Get trained to train

U Skanda Aithal *

Abstract— **Object detection is a fundamental task in computer vision that involves identifying and localizing objects of interest within digital images or video frames.**

Machine learning algorithms play a crucial role in object detection by automating the process of learning and recognizing patterns within visual data. These algorithms analyze a large amount of labeled training data to extract meaningful features and create models that can generalize to new, unseen data. There are several popular machine learning algorithms used for object detection, each with its unique strengths and characteristics.

One widely used algorithm for object detection is the region-based convolutional neural network (R-CNN) family. R-CNN approaches generate a set of region proposals within an image and then classify each proposal as either containing an object or not.The one I have used is an algorithm of this family, FasterRCNN.

Machine learning algorithms have significantly advanced object detection, enabling accurate and efficient identification and localization of objects in images and videos. There are other algorithms like YOLO, SSD and others which are efficient and have pushed the object detection performance.

I. INTRODUCTION

The objective is to train machine learning models to decipher the unmarked images and construct a model that aligns with the guidelines given in the commandments. The model should be trained on Colab notebook using pytorch. The bounding boxes of the dataset provided is to be created using LabelImg.

The first algorithm I implemented was a custom algorithm using resnet50 as the base model. I used a few linear, ReLU and a sigmoid layer to start with and I tried a lot of combinations of these. But they failed to give me the expected results maybe because very small dataset as I had not tried augmentations then. Later I switched to use FasterRCNN model.

II. PROBLEM STATEMENT

The major objective is to train a Machine Learning Algorithm on object detection using provided dataset.

In this scenario, we find ourselves in a distant future where artificial intelligence has become deeply ingrained in society, and we are part of a lineage that embodies the merging of ancient wisdom and advanced technology. As a transcendent individual, we embark on a quest to uncover the mysteries of existence, traversing realms where destiny and knowledge intertwine.

One day, driven by an insatiable curiosity, we stumble upon a cryptic miniature box labeled "Universal-Serial-Bus," which contains a physical data storage medium. It is a revelation for us, as it was the first time seeing such a physical storage. Upon accessing the contents of the storage device, we discover a folder filled with unlabeled images, accompanied by a text containing commandments bestowed by an ancient philosopher. The objective is to apply the expertise in training machine learning models to decipher the unmarked images and construct a model that aligns with the guidelines set forth by the ancient philosopher. The commandments, although shrouded in mystery, serve as your guiding principles in this endeavor. The text instructed us to overcome the confines of personal computer and train the model on Colab's gpu using pytorch. And also to use the eyes of LabelImg which helps in drawing bounding boxes in unlabelled images and get that data in pascal voc format which is generally used.

The final model should be able to draw a bounding box around the balls in the input images.

III. RELATED WORK

The algorithms like RCNN, YOLO, SSD and others can be used. We can also use augmentations to improve the dataset. SSD algorithm is said to be relatively simpler and faster. You Only Look Once (YOLO) is a widely used algorithm which is known for its object detection characteristic. My initial attempt to use an algorithm based on resnet50 can also be used if we have a larger dataset. If the dataset is small it is suggested to use less complex algorithms to avoid over-fitting. And also we can introduce a Dropout layer to improve efficiency.

IV. INITIAL ATTEMPTS

Initial attempt of model developed using resnet50 as base model. I had used a few linear, ReLU layers and finally a sigmoid layer. I also later tried reducing the comlpexity by reducing the layers and added dropout layers to increase the efficiency. But they did not give me expected results.

V. FINAL APPROACH

FasterRCNN algorithm has been used

The first cell sets up various configurations and parameters for training a machine learning model on image data. The variables numEpoch, batchSize, and reSize are defined to control the number of training epochs, batch size for each iteration, and the desired size to which the input images will be resized, respectively. The code also checks for the availability of a CUDA-enabled GPU device and assigns the device accordingly. If a GPU is available, the model training will be performed on the GPU; otherwise, it will utilize the CPU.train and valid variable specify the directory

*

paths for the training and validation data. The ”trainData” directory is assigned to the variable train, while the ”testData” directory is assigned to valid. These directories are expected to contain the corresponding image data for training and validation. The variable classes is a list that defines the different classes or categories present in the dataset. In this case, there are two classes: ”Background” and ”Ball”. The VISUALIZE TRANSFORMED IMAGES flag is a boolean variable that determines whether transformed images during training should be visualized or not. If set to True, the transformed images will be displayed as per the later part of the code. The output variable specifies the directory path where the model and the training graphs will be stored. The save plot epoch and save model epoch variables control the frequency at which the model and the graphs will be saved.

