Image Classification documentation

- For the image classification modeling, we are using the Keras library

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library

For the image classification we are using ResNet model to classify the images in CIFAR-10

ResNet, short for Residual Networks, is a classic neural network used **as a backbone for many computer vision tasks**. This model was the winner of the ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully.

Installing the required modules

```
!pip install -q keras
!pip install image-classifiers
!pip install git+https://github.com/qubvel/classification_models.git
```

Importing The libraries

```
import keras
from keras.datasets import cifar10
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
import numpy as np
%matplotlib inline
```

```
import matplotlib.pyplot as plt
from keras import optimizers
from keras.callbacks import ModelCheckpoint
from keras.applications.vgg16 import preprocess_input, decode_predictions
from keras.preprocessing import image
import keras.backend as K
import cv2
import sys
```

After we are done importing the required library, we can write a code to load the dataset
For the image classification, we are using CIFAR-10 dataset

To use keras in python for machine learning we import it using the given code

```
import keras
```

We are using Numpy Library to convert the List into an Array in this project

```
import numpy as np
```

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Matplotlib library is being used to plot the graph of the accuracy and loss of the model

```
import matplotlib.pyplot as plt
```

Matplotlib is a comprehensive library for **creating static, animated, and interactive visualizations in Python**.

```
batch_size = 128
```

The batch size is **a number of samples processed before the model is updated**

```
n_classes = 10 #Because CIFAR10 has 10 classes
epochs = 20
```

The number of epochs is the number of complete passes through the training dataset. The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.

After the dataset is loaded we split the dataset into Train and Test sets. This is done using the code given below

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

```
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

Training Dataset

x_train is the feature of the train dataset

y_train is the label of the train dataset

Test Dataset

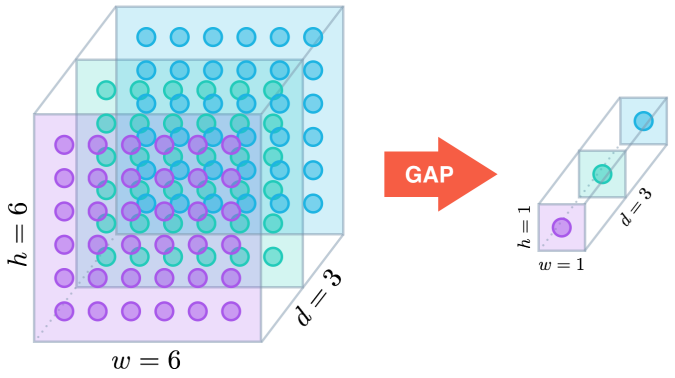
x_test is the feature of the test dataset

y_test is the label of the test dataset

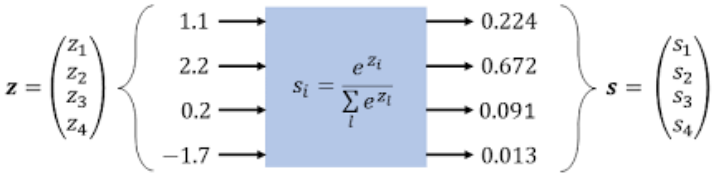
```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, n_classes)
y_test = keras.utils.to_categorical(y_test, n_classes)

print(np.mean(x_test[0]))
print(np.mean(x_train[0]))


# normalize inputs from 0-255 to 0.0-1.0
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train / 255.0
x_test = x_test / 255.0



print(np.mean(x_test[0]))
print(np.mean(x_train[0]))
```

Using the ResNet50 Architecture

```
base_model = ResNet50(input_shape=(32,32,3), weights='imagenet',
include_top=False)
```




```
x = keras.layers.GlobalAveragePooling2D()(base_model.output)
```




```
output = keras.layers.Dense(n_classes, activation='softmax')(x)
model = keras.models.Model(inputs=[base_model.input], outputs=[output])
```

Stochastic Gradient Descent (SGD)

Stochastic gradient descent is an optimization algorithm often used in machine learning applications to find the model parameters that correspond to the best fit between predicted and actual outputs

```
sgd = optimizers.SGD(lr=0.1, decay=0.0001, momentum=0.9, nesterov=True)
```

categorical_crossentropy: Used as a loss function for multi-class classification model where there are two or more output labels. The output label is assigned one-hot category encoding value in form of 0s and 1.

