Image Classification documentation

- For the image classification modeling, we are using the TensorFlow framework

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks

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

```
import tensorflow as tf
```

To check the version of TensorFlow we use the code given below

```
# Display the version
print(tf.__version__)
```

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**.

We are using the Keras layers API and using it we are importing

`Input, Conv2D , Dense, Flatten, Dropout`

```
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
```

`Input` : this method take shape of the input dataset

`Conv2D` : 2D convolution layer (e.g. spatial convolution over images).

`Dense` : it decides the number of neurons and connects the Neural network

`Flatten` : Flattens the input. Does not affect the batch size.

`Dropout` : It deactivates some parts of the neurons, which helps prevent overfitting

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

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.

```
cifar10 = tf.keras.datasets.cifar10
```

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()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

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

Visualize The Data

Now we can visualize the data we imported from CIFAR-10 using matplotlib library

```
# visualize data by plotting images
fig, ax = plt.subplots(10, 10)
```

plt.subplot(10,10) decides the size of each image plotted
This for loop creates an image of a 5x5 matrix

```
k = 0
for i in range(5):
    for j in range(5):
        ax[i][j].imshow(x_train[k], aspect='auto')
        k += 1
plt.show()

# number of classes
```

The below code displays the number of classes in the dataset

```
K = len(set(y_train))
# calculate the total number of classes
# for the output layer
print("number of classes:", K)
```

Python code to create a model for the image classification

```
# Build the model using the functional API
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dropout(0.2)(x)
# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
```

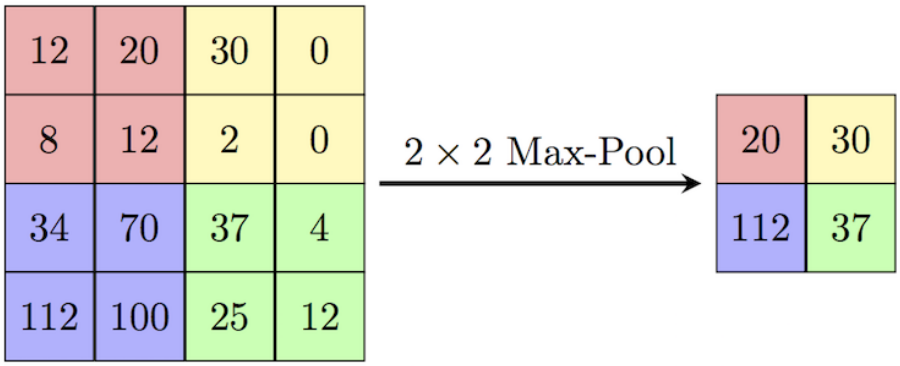
The below code is the last hidden layer that is using softmax as an activation

```
# last hidden layer i.e output layer
x = Dense(K, activation='softmax')(x)
model = Model(i, x)
```

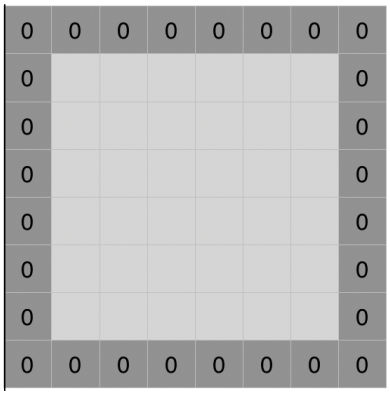
mode.summary() display the architecture of the model

```
# model description
model.summary()
```

- **Batch normalization** is a method we can use to normalize the inputs of each layer, in order to fight the internal covariate shift problem
- **Max pooling** is a type of operation that is typically added to CNNs following individual convolutional layers. When added to a model, max pooling **reduces the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer**.



- **Padding** is simply a process of adding layers of zeros to our input images so as to avoid the problems



Zero-padding added to image

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Model Training

```
r = model.fit(
  x_train, y_train, validation_data=(x_test, y_test), epochs=50)
```

An epoch is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model.

We have a limited number of datasets, so we can increase the dataset amount by rotating, moving, shifting zooming the same image and creating multiple variants of it

Now we can check how our model performs after the Train, it can be determined by plotting the graph of accuracy and val_accuracy.

```
# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc', color='red')
plt.plot(r.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
```

