [20]:
```
type(label['shapes'])
```

Out[20]:
```
list
```

In [21]:
```
label['shapes']
```

Out[21]:
```
[{'label': 'BALL',
  'points': [[3.181818181818187, 270.0],
   [295.9090909090909, 644.5454545454545]],
  'group_id': None,
  'description': '',
  'shape_type': 'rectangle',
  'flags': {},
  'mask': None}]
```

In [22]:
```
label['shapes'][0]
```

Out[22]:
```
{'label': 'BALL',
 'points': [[3.181818181818187, 270.0],
  [295.9090909090909, 644.5454545454545]],
 'group_id': None,
 'description': '',
 'shape_type': 'rectangle',
 'flags': {},
 'mask': None}
```

In [23]:
```
label['shapes'][0]['label']
```

Out[23]:
```
'BALL'
```

In [24]:
```
label['shapes'][0]['shape_type']
```

Out[24]:
```
'rectangle'
```

In [25]:
```
label['shapes'][0]['points']
```

Out[25]:
```
[[3.181818181818187, 270.0], [295.9090909090909, 644.5454545454545]]
```

In [26]:
```
coords = [0,0,0,0]
coords[0] = label['shapes'][0]['points'][0][0]
coords[1] = label['shapes'][0]['points'][0][1]
coords[2] = label['shapes'][0]['points'][1][0]
coords[3] = label['shapes'][0]['points'][1][1]
```

In [27]:
```
coords
```

Out[27]:
```
[3.181818181818187, 270.0, 295.9090909090909, 644.5454545454545]
```

In [28]:
```
coords = list(np.divide(coords,[840,840,840,840]))
```

In [29]:
```
coords
```

Out[29]:
```
[0.003787878787878794,
 0.32142857142857145,
 0.35227272727272724,
 0.7673160173160173]
```

In [30]:
```
augmented = augmentor(image=img, bboxes=[coords], class_labels=['BALL'])
```

In [31]:
```
augmented
```

Out[31]:
```
{'image': array([[[100, 100, 100],
         [100, 100, 100],
         [100,  99, 101],
         ...,
         [ 79,  86,  79],
         [ 79,  86,  79],
         [ 79,  86,  79]],

        [[104, 104, 104],
         [102, 104, 105],
         [102, 104, 105],
         ...,
         [ 77,  84,  77],
         [ 77,  84,  77],
         [ 77,  84,  77]],

        [[103, 105, 105],
         [103, 107, 108],
         [106, 110, 111],
         ...,
         [ 75,  82,  75],
         [ 76,  83,  76],
         [ 76,  83,  76]],

        ...,

        [[  0,   0,   2],
         [  0,   0,   1],
         [  0,   0,   1],
         ...,
         [152, 151, 141],
         [151, 150, 140],
         [151, 150, 140]],

        [[  0,   0,   2],
         [  0,   0,   1],
         [  0,   0,   1],
         ...,
         [152, 151, 141],
         [152, 151, 141],
         [151, 150, 140]],

        [[  0,   0,   2],
         [  0,   0,   2],
         [  0,   0,   1],
         ...,
         [151, 150, 140],
         [150, 149, 139],
         [150, 149, 139]]], dtype=uint8),
 'bboxes': [(0.6601136363636364, 0.31, 1.0, 0.7781818181818181)],
 'class_labels': ['BALL']}
```

