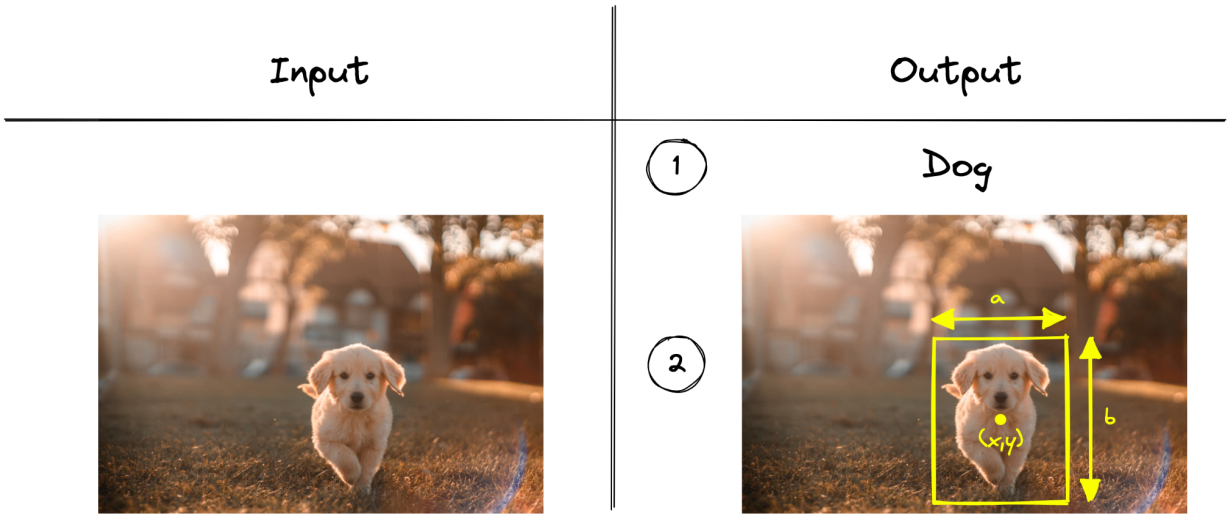
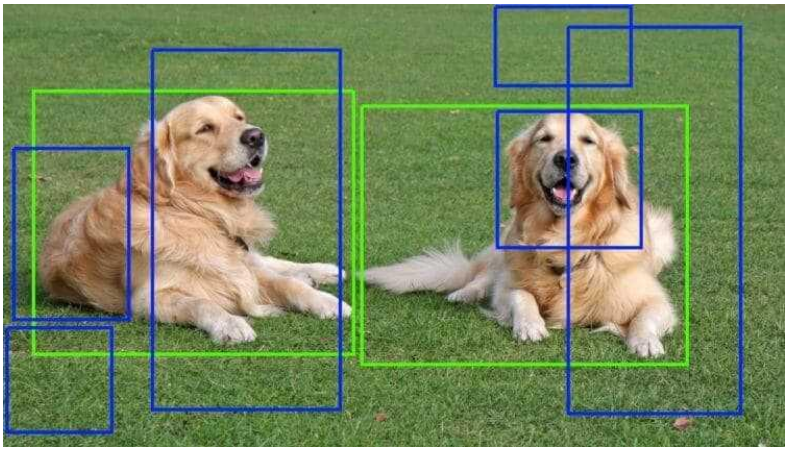
Image Localization documentation

Image localization is **a spin-off of regular CNN vision algorithms**. These algorithms predict classes with discrete numbers. In object localization, the algorithm predicts a set of 4 continuous numbers, namely, x coordinate, y coordinate, height, and width, to draw a bounding box around an object of interest.





Importing the Required modules

```
import numpy as np
import glob
import PIL
import cv2
import xmltodict
import random
from tqdm import tqdm
from PIL import ImageDraw
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, MaxPool2D, LeakyReLU, Dense,
Flatten, BatchNormalization, Dropout
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model, load_model
from tensorflow.keras import backend as K
from tensorflow.keras.utils import plot_model
```

numpy : this is being used here to convert the list into array and do some array operations on image

glob : the glob module is used to retrieve files/pathnames matching a specified pattern.

cv2 : it is used to read image , in the form of array

ImageDraw : it is used to draw rectangle in image to localize the object in it

Path of the images for the training

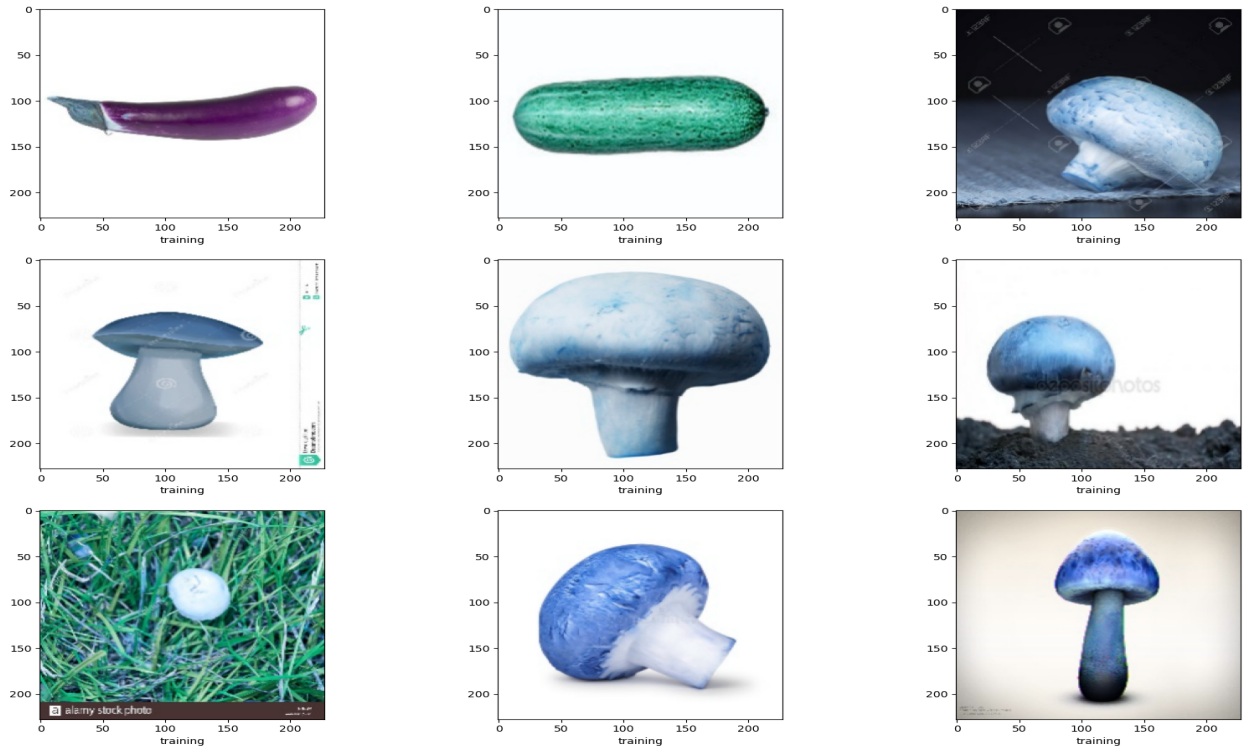
```
path = "training_images"
```

The function (**normalize**) is used to **resize** the image from its original dimension to **228 x 228** using **cv2.resize()** method and append the image into a list images and also put the class name of the image into names list.

```
def normalize(path):
    images=[]
    names =[]
    for file in tqdm(glob.glob(path +"/*.jpg")):
        image = cv2.resize(cv2.imread(file), (228,228))
        image = np.array(image)
        name = file.split('/')[-1].split('_')[0]
        images.append(image)
        names.append(name)
    return images,names

images,names = normalize(path)
```

This part of code display 9 randomly picked images from the images list also mention the class of the image as label , size of each image displayed assigned in **figsize = (20,15) ,** it usages matplotlib.pyplot library to show the image

```
fig = plt.figure(figsize=(20,15))
for i in range(9):
    r = random.randint(1,186)
    plt.subplot(3,3,i+1)
    plt.imshow(images[r])
    plt.xlabel(names[r])

plt.show()
```



It creates a rectangle box around the object detected

```
def get_bbox(xml_path):
    bboxes =[]
    classnames=[]
    for file in tqdm(glob.glob(xml_path + "/*.xml")):
        x = xmltodict.parse(open(file, 'rb'))
        bbox = x['annotation']['object']['bndbox']
        name = x['annotation']['object']['name']
        bbox =
np.array([int(bbox['xmin']),int(bbox['ymin']),int(bbox['xmax']),int(bbox['
ymax'])])
        bbox2 = [None]*4
        bbox2[0] = bbox[0]
        bbox2[1] = bbox[1]
        bbox2[2] = bbox[2]
        bbox2[3] = bbox[3]
        bbox2 = np.array(bbox2)/228
        bboxes.append(bbox2)
        classnames.append(name)
    return np.array(bboxes), classnames
bboxes, classnames = get_bbox(path)
```

LabelBinarizer turns every variable into binary within a matrix where that variable is indicated as a column.

```
encoder = LabelBinarizer()
classnames = encoder.fit_transform(classnames)
Y = np.concatenate([bboxes, classnames], axis=1)
X = np.array(images)
```

`fit_transform`**(y)**	Fit label binarizer/transform multi-class labels to binary labels.

Here we dump the `LabelBinarizer into labelbinarizer.pkl` pickle file so that we can use it later without running the code again just by loading the file.

```
import pickle
with open('labelbinarizer.pkl', 'wb') as file:
    # A new file will be created
    pickle.dump(encoder, file)
```

Here we split the loaded data into Train and Test dataset for training and testing
`X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.1)`

Intersection over Union (IoU)

```
def calculate_iou(target, pred):
    xA = K.maximum(target[:,0], pred[:,0])
    yA = K.maximum(target[:,1], pred[:,1])
    xB = K.minimum(target[:,2], pred[:,2])
    yB = K.minimum(target[:,3], pred[:,3])
    interArea = K.maximum(0.0, xB-xA)*K.maximum(0.0,yB-yA)
    boxAarea = (target[:,2]-target[:,0])*(target[:,3]-target[:,1])
    boxBarea = (pred[:,2]-pred[:,0]) * (pred[:,3]-pred[:,1])

    iou = interArea / (boxAarea+boxBarea - interArea)
    return iou
```

Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset

Mean squared error (MSE) measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values

Here custom_loss is the sum of MSE and (1-IOU)

```
def custom_loss(y_true, y_pred):
    mse = tf.losses.mean_squared_error(y_true, y_pred)
    iou = calculate_iou(y_true , y_pred)
    return mse + (1-iou)

def iou_metric(y_true, y_pred):
    return calculate_iou(y_true, y_pred)
```

Initializing parameters for the model

Dimension of the Image

```
input_shape = ( 228 , 228 , 3 )
```

Percentage of the neurons to deactivate

```
dropout_rate = 0.5
```

Number of classes

```
classes = 3

alpha = 0.2
prediction_units = 4+ classes
```

Neural Network

```
def block1(filters,X):
    x = Conv2D(filters, kernel_size=(3,3), strides=1 )(X)
    x = LeakyReLU(alpha)(x)
    x = Conv2D(filters, (3,3), strides=1)(x)
    x = LeakyReLU(alpha)(x)
    x = MaxPool2D((2,2))(x)
    return x

def block2(units,X):
    x = Dense(units)(X)
    x = LeakyReLU(alpha)(x)
    return x

def mymodel():
    model_input = Input(shape=(228,228,3))
    x= block1(16, model_input)
    x= block1(32,x)
    x = block1(64,x)
    x= block1(128,x)
    x = block1(256,x)

    x = Flatten()(x)
    x = block2(1240, x)
    x = block2(640, x)
    x = block2(480,x)
    x = block2(120,x)
    x = block2(62,x)
    model_outputs = Dense(prediction_units)(x)
    model = Model(inputs=[model_input], outputs=[model_outputs])
    model.compile( tf.keras.optimizers.Adam(0.0001),
                  loss=custom_loss,
                  metrics=[iou_metric])
    return model
model = mymodel()
```

ModelCheckpoint callback is used in conjunction with training using `model.fit()` to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.

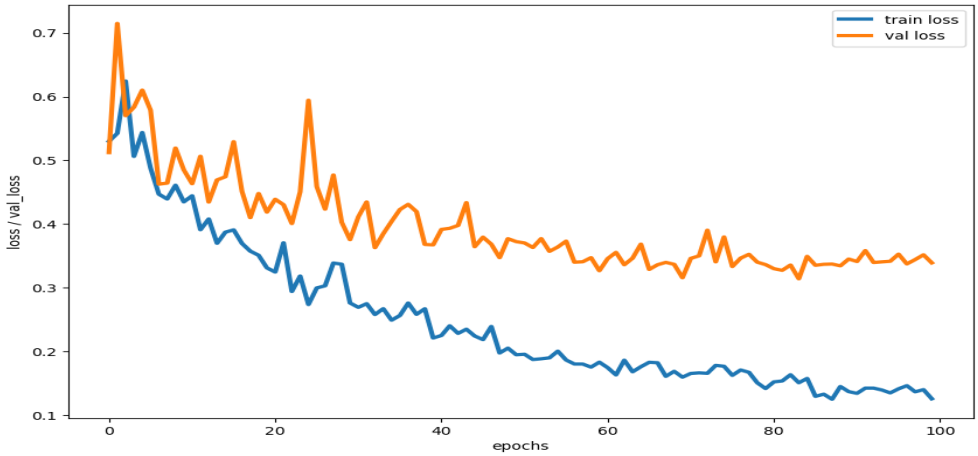
```
from tensorflow.keras.callbacks import ModelCheckpoint

filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
```

Model Training

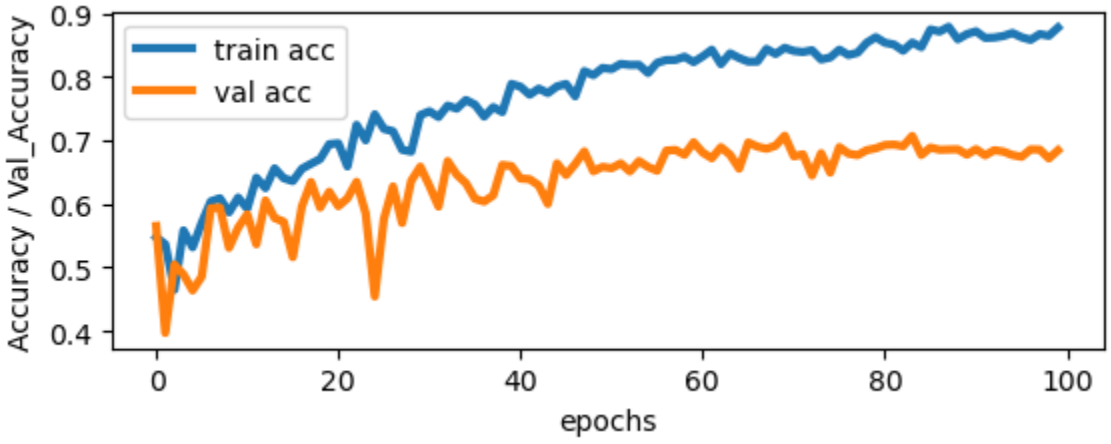
```
history = model.fit( X_train ,Y_train ,validation_data=( X_test , Y_test),
    epochs=100,
    batch_size=3)
loss = history.history['loss']
val_loss = history.history['val_loss']

acc = history.history['iou_metric']
val_acc = history.history['val_iou_metric']
```

Plotting the Trained model loss and accuracy score

```
plt.figure(figsize=(10,15))
plt.subplot(2,1,1)
plt.plot(loss , linewidth=3 ,label='train loss')
plt.plot(val_loss , linewidth=3, label='val loss')
plt.xlabel('epochs')
plt.ylabel('loss / val_loss')
plt.legend()
```



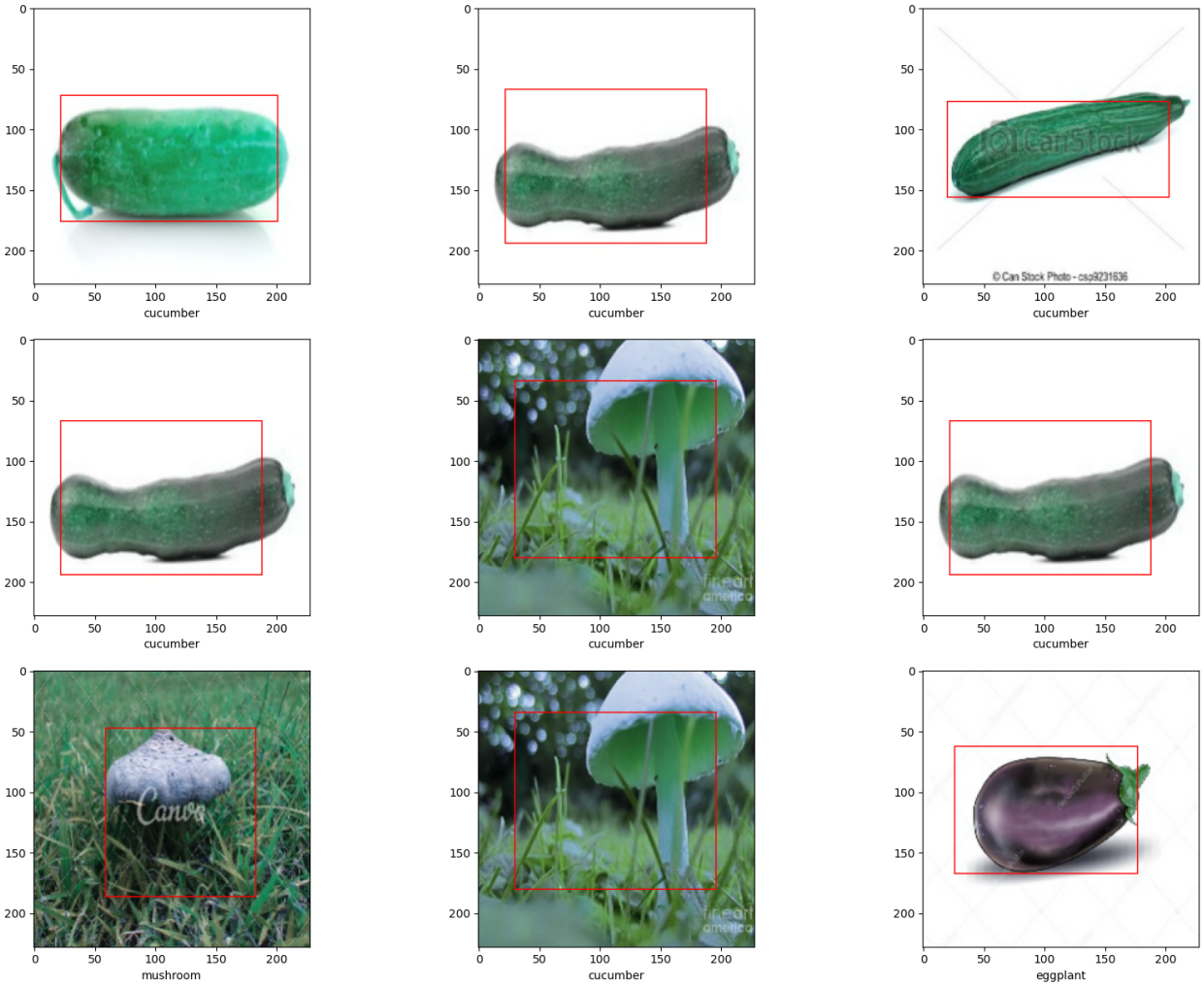

```
plt.subplot(2,1,2)
plt.plot(acc , linewidth=3 ,label='train acc')
plt.plot(val_acc , linewidth=3, label='val acc')
plt.xlabel('epochs')
plt.ylabel('Accuracy / Val_Accuracy')
plt.legend()
```



Drawbox function is used to draw a rectangle in the image

```
def drawbox(model,image, y_true, le):
    img = tf.cast(np.expand_dims(image, axis=0), tf.float32)
    y_true = np.expand_dims(y_true, axis=0)
    #prediction
    predict = model.predict(img)
    #Box coordinates
    Y_test_box =y_true[...,0:4]*228
    pred_box = predict[...,0:4]*228
    x = pred_box[0][0]
    y = pred_box[0][1]
    w = pred_box[0][2]
    h = pred_box[0][3]
    #get class name
    trans= le.inverse_transform(predict[...,4:])
    im = PIL.Image.fromarray(image)
    draw=ImageDraw.Draw(im)
    draw.rectangle([x,y,w,h], outline='red')
    plt.xlabel(trans[0])
    plt.imshow(im)
    iou = calculate_iou(Y_test_box, pred_box)
```

```
Predict 9 images and
fig = plt.figure(figsize=(20,15))

for i in range(9):
    r = random.randint(1,10)
    plt.subplot(3,3,i+1)
    drawbox(model,X_test[r], Y_test[r], encoder)
plt.show()
```



Save the model

model.save("image_localization_model.h5")