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:\\images_2\\images\\*.jpg')
```

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

Out[3]:
```
b'D:\\images_2\\images\\006.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([[[180, 180, 190],
        [179, 182, 191],
        [179, 183, 194],
        ...,
        [159, 173, 173],
        [163, 177, 178],
        [166, 180, 181]],

       [[180, 180, 188],
        [177, 180, 187],
        [174, 181, 189],
        ...,
        [166, 180, 180],
        [167, 181, 182],
        [167, 181, 182]],

       [[179, 182, 187],
        [176, 181, 185],
        [173, 180, 186],
        ...,
        [170, 184, 184],
        [169, 183, 183],
        [164, 178, 178]],

       ...,

       [[105,  90,  67],
        [105,  87,  63],
        [107,  85,  61],
        ...,
        [135,  99,  67],
        [127,  91,  57],
        [138, 104,  69]],

       [[118,  98,  71],
        [113,  92,  65],
        [113,  87,  62],
        ...,
        [137, 101,  69],
        [125,  91,  56],
        [131,  99,  61]],

       [[121, 100,  71],
        [113,  89,  61],
        [113,  85,  61],
        ...,
        [138, 104,  69],
        [128,  94,  59],
        [128,  96,  58]]], dtype=uint8)
```

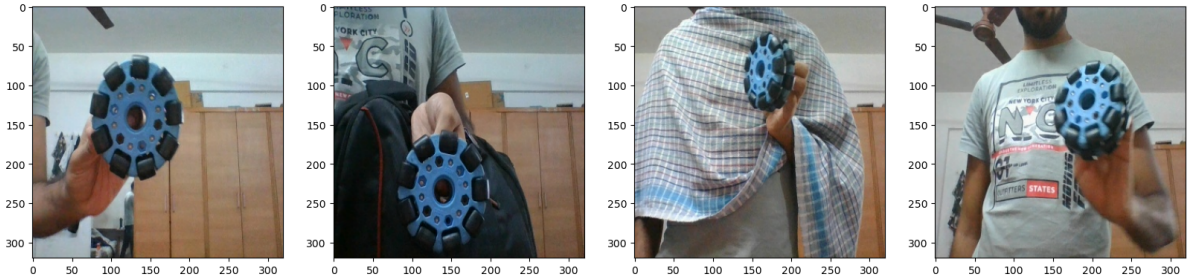
In [7]:
```
type(images)
```

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]:
```
50*.7
```

Out[11]:
```
35.0
```

In [12]:
```
50*.15
```

Out[12]:
```
7.5
```

In [13]:
```
import os
for folder in ['train','test','val']:
    for file in os.listdir(os.path.join('D:\\images_2',folder,'images')):
        filename = file.split('.')[0]+'.json'
        existing_filepath = os.path.join('D:\\images_2','labels',filename)
        if os.path.exists(existing_filepath):
            new_filepath = os.path.join('D:\\images_2',folder,'labels',filename)
            os.replace(existing_filepath,new_filepath)
```

In [14]:
```
import albumentations as alb
```

In [15]:
```
augmentor = alb.Compose([alb.RandomCrop(width=320, height=320),
                         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_labels']))
```

In [16]:
```
import cv2
img = cv2.imread(os.path.join('D:\\images_2','train','images','002.jpg'))
```

In [17]:
```
img
```

Out[17]:
```
array([[[202, 189, 187],
        [204, 190, 184],
        [209, 193, 181],
        ...,
        [194, 179, 176],
        [197, 176, 178],
        [197, 176, 178]],

       [[205, 193, 189],
        [209, 196, 188],
        [213, 197, 184],
        ...,
        [195, 180, 177],
        [196, 178, 177],
        [198, 177, 179]],

       [[205, 196, 187],
        [208, 198, 188],
        [218, 205, 189],
        ...,
        [196, 181, 178],
        [197, 179, 178],
        [197, 179, 178]],

       ...,

       [[ 13,  11,  10],
        [ 13,  14,  12],
        [ 14,  16,  17],
        ...,
        [ 69,  93, 139],
        [ 61,  85, 131],
        [ 77, 102, 146]],

       [[ 17,  13,  12],
        [ 17,  15,  14],
        [ 17,  16,  18],
        ...,
        [ 71,  97, 143],
        [ 65,  91, 137],
        [ 72,  99, 143]],