```
model.compile(optimizer='SGD', loss='categorical_crossentropy',
metrics=['accuracy'])
```

mode.summary() display the architecture of the model

```
model.summary()
```

```
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
import math
```

```
# learning rate schedule
def step_decay(epoch):
  initial_lrate = 0.1
  drop = 0.7
  epochs_drop = 7.0
  lrate = initial_lrate * math.pow(drop,
math.floor((1+epoch)/epochs_drop))
  return lrate



for i in range(1,26):
  print("Epoch "+str(i)+ " : ",step_decay(i))
```

This function keeps the initial learning rate for the first 26 epochs and decreases it exponentially after that.

```
from keras.callbacks import LearningRateScheduler
lrate = LearningRateScheduler(step_decay)

# checkpoint
filepath="Assignment_5.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
```

Model Training

```
model.fit(x_train, y_train, batch_size=batch_size,
validation_data=(x_test, y_test), epochs=epochs,
verbose=1,callbacks=[lrate,checkpoint])

scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

Saving the Model Weight

```
# Save the trained weights in to .h5 format
model.save_weights("ResNet_CIFAR10_Weights_Best.h5")
print("Saved model to disk")
```

Model Saving

```
# Save the trained model in to .h5 format
model.save("ResNet_CIFAR10_Model_Best.h5")
print("Saved model to disk")
```

Predicting the 10000 images

```
y_pred = model.predict(x_test, verbose = 1)
```

The maximum value along a given axis.

```
y_pred_classes = np.argmax(y_pred, axis=1)
y_pred_max_probas = np.max(y_pred, axis=1)
```

Loading the saved model

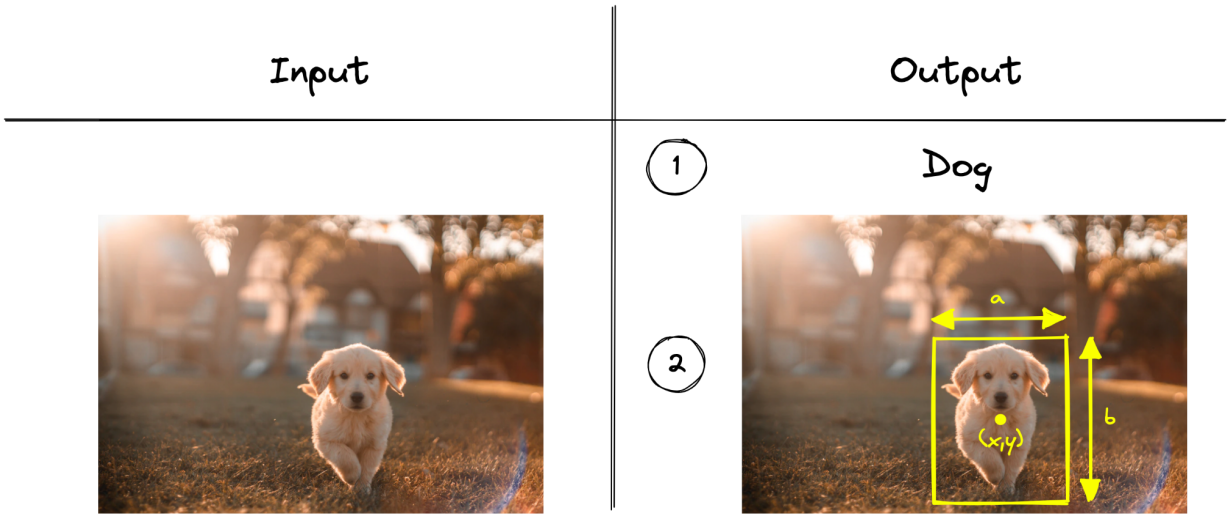
```
model = keras.models.load_model('/content/ResNet_CIFAR10_Model_Best.h5')
```