In [32]:
```
augmented.keys()
```

Out[32]:
```
dict_keys(['image', 'bboxes', 'class_labels'])
```

In [33]:
```
augmented['image'].shape
```

Out[33]:
```
(800, 800, 3)
```

In [34]:
```
augmented['bboxes']
```

Out[34]:
```
[(0.6601136363636364, 0.31, 1.0, 0.7781818181818181)]
```

In [35]:
```
augmented['bboxes'][0][:2]
```

Out[35]:
```
(0.6601136363636364, 0.31)
```

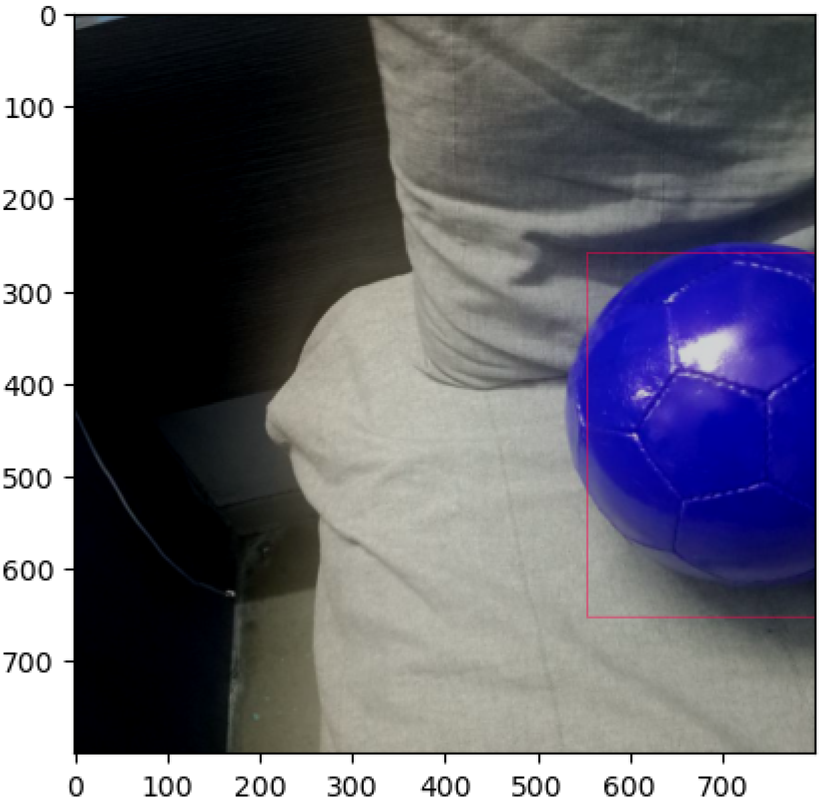
In [36]:
```
augmented['bboxes'][0][2:]
```

Out[36]:
```
(1.0, 0.7781818181818181)
```

In [37]:
```
cv2.rectangle(augmented['image'],
              tuple(np.multiply(augmented['bboxes'][0][:2], [840,840]).asty
              tuple(np.multiply(augmented['bboxes'][0][2:], [840,840]).asty
                    (255,0,77), 1)

plt.imshow(augmented['image'])
```

Out[37]:
```
<matplotlib.image.AxesImage at 0x227af97ddd0>
```



In [38]:

```
# for partition in ['train','test','val']:
#     for image in os.listdir(os.path.join('D:\\data', partition, 'images')
#         img = cv2.imread(os.path.join('D:\\data', partition, 'images', im

#         coords = [0,0,0.00001,0.00001]
#         label_path = os.path.join('D:\\data', partition, 'labels', f'{ima
#         if os.path.exists(label_path):
#             with open(label_path, 'r') as f:
#                 label = json.load(f)

#             coords[0] = label['shapes'][0]['points'][0][0]
#             coords[1] = label['shapes'][0]['points'][0][1]
#             coords[2] = label['shapes'][0]['points'][1][0]
#             coords[3] = label['shapes'][0]['points'][1][1]
#             coords = list(np.divide(coords, [840,840,840,840]))

#         try:
#             for x in range(60):
#                 augmented = augmentor(image=img, bboxes=[coords], class_l
#                 cv2.imwrite(os.path.join('D:\\agm_data', partition, 'imag

#                 annotation = {}
#                 annotation['image'] = image


#                 if os.path.exists(label_path):
#                     if len(augmented['bboxes']) == 0:
#                         annotation['bbox'] = [0,0,0,0]
#                         annotation['class'] = 0
#                     else:
#                         annotation['bbox'] = augmented['bboxes'][0]
#                         annotation['class'] = 1
#                 else:
#                     annotation['bbox'] = [0,0,0,0]
#                     annotation['class'] = 0


#                 with open(os.path.join('D:\\agm_data', partition, 'labels
#                     json.dump(annotation, f)

#         except Exception as e:
#             print(e)
```

In [39]:

```
train_images = tf.data.Dataset.list_files('D:\\agm_data\\train\\images\\*.j
train_images = train_images.map(load_image)
train_images = train_images.map(lambda x: tf.image.resize(x, (120,120)))
train_images = train_images.map(lambda x: x/255)
```

In [40]:

```
test_images = tf.data.Dataset.list_files('D:\\agm_data\\test\\images\\*.jpg
test_images = test_images.map(load_image)
test_images = test_images.map(lambda x: tf.image.resize(x, (120,120)))
test_images = test_images.map(lambda x: x/255)
```

In [41]:

```
val_images = tf.data.Dataset.list_files('D:\\agm_data\\val\\images\\*.jpg',
val_images = val_images.map(load_image)
val_images = val_images.map(lambda x: tf.image.resize(x, (120,120)))
val_images = val_images.map(lambda x: x/255)
```