       [[ 17,  13,  12],
        [ 16,  14,  13],
        [ 19,  16,  18],
        ...,
        [ 74, 100, 146],
        [ 67,  93, 139],
        [ 65,  92, 136]]], dtype=uint8)
```

In [18]:
```
with open(os.path.join('D:\\images_2', 'train', 'labels', '002.json'), 'r') as f:
    label = json.load(f)
```

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

Out[19]:
```
[[29.965277777777757, 130.48611111111111], [236.15234375, 319.0]]
```

In [20]:
```
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 [21]:
```
coords
```

Out[21]:
```
[29.965277777777757, 130.48611111111111, 236.15234375, 319.0]
```

In [22]:
```
coords = list(np.divide(coords,[320,320,320,320]))
```

In [23]:
```
coords
```

Out[23]:
```
[0.0936414930555555, 0.4077690972222222, 0.73797607421875, 0.996875]
```

In [24]:
```
augmented = augmentor(image=img, bboxes=[coords], class_labels=['face'])
```

In [25]:
```
augmented.keys()
```

Out[25]:
```
dict_keys(['image', 'bboxes', 'class_labels'])
```

In [26]:
```
augmented['bboxes'][0][2:]
```

Out[26]:
```
(0.9063585069444445, 0.5922309027777778)
```

In [27]:
```
augmented['bboxes']
```

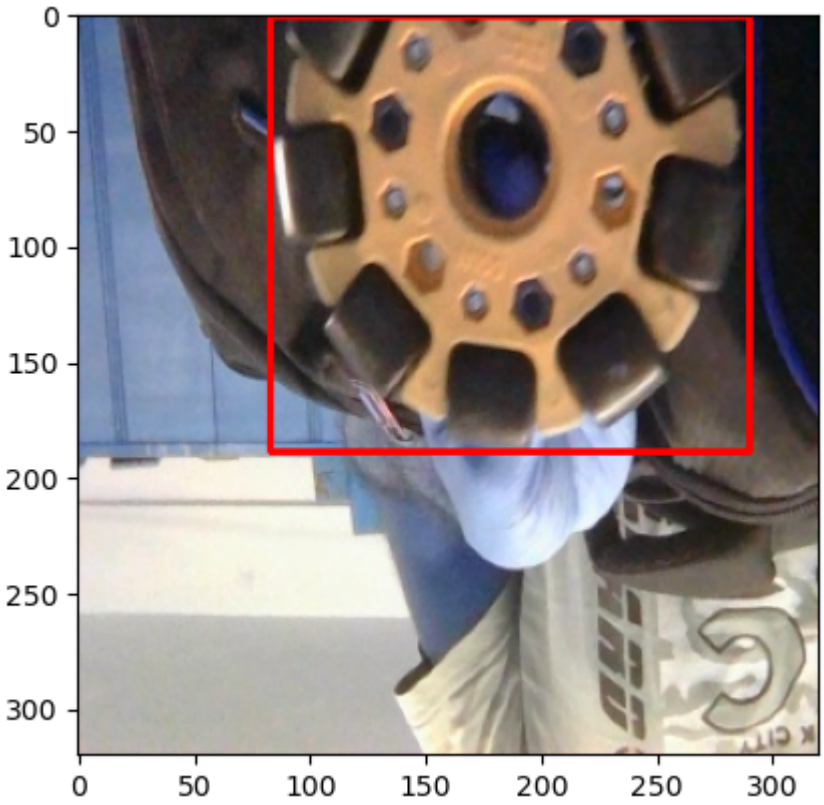
Out[27]:
```
[(0.26202392578125,
  0.0031250000000000444,
  0.9063585069444445,
  0.5922309027777778)]
```

In [28]:
```
augmented['image']
```

Out[28]:
```
array([[[ 65,  92, 136],
        [ 67,  93, 139],
        [ 74, 100, 146],
        ...,
        [ 19,  16,  18],
        [ 16,  14,  13],
        [ 17,  13,  12]],

       [[ 72,  99, 143],
        [ 65,  91, 137],
        [ 71,  97, 143],
        ...,
        [ 17,  16,  18],
        [ 17,  15,  14],
        [ 17,  13,  12]],

       [[ 77, 102, 146],
        [ 61,  85, 131],
        [ 69,  93, 139],
        ...,
        [ 14,  16,  17],
        [ 13,  14,  12],
        [ 13,  11,  10]],

       ...,

       [[197, 179, 178],
        [197, 179, 178],
        [196, 181, 178],
        ...,
        [218, 205, 189],
        [208, 198, 188],
        [205, 196, 187]],

       [[198, 177, 179],
        [196, 178, 177],
        [195, 180, 177],
        ...,
        [213, 197, 184],
        [209, 196, 188],
        [205, 193, 189]],

       [[197, 176, 178],
        [197, 176, 178],
        [194, 179, 176],
        ...,
        [209, 193, 181],
        [204, 190, 184],
        [202, 189, 187]]], dtype=uint8)
```