Testing with the labels displayed

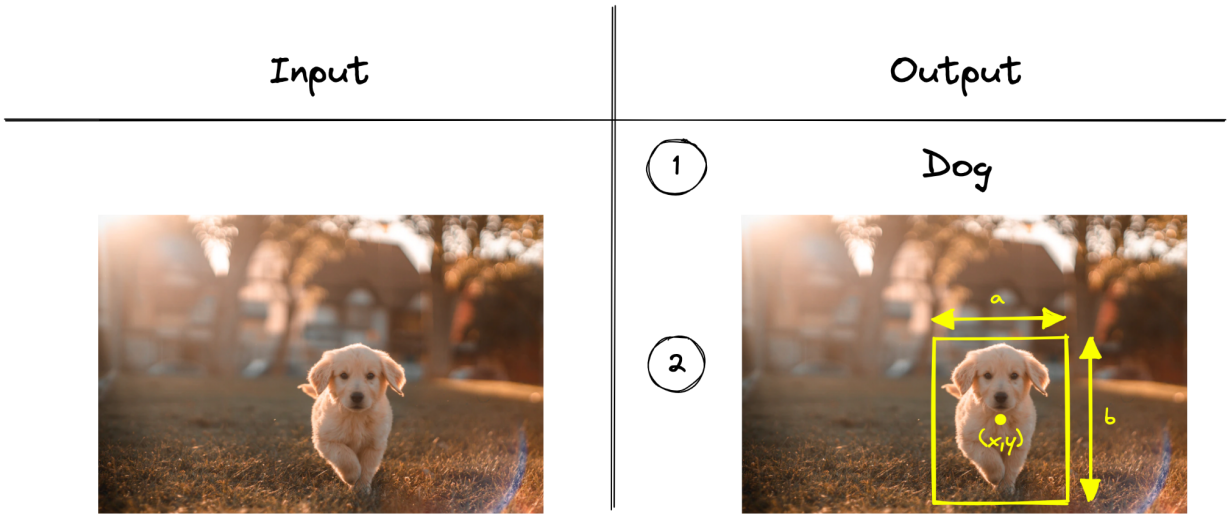
```
# label mapping
labels = '''airplane automobile bird cat deerdog frog horseship
truck'''.split()
# select the image from our test dataset
image_number = 0
# display the image
plt.imshow(x_test[image_number])
# load the image in an array
n = np.array(x_test[image_number])
# reshape it
p = n.reshape(1, 32, 32, 3)
# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]
# load the original label
original_label = labels[y_test[image_number]]
# display the result
print("Original label is {} and predicted label is {}".format(
    original_label, predicted_label))
```

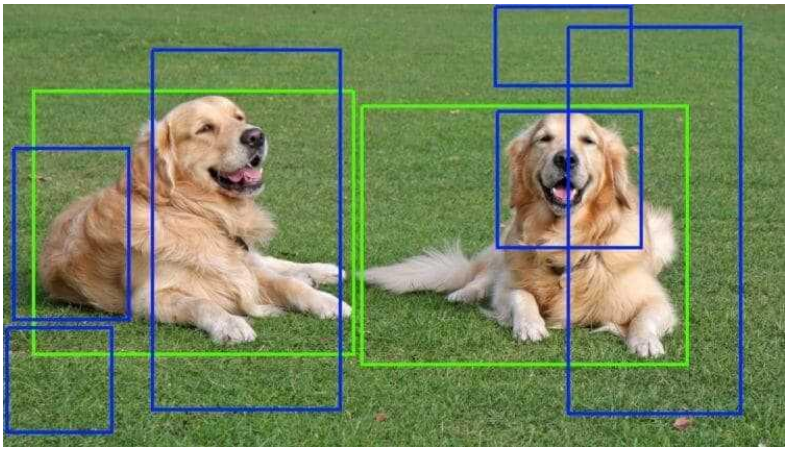
Save the Model with .h5 extension

```
# save the model
model.save('imgclassification.h5')
```