In [42]:

```
train_images.as_numpy_iterator().next()
```

Out[42]:

```
array([[[0.3691721 , 0.33387798, 0.33779955],
        [0.3745098 , 0.3392157 , 0.34313726],
        [0.3647059 , 0.34117648, 0.34117648],
        ...,
        [0.10980392, 0.05490196, 0.05098039],
        [0.11601307, 0.05326797, 0.05718954],
        [0.11764706, 0.05490196, 0.05882353]],

       [[0.38104573, 0.3477124 , 0.3457516 ],
        [0.3882353 , 0.35588235, 0.3509804 ],
        [0.37254903, 0.34901962, 0.3529412 ],
        ...,
        [0.10653603, 0.05163407, 0.04967328],
        [0.11568628, 0.05294118, 0.05686275],
        [0.11960784, 0.05686275, 0.06078431]],

       [[0.40784314, 0.36862746, 0.36078432],
        [0.39607844, 0.36078432, 0.3509804 ],
        [0.38104573, 0.34967318, 0.34183004],
        ...,
        [0.11318076, 0.0582788 , 0.0582788 ],
        [0.11176471, 0.05686275, 0.05686275],
        [0.11361659, 0.05871462, 0.05871462]],

       ...,

       [[0.46666676, 0.4862746 , 0.4627452 ],
        [0.46535957, 0.4849674 , 0.461438  ],
        [0.48235285, 0.5019607 , 0.48627442],
        ...,
        [0.38104582, 0.3457517 , 0.3104576 ],
        [0.36764717, 0.33235306, 0.29705894],
        [0.3764706 , 0.3529412 , 0.30588236]],

       [[0.4640523 , 0.48366013, 0.46013072],
        [0.46078432, 0.48039216, 0.45686275],
        [0.46503267, 0.4846405 , 0.46895424],
        ...,
        [0.39281052, 0.3575164 , 0.3222223 ],
        [0.39313725, 0.35784313, 0.32254902],
        [0.38267967, 0.34738556, 0.3120914 ]],

       [[0.43812662, 0.45773447, 0.43420506],
        [0.44509804, 0.46470588, 0.44117647],
        [0.45599118, 0.47559902, 0.45991275],
        ...,
        [0.40838766, 0.37309355, 0.33779943],
        [0.40032673, 0.3650326 , 0.32973847],
        [0.403377  , 0.36808288, 0.33278877]]], dtype=float32)
```

In [43]:

```
def load_labels(label_path):
    with open(label_path.numpy(), 'r', encoding = "utf-8") as f:
        label = json.load(f)

    return [label['class']], label['bbox']
```

In [44]:

```
train_labels = tf.data.Dataset.list_files('D:\\agm_data\\train\\labels\\*.j
train_labels = train_labels.map(lambda x: tf.py_function(load_labels, [x],
```

In [45]:

```
test_labels = tf.data.Dataset.list_files('D:\\agm_data\\test\\labels\\*.jso
test_labels = test_labels.map(lambda x: tf.py_function(load_labels, [x], [t
```

In [46]:

```
val_labels = tf.data.Dataset.list_files('D:\\agm_data\\val\\labels\\*.json'
val_labels = val_labels.map(lambda x: tf.py_function(load_labels, [x], [tf.
```

In [47]:

```
train_labels.as_numpy_iterator().next()
```

Out[47]:

```
(array([1], dtype=uint8),
 array([0.7383, 0.6904, 0.9233, 0.8755], dtype=float16))
```

In [48]:

```
len(train_images), len(train_labels), len(test_images), len(test_labels), l
```

Out[48]:

```
(1320, 1320, 180, 180, 300, 300)
```

In [49]:

```
train = tf.data.Dataset.zip((train_images, train_labels))
train = train.shuffle(14000)
train = train.batch(8)
train = train.prefetch(4)
```

In [50]:

```
test = tf.data.Dataset.zip((test_images, test_labels))
test = test.shuffle(200)
test = test.batch(8)
test = test.prefetch(4)
```

In [51]:

```
val = tf.data.Dataset.zip((val_images, val_labels))
val = val.shuffle(200)
val = val.batch(8)
val = val.prefetch(4)
```

```
In [52]: train.as_numpy_iterator().next()[0]
```

```
Out[52]: array([[[[0.35250548, 0.40980393, 0.34411764],
                  [0.37973857, 0.4366013 , 0.38333333],
                  [0.35228756, 0.41503266, 0.36405227],
                  ...,
                  [0.8437909 , 0.8398693 , 0.7575164 ],
                  [0.84444445, 0.8405229 , 0.7581699 ],
                  [0.851634  , 0.84771246, 0.7653595 ]],

                 [[0.5480392 , 0.5147059 , 0.36568627],
                  [0.51666665, 0.48039216, 0.34313726],
                  [0.5297386 , 0.4866013 , 0.35326797],
                  ...,
                  [0.845098  , 0.84117645, 0.7588235 ],
                  [0.84705883, 0.84313726, 0.7607843 ],
                  [0.8509804 , 0.84705883, 0.7647059 ]],

                 [[0.5089325 , 0.45795208, 0.32069716],
                  [0.51666665, 0.46568626, 0.32843137],
                  [0.5253812 , 0.48616558, 0.3449891 ],
```

```
In [53]: train.as_numpy_iterator().next()[0].shape
```

```
Out[53]: (8, 120, 120, 3)
```

```
In [54]: train.as_numpy_iterator().next()[1]
```

```
Out[54]: (array([[1],
                 [1],
                 [1],
                 [1],
                 [1],
                 [1],
                 [1],
                 [1]], dtype=uint8),
          array([[0.783  , 0.743  , 1.     , 1.     ],
                 [0.5615 , 0.885  , 0.7285 , 1.     ],
                 [0.1174 , 0.263  , 0.5083 , 0.6562 ],
                 [0.525  , 0.4172 , 0.6694 , 0.564  ],
                 [0.7637 , 0.8027 , 0.9263 , 0.9717 ],
                 [0.11316, 0.661  , 0.7207 , 1.     ],
                 [0.4436 , 0.7085 , 0.7085 , 0.9854 ],
                 [0.     , 0.743  , 0.2191 , 1.     ]], dtype=float16))
```

```
In [55]: data_samples = train.as_numpy_iterator()
```

```
In [56]: res = data_samples.next()
```

```
In [57]: print(cv2.__version__)
```

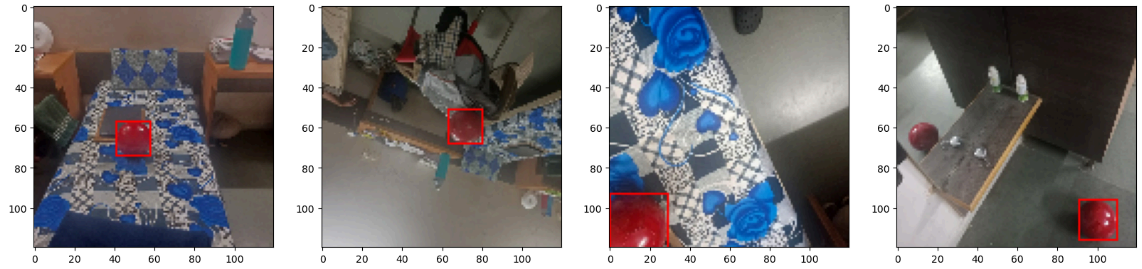
```
4.8.0
```

In [58]:
```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx in range(4):
    sample_image = res[0][idx]
    sample_coords = res[1][1][idx]

    cv2.rectangle(sample_image,
                  tuple(np.multiply(sample_coords[:2], [120,120]).astype(in
                  tuple(np.multiply(sample_coords[2:], [120,120]).astype(in
                        (255,0,0), 1)

    ax[idx].imshow(sample_image)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```



In [59]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Dense, GlobalMaxPooling2
from tensorflow.keras.applications import VGG16
```

In [60]:
```
vgg = VGG16(include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appl
ications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://
storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_t
f_dim_ordering_tf_kernels_notop.h5)
58889256/58889256 [==============================] - 129s 2us/step
```

In [61]:
```
vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0         
                                                                 
 block1_conv1 (Conv2D)       (None, None, None, 64)    1792      
                                                                 
 block1_conv2 (Conv2D)       (None, None, None, 64)    36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, None, None, 64)    0         
                                                                 
 block2_conv1 (Conv2D)       (None, None, None, 128)   73856     
                                                                 
 block2_conv2 (Conv2D)       (None, None, None, 128)   147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, None, None, 128)   0         
                                                                 
 block3_conv1 (Conv2D)       (None, None, None, 256)   295168    
                                                                 
 block3_conv2 (Conv2D)       (None, None, None, 256)   590080    
                                                                 
 block3_conv3 (Conv2D)       (None, None, None, 256)   590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, None, None, 256)   0         
                                                                 
 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, None, None, 512)   0         
                                                                 
 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, None, None, 512)   0         
                                                                 
=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [62]:

```
def build_model():
    input_layer = Input(shape=(120,120,3))

    vgg = VGG16(include_top=False)(input_layer)

    # Classification Model
    f1 = GlobalMaxPooling2D()(vgg)
    class1 = Dense(2048, activation='relu')(f1)
    class2 = Dense(1, activation='sigmoid')(class1)

    # Bounding box model
    f2 = GlobalMaxPooling2D()(vgg)
    regress1 = Dense(2048, activation='relu')(f2)
    regress2 = Dense(4, activation='sigmoid')(regress1)

    balltracker = Model(inputs=input_layer, outputs=[class2, regress2])
    return balltracker
```

In [63]:

```
balltracker = build_model()
```

In [64]:
```
balltracker.summary()
```

```
Model: "model"
__________________________________________________________________________
________________________
 Layer (type)                Output Shape                 Param #   Connec
ted to
==========================================================================
========================
 input_2 (InputLayer)        [(None, 120, 120, 3)]        0         []

 vgg16 (Functional)          (None, None, None, 512)      1471468   ['inpu
t_2[0][0]']
                                                          8

 global_max_pooling2d (Glob  (None, 512)                  0         ['vgg1
6[0][0]']
 alMaxPooling2D)

 global_max_pooling2d_1 (Gl  (None, 512)                  0         ['vgg1
6[0][0]']
 obalMaxPooling2D)

 dense (Dense)               (None, 2048)                 1050624   ['glob
al_max_pooling2d[0][0]']

 dense_2 (Dense)             (None, 2048)                 1050624   ['glob
al_max_pooling2d_1[0][0
                                                                    ']

 dense_1 (Dense)             (None, 1)                    2049      ['dens
e[0][0]']

 dense_3 (Dense)             (None, 4)                    8196      ['dens
e_2[0][0]']

==========================================================================
========================
Total params: 16826181 (64.19 MB)
Trainable params: 16826181 (64.19 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________
________________________
```

In [65]:
```
X,y = train.as_numpy_iterator().next()
```

```
In [66]: X
```

```
Out[66]: array([[[[0.8092593 , 0.7974946 , 0.7700436 ],
                  [0.80359477, 0.79183006, 0.7643791 ],
                  [0.80119824, 0.78943354, 0.76198256],
                  ...,
                  [0.36535916, 0.29084936, 0.26731995],
                  [0.46732026, 0.41241828, 0.3830065 ],
                  [0.46263683, 0.42342114, 0.38812703]],

                 [[0.80784315, 0.79607844, 0.76862746],
                  [0.80588233, 0.7941176 , 0.76666665],
                  [0.8117647 , 0.8       , 0.77254903],
                  ...,
                  [0.3663396 , 0.2918298 , 0.2663396 ],
                  [0.46372548, 0.40686274, 0.37843138],
                  [0.48104462, 0.44182894, 0.40653482]],

                 [[0.79607844, 0.79607844, 0.7647059 ],
                  [0.7964052 , 0.7964052 , 0.76503265],
                  [0.8117647 , 0.8       , 0.77254903],
```

```
In [67]: y
```

```
Out[67]: (array([[1],
                 [1],
                 [1],
                 [1],
                 [1],
                 [1],
                 [1],
                 [1]], dtype=uint8),
          array([[0.3992  , 0.      , 0.989   , 0.3857  ],
                 [0.7715  , 0.3281  , 1.      , 0.553   ],
                 [0.10815 , 0.6846  , 0.716   , 1.      ],
                 [0.3416  , 0.368   , 0.488   , 0.511   ],
                 [0.4548  , 0.727   , 0.7197  , 1.      ],
                 [0.0909  , 0.1842  , 0.875   , 0.9604  ],
                 [0.      , 0.4507  , 0.2283  , 0.676   ],
                 [0.001364, 0.8345  , 0.1967  , 1.      ]], dtype=float16))
```