In [29]:
```
cv2.rectangle(augmented['image'],
              tuple(np.multiply(augmented['bboxes'][0][:2], [320,320]).astype(int)),
              tuple(np.multiply(augmented['bboxes'][0][2:], [320,320]).astype(int)),
                    (255,0,0), 2)

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

Out[29]:
```
<matplotlib.image.AxesImage at 0x17356a8b370>
```



In [30]:
```
for partition in ['train','test','val']:
    for image in os.listdir(os.path.join('D:\\images_2', partition, 'images')):
        img = cv2.imread(os.path.join('D:\\images_2', partition, 'images', image))

        coords = [0,0,0.00001,0.00001]
        label_path = os.path.join('D:\\images_2', partition, 'labels', f'{image.spl
        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, [320,320,320,320]))

        try:
            for x in range(34):
                augmented = augmentor(image=img, bboxes=[coords], class_labels=['fa
                cv2.imwrite(os.path.join('D:\\aug_data', partition, 'images', f'{im

                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:\\aug_data', partition, 'labels', f'{imag
```

```
                    json.dump(annotation, f)

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

In [31]:
```
train_images = tf.data.Dataset.list_files('D:\\aug_data\\train\\images\\*.jpg', shu
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 [32]:
```
test_images = tf.data.Dataset.list_files('D:\\aug_data\\test\\images\\*.jpg', shuff
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 [33]:
```
val_images = tf.data.Dataset.list_files('D:\\aug_data\\val\\images\\*.jpg', shuffle
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 [34]:
```
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 [35]:
```
train_labels = tf.data.Dataset.list_files('D:\\aug_data\\train\\labels\\*.json', sh
train_labels = train_labels.map(lambda x: tf.py_function(load_labels, [x], [tf.uint
```

In [36]:
```
test_labels = tf.data.Dataset.list_files('D:\\aug_data\\test\\labels\\*.json', shuf
test_labels = test_labels.map(lambda x: tf.py_function(load_labels, [x], [tf.uint8,
```

In [37]:
```
val_labels = tf.data.Dataset.list_files('D:\\aug_data\\val\\labels\\*.json', shuffl
val_labels = val_labels.map(lambda x: tf.py_function(load_labels, [x], [tf.uint8, t
```

In [38]:
```
train_labels.as_numpy_iterator().next()
```

Out[38]:
```
(array([1], dtype=uint8),
 array([0.262   , 0.003124, 0.9062  , 0.5923  ], dtype=float16))
```

In [39]:
```
len(train_images), len(train_labels), len(test_images), len(test_labels), len(val_i
```

Out[39]:
```
(1190, 1190, 420, 420, 272, 272)
```

In [40]:
```
train = tf.data.Dataset.zip((train_images, train_labels))
train = train.shuffle(3000)
train = train.batch(4)
train = train.prefetch(4)
```

In [41]:
```
test = tf.data.Dataset.zip((test_images, test_labels))
test = test.shuffle(1300)
test = test.batch(4)
test = test.prefetch(4)
```

In [42]:
```
val = tf.data.Dataset.zip((val_images, val_labels))
val = val.shuffle(1000)
val = val.batch(4)
val = val.prefetch(4)
```

In [43]:
```
train.as_numpy_iterator().next()[1]
```

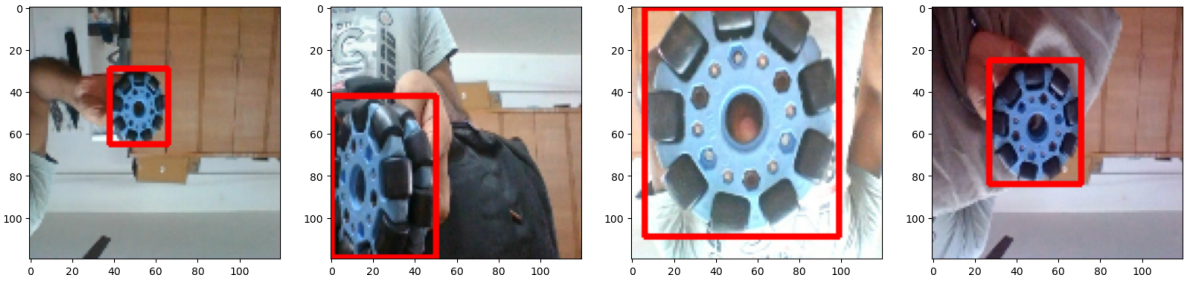
Out[43]:
```
(array([[1],
        [1],
        [1],
        [1]], dtype=uint8),
 array([[0.3594 , 0.3003 , 0.814  , 0.8896 ],
        [0.01009, 0.00846, 0.495  , 0.6543 ],
        [0.4702 , 0.3513 , 0.6787 , 0.676  ],
        [0.3345 , 0.4675 , 0.5625 , 0.767  ]], dtype=float16))
```