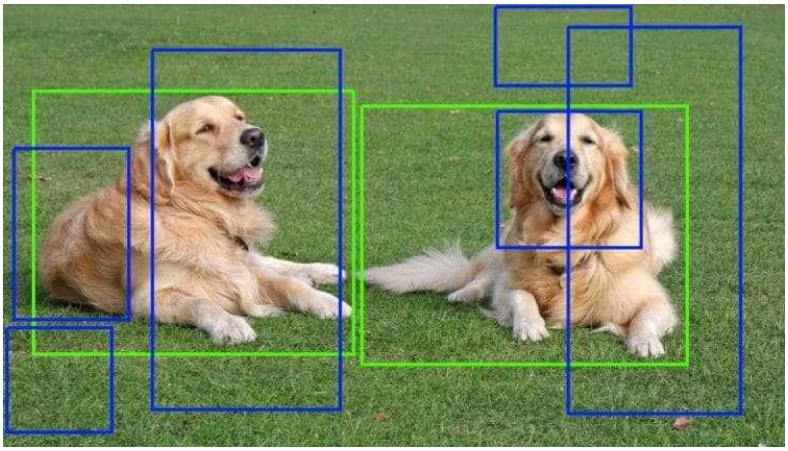
Predicting the first image

```
model.predict(np.array([x_test[0]]))
```

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

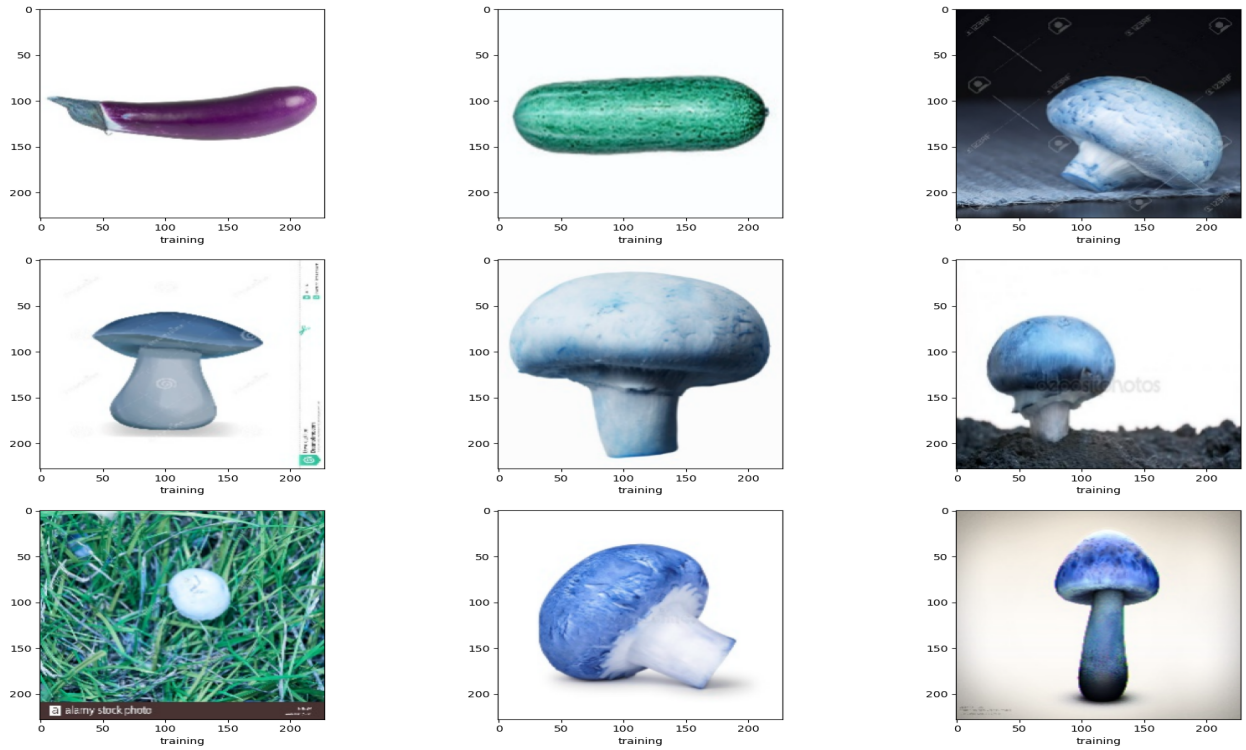
```
path = "training_images"
```

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.

