create model folders

```
from google.colab import drive
drive.mount('/content/drive')

import os

# Path to your project folder in Drive
drive_project_dir = '/content/drive/MyDrive/DL-CNN'

# Create the project folder and common subfolders
os.makedirs(drive_project_dir, exist_ok=True)
os.makedirs(os.path.join(drive_project_dir, 'dataset'), exist_ok=True)
os.makedirs(os.path.join(drive_project_dir, 'models'), exist_ok=True)
os.makedirs(os.path.join(drive_project_dir, 'outputs'), exist_ok=True)
```

```
Mounted at /content/drive
```

```
#saving the model
model.save('/content/drive/MyDrive/DL-CNN/models/my_trained_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file
```

downloading the dataset

```
# Download Pascal VOC 2007 Train/Val set (about 450MB)
!wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar

# Extract the archive
!tar -xf VOCtrainval_06-Nov-2007.tar

# The data is now in the folder: ./VOCdevkit/VOC2007/
```

```
--2025-04-28 13:46:49--  http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
Resolving host.robots.ox.ac.uk (host.robots.ox.ac.uk)... 129.67.94.152
Connecting to host.robots.ox.ac.uk (host.robots.ox.ac.uk)|129.67.94.152|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 460032000 (439M) [application/x-tar]
Saving to: ‘VOCtrainval_06-Nov-2007.tar’

VOCtrainval_06-Nov- 100%[===================>] 438.72M  29.0MB/s    in 15s

2025-04-28 13:47:04 (29.3 MB/s) - ‘VOCtrainval_06-Nov-2007.tar’ saved [460032000/460032000]
```

EXRACTING

```
import os

base_dir = './VOCdevkit/VOC2007'
print("Image directory:", os.path.join(base_dir, 'JPEGImages'))
print("Annotation directory:", os.path.join(base_dir, 'Annotations'))
print("ImageSets directory (splits):", os.path.join(base_dir, 'ImageSets/Main'))

# List a few files in each directory
print("Sample images:", os.listdir(os.path.join(base_dir, 'JPEGImages'))[:5])
print("Sample annotations:", os.listdir(os.path.join(base_dir, 'Annotations'))[:5])
print("Sample image sets:", os.listdir(os.path.join(base_dir, 'ImageSets/Main'))[:5])
```

```
Image directory: ./VOCdevkit/VOC2007/JPEGImages
Annotation directory: ./VOCdevkit/VOC2007/Annotations
ImageSets directory (splits): ./VOCdevkit/VOC2007/ImageSets/Main
Sample images: ['006618.jpg', '009421.jpg', '000194.jpg', '009780.jpg', '002334.jpg']
Sample annotations: ['004274.xml', '009687.xml', '007029.xml', '004570.xml', '007481.xml']
Sample image sets: ['motorbike_train.txt', 'bus_train.txt', 'bicycle_trainval.txt', 'bus_val.txt', 'sheep_val.txt']
```

VISUALIZING

```
import xml.etree.ElementTree as ET
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt

img_dir = os.path.join(base_dir, 'JPEGImages')
ann_dir = os.path.join(base_dir, 'Annotations')
```

```
# Choose a sample annotation file
sample_ann = os.listdir(ann_dir)[0]
tree = ET.parse(os.path.join(ann_dir, sample_ann))
root = tree.getroot()
img_filename = root.find('filename').text

# Open the image
img_path = os.path.join(img_dir, img_filename)
image = Image.open(img_path)
draw = ImageDraw.Draw(image)

# Draw all bounding boxes
for obj in root.findall('object'):
    bbox = obj.find('bndbox')
    xmin = int(bbox.find('xmin').text)
    ymin = int(bbox.find('ymin').text)
    xmax = int(bbox.find('xmax').text)
    ymax = int(bbox.find('ymax').text)
    label = obj.find('name').text
    draw.rectangle([xmin, ymin, xmax, ymax], outline='red', width=2)
    draw.text((xmin, ymin), label, fill='red')

plt.figure(figsize=(7, 7))
plt.imshow(image)
plt.axis('off')
plt.title(f"Image: {img_filename}")
plt.show()
```

