Building a deep learning model for object detection using Pytorch library

Lakshmikar Reddy Polamreddy Yeshiva University
lakshmikarpolamreddy@gmail.com

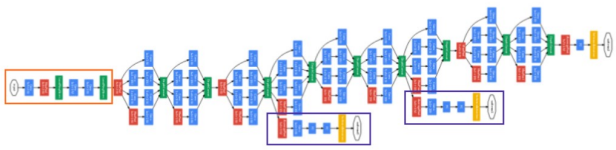
Abstract

An attempt has been made to build a deep learning neural network model in Pytorch library for the sake of object detection. To develop our model, we have applied transfer learning technique by using Googlenet pretrained model. This model has been trained on 1042 images having cowstall numbers. We have passed coordinates of bounding box and cowstall numbers as labels. After training our model for 500 epochs, we have used it to predict the results of 261 unseen images. Our model has yielded two outputs- one for predicting the cowstall number on the images and the next for predicting the bounding box coordinates. With this model, we have achieved an acccuracy of 45 percent in case of cowstall numbers. In future, I would like to develop a similar model to predict objects in videos.

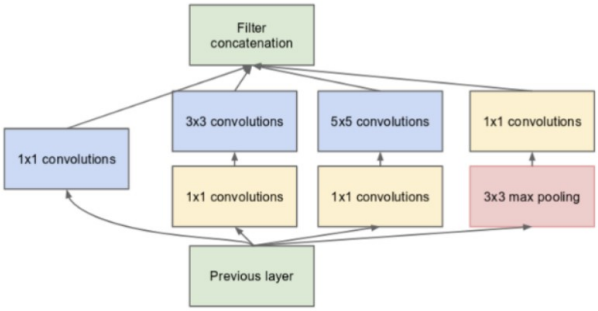
1. Introduction

Object detection refers to the task of assigning a label and a bounding box to all the objects in an image. This can be done by building deep a learning neural network model using libraries such as Keras, Tensorflow and Pytorch. For this project, we have used Pytorch because of its better debugging capabilities. In addition to this, Pytorch is faster than Keras especially when dealing with large datasets.

Instead of developing the model from scratch, we have exploited transfer learning technique which refers to the use of a pretrained models such as GoogleNet, Resnet50, Yolo and many more. In this case, we preferred to use GoogleNet. Its architecture [1] is 22 layers deep, with 27 pooling layers included. There are 9 inception modules stacked linearly in total. The ends of the inception modules are connected to the global average pooling layer. Below is a zoomed-out image of the full GoogleNet architecture



Since neural networks are time-consuming and expensive to train, the authors limit the number of input channels by adding an extra (1 × 1) convolution before the (3 × 3) and (5 × 5) convolutions to reduce the dimensions of the network and perform faster computations. Below is an image showing a Naive Inception Module with this addition



The detailed architecture and parameters of GoogleNet for all the convolution and pooling layers, and inception modules are explained in the image below

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

We have used this pretrained model and further trained it on our input cowstall images for 500 epochs. This trained model has been used to predict the results on unseen cowstall images.

2. Related Work

A contestant [3] who participated in the Global Wheat Detection Competion held in Kaggle platform developed a model to detect the number and size of wheat heads in an image. The dataset used here has just 2 classes with one being background. FastRCNN with Resnet-50 pretrained model has been used to develop final model.

Pytorch.org tutorials [2] have been very helpful to understand how to prepare datasets and how to pass them to a dataloader in pytorch library for custom datsets.

3. Methods

We have considered two types of losses in this case - Cross Entropy Loss for stall numbers and Regression loss for bounding box coordinates Below loss is calculated for each stall number per observation and sum the result.

$$\text{Cross Entropy Loss} = \sum_{c=1}^{M}(y_{o,c}log(p_{o,c}), \qquad (1)$$

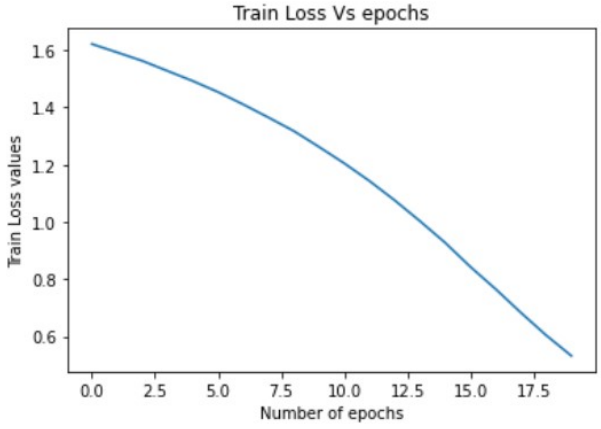
where M is number of classes, y is binary indicator (0 or 1) if class label c is the correct classification for observation o and p is the predicted probability observation o is of class c

Below Regression loss is calculated for each coordinate of the bounding boxes and sum the result

$$\text{Regression Loss} = .5\sum(x-y)^2 \qquad (2)$$

4. Results

Below graph shows loss value for training images versus number of epochs for my model that was run for 500 epochs. This has been plotted with number of epochs on X-axis and loss values on Y-axis. Here, the loss constitutes the sum of both corss entropy loss and regression loss. The training loss for this model has reduced from 6.2 to 0.5 after 100 epochs and it seems to have converged, it has just given 45 percent accuracy on unseen test images in predicting the correct cowstall numbers.



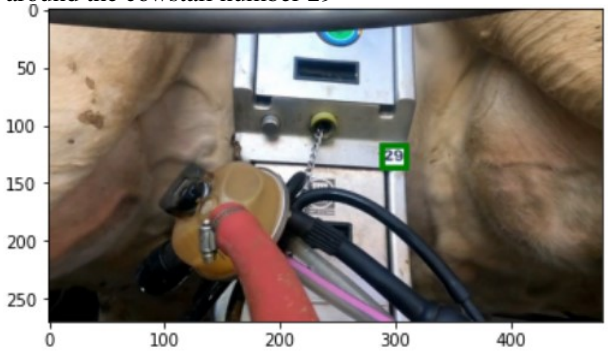
Validation accuracy obtained by this model : 59 percent

After testing this model on 261 unseen images, it has shown an accuracy of 45 percent.

4.1. Datasets

We have considered 1042 images containing cowstall numbers ranging from 1 to 60 for training my model. 916 images have been randomly selected for train data set and 126 images for validation data set. A batch size of 16 images have been selected for running each epoch. Along with the images, we have been provided a csv file containing the box coordinates and class numbers