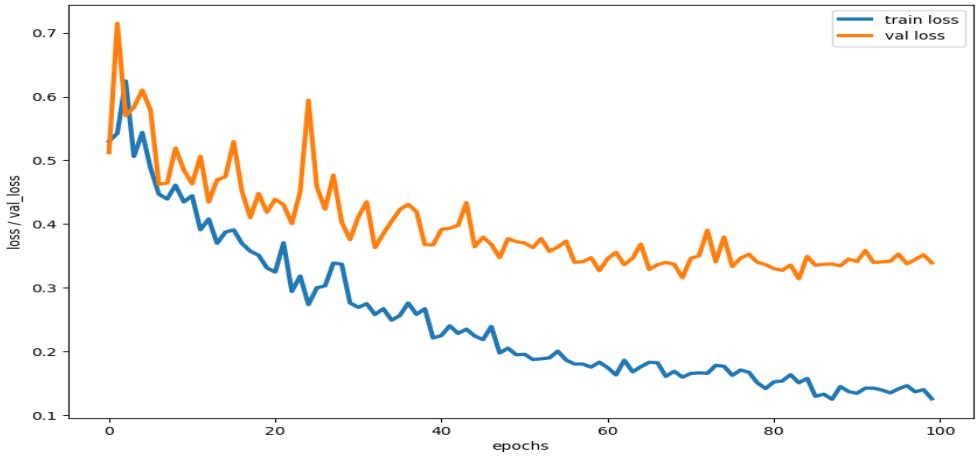
```
from tensorflow.keras.callbacks import ModelCheckpoint

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

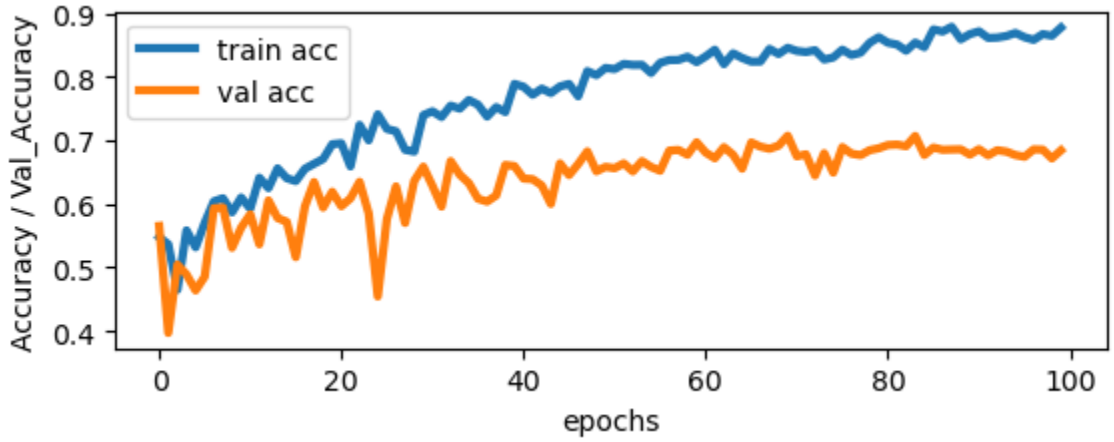
```
history = model.fit( X_train ,Y_train ,validation_data=( X_test , Y_test),
    epochs=100,
    batch_size=3)
loss = history.history['loss']
val_loss = history.history['val_loss']