The second cell has some utility functions and classes which are going to be used later on. The Averager class initializes with currTotal and iterations variables set to zero. The send method adds a new value to the current total and increments the iteration count. The value property calculates and returns the average value based on the current total and iterations. The reset method resets the iteration count and current total to zero. The collate fn function just returns a tuple of lists which is used in DataLoader function. The getTrainTransform function returns an augmentation pipeline for training data. It uses the A.Compose function from the Albumentations library to create a sequence of transformations. The transformations include random flipping, random 90-degree rotation, motion blur, median blur, and Gaussian blur. The ToTensorV2 transformation converts the augmented image and associated bounding boxes to PyTorch tensors. The bounding box parameters are specified to be in Pascal VOC format. The getValidTransform function returns an augmentation pipeline for validation data. It also uses A.Compose to create a sequence of transformations, which in this case only includes converting the image and bounding boxes to tensors. The showTransformedImage function is used for visualizing transformed images. It takes a trainLoader as input, which is a data loader for training data. If the VISUALIZE TRANSFORMED IMAGES flag is set to True, this function selects a batch of images and targets from the trainLoader. It then converts the images and targets to the appropriate device (GPU or CPU), extracts the bounding box information, and draws rectangles on the images using OpenCV. The transformed images are displayed using cv2.imshow. This function is primarily intended for debugging and visualizing the effects of the augmentation pipeline during training.

The next cell defines a custom dataset class named CustomData. The CustomData class is a subclass of the torch.utils.data.Dataset class, which is used for creating custom datasets in PyTorch. It takes several arguments in its constructor, including dirPath (the directory path containing the dataset), width and height (the desired dimensions for resizing images), classes (a list of class labels), and transforms (optional data augmentation transformations). The class initializes by obtaining the paths of the image files in the specified directory, sorting them, and storing them in the allImages variable.The getitem method is overridden to return a specific image and its associated targets (bounding boxes and labels) based on the provided index idx. It reads the image using OpenCV, converts it to RGB format, resizes it to the desired dimensions, and normalizes the pixel values. It then parses the corresponding XML annotation file to extract the bounding box coordinates and labels. The bounding box coordinates are adjusted according to the resized image dimensions. The boxes and labels are converted to PyTorch tensors, and the target dictionary is constructed with the relevant information. If data augmentation transforms are provided, they are applied to the image and bounding boxes. Finally, the preprocessed image and target are returned as the output. The len method returns the total number of images in the dataset. After defining the CustomData class, an instance trainDS is created with the specified parameters, including the getTrainTransform function for data augmentation. Similarly, an instance validDS is created for the validation dataset with the getValidTransform function. The visualize sample function is defined to visualize a sample image and its associated bounding boxes and labels. It takes an image and target dictionary as input. It draws rectangles around the bounding boxes and adds labels to the image using OpenCV functions. Lastly, a loop is used to visualize a few samples from the dataset by retrieving the image and target at each index, and passing them to the visualize sample function. But the samples here appear black since the pixel values of the are divided by 255.

The next cell defines a function named createModel that creates a Faster R-CNN (Region-based Convolutional Neural Network) model for object detection. It utilizes the torchvision library, which provides pre-trained models and utilities for computer vision tasks. First, it loads the pre-trained Faster R-CNN model, fasterrcnn resnet50 fpn, from the torchvision models. This model is pre-trained on a large-scale dataset and has a ResNet-50 backbone with a feature pyramid network (FPN) architecture. Next, it retrieves the number of input features for the classifier from model.roi heads.box predictor.cls score.in features. This feature dimension represents the output of the last convolutional layer before the classifier. Then, it replaces the existing box predictor of the model's region of interest (ROI) heads with a new FastRCNNPredictor. The FastRCNNPredictor takes the extracted features as input and generates class scores and bounding box predictions for the specified number of classes. Finally, the modified model with the updated classifier is returned.

The next cell is for training the model. The code defines a train function that takes a trainDataLoader and a model as inputs. This function performs the training loop over the training data. It initializes a progress bar (progBar) to track the training progress. Iterates over the training data using the progress bar. It sets the gradients to zero (optimizer.zero grad()). Retrieves the images and targets from the data, sends the images to the device (presumably a GPU) and converts the targets to device as well. The code then computes the loss of the model on the images and targets. It computes the

total loss by summing up individual losses and stores the loss value in a list (train loss list). It performs backpropagation and optimization (losses.backward() and optimizer.step()). The code then updates the progress bar description with the current loss value. The code also defines a validate function that takes a validDataLoader (presumably a data loader for validation data) and a model as inputs. This function performs the validation loop over the validation data. The steps inside the function are similar to the train function, with the key difference being that gradients are not computed or updated since it's the validation phase.

A model is created using createModel function with a specified number of classes (numClasses) and moved to the device. The model parameters are extracted (params) and an optimizer is created using stochastic gradient descent (torch.optim.SGD). Several variables are initialized, including train loss hist and val loss hist for storing loss values during training and validation, train itr and val itr for keeping track of iterations, and empty lists train loss list and val loss list to store loss values for plotting. If VISUALIZE TRANSFORMED IMAGES is set to true, a function called showTransformedImage is called with trainLoader as an argument to visualize the transformed images. The code enters a loop over the epochs specified by numEpoch. For each epoch:

Loss history objects are reset.

Two subplots are created for visualizing the training and validation losses.

The current time is recorded.

The train function is called with trainLoader and model as arguments, and the returned training loss values are stored.

The validate function is called with validLoader and model as arguments, and the returned validation loss values are stored.

The training and validation losses for the current epoch are printed.

The time taken for the epoch is calculated and printed.

If the current epoch is a multiple of save model epoch, the model's state is saved to a file.

If the current epoch is a multiple of save plot epoch, the training and validation loss plots are created and saved to files.

If it is the last epoch, the plots and model state are saved again.

All plot figures are closed.

The next cell is for testing the saved models. An unlabelled image is passed to the model after necessary transformations and an output is obtained.

VI. Results and Observation

The initial model I have used did not give me expected output. The box drawn were not covering the ball,a lot of times they were not even close to the ball. I tried to improve that model but I failed. Then I decided to use an RCNN model. I trained the model for 10 iterations and got satisfactory results. One of the outputs is given. The details of the last iteration of the training is also given.

Fig. 1. Training iteration

Fig. 2. Validation iteration

VII. Future Work

The model can be trained for more iterations. A larger dataset would also help in improving the accuracy of the model. A different algorithm can also be used like SSD or YOLO. SSD algorithm is said to provide a balance between accuracy and speed, so it can be used.

CONCLUSION

The problem statement was just to train an ML model on Google Colab using PyTorch. For the process of labelling LabelImg was to be used. One problem was to choose the model to work on for object detection. The first model I chose did not work for me, so I decided to use RCNN algorithm. We were supposed to use GPU to train the model. That was actually helpful. Initially I tried training the model on a CPU but it took around 10 to 20 minutes per epoch. But then I used GPU which took less than a minute per epoch. The output of the RCNN model was satisfactory but the model I initially worked did not give me the expected result. Debugging the first model was also a problem. I could not figure out if it was a problem of having a small dataset or a problem in the algorithm itself. I tried modifying the model a lot, but still there was no improvement. So I finally decided to change the model and then the results were satisfactory.

The task of object detection plays a very big role in autonomous robotics. It helps in obstacle avoidance in autonomous vehicles. It is also used in drones or bots which are used for search operations to identify the objects. Machine Learning Algorithms in general is also widely used in robotics since they help bots in decision making and in interacting with dynamic environments.

References

[1] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, Bo Ma, "A Review of Yolo Algorithm Developments", Procedia Computer Science 199, 1066–1073, 2022

[2] https://arxiv.org/abs/1512.02325

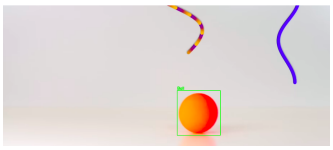

Fig. 3. An output