In [44]:
```
data_samples = train.as_numpy_iterator()
```

In [45]:
```
res = data_samples.next()
```

In [46]:
```
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(int)),
                  tuple(np.multiply(sample_coords[2:], [120,120]).astype(int)),
                        (255,0,0), 2)

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

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



In [47]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Dense, GlobalMaxPooling2D
from tensorflow.keras.applications import VGG16
```

In [48]:
```
vgg = VGG16(include_top=False)
```

In [49]:
```
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: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [50]:

```python
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)
```

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

In [51]:
```
wheeltracker = build_model()
```

In [52]:
```
wheeltracker.summary()
```

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

 vgg16 (Functional)             (None, None, None,   14714688    ['input_2[0][0]']
                                512)

 global_max_pooling2d (GlobalMa  (None, 512)         0           ['vgg16[0][0]']
 xPooling2D)

 global_max_pooling2d_1 (Global  (None, 512)         0           ['vgg16[0][0]']
 MaxPooling2D)

 dense (Dense)                  (None, 2048)         1050624     ['global_max_pool
ing2d[0][0]']

 dense_2 (Dense)                (None, 2048)         1050624     ['global_max_pool
ing2d_1[0][0]']

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

 dense_3 (Dense)                (None, 4)            8196        ['dense_2[0][0]']

==================================================================================================
Total params: 16,826,181
Trainable params: 16,826,181
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [53]:
```
X,y = train.as_numpy_iterator().next()
```

In [54]:
```
X.shape
```

Out[54]:
```
(4, 120, 120, 3)
```

In [55]:
```
classes, coords = wheeltracker.predict(X)
```

```
1/1 [==============================] - 1s 808ms/step
```

In [56]:
```
classes,coords
```

Out[56]:
```
(array([[0.63973665],
        [0.6732118 ],
        [0.5954127 ],
        [0.49985802]], dtype=float32),
 array([[0.3698088 , 0.2248449 , 0.62256306, 0.51001376],
        [0.38658082, 0.2305657 , 0.55798966, 0.42848614],
        [0.3334335 , 0.20294228, 0.5998287 , 0.55069065],
        [0.36772338, 0.27767774, 0.69055504, 0.5154221 ]], dtype=float32))
```

In [57]:
```
batches_per_epoch = len(train)
lr_decay = (1./0.75 -1)/batches_per_epoch
```

In [58]:
```
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 [59]:
```
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_true-h_pred

    return delta_coord + delta_size
```

In [60]:
```
classloss = tf.keras.losses.BinaryCrossentropy()
regressloss = localization_loss
```

In [61]:
```
localization_loss(y[1], coords)
```

Out[61]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9014548>
```

In [62]:
```
classloss(y[0], classes)
```

Out[62]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.5135812>
```

In [63]:
```
regressloss(y[1], coords)
```

Out[63]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.9014548>
```

In [64]:
```
class WheelTracker(Model):
    def __init__(self, eyetracker,  **kwargs):
        super().__init__(**kwargs)
        self.model = eyetracker

    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), coords)

            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, "regress_los

    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), coords)
        total_loss = batch_localizationloss+0.5*batch_classloss

