**Object Detection using YoloV3:**

YOLO, short for You Only Look Once, is a popular State of the Art Model for Object Detection. The algorithm is a one stage approach that uses convolutional neural network to predict bounding boxes coordinates on image dataset. The outputs include class type and confidence score.

There are several versions­ of Yolo and the latest version is YoloV4 available on [pjreddie](https://github.com/pjreddie/darknet#darknet) and [AlexeyAB](https://github.com/AlexeyAB/darknet) github repositories. For this project, we have used the keras-yolov3 version from Huynh Ngoc Anh available at the following link:

[experiencor/keras-yolo3: Training and Detecting Objects with YOLO3 (github.com)](https://github.com/experiencor/keras-yolo3)

This implementation is based on darknet architecture available from above official repositories. We have also referred the following blog article to get a basic understanding of the yolo model and its usage: [How to Perform Object Detection With YOLOv3 in Keras (machinelearningmastery.com)](https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/)

There are several steps in using the yolov3-model for pneumonia detection.

1. Clone the repository
2. Install the dependencies
3. Detect a test image
4. Custom object detection for pneumonia use case
   1. Pneumonia dataset preparation – images and annotations
   2. Edit the configuration file
   3. Generate anchors for the pneumonia dataset
   4. Train data on pneumonia dataset
   5. Perform detection using trained weights
5. Observations and next steps

The above steps are detailed below:

1. Clone the repository:

We cloned both the repositories to experiment with the functionality. However, we used the experiencor version for subsequent processing as the configuration process for custom object training is relatively simpler.

# !git clone https://github.com/pjreddie/darknet­­

# !git clone https://github.com/experiencor/keras-yolo3.git

The above commands create a folder structure such as shown below:

|  |  |
| --- | --- |
| A picture containing graphical user interface  Description automatically generated | Graphical user interface, application  Description automatically generated |

1. Install the dependencies:

This is an important step as the yolov3 implementation code in python has references to functionality from these libraries. The tensorflow(1.15), keras(2.3.1), hdf5(2.10.0) and other libraries need to be at the required versions in order for the code to work.

!pip install -r requirements.txt

It would be to good to create

A virtual environment can be created to install the dependencies so that we can keep the ongoing development related to this program separate from other programs which may have other dependencies.

1. Detect a test image:

Before training yolov3 network on the pneumonia custom dataset, it is important to check whether the installation and dependencies have completed properly. We can do this by predicting the result for a test image.

Before detecting a test image, we need to provide name scope by adding prefix tensorflow to all the import statements where keras libraries are mentioned. Once this is done, we test the yolov3 model on a test image named dog.jpg.

!python yolo3\_one\_file\_to\_detect\_them\_all.py -w '/content/drive/MyDrive/Project Datasets/Capstone Project/darknet/yolov3.weights' -i '/content/drive/MyDrive/Project Datasets/Capstone Project/darknet/darknet/data/dog.jpg'

The above code uses the default yolov3 weights available from darknet repository and the dog image available as part of data directory.

This provides the following output:

bicycle: 99.34676885604858%

dog: 98.6376941204071%

truck: 92.89804697036743%

Then, we use matplotlib and cv2 to display the image and bounding boxes.

Graphical user interface

Description automatically generated

1. Custom object detection for pneumonia dataset
2. Data Preparation –

This step includes creation of separation folders for train and validation images and their corresponding annotations. If the validation folder is empty, then training set will be automatically splitted in the ratio of 0.8.

For this project, we used only a subset of images for training purpose considering memory requirements and speed of execution. Dicom images were preprocessed separately and converted to jpg images of size 256 x 256 and used as the input. The following code copies subset of such images into the training\_images folder

# Code to copy a subset of images from source folder to train\_images\_present

import shutil, os

source\_dir = "/content/drive/MyDrive/Project Datasets/Capstone Project/jpg\_reshaped\_train/present/"

dest\_dir = "/content/drive/MyDrive/Project Datasets/Capstone Project/train\_images\_present/"

files = os.listdir(source\_dir)[:500]

for f in files:

    shutil.copy(source\_dir + f, dest\_dir)

Once images are available, the next step is to create annotations in VOC format. Pascal-voc-writer was used to create annotations in VOC format. Another key input to generate annotations includes the bounding box coordinates for the train image set. This data was preprocessed separately and kept ready in a pickled file which was loaded and stored in a dataframe. A brief stub was written to read the dataframe and convert the bounding box inputs into VOC format. This included consideration for images with multiple bounding boxes in a single patient image.

Example of one such annotation file generated is shown below:

<annotation>

<folder>train\_images\_present</folder>

<filename>c37cfadd-1e52-4704-9831-403826ad2974.jpg</filename>

<path>/content/drive/MyDrive/Project Datasets/Capstone Project/train\_images\_present/c37cfadd-1e52-4704-9831-403826ad2974.jpg</path>

<source>

<database>Unknown</database>

</source>

<size>

<width>256</width>

<height>256</height>

<depth>3</depth>

</size>

<segmented>0</segmented>

<object>

<name>opacity</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<difficult>0</difficult>

<bndbox>

<xmin>60.0</xmin>

<ymin>39.0</ymin>

<xmax>129.0</xmax>

<ymax>174.0</ymax>

</bndbox>

</object> <object>

<name>opacity</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<difficult>0</difficult>

<bndbox>

<xmin>167.0</xmin>

<ymin>40.0</ymin>

<xmax>237.0</xmax>

<ymax>157.0</ymax>

</bndbox>

</object>

</annotation>

1. Edit the configuration file:

This is one of the most important steps as the configuration file is the key for generating image anchors and for training process.

|  |  |
| --- | --- |
| {  "model" : {  "min\_input\_size": 288,  "max\_input\_size": 448,  "anchors": [56, 52, 65,97, 71,146, 85,68, 94,194, 98,138, 107,102, 119,168, 122,221],  "labels": ["opacity"]  },  "train": {  "train\_image\_folder": "/content/drive/MyDrive/Project Datasets/Capstone Project/train\_images\_present/",  "train\_annot\_folder": "/content/drive/MyDrive/Project Datasets/Capstone Project/train\_annotations\_present/",  "cache\_name": "pneumonia\_train.pkl",  "pretrained\_weights": "backend.h5",  "train\_times": 8,  "batch\_size": 8,  "learning\_rate": 1e-4,  "nb\_epochs": 10,  "warmup\_epochs": 3,  "ignore\_thresh": 0.5,  "gpus": "0",  "grid\_scales": [1,1,1],  "obj\_scale": 5,  "noobj\_scale": 1,  "xywh\_scale": 1,  "class\_scale": 1,  "tensorboard\_dir": "logs",  "saved\_weights\_name": "pneumonia\_detection.h5",  "debug": true  },  "valid": {  "valid\_image\_folder": "",  "valid\_annot\_folder": "",  "cache\_name": "",  "valid\_times": 1  }  } | Model section has the following parameters:   * Min\_input\_size and Max\_input\_size are used to define the range within which the images are resized * The anchor box sizes are used to improve the quality of object detection. The coordinates of anchor boxes mentioned in the configuration file are obtained after running the command described in 4c below. * Labels tag is updated with one class name ‘opacity’ corresponding to pneumonia dataset.   Train section has the following parameters:   * Train\_image\_folder – location of images to be trained with full path * Train\_annot\_folder – location of image annotations which correspond 1:1 with the images itself * Cache\_name – the name of the file where annotation results are stored. This will get created first time we run this * Pretrained\_weights – we need to use the backend weights available from darknet repository * Train\_times - the number of time to cycle through the training set. Retained the default value of 8 * Batch\_size - # the number of images to read in each batch. This is a very important parameter to fine tune considering available memory * Learning Rate - # the base learning rate of the default Adam rate scheduler * Nb\_epochs – number of epochs to train the model * Gpus – flag to indicate to use gpu when the value is 0 * Saved\_weights\_name – file to store the trained model with weights * Default values chosen for remaining parameters   Since there is not separate validation dataset, we have not set these parameters |

1. Generate anchors for the pneumonia dataset:

Anchors can be generated using the following command:

!python gen\_anchors.py -c config\_pneumonia.json

The script gen\_achors.py generated the following result:

Average IOU for 9 anchors: 0.83

56,52, 65,97, 71,146, 85,68, 94,194, 98,138, 107,102, 119,168, 122,221

1. Train data on pneumonia dataset:

Training performed using the following command:

!python train.py -c config\_pneumonia.json

Once training is completed, the script prints out mAP value.

1. Perform detection using trained weights:

Training perform on a test image using the following command saves the result image with predicted bounding boxes in the output folder(default).

­­!python predict.py -c config\_pneumonia.json -i '/content/drive/MyDrive/Project Datasets/Capstone Project/train\_images\_present/c37e4ade-605d-4b78-ada5-231c82b72b43.jpg'

A picture containing text

Description automatically generated

1. Observations and next steps:

The mAP value is very low considering the number of images trained. By training additional images and increasing number of epochs, adjusting the values of object threshold and nms threshold, we can increase prediction quality.