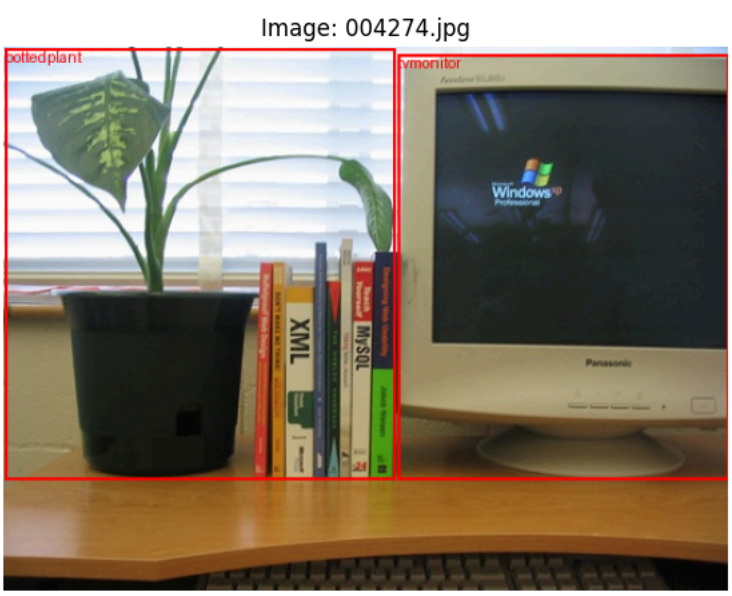


IMAGE PREPROCESSING "RESIZING & *italicized text*"

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

def preprocess_image(img_path, target_size=(128, 128)):
    img = load_img(img_path, target_size=target_size)
    img_arr = img_to_array(img) / 255.0  # Normalize to [0,1]
    return img_arr

# Example usage
sample_img = preprocess_image(img_path)
print("Processed image shape:", sample_img.shape)
```

```
Processed image shape: (128, 128, 3)
```

EXPLORING

```
### **Step 1: Parse All Annotations**
import os
import xml.etree.ElementTree as ET
import pandas as pd

img_dir = './VOCdevkit/VOC2007/JPEGImages'
```

```
ann_dir = './VOCdevkit/VOC2007/Annotations'

records = []

for ann_file in os.listdir(ann_dir):
    tree = ET.parse(os.path.join(ann_dir, ann_file))
    root = tree.getroot()
    img_filename = root.find('filename').text
    img_path = os.path.join(img_dir, img_filename)

    for obj in root.findall('object'):
        label = obj.find('name').text
        bbox = obj.find('bndbox')
        xmin = int(bbox.find('xmin').text)
        ymin = int(bbox.find('ymin').text)
        xmax = int(bbox.find('xmax').text)
        ymax = int(bbox.find('ymax').text)
        records.append([img_path, label, xmin, ymin, xmax, ymax])

# Create a DataFrame for easy handling
df = pd.DataFrame(records, columns=['image', 'label', 'xmin', 'ymin', 'xmax', 'ymax'])
print(df.head())
print("Total samples:", len(df))
```

```
                                    image        label  xmin  ymin  xmax  \
0  ./VOCdevkit/VOC2007/JPEGImages/004274.jpg    tvmonitor   272     5   500
1  ./VOCdevkit/VOC2007/JPEGImages/004274.jpg  pottedplant     1     1   270
2  ./VOCdevkit/VOC2007/JPEGImages/009687.jpg       person   262    60   500
3  ./VOCdevkit/VOC2007/JPEGImages/009687.jpg       person     1    75   278
4  ./VOCdevkit/VOC2007/JPEGImages/009687.jpg       bottle   405   165   500

   ymax
0   298
1   298
2   375
3   375
4   273
Total samples: 15662
```

OFFICIAL VALIDATION

```
import os

split_dir = './VOCdevkit/VOC2007/ImageSets/Main'
print("Split files available:", os.listdir(split_dir))
```

```
Split files available: ['motorbike_train.txt', 'bus_train.txt', 'bicycle_trainval.txt', 'bus_val.txt', 'sheep_val.txt',
```

SPKITING

```
import os
import xml.etree.ElementTree as ET
import pandas as pd

# Paths
img_dir = './VOCdevkit/VOC2007/JPEGImages'
ann_dir = './VOCdevkit/VOC2007/Annotations'
split_dir = './VOCdevkit/VOC2007/ImageSets/Main'