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()
```



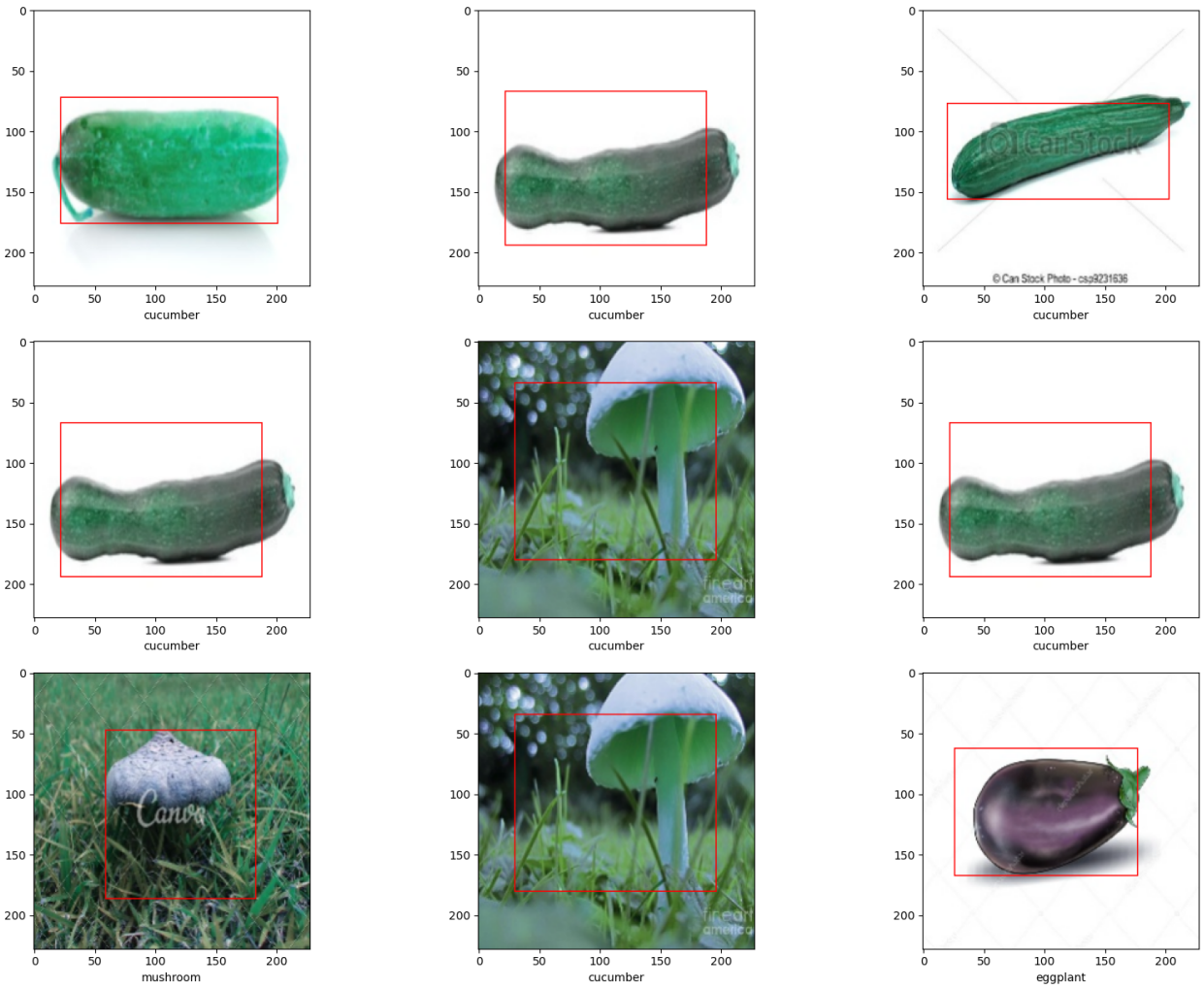

```
plt.subplot(2,1,2)
plt.plot(acc , linewidth=3 ,label='train acc')
plt.plot(val_acc , linewidth=3, label='val acc')
plt.xlabel('epochs')
plt.ylabel('Accuracy / Val_Accuracy')
plt.legend()
```



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.h5")
```