AUTOMATIC NUMBER PLATE RECOGNITION

Aditi Adhikary *(26/07/2022)*

Dataset used: - "**Indian vehicle license plate dataset**"
[Consists of – Google images, State-wise vehicle images, Video Images]

Source: - https://www.kaggle.com/datasets/saisirishan/indian-vehicle-dataset?resource=download

Download via: - Download_kaggle_dataset.ipynb [username and key required via Kaggle account]

Files: -

I. _200_Object_Detection_indian-vehicle-dataset-Copy1 (.ipynb file)
II. Make Predictions-Copy1 (.ipynb file)
III. xml_to_csv5.py
IV. ./google_images directory [For training purpose, under the *Indian Vehicle Dataset*]
V. convert_image_type_code.py [Additional, for pre-processing: If required]
VI. Download_kaggle_dataset.ipynb (.ipynb file)

PROJECT ARCHITECTURE: -

1. Labelling the images and obtaining the *.xml files*.
2. Parsing Information from XML
3. Parsing data from XML and Converting it into CSV
4. Verify the Data
5. Data Processing
6. Build and Train Transfer Learning
7. Make Bounding Box Prediction
8. OCR [Extract Characters]

I. Labelling the images: -

To label images, we use the LabelImg image annotation tool. Download the labelImg from GitHub and follow the instruction to install the package. (https://github.com/tzutalin/labelImg)

1. Build and launch using the instructions above.
2. Click 'Change default saved annotation folder' in Menu/File
3. Click 'Open Dir'
4. Click 'Create RectBox'
5. Click and release left mouse to select a region to annotate the rect box
6. You can use right mouse to drag the rect box to copy or move it.
7. Save the output (Ctrl +S) in XML.
8. Save all of them under a common label such as "*numberplate*", etc.

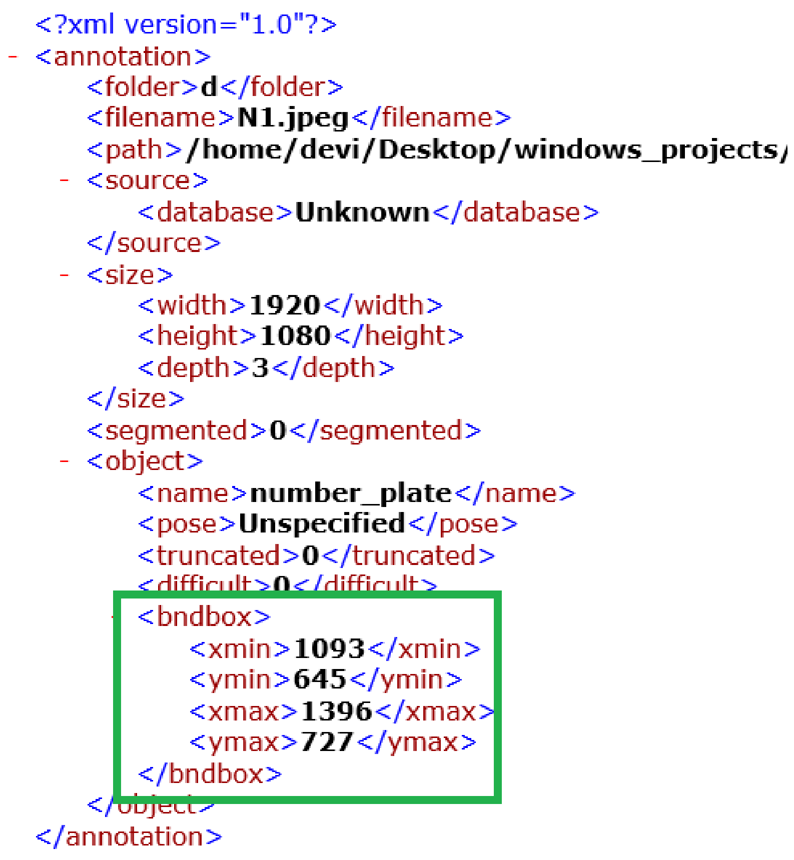
The annotation will be saved to the folder you specify. The labelling process has a direct impact on the accuracy of the model.

Hotkeys: -

w	Create a rect box
d	Next image
a	Previous image
del	Delete the selected rect box
Ctrl++	Zoom in
Ctrl--	Zoom out
↑→↓←	Keyboard arrows to move selected rect box

II. Parsing Information from XML: -

We will take the useful information from the label which are the diagonal points of the rectangle box or **bounding box** which are *xmin, ymin, xmax, ymax* respectively as shown in **figure**. This is available in XML. So, we need to extract the information and save it in any convenient format



```
  <?xml version="1.0"?>
- <annotation>
      <folder>d</folder>
      <filename>N1.jpeg</filename>
      <path>/home/devi/Desktop/windows_projects/
    - <source>
          <database>Unknown</database>
      </source>
    - <size>
          <width>1920</width>
          <height>1080</height>
          <depth>3</depth>
      </size>
      <segmented>0</segmented>
    - <object>
          <name>number_plate</name>
          <pose>Unspecified</pose>
          <truncated>0</truncated>
          <difficult>0</difficult>
        - <bndbox>
              <xmin>1093</xmin>
              <ymin>645</ymin>
              <xmax>1396</xmax>
              <ymax>727</ymax>
          </bndbox>
      </object>
  </annotation>
```

III. Parsing data from XML and Converting it into CSV: -

`[ Code: - xml_to_csv5.py ]`

Here, we use **xml.etree** python library to parse the data from XML and also import pandas and glob. Using glob we first get all the XML files that are produced during labelling.

```
path = glob('./[NAME_OF_IMAGE_DIRECTORY]/*.xml')
```

In the code, we individually take each file and parse into xml.etree and find the object -> **bndbox** which is in line 2 to 7.

Then we extract '*xmin,xmax,ymin,ymax*' and saved those values in the dictionary. After that we convert it into a pandas data frame and save that into a CSV file (*labels.csv*).

With the above code, we extract the diagonal position of each image and convert the data from an unstructured to a structured format.

IV. Verify the Data: -

To verify whether the information we got is valid or not. For that just verify the bounding box is appearing properly for a given image. Here I consider the image at file_path = image_path[0] and the corresponding diagonal position can found in **df.**

```
cv2.rectangle(img,(140,210),(334,252),(0,255,0),3)
cv2.namedWindow('example',cv2.WINDOW_NORMAL)
cv2.imshow('example',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here, the values of (xmin, xmax),(ymin, ymax) in cv2.rectangle are pre-specified beforehand for verification of Output purpose.

V. Data Pre-processing: -

In this process we will take each and every image and convert it into an array using **OpenCV** and resize the image into 224 x 224 which is the standard compatible size of the pre-trained **transfer learning model.**

1. After that, we normalize the image just by dividing with maximum number as we know that the maximum number for an 8-bit image is

$$2^8 - 1 = 255$$

2. That the reason we will divide our image 255.0. The way of diving an array with the maximum value is called **Normalization** (Min-Max Scaler).
3. We also need to normalize our labels too. Because for the deep learning model the output range should be between 0 to 1.
4. For normalizing labels, we need to divide the diagonal points with the width and height of the image. And finally values in a python list.

We will convert the list into an array using Numpy, and then split the data into training and testing set using **sklearn**.

```
          x_train,x_test,y_train,y_test =
train_test_split(X,y,train_size=0.8,random_state=0)
```

VI. Build and Train Transfer Learning: -

Here we will use the **InceptionResNetV2** model with pre-trained weights and train this to our data.

The model we are building is an object detection model and the number of output expected from the is model -> 4, which are the diagonal points.

Hence we add an *embedding neural network layer* to the transfer learning model.

- Now we compile the model and train the model.
- Save the model with the desired name via model.save().

#It is preferable to create a small function for plotting metrics.: -

Using *matplotlib*, we plot a graph of loss vs Epochs and observe the values of: -

1) Training loss (**train_loss**), and,

2) Validation loss (**val_loss**).

#We can also observe: -

#**model.evaluate():** That is for evaluating the already trained model using the validation (or test) data and the corresponding labels.

#Returns the loss value and metrics values for the model.

[model.evaluate(x_test,y_test)]

- Lower the loss, better the model generated.

VII. Make Bounding Box Prediction: -

In this step, we use the model generated to make predictions on images that were unseen.

1. We 1st load the model via *tf.keras.models.load_model()*
2. Test predictions on a single image from the given dataset.
3. We also include a preprocessing step while loading the image [via 'ImageEnhance' from the PIL library]
4. We then create a model pipeline with a user-defined function (called as *object_detection*) in order to combine the previous two steps.
5. Once we have done with the Object Detection model training process, then using this model we will crop the image which contains the license plate which is also called the region of interest (ROI) [that is a NumPy array]

VIII. OCR : -

1. Import the Pytesseract library.
2. Additional we may apply: - grayscale, Otsu's threshold, invert, processing specifically to the region of interest.
3. Pass the ROI to Optical Character Recognition API Tesseract in Python (**PyTesseract**). that will extract text from the region of interest

References: -

1. https://medium.com/codex/7-steps-to-build-automatic-number-plate-recognition-in-python-53e34c9ae583
2. https://github.com/heartexlabs/labelImg

3. https://cloudxlab.com/blog/number-plate-reader/
4. https://www.analyticsvidhya.com/blog/2021/04/how-to-download-kaggle-datasets-using-jupyter-notebook/
5. https://pythonexamples.org/python-pillow-adjust-image-sharpness/
6. https://www.geeksforgeeks.org/python-opencv-cv2-rectangle-method/
7. https://www.kaggle.com/datasets/saisirishan/indian-vehicle-dataset?resource=download