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

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

In [65]:
```
model = WheelTracker(wheeltracker)
```

In [66]:
```
model.compile(opt, classloss, regressloss)
```

In [67]:
```
logdir='logs'
```

In [68]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

In [69]:
```
hist = model.fit(train, epochs=10, validation_data=val, callbacks=[tensorboard_call
```

```
Epoch 1/10
298/298 [==============================] - 272s 887ms/step - total_loss: 0.0905 -
class_loss: 0.0047 - regress_loss: 0.0882 - val_total_loss: 0.0854 - val_class_los
s: 1.8344e-05 - val_regress_loss: 0.0854
Epoch 2/10
298/298 [==============================] - 225s 753ms/step - total_loss: 0.0125 -
class_loss: 2.0134e-05 - regress_loss: 0.0125 - val_total_loss: 0.0476 - val_class
_loss: 4.0829e-06 - val_regress_loss: 0.0476
Epoch 3/10
298/298 [==============================] - 222s 744ms/step - total_loss: 0.0133 -
class_loss: 6.0874e-06 - regress_loss: 0.0133 - val_total_loss: 0.0090 - val_class
_loss: 1.2368e-06 - val_regress_loss: 0.0090
Epoch 4/10
298/298 [==============================] - 219s 732ms/step - total_loss: 0.0062 -
class_loss: 2.8463e-06 - regress_loss: 0.0062 - val_total_loss: 0.0105 - val_class
_loss: 1.0133e-06 - val_regress_loss: 0.0105
Epoch 5/10
298/298 [==============================] - 218s 729ms/step - total_loss: 0.0071 -
class_loss: 1.9241e-06 - regress_loss: 0.0071 - val_total_loss: 0.0552 - val_class
_loss: 1.8477e-06 - val_regress_loss: 0.0552
Epoch 6/10
298/298 [==============================] - 219s 733ms/step - total_loss: 0.0055 -
class_loss: 1.3490e-06 - regress_loss: 0.0055 - val_total_loss: 0.0264 - val_class
_loss: 1.2517e-06 - val_regress_loss: 0.0264
Epoch 7/10
298/298 [==============================] - 221s 739ms/step - total_loss: 0.0065 -
class_loss: 8.9168e-07 - regress_loss: 0.0065 - val_total_loss: 0.0123 - val_class
_loss: 0.0000e+00 - val_regress_loss: 0.0123
Epoch 8/10
298/298 [==============================] - 220s 736ms/step - total_loss: 0.0046 -
class_loss: 6.4718e-07 - regress_loss: 0.0046 - val_total_loss: 0.0027 - val_class
_loss: 5.9605e-08 - val_regress_loss: 0.0027
Epoch 9/10
298/298 [==============================] - 221s 738ms/step - total_loss: 0.0073 -
class_loss: 7.0056e-07 - regress_loss: 0.0073 - val_total_loss: 0.0027 - val_class
_loss: 5.3644e-07 - val_regress_loss: 0.0027
Epoch 10/10
298/298 [==============================] - 222s 743ms/step - total_loss: 0.0068 -
class_loss: 7.2388e-07 - regress_loss: 0.0068 - val_total_loss: 0.0073 - val_class
_loss: 5.3644e-07 - val_regress_loss: 0.0073
```