```
In [68]: X.shape
```

```
Out[68]: (8, 120, 120, 3)
```

```
In [69]: y[0].shape
```

```
Out[69]: (8, 1)
```

```
In [70]: classes, coords = balltracker.predict(X)
```

```
1/1 [==============================] - 0s 451ms/step
```

In [71]:
```
classes, coords
```

Out[71]:
```
(array([[0.45654562],
        [0.43268237],
        [0.44927618],
        [0.40351665],
        [0.45899197],
        [0.42935258],
        [0.3798139 ],
        [0.50452715]], dtype=float32),
 array([[0.41460603, 0.64592713, 0.44026262, 0.49119583],
        [0.46283156, 0.6263228 , 0.38800925, 0.54358184],
        [0.4147436 , 0.6556945 , 0.42167833, 0.52570707],
        [0.44159696, 0.6357415 , 0.42227796, 0.48863956],
        [0.40326825, 0.6174791 , 0.47073182, 0.5321717 ],
        [0.37899   , 0.6706817 , 0.48252463, 0.5429722 ],
        [0.56082004, 0.5283149 , 0.36820418, 0.6373803 ],
        [0.4740559 , 0.6543531 , 0.39022642, 0.5799495 ]], dtype=float32))
```

In [72]:
```
classes
```

Out[72]:
```
array([[0.45654562],
       [0.43268237],
       [0.44927618],
       [0.40351665],
       [0.45899197],
       [0.42935258],
       [0.3798139 ],
       [0.50452715]], dtype=float32)
```

In [73]:
```
len(train)
```

Out[73]:
```
165
```

In [74]:
```
batches_per_epoch = len(train)
lr_decay = (1./0.75 -1)/batches_per_epoch
```

In [75]:
```
import tensorflow as tf

initial_learning_rate = 0.0001
lr_decay = 1e-6  # Example decay value

# Define the learning rate schedule with decay
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=10000,  # Adjust decay steps as needed
    decay_rate=0.96,
    staircase=True)

# Create the Adam optimizer using the learning rate schedule
opt = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
```

In [76]:

```
def localization_loss(y_true, yhat):
    delta_coord = tf.reduce_sum(tf.square(y_true[:,:2] - yhat[:,:2]))

    h_true = y_true[:,3] - y_true[:,1]
    w_true = y_true[:,2] - y_true[:,0]

    h_pred = yhat[:,3] - yhat[:,1]
    w_pred = yhat[:,2] - yhat[:,0]

    delta_size = tf.reduce_sum(tf.square(w_true - w_pred) + tf.square(h_tru

    return delta_coord + delta_size
```

In [77]:

```
classloss = tf.keras.losses.BinaryCrossentropy()
regressloss = localization_loss
```

In [78]:

```
localization_loss(y[1], coords).numpy()
```

Out[78]:

```
4.931039
```

In [79]:

```
classloss(y[0], classes).numpy()
```

Out[79]:

```
0.82573473
```

In [80]:

```
regressloss(y[1], coords).numpy()
```

Out[80]:

```
4.931039
```

In [81]:

```
class BallTracker(Model):
    def __init__(self, balltracker,  **kwargs):
        super().__init__(**kwargs)
        self.model = balltracker

    def compile(self, opt, classloss, localizationloss, **kwargs):
        super().compile(**kwargs)
        self.closs = classloss
        self.lloss = localizationloss
        self.opt = opt

    def train_step(self, batch, **kwargs):

        X, y = batch

        with tf.GradientTape() as tape:
            classes, coords = self.model(X, training=True)

            batch_classloss = self.closs(y[0], classes)
            batch_localizationloss = self.lloss(tf.cast(y[1], tf.float32),

            total_loss = batch_localizationloss+0.5*batch_classloss

            grad = tape.gradient(total_loss, self.model.trainable_variables

        opt.apply_gradients(zip(grad, self.model.trainable_variables))

        return {"total_loss":total_loss, "class_loss":batch_classloss, "reg

    def test_step(self, batch, **kwargs):
        X, y = batch

        classes, coords = self.model(X, training=False)

        batch_classloss = self.closs(y[0], classes)
        batch_localizationloss = self.lloss(tf.cast(y[1], tf.float32), coor
        total_loss = batch_localizationloss+0.5*batch_classloss

        return {"total_loss":total_loss, "class_loss":batch_classloss, "reg

    def call(self, X, **kwargs):
        return self.model(X, **kwargs)
```

In [82]:

```
model = BallTracker(balltracker)
```

In [83]:

```
model.compile(opt, classloss, regressloss)
```

In [84]:

```
logdir='logs'
```

In [85]:

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

```
In [86]: hist = model.fit(train, epochs=10, validation_data=val, callbacks=[tensorbo
```