# Parse annotations
records = []
for ann_file in os.listdir(ann_dir):
    tree = ET.parse(os.path.join(ann_dir, ann_file))
    root = tree.getroot()
    img_filename = root.find('filename').text
    img_path = os.path.join(img_dir, img_filename)

    for obj in root.findall('object'):
        label = obj.find('name').text
        bbox = obj.find('bndbox')
        xmin = int(bbox.find('xmin').text)
        ymin = int(bbox.find('ymin').text)
        xmax = int(bbox.find('xmax').text)
        ymax = int(bbox.find('ymax').text)
        records.append([img_path, label, xmin, ymin, xmax, ymax])

# Create DataFrame
df = pd.DataFrame(records, columns=['image', 'label', 'xmin', 'ymin', 'xmax', 'ymax'])
```

```
# Function to get image ID
def get_image_id(path):
    return os.path.splitext(os.path.basename(path))[0]

df['image_id'] = df['image'].apply(get_image_id)

# Read train and val splits
with open(os.path.join(split_dir, 'train.txt')) as f:
    train_ids = set(line.strip() for line in f.readlines())
with open(os.path.join(split_dir, 'val.txt')) as f:
    val_ids = set(line.strip() for line in f.readlines())

# Assign split
df['split'] = df['image_id'].apply(
    lambda x: 'train' if x in train_ids else ('val' if x in val_ids else 'unknown')
)

# Show results
print(df['split'].value_counts())
print(df.head())
```

```
split
train    7844
val      7818
Name: count, dtype: int64
                                    image        label  xmin  ymin  xmax  \
0  ./VOCdevkit/VOC2007/JPEGImages/004274.jpg    tvmonitor   272     5   500
1  ./VOCdevkit/VOC2007/JPEGImages/004274.jpg  pottedplant     1     1   270
2  ./VOCdevkit/VOC2007/JPEGImages/009687.jpg       person   262    60   500
3  ./VOCdevkit/VOC2007/JPEGImages/009687.jpg       person     1    75   278
4  ./VOCdevkit/VOC2007/JPEGImages/009687.jpg       bottle   405   165   500

   ymax image_id split
0   298   004274   val
1   298   004274   val
2   375   009687   val
3   375   009687   val
4   273   009687   val
```

Data Augmentation & Data Loader Preparation

```
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array

def preprocess_image_and_bbox(img_path, bbox, target_size=(128, 128)):
    # Load and resize image
    img = load_img(img_path, target_size=target_size)
    img_arr = img_to_array(img) / 255.0

    # Get original image size for bbox scaling
    orig_img = Image.open(img_path)
    orig_w, orig_h = orig_img.size
    # Scale bbox
    xmin, ymin, xmax, ymax = bbox
    scale_x = target_size[0] / orig_w
    scale_y = target_size[1] / orig_h
    xmin = int(xmin * scale_x)
    xmax = int(xmax * scale_x)
    ymin = int(ymin * scale_y)
    ymax = int(ymax * scale_y)
    bbox_scaled = [xmin, ymin, xmax, ymax]

    return img_arr, bbox_scaled

# Example: Load a batch of training samples
sample_df = df[df['split'] == 'train'].sample(4)
batch_images = []
batch_bboxes = []
batch_labels = []

for _, row in sample_df.iterrows():
    img_arr, bbox = preprocess_image_and_bbox(row['image'], [row['xmin'], row['ymin'], row['xmax'], row['ymax']])
    batch_images.append(img_arr)
    batch_bboxes.append(bbox)
    batch_labels.append(row['label'])  # You can one-hot encode labels if needed

batch_images = np.array(batch_images)
batch_bboxes = np.array(batch_bboxes)

print("Batch images shape:", batch_images.shape)
```

```
print("Batch bboxes shape:", batch_bboxes.shape)
print("Batch labels:", batch_labels)
```

```
Batch images shape: (4, 128, 128, 3)
Batch bboxes shape: (4, 4)
Batch labels: ['boat', 'chair', 'sofa', 'sheep']
```

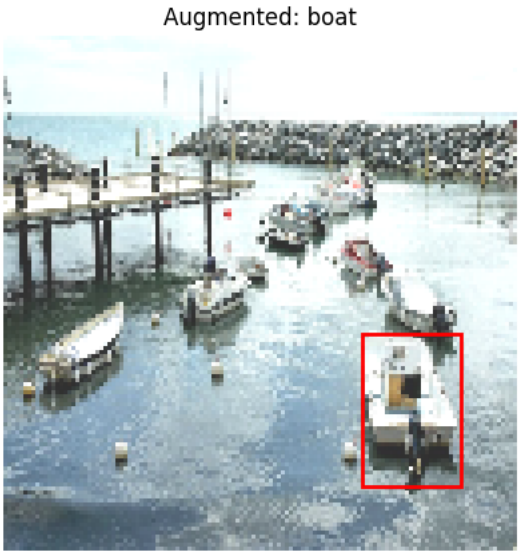
Data Augmentation for Robustness

Data augmentation makes your model more robust and less likely to overfit. For object detection, you should use augmentations that transform both the image and the bounding boxes in sync.

```
!pip install albumentations --quiet
```

```
#Augmenting Images and Bounding Boxes with Albumentations
import albumentations as A
from albumentations.pytorch import ToTensorV2

# Define an augmentation pipeline
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.05, rotate_limit=15, p=0.5),
    A.Resize(128, 128),
],
    bbox_params=A.BboxParams(format='pascal_voc', label_fields=['category_ids'])
)

# Example function for a single image + bbox + label
def augment_image(img_path, bbox, label):
    image = np.array(Image.open(img_path).convert("RGB"))
    # Albumentations expects bboxes as a list of [xmin, ymin, xmax, ymax]
    augmented = transform(image=image, bboxes=[bbox], category_ids=[label])
    aug_img = augmented['image']
    aug_bbox = augmented['bboxes'][0]
    aug_label = augmented['category_ids'][0]
    return aug_img, aug_bbox, aug_label

# Let's try it for one sample
sample = sample_df.iloc[0]
img_path = sample['image']
bbox = [sample['xmin'], sample['ymin'], sample['xmax'], sample['ymax']]
label = sample['label']

# Map label to integer for augmentation (required by Albumentations)
class2idx = {label: idx for idx, label in enumerate(df['label'].unique())}
label_idx = class2idx[label]

aug_img, aug_bbox, aug_label = augment_image(img_path, bbox, label_idx)
print("Augmented bbox:", aug_bbox)

# Visualize the augmented image and bbox
aug_img_vis = (aug_img * 255).astype(np.uint8) if aug_img.max() <= 1.0 else aug_img.astype(np.uint8)
plt.figure(figsize=(5,5))
plt.imshow(aug_img_vis)
plt.gca().add_patch(
    plt.Rectangle(
        (aug_bbox[0], aug_bbox[1]),
        aug_bbox[2] - aug_bbox[0],
        aug_bbox[3] - aug_bbox[1],
        fill=False, color='red', linewidth=2
    )
)
plt.title(f"Augmented: {list(class2idx.keys())[list(class2idx.values()).index(aug_label)]}")
plt.axis('off')
plt.show()
```

```
Augmented bbox: [89.08799743652344, 73.63855743408203, 113.66400146484375, 111.42168426513672]
/usr/local/lib/python3.11/dist-packages/albumentations/core/validation.py:87: UserWarning: ShiftScaleRotate is a special
  original_init(self, **validated_kwargs)
```



Create a Custom Data Generator with Augmentation

```
import numpy as np
import albumentations as A
from tensorflow.keras.utils import to_categorical

class PascalVOCAugmentGenerator:
    def __init__(self, df, batch_size, classes, augment=True, shuffle=True):
        self.df = df.reset_index(drop=True)
        self.batch_size = batch_size
        self.classes = classes
        self.augment = augment
        self.shuffle = shuffle

        self.class2idx = {c: i for i, c in enumerate(classes)}
        self.transform = A.Compose([
            A.HorizontalFlip(p=0.5),
            A.RandomBrightnessContrast(p=0.2),
            A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.05, rotate_limit=15, p=0.5),
            A.Resize(128, 128),
        ], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['category_ids']))

    def __len__(self):
        return int(np.ceil(len(self.df) / self.batch_size))

    def __getitem__(self, idx):
        batch_df = self.df.iloc[idx*self.batch_size : (idx+1)*self.batch_size]
        batch_images, batch_bboxes, batch_labels = [], [], []

        for _, row in batch_df.iterrows():
            image = np.array(Image.open(row['image']).convert("RGB"))
            bbox = [row['xmin'], row['ymin'], row['xmax'], row['ymax']]
            label = self.class2idx[row['label']]

            if self.augment:
                transformed = self.transform(image=image, bboxes=[bbox], category_ids=[label])
                image = transformed['image']
                bbox = transformed['bboxes'][0]
                label = transformed['category_ids'][0]

            batch_images.append(image / 255.0 if image.max() > 1.0 else image)  # Normalize
            batch_bboxes.append(bbox)
            batch_labels.append(to_categorical(label, num_classes=len(self.classes)))

        return np.stack(batch_images), np.array(batch_bboxes), np.stack(batch_labels)

    def on_epoch_end(self):
        if self.shuffle:
            self.df = self.df.sample(frac=1).reset_index(drop=True)
```

```
# List of all unique classes
all_classes = sorted(df['label'].unique())

# Split dataframes
train_df = df[df['split'] == 'train']
val_df = df[df['split'] == 'val']

# Instantiate generators
train_gen = PascalVOCAugmentGenerator(train_df, batch_size=16, classes=all_classes, augment=True, shuffle=True)
val_gen = PascalVOCAugmentGenerator(val_df, batch_size=16, classes=all_classes, augment=False, shuffle=False)

# Example: Get a batch from train_gen
batch_imgs, batch_bboxes, batch_labels = train_gen.__getitem__(0)
print("Batch images shape:", batch_imgs.shape)
print("Batch bboxes shape:", batch_bboxes.shape)
print("Batch labels shape (one-hot):", batch_labels.shape)
```

Build a Multi-Task CNN (Classification + Bounding Box Regression)

```
!pip install albumentations --quiet
```

```
!tar -xf VOCtrainval_06-Nov-2007.tar
```

```
import os

print("JPEGImages directory exists:", os.path.exists('./VOCdevkit/VOC2007/JPEGImages'))
print("Annotations directory exists:", os.path.exists('./VOCdevkit/VOC2007/Annotations'))
print("ImageSets/Main directory exists:", os.path.exists('./VOCdevkit/VOC2007/ImageSets/Main'))
```

```
JPEGImages directory exists: True
Annotations directory exists: True
ImageSets/Main directory exists: True
```

```
!pip install tensorflow --quiet
# 1. Imports & Setup
import os
import xml.etree.ElementTree as ET
import numpy as np
import pandas as pd
from PIL import Image
import albumentations as A
from tensorflow.keras.utils import to_categorical, Sequence
import tensorflow as tf
from tensorflow.keras import layers, Model, Input

# 2. Parse Pascal VOC Annotations to DataFrame
img_dir = './VOCdevkit/VOC2007/JPEGImages'
ann_dir = './VOCdevkit/VOC2007/Annotations'
split_dir = './VOCdevkit/VOC2007/ImageSets/Main'

records = []
for ann_file in os.listdir(ann_dir):
    tree = ET.parse(os.path.join(ann_dir, ann_file))
    root = tree.getroot()
    img_filename = root.find('filename').text
    img_path = os.path.join(img_dir, img_filename)
    for obj in root.findall('object'):
        label = obj.find('name').text
        bbox = obj.find('bndbox')
        xmin = int(bbox.find('xmin').text)
        ymin = int(bbox.find('ymin').text)
        xmax = int(bbox.find('xmax').text)
        ymax = int(bbox.find('ymax').text)
        records.append([img_path, label, xmin, ymin, xmax, ymax])
df = pd.DataFrame(records, columns=['image', 'label', 'xmin', 'ymin', 'xmax', 'ymax'])

def get_image_id(path):
    return os.path.splitext(os.path.basename(path))[0]
df['image_id'] = df['image'].apply(get_image_id)

with open(os.path.join(split_dir, 'train.txt')) as f:
    train_ids = set(line.strip() for line in f.readlines())
with open(os.path.join(split_dir, 'val.txt')) as f:
    val_ids = set(line.strip() for line in f.readlines())
```

```
df['split'] = df['image_id'].apply(lambda x: 'train' if x in train_ids else ('val' if x in val_ids else 'unknown'))

# 3. Albumentations Data Generator (robust to empty bbox)
class PascalVOCAugmentGenerator(Sequence):
    def __init__(self, df, batch_size, classes, augment=True, shuffle=True):
        self.df = df.reset_index(drop=True)
        self.batch_size = batch_size
        self.classes = classes
        self.augment = augment
        self.shuffle = shuffle
        self.class2idx = {c: i for i, c in enumerate(classes)}
        self.transform = A.Compose([
            A.HorizontalFlip(p=0.5),
            A.RandomBrightnessContrast(p=0.2),
            A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.05, rotate_limit=15, p=0.5),
            A.Resize(128, 128),
        ], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['category_ids']))

    def __len__(self):
        return int(np.ceil(len(self.df) / self.batch_size))

    def __getitem__(self, idx):
        batch_df = self.df.iloc[idx*self.batch_size : (idx+1)*self.batch_size]
        batch_images, batch_bboxes, batch_labels = [], [], []
        for _, row in batch_df.iterrows():
            image = np.array(Image.open(row['image']).convert("RGB"))
            bbox = [row['xmin'], row['ymin'], row['xmax'], row['ymax']]
            label = self.class2idx[row['label']]
            keep = False
            tries = 0
            # Try to get a valid augmentation (non-empty bbox) up to 3 times
            while not keep and tries < 3:
                if self.augment:
                    transformed = self.transform(image=image, bboxes=[bbox], category_ids=[label])
                    if transformed['bboxes']:
                        image = transformed['image']
                        bbox = transformed['bboxes'][0]
                        label = transformed['category_ids'][0]
                        keep = True
                    else:
                        tries += 1
                else:
                    keep = True
            # If fails 3 times, skip this sample
            if not keep:
                continue
            batch_images.append(image / 255.0 if image.max() > 1.0 else image)
            batch_bboxes.append(bbox)
            batch_labels.append(to_categorical(label, num_classes=len(self.classes)))
        # If batch is empty, repeat previous batch
        if len(batch_images) == 0:
            return self.__getitem__((idx - 1) % self.__len__())
        return np.stack(batch_images), {'class_output': np.stack(batch_labels), 'bbox_output': np.array(batch_bboxes)}

    def on_epoch_end(self):
        if self.shuffle:
            self.df = self.df.sample(frac=1).reset_index(drop=True)


#keep alive
```

Prevent Colab from disconnecting (run this in its own cell)

```
#@title Prevent Colab from disconnecting (run this in its own cell)
%%javascript
function ClickConnect(){
  console.log("Clicking reconnect button");
  document.querySelector("colab-toolbar-button#connect").click()
}
setInterval(ClickConnect, 60000)
```

MAYBE

```
#bring important libraries

# 4. Prepare Generators
```

```
all_classes = sorted(df['label'].unique())
train_df = df[df['split'] == 'train']
val_df = df[df['split'] == 'val']
train_gen = PascalVOCAugmentGenerator(train_df, batch_size=16, classes=all_classes, augment=True, shuffle=True)
val_gen = PascalVOCAugmentGenerator(val_df, batch_size=16, classes=all_classes, augment=False, shuffle=False)

# 5. Build Multi-Task CNN Model
input_img = Input(shape=(128, 128, 3))
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2))(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
class_output = layers.Dense(len(all_classes), activation='softmax', name='class_output')(x)
bbox_output = layers.Dense(4, activation='linear', name='bbox_output')(x)
model = Model(inputs=input_img, outputs=[class_output, bbox_output])
model.compile(optimizer='adam',
              loss={'class_output': 'categorical_crossentropy', 'bbox_output': 'mse'},
              metrics={'class_output': 'accuracy'})
model.summary()

# 6. Train Model
history = model.fit(
    train_gen,
    epochs=10,
    validation_data=val_gen
)
def __getitem__(self, idx):
    ...
    if len(batch_images) == 0:
        print(f"!! Empty batch at idx {idx}, retrying previous.")
        return self.__getitem__((idx - 1) % self.__len__())
    # Debug: print all shapes before stacking
    for i, im in enumerate(batch_images):
        if im.shape != (128, 128, 3):
            print(f"!! Batch {idx} image {i} shape {im.shape}")
    for i, bbox in enumerate(batch_bboxes):
        if len(bbox) != 4:
            print(f"!! Batch {idx} bbox {i} wrong length: {bbox}")
    return np.stack(batch_images), {'class_output': np.stack(batch_labels), 'bbox_output': np.array(batch_bboxes)}
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_3 (Conv2D)	(None, 128, 128, 32)	896	input_layer_1[0]…
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 64)	18,496	max_pooling2d_3[…
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 64)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 128)	73,856	max_pooling2d_4[…
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 128)	0	conv2d_5[0][0]
flatten_1 (Flatten)	(None, 32768)	0	max_pooling2d_5[…
dense_1 (Dense)	(None, 256)	8,388,864	flatten_1[0][0]
class_output (Dense)	(None, 20)	5,140	dense_1[0][0]
bbox_output (Dense)	(None, 4)	1,028	dense_1[0][0]

Total params: 8,488,280 (32.38 MB)
Trainable params: 8,488,280 (32.38 MB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 367s 742ms/step - bbox_output_loss: 1457.0222 - class_output_accuracy: 0.2137 - class_outpu
Epoch 2/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 361s 735ms/step - bbox_output_loss: 1026.1390 - class_output_accuracy: 0.2759 - class_outpu
Epoch 3/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 340s 693ms/step - bbox_output_loss: 972.7927 - class_output_accuracy: 0.2431 - class_output
Epoch 4/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 352s 718ms/step - bbox_output_loss: 997.6590 - class_output_accuracy: 0.2762 - class_output
Epoch 5/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 363s 740ms/step - bbox_output_loss: 988.2253 - class_output_accuracy: 0.2860 - class_output
Epoch 6/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 379s 773ms/step - bbox_output_loss: 956.0208 - class_output_accuracy: 0.2663 - class_output
Epoch 7/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 429s 873ms/step - bbox_output_loss: 938.0563 - class_output_accuracy: 0.3049 - class_output
Epoch 8/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 357s 728ms/step - bbox_output_loss: 932.4114 - class_output_accuracy: 0.2946 - class_output
Epoch 9/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 352s 717ms/step - bbox_output_loss: 933.9796 - class_output_accuracy: 0.2897 - class_output
Epoch 10/10
491/491 ━━━━━━━━━━━━━━━━━━━━ 0s 540ms/step - bbox_output_loss: 908.5253 - class_output_accuracy: 0.3172 - class_output_l
```

saving the model

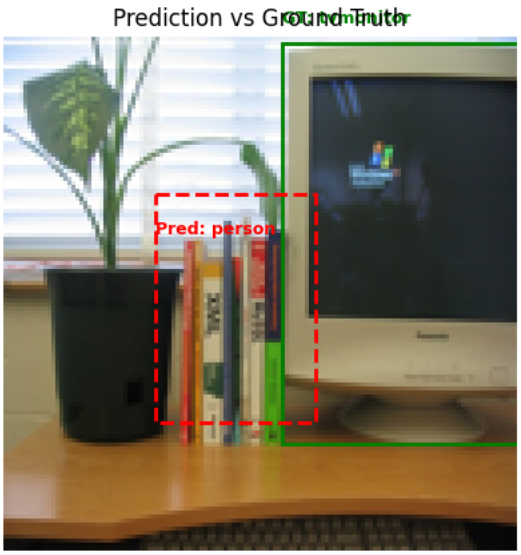
```
model.save('/content/drive/MyDrive/DL-CNN/models/my_final_model.h5')
```

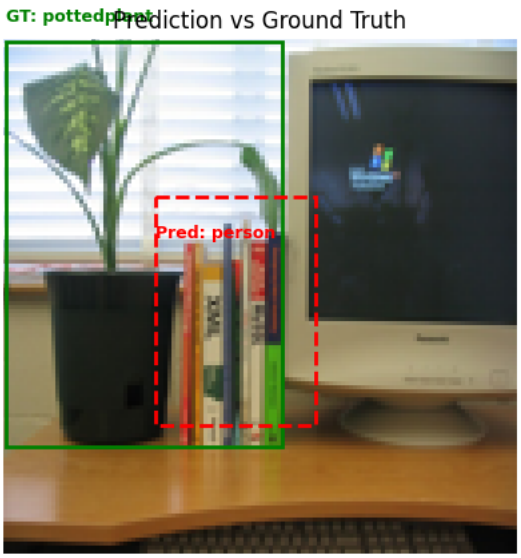
Visualize Model Predictions (Predicted vs. Ground Truth Bounding Boxes & Classes)

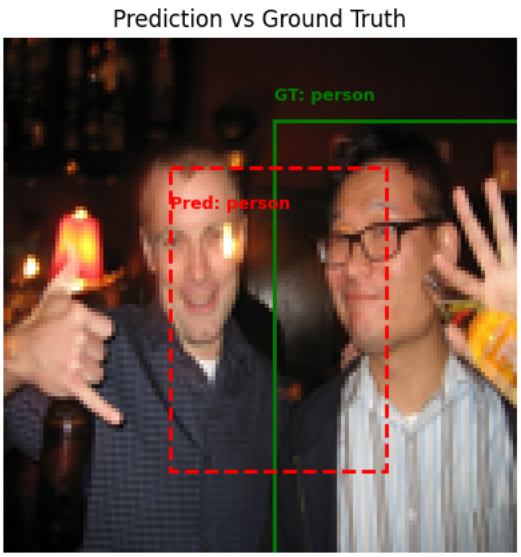
```
import matplotlib.pyplot as plt
import numpy as np

def plot_predictions(model, generator, class_names, batch_index=0, num_images=5):
    batch_imgs, targets = generator.__getitem__(batch_index)
    y_true_cls = np.argmax(targets['class_output'], axis=1)
    y_true_bbox = targets['bbox_output']
    preds = model.predict(batch_imgs)
    y_pred_cls = np.argmax(preds[0], axis=1)
    y_pred_bbox = preds[1]

    for i in range(min(num_images, batch_imgs.shape[0])):
        img = batch_imgs[i]
        gt_bbox = y_true_bbox[i]
        pred_bbox = y_pred_bbox[i]
        gt_label = class_names[y_true_cls[i]]
```

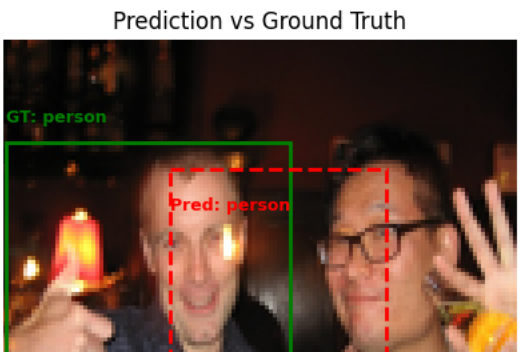
```
            pred_label = class_names[y_pred_cls[i]]

            plt.figure(figsize=(5,5))
            plt.imshow(np.clip(img, 0, 1))
            ax = plt.gca()
            # Ground truth box (green)
            ax.add_patch(plt.Rectangle((gt_bbox[0], gt_bbox[1]),
                                       gt_bbox[2]-gt_bbox[0], gt_bbox[3]-gt_bbox[1],
                                       fill=False, edgecolor='green', linewidth=2, label='GT'))
            ax.text(gt_bbox[0], gt_bbox[1]-5, f"GT: {gt_label}", color='green', fontsize=9, weight='bold')
            # Predicted box (red)
            ax.add_patch(plt.Rectangle((pred_bbox[0], pred_bbox[1]),
                                       pred_bbox[2]-pred_bbox[0], pred_bbox[3]-pred_bbox[1],
                                       fill=False, edgecolor='red', linewidth=2, linestyle='--', label='Pred'))
            ax.text(pred_bbox[0], pred_bbox[1]+10, f"Pred: {pred_label}", color='red', fontsize=9, weight='bold')
            plt.axis('off')
            plt.title(f"Prediction vs Ground Truth")
            plt.show()

    # Usage example (try batch_index=0 or another index if needed)
    plot_predictions(model, val_gen, all_classes, batch_index=0, num_images=5)
```

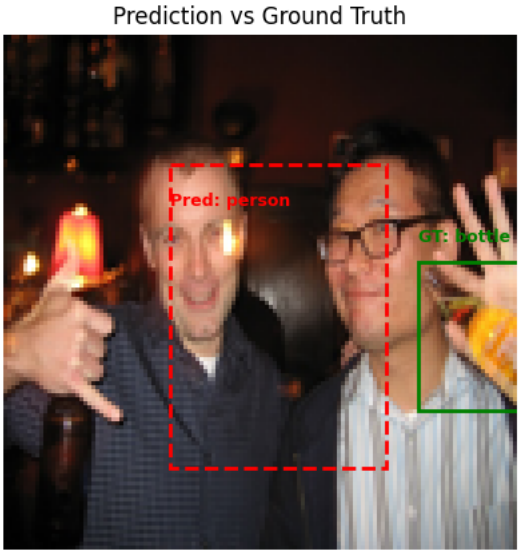
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 192ms/step
```











Plot Training and Validation Loss/Accuracy Curves

```
import matplotlib.pyplot as plt

# Plot classification accuracy
plt.figure(figsize=(10,5))
plt.plot(history.history['class_output_accuracy'], label='Train Acc')
plt.plot(history.history['val_class_output_accuracy'], label='Val Acc')
plt.xlabel('Epoch')
plt.ylabel('Classification Accuracy')
plt.legend()
plt.title('Classification Accuracy over Epochs')
plt.show()

# Plot loss (for both heads and total loss)
plt.figure(figsize=(10,5))
plt.plot(history.history['loss'], label='Train Total Loss')
plt.plot(history.history['val_loss'], label='Val Total Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Total Loss over Epochs')
plt.show()
```