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

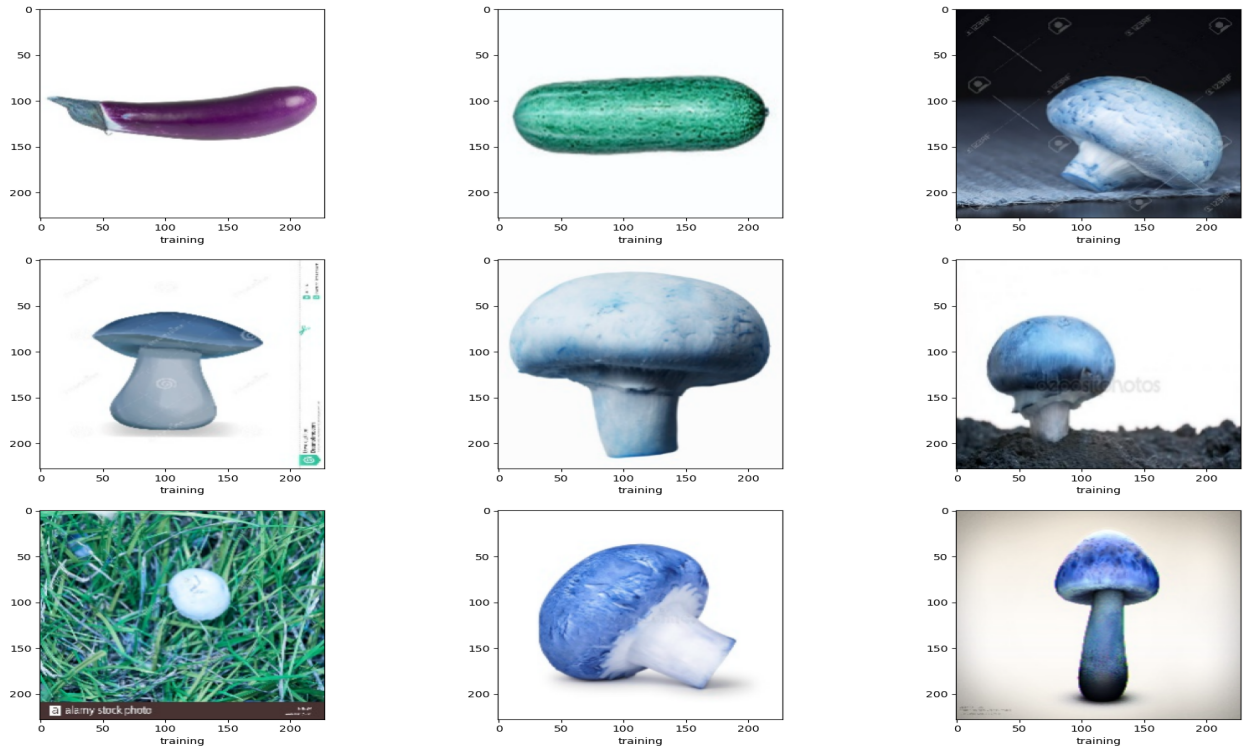
```
path = "training_images"
```

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.

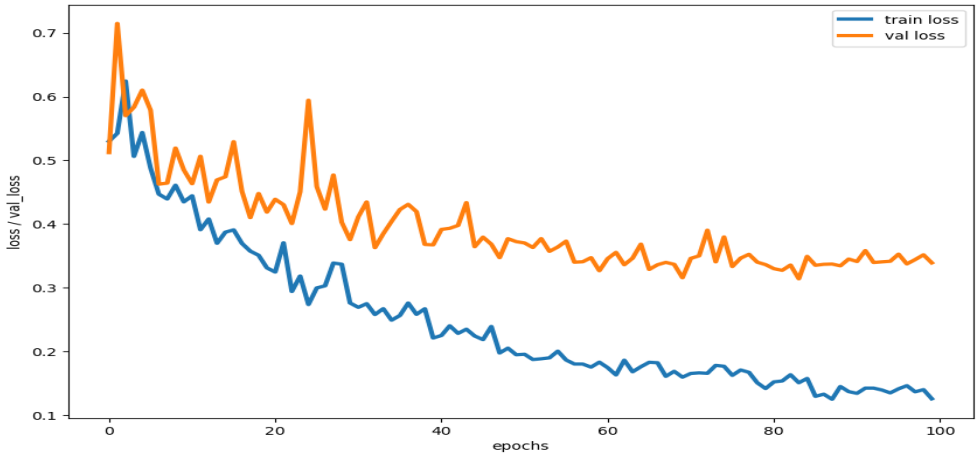
```
from tensorflow.keras.callbacks import ModelCheckpoint

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

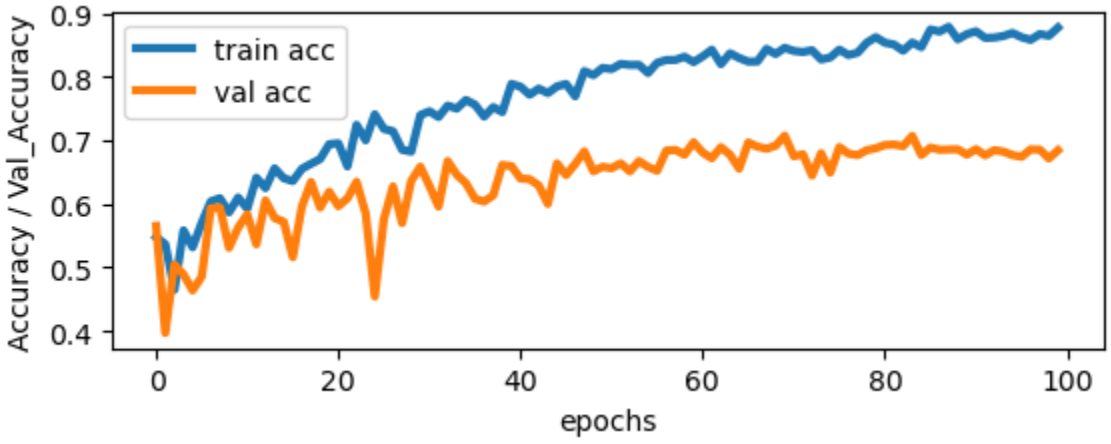
```
history = model.fit( X_train ,Y_train ,validation_data=( X_test , Y_test),
    epochs=100,
    batch_size=3)
loss = history.history['loss']
val_loss = history.history['val_loss']

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()
```



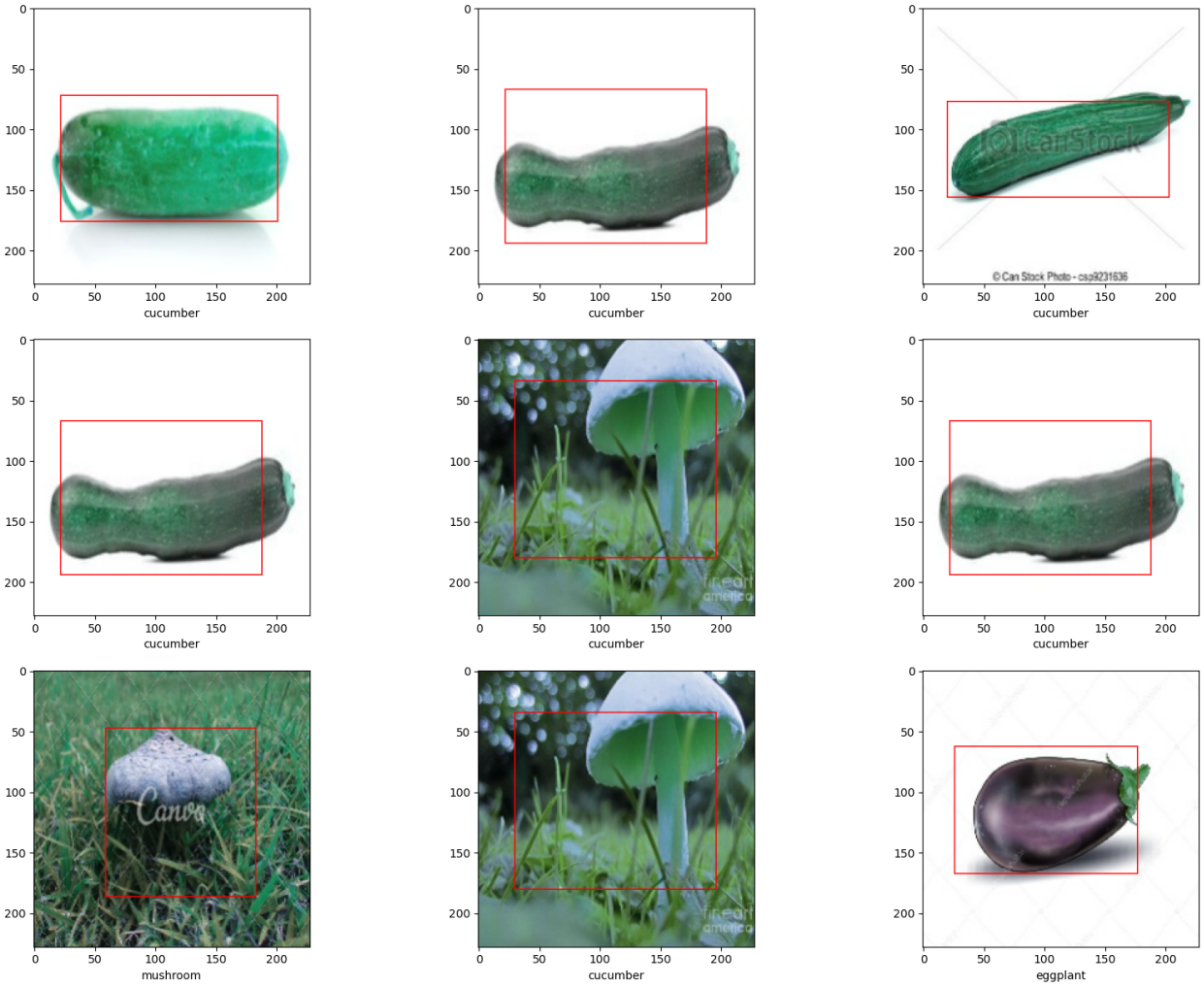

```
plt.subplot(2,1,2)
plt.plot(acc , linewidth=3 ,label='train acc')
plt.plot(val_acc , linewidth=3, label='val acc')
plt.xlabel('epochs')
plt.ylabel('Accuracy / Val_Accuracy')
plt.legend()
```



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")