```
Epoch 1/10
165/165 [==============================] - 213s 1s/step - total_loss: 0.35
26 - class_loss: 0.0160 - regress_loss: 0.3446 - val_total_loss: 0.4270 -
val_class_loss: 0.0012 - val_regress_loss: 0.4264
Epoch 2/10
165/165 [==============================] - 207s 1s/step - total_loss: 0.03
53 - class_loss: 1.1402e-04 - regress_loss: 0.0353 - val_total_loss: 0.400
1 - val_class_loss: 1.3707e-04 - val_regress_loss: 0.4000
Epoch 3/10
165/165 [==============================] - 204s 1s/step - total_loss: 0.01
93 - class_loss: 4.2402e-05 - regress_loss: 0.0192 - val_total_loss: 0.196
6 - val_class_loss: 2.0490e-05 - val_regress_loss: 0.1966
Epoch 4/10
165/165 [==============================] - 205s 1s/step - total_loss: 0.02
15 - class_loss: 2.3400e-05 - regress_loss: 0.0215 - val_total_loss: 0.324
6 - val_class_loss: 3.8804e-05 - val_regress_loss: 0.3246
Epoch 5/10
165/165 [==============================] - 203s 1s/step - total_loss: 0.01
09 - class_loss: 1.1058e-05 - regress_loss: 0.0108 - val_total_loss: 0.349
6 - val_class_loss: 1.0610e-05 - val_regress_loss: 0.3496
Epoch 6/10
165/165 [==============================] - 205s 1s/step - total_loss: 0.01
19 - class_loss: 8.0457e-06 - regress_loss: 0.0119 - val_total_loss: 0.179
0 - val_class_loss: 1.1623e-06 - val_regress_loss: 0.1790
Epoch 7/10
165/165 [==============================] - 203s 1s/step - total_loss: 0.01
08 - class_loss: 6.1363e-06 - regress_loss: 0.0108 - val_total_loss: 0.090
5 - val_class_loss: 1.0431e-07 - val_regress_loss: 0.0905
Epoch 8/10
165/165 [==============================] - 205s 1s/step - total_loss: 0.00
95 - class_loss: 5.6585e-06 - regress_loss: 0.0095 - val_total_loss: 0.205
2 - val_class_loss: 4.7684e-06 - val_regress_loss: 0.2052
Epoch 9/10
165/165 [==============================] - 205s 1s/step - total_loss: 0.01
37 - class_loss: 4.6109e-06 - regress_loss: 0.0137 - val_total_loss: 0.219
1 - val_class_loss: 3.4273e-06 - val_regress_loss: 0.2191
Epoch 10/10
165/165 [==============================] - 206s 1s/step - total_loss: 0.00
87 - class_loss: 4.1558e-06 - regress_loss: 0.0087 - val_total_loss: 0.151
0 - val_class_loss: 9.6858e-07 - val_regress_loss: 0.1510
```

In [87]:

```
hist.history
```

```
Out[87]: {'total_loss': [0.04746120795607567,
           0.03105759434401989,
           0.045612622052431 11,
           0.006442650686949491 5,
           0.00792628340423107 1,
           0.01293778792023658 8,
           0.00628854800015687 9,
           0.00663907919079065 3,
           0.01914985850453376 8,
           0.01841533929109573 4],
          'class_loss': [0.00053098192438483 24,
           0.00019549595890566707,
           3.9488094216721947e-07,
           2.928387220890727e-05,
           4.20216110796900 5e-06,
           1.4499578355753329e-05,
           1.4901162970204496e-08,
           1.4052081496629398e-05,
           0.0,
           5.885971745556162e-07],
          'regress_loss': [0.04719571769237518,
           0.030959846451878548,
           0.045612424612045 29,
           0.006428008899092674,
           0.007924182340502739,
           0.01293053850531578,
           0.0062885405495762825,
           0.006632053293287754,
           0.019149858504533768,
           0.018415044993162155],
          'val_total_loss': [0.4269649088382721,
           0.4001111686229706,
           0.19658887386322021,
           0.3245813250541687,
           0.3495769500732422,
           0.17898453772068024,
           0.09052697569131851,
           0.2051771879196167,
           0.21908409893512726,
           0.15103386342525482],
          'val_class_loss': [0.0012077523861080408,
           0.0001370721583953127 3,
           2.0489709640969522e-05,
           3.880390431731939e-05,
           1.0609742275846656e-05,
           1.1622920510490076e-06,
           1.0430814256778831e-07,
           4.768402959598461e-06,
           3.4272818538738647e-06,
           9.685772965895012e-07],
          'val_regress_loss': [0.4263610243797302,
           0.4000426232814789,
           0.19657862186431885,
           0.3245619237422943,
           0.3495716452598572,
           0.17898395657539368,
           0.09052692353725433,
           0.20517480373382568,
           0.21908238530158997,
           0.15103337168693542]}
```

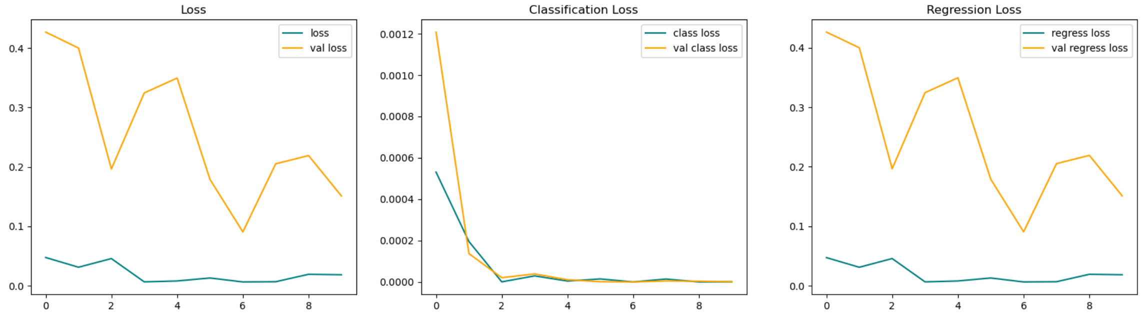
```
In [88]: fig, ax = plt.subplots(ncols=3, figsize=(20,5))