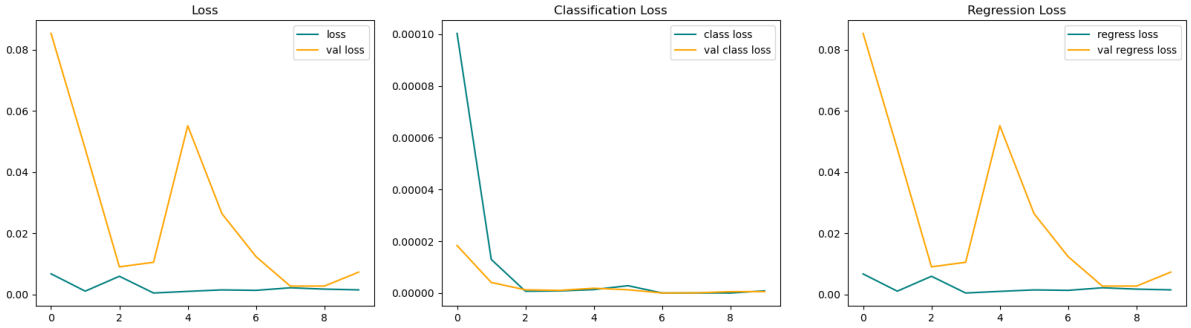
In [70]:
```
hist.history
```

Out[70]:

```
{'total_loss': [0.0067139542661607265,
  0.0010398124577477574,
  0.0059084463864564896,
  0.0004323918838053941 7,
  0.0009732298785820603,
  0.0014478960074484348,
  0.0013017215533182025,
  0.002146644052118063,
  0.001702834153547883,
  0.0014607426710426807],
 'class_loss': [0.00010026506788562983,
  1.2964095731149428e-05,
  6.854538696643431e-07,
  8.046631023717055e-07,
  1.3113038903611596e-06,
  2.8312267659202917e-06,
  0.0,
  2.9802322387695312e-08,
  0.0,
  8.344657089764951e-07],
 'regress_loss': [0.006663821637630463,
  0.0010333304526284337,
  0.005908103659749031,
  0.0004319895524531603,
  0.000972574227489531,
  0.0014464803971350193,
  0.0013017215533182025,
  0.002146629150956869,
  0.001702834153547883,
  0.0014603254385292 53],
 'val_total_loss': [0.08537231385707855,
  0.04760414734482765,
  0.00899313110858202,
  0.010493303649127483,
  0.055169083178043365,
  0.02640898898243904,
  0.01231568306684494,
  0.0026930654421448708,
  0.0027068983763456345,
  0.007261715363711119],
 'val_class_loss': [1.8343856936553493e-05,
  4.082930445292732e-06,
  1.2367975159577327e-06,
  1.0132800980500178e-06,
  1.8477462617738638e-06,
  1.251698790838418e-06,
  0.0,
  5.960465188081798e-08,
  5.364421440390288e-07,
  5.364420871956099e-07],
 'val_regress_loss': [0.08536314219236374,
  0.047602105885744095,
  0.008992512710392475,
  0.010492797009646893,
  0.05516815930604935,
  0.02640836313366 89,
  0.01231568306684494,
  0.002693035639822483,
  0.002706630155444145,
  0.007261447142809 6294]}
```


In [71]:
```
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 loss')
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 regress los
ax[2].title.set_text('Regression Loss')
ax[2].legend()

plt.show()
```



In [72]:
```
test_data = test.as_numpy_iterator()
```

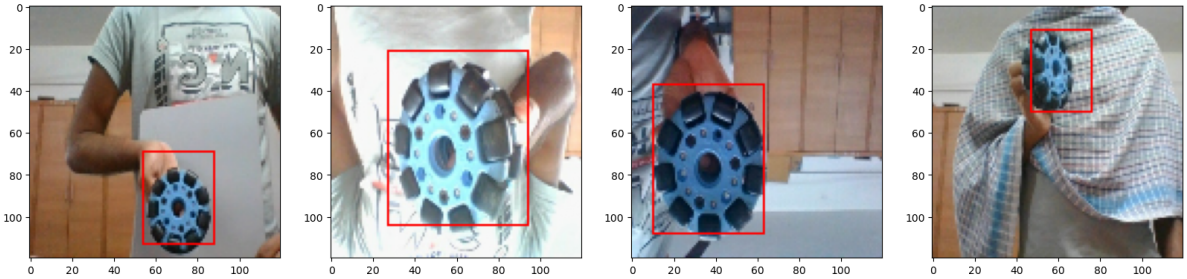
In [98]:
```
test_sample = test_data.next()
```

In [99]:
```
yhat = wheeltracker.predict(test_sample[0])
```

```
1/1 [==============================] - 0s 100ms/step
```

In [100…
```
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]).astype(int)),
                      tuple(np.multiply(sample_coords[2:], [120,120]).astype(int)),
                            (255,0,0), 1)

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

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



In [92]:
```
from tensorflow.keras.models import load_model
```

In [87]:
```
wheeltracker.save('wheeltracker.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to
be built. `model.compile_metrics` will be empty until you train or evaluate the mo
del.
```

In [88]:
```
wheeltracker = load_model('wheeltracker.h5')
```

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

In [80]:
```
cap = cv2.VideoCapture(1)
while cap.isOpened():
    _ , frame = cap.read()
    frame = frame[50:500, 50:500,:]

    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    resized = tf.image.resize(rgb, (120,120))

    yhat = wheeltracker.predict(np.expand_dims(resized/255,0))
    sample_coords = yhat[1][0]

    if yhat[0] > 0.5:
        # Controls the main rectangle
        cv2.rectangle(frame,
                      tuple(np.multiply(sample_coords[:2], [450,450]).astype(int)),
                      tuple(np.multiply(sample_coords[2:], [450,450]).astype(int)),
                            (255,0,0), 2)
        # Controls the label rectangle
        cv2.rectangle(frame,
                      tuple(np.add(np.multiply(sample_coords[:2], [450,450]).astype
                                   [0,-30])),
                      tuple(np.add(np.multiply(sample_coords[:2], [450,450]).astype
                                   [80,0])),
                            (255,0,0), -1)

        # Controls the text rendered
        cv2.putText(frame, 'face', tuple(np.add(np.multiply(sample_coords[:2], [450
                                            [0,-5])),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

    cv2.imshow('EyeTrack', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

In [ ]: