Bounding box detection - Racoon data

Data files

- images_racoon.rar: contain images of racoons
- train_labels.cv: contains coordinates for bounding box for every image

Import the necessary libraries

```
import tensorflow as tf
import csv
import numpy as np
from PIL import Image
import pandas as pd
from keras import Model
from keras.applications.mobilenet import MobileNet, preprocess_input
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, Callback
from keras.layers import Conv2D, Reshape
from keras.utils import Sequence
from keras.backend import epsilon
```

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

Change directory

```
path = '/content/drive/My Drive/Lab/Lab9'
```

Load the training data from train.csv file

```
train=pd.read_csv(path+"/train_labels.csv")
```

Print the shape of the train dataset

```
train.head()
```

```
train.shape
```

Declare a variable IMAGE_SIZE = 128 as we will be using MobileNet which will be * 128

```
IMAGE_SIZE = 128
TRAIN_CSV = path+"/train_labels.csv"
```

With the help of csv.reader write a for loop which can load the train.csv file and s x0,y0,x1,y1 in induvidual variables.

1. Create a list variable known as 'path' which has all the path for all the training images
2. Create an array 'coords' which has the resized coordinates of the bounding box for the training images

Note: All the training images should be downsampled to 128 * 128 as it is the input shape of MobileN Object detection). Hence the corresponding coordinates of the bounding boxes should be changed to 128 * 128

```
!pip install patool
import patoolib
patoolib.extract_archive("/content/drive/My Drive/Lab/Lab9/images_racoon.rar")
```

```
import csv
with open(TRAIN_CSV, 'r') as csvfile:
    paths = []
    coords = np.zeros((sum(1 for line in csvfile), 4))
    reader = csv.reader(csvfile, delimiter=',')
    csvfile.seek(0)
    next(reader)

    for col, row in enumerate(reader):

        for i, r in enumerate(row[1:7]): # Parse row with seven entities
            row[i+1] = r

        path, image_width, image_height,_, x0, y0, x1, y1, = row
        path = "images/" + path
        coords[col, 0] = int(x0) * IMAGE_SIZE / int(image_width) # Normalize bounding box by
        coords[col, 1] = int(y0) * IMAGE_SIZE / int(image_height) # Normalize bounding box by
        coords[col, 2] = (int(x1) - int(x0)) * IMAGE_SIZE / int(image_width) # Normalize boun
        coords[col, 3] = (int(y1) - int(y0)) * IMAGE_SIZE / int(image_height)
        paths.append(path)
```

```
paths
```

Write a for loop which can load all the training images into a variable 'batch_imag the 'paths' variable

Note: Convert the image to RGB scale as the MobileNet accepts 3 channels as inputs

```
batch_images = np.zeros((len(paths), IMAGE_SIZE, IMAGE_SIZE, 3), dtype=np.float32)
for i, f in enumerate(paths):
    img = Image.open(f) # Read image
    img = img.resize((IMAGE_SIZE, IMAGE_SIZE)) # Resize image
    img = img.convert('RGB')
    batch_images[i] = preprocess_input(np.array(img, dtype=np.float32))


batch_images.shape
```

Import MobileNet and load MobileNet into a variable named 'model' which takes 3. Freeze all the layers. Add convolution and reshape layers at the end to ensure t

```
ALPHA = 1.0 # Width hyper parameter for MobileNet (0.25, 0.5, 0.75, 1.0). Higher width means

IMAGE_SIZE = 128 # MobileNet takes images of size 128*128*3

EPOCHS = 10 # Number of epochs. I got decent performance with just 5.
BATCH_SIZE = 32 # Depends on your GPU or CPU RAM.


model = MobileNet(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3), include_top=False, alpha=ALPHA) #
# Do not include classification (top) layer

# to freeze layers, except the new top layer, of course, which will be added below
for layer in model.layers:
    layer.trainable = False

# Add new top layer which is a conv layer of the same size as the previous layer so that only
x = model.layers[-1].output
x = Conv2D(4, kernel_size=4, name="coords")(x)
# In the line above kernel size should be 3 for img size 96, 4 for img size 128, 5 for img si
x = Reshape((4,))(x) # These are the 4 predicted coordinates of one BBox


model = Model(inputs=model.input, outputs=x)


model.summary()
```

Define a custom loss function IoU which calculates Intersection Over Union

```
def loss(gt,pred):
    intersections = 0
    unions = 0
    diff_width = np.minimum(gt[:,0] + gt[:,2], pred[:,0] + pred[:,2]) - np.maximum(gt[:,0], p
    diff_height = np.minimum(gt[:,1] + gt[:,3], pred[:,1] + pred[:,3]) - np.maximum(gt[:,1],
    intersection = diff_width * diff_height

    # Compute union
    area_gt = gt[:,2] * gt[:,3]
    area_pred = pred[:,2] * pred[:,3]
    union = area_gt + area_pred - intersection
```

```
#     Compute intersection and union over multiple boxes
    for j, _ in enumerate(union):
        if union[j] > 0 and intersection[j] > 0 and union[j] >= intersection[j]:
            intersections += intersection[j]
            unions += union[j]

    # Compute IOU. Use epsilon to prevent division by zero
    iou = np.round(intersections / (unions + epsilon()), 4)
    iou = iou.astype(np.float32)
    return iou

def IoU(y_true, y_pred):
    iou = tf.py_func(loss, [y_true, y_pred], tf.float32)
    return iou
```

Write model.compile function & model.fit function with:

1. Optimizer = Adam, Loss = 'mse' and metrics = IoU
2. Epochs = 30, batch_size = 32, verbose = 1

```
coords=coords[0:173]


model.compile(optimizer='Adam', loss='mse', metrics=[IoU]) # Regression loss is MSE

#checkpoint = ModelCheckpoint("model-{val_iou:.2f}.h5", verbose=1, save_best_only=True,
#                             save_weights_only=True, mode="max", period=1) # Checkpoint bes
#stop = EarlyStopping(monitor="val_iou", patience=PATIENCE, mode="max") # Stop early, if the
#reduce_lr = ReduceLROnPlateau(monitor="val_iou", factor=0.2, patience=10, min_lr=1e-7, verbc
# Reduce learning rate if Validation IOU does not improve

model.fit(batch_images,coords,
            epochs=30,batch_size = 32,
            verbose=1)
```

```
batch_images.shape
```

Pick a test image from the given data

```
test_img="/content/images/raccoon-112.jpg"
```

```
import cv2
unscaled = cv2.imread(test_img)
```

Resize the image to 128 * 128 and preprocess the image for the MobileNet mode

```
unscaled.shape
```

```
image_height, image_width, _ = unscaled.shape
image = cv2.resize(unscaled, (IMAGE_SIZE, IMAGE_SIZE)) # Rescaled image to run the network
feat_scaled = preprocess_input(np.array(image, dtype=np.float32))
```

Predict the coordinates of the bounding box for the given test image

```
region = model.predict(x=np.array([feat_scaled]))[0]
```

Plot the test image using .imshow and draw a boundary box around the image wi obtained from the model

```
x0 = int(region[0] * image_width / IMAGE_SIZE) # Scale the BBox
y0 = int(region[1] * image_height / IMAGE_SIZE)

x1 = int((region[2]) * image_width / IMAGE_SIZE)
y1 = int((region[3]) * image_height / IMAGE_SIZE)


import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import numpy as np


# Create figure and axes
fig,ax = plt.subplots(1)

# Display the image
ax.imshow(unscaled)

# Create a Rectangle patch
rect = patches.Rectangle((x0, y0), (x1 - x0) , (y1 - y0) , linewidth=2, edgecolor='r', facecc

# Add the patch to the Axes
ax.add_patch(rect)

plt.show()
```

Time Series Prediction using LSTM

Download Data

Link: https://datamarket.com/data/set/2324/daily-minimum-temperatures-in-melbourne-australia-19

Description

Daily minimum temperatures in Melbourne, Australia, 1981-1990

Units: Degrees Celcius

Steps before loading

- Rename the column name with temprature values to "Temprature"
- In the last, there is one extra row in the data, remove it by opening the file and save it again.
- There are some values in Temprature column which have a "?" before them, they will give error, r the file
- If you don't want to do these steps, just load the data file given by Great Learning.

Mount google drive

Change your present working directory

```
path="/content/drive/My Drive/Lab/Lab9/"
```

Load your data file

```
data=pd.read_csv(path+"daily-minimum-temperatures-in-me.csv")
```

Plot data

```
data.plot()
```

Descibe your dataframe

```
data.describe()
```

Check for null values

```
data.isnull().any()
```

Drop null values

```
data.shape
```

```
data.dropna(axis=0, how='any',inplace=True)
```

```
data.shape
```

Get the representation of the distribution of data in the form of histogram

```
data.hist()
```

Check the maximum and minimum values

```
data.max()
```

```
data.min()
```

Normalize the data

```
data=data.drop("Date",axis=1)
```

```
data.head()
```

```
type(data)
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler(feature_range=(0,1))
scaled=scale.fit_transform(data)
```

Check the maximum and minimum values of scaled data

```
print(scaled.max())
print(scaled.min())
```

Look into some of the scaled values

```
scaled[23:30]
```

Split data into Training and Testing

```
train_size = int(len(scaled) * 0.70)
test_size = len(scaled - train_size)
```

Print train and test size

```
train, test = scaled[0:train_size, :], scaled[train_size: len(scaled), :]
print('train: {}\ntest: {}'.format(len(train), len(test)))
```

Create the sequential data

Map the temprature at a particular time t to the temprature at time t+n, where n is any number you de

For example: to map tempratures of consecutive days, use t+1, i.e. loop_back = 1

Define your function to create dataset

```
def create_dataset(dataset, window=1):

    dataX, dataY = [], []

    for i in range(len(dataset)-window):

        a = dataset[i:(i+window), 0]
        dataX.append(a)
        dataY.append(dataset[i + window, 0])

    return np.array(dataX), np.array(dataY)
```

Use function to get training and test set

```
window_size = 1
X_train, y_train = create_dataset(train, window_size)
X_test, y_test = create_dataset(test, window_size)
```

Transform the prepared train and test input data into the expected structure using numpy.

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
print(X_train.shape)
print(X_test.shape)
```

```
print(X_test.shape)
```

Define Model

Define sequntial model, add LSTM layer and compile the model

```
tf.keras.backend.clear_session()
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(32, input_shape=(window_size, 1)))
model.add(tf.keras.layers.Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Summarize your model

```
model.summary()
```

Train the model

```
model.fit(X_train, y_train, epochs=200, validation_data=(X_test, y_test), batch_size=40)
```

Make Predictions and Evaluate your model

```
trainPredict = model.predict(X_train)
```

```
trainPredict = model.predict(X_train)
testPredict = model.predict(X_test)
```

```
trainPredict = model.predict(X_train)
testPredict = model.predict(X_test)
trainPredict = scale.inverse_transform(trainPredict)
testPredict = scale.inverse_transform(testPredict)
```

Plot the results

```
trainPredictPlot = np.empty_like(scaled)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[window_size:len(trainPredict)+window_size, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = np.empty_like(scaled)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(window_size*2):len(scaled), :] = testPredict
# plot baseline and predictions
plt.figure(figsize=(20,10))
plt.plot(scale.inverse_transform(scaled))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```