         ax[0].plot(hist.history['total_loss'], color='teal', label='loss')
         ax[0].plot(hist.history['val_total_loss'], color='orange', label='val loss'
         ax[0].title.set_text('Loss')
         ax[0].legend()

         ax[1].plot(hist.history['class_loss'], color='teal', label='class loss')
         ax[1].plot(hist.history['val_class_loss'], color='orange', label='val class
         ax[1].title.set_text('Classification Loss')
         ax[1].legend()

         ax[2].plot(hist.history['regress_loss'], color='teal', label='regress loss'
         ax[2].plot(hist.history['val_regress_loss'], color='orange', label='val reg
         ax[2].title.set_text('Regression Loss')
         ax[2].legend()

         plt.show()
```



```
In [89]: test_data = test.as_numpy_iterator()
```

```
In [90]: test_sample = test_data.next()
```

```
In [91]: yhat = balltracker.predict(test_sample[0])
```

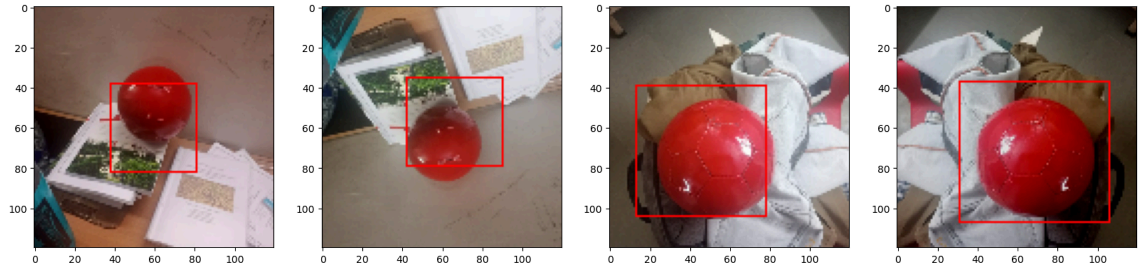
```
1/1 [==============================] - 0s 205ms/step
```

In [92]:
```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx in range(4):
    sample_image = test_sample[0][idx]
    sample_coords = yhat[1][idx]

    if yhat[0][idx] > 0.9:
        cv2.rectangle(sample_image,
                      tuple(np.multiply(sample_coords[:2], [120,120]).astyp
                      tuple(np.multiply(sample_coords[2:], [120,120]).astyp
                            (255,0,0), 1)

    ax[idx].imshow(sample_image)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] fo
r floats or [0..255] for integers).
```



In [93]:
```
from tensorflow.keras.models import load_model
```

In [94]:
```
balltracker.save('balltracker.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics hav
e yet to be built. `model.compile_metrics` will be empty until you train o
r evaluate the model.

C:\Users\sanke\anaconda3\Lib\site-packages\keras\src\engine\training.py:30
00: UserWarning: You are saving your model as an HDF5 file via `model.save
()`. This file format is considered legacy. We recommend using instead the
native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

In [95]:
```
balltracker = load_model('balltracker.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file, so th
e model was *not* compiled. Compile it manually.
```

In [ ]: