Tools and Libraries

Python Modules and Libraries

NumPy	|import numpy as np
OpenCV	|import cv2
Matplotlib	|import matplotlib.pyplot as plt
Pickle	|import pickle
XLRD	|import xlrd
Keras	|import keras
XLWT	|import xlwt
PyDrive	|For importing and exporting files between Google Drive and Google Colaboratory

Tools

Jupyter Notebook
Google Colab
Excel

Approach

Data

We convert each **Image** from coloured to grayscale and resize from **(640, 480, 3)** to **(100, 100, 1).**
For this we used OpenCV.

So we create a 4D Tensor of Training Images **(12000, 100, 100, 1),** Training Labels **(12000, 4)** and 4D Tensor of Validation Images **(2000, 100, 100, 1),** Validation Labels **(2000, 4)**

The labels given to us in '*training.csv*' were – **x1, x2, y1, y2** which we converted to – **x, y w, h**

Where -

x = (x2 + x1)/2	|X-Coordinate of Centre of Bounding Box
y = (y2 + y1)/2	|Y-Coordinate of Centre of Bounding Box
w = (x2 – x1)	|Width of Bounding Box
h = (y2 – y1)	|Height of Bounding Box

We then multiply these label values by a factor to compensate for the '*resize*' operation that we do over Images -

sx	|After resize x-dimension (100 in our case)
sy	|After resize y-dimension (100 in our case)
fx = sx/640	|Factor to get the Coordinates for 'sx' size of x-dimension
fy = sy/480	|Factor to get the Coordinates for 'sy' size of y-dimension
fx' = (1/sx)*fx	|Factor to Standardize the labels between 0 and 1
fy' = (1/sy)*fy	|Factor to Standardize the labels between 0 and 1
x = fx'*x	|x-coordinate of bounding-box centre (will use in label)
y = fy'*y	|y-coordinate of bounding-box centre (will use in label)
w = fx'*w	|width of bounding-box centre (will use in label)
h = fy'*h	|height of bounding-box centre (will use in label)

We then divide the Image **matrix** (of size 100x100x1) obtained, by **255. (Standardization)** So that the matrix range is limited between **0** and **1**.

We used **Data Augmentation** to increase our dataset size, reduce overfitting and increase accuracy.

Model

We used '*keras*' to build our model. We used the Functional API this time.

Techniques involved -

Batch Normalization	
Residual Layers	|We also use Residual Layers in our model
Regularization	|We use 'Dropout'
Sub-Sampling	|We use 'Average-Pooling'
Optimization	|We use 'Adam'

How to Run the Code (source_submit.ipynb/.py)

Code is available in two formats, '**.ipynb**' and '.**py**'

One will have to first import the necessary libraries.

We have provided 4 variables - *'x_train', 'y_train', 'x_valid', 'y_valid'*

'x_train' and 'x_valid' are 4D Tensors 'y_train' and 'y_valid' are 2D Tensors

All of them are initialized with '0'

Now to train the model we have to feed it with actual 12000 images in the form of x_train (pre-processed) and 2000 images in x_valid and their respective labels in y_train and y_labels

Kindly refer Approach->Data to know how to make the image labels

It is upon the evaluator to choose the dataset and save them as in the above format in those 4 variables.

Then run the Training Phase and train the model.

But one can also skip the training phase and load the model that we trained and then evaluate it on their own set of dataset

One can use **predict_640x480x3(image, model)**

In which we input two arguments – Original 640x480x3 image and the trained model to show the image with coordinates marked on them.

Google Drive Links –

d2c_data.tar - https://drive.google.com/open?id=1RVoKzP6IeulTmuLRg6cgsoVEr3RcJAe-
model_0028.h5 - https://drive.google.com/open?id=1Ed81aWjZ_tH_CdjbC7j2RddalnuWYpmh

'd2c_data.tar' contains - *x_train.pickle*, *y_train.pickle*, *x_valid.pickle*, *y_valid.pickle*
'model_0028.h5 – contains the weights of our model (obtained after training)

Special Mentions

Google Colaboratory |https://colab.research.google.com/

We used the freely available GPU provided by Google Colaboratory Research Project to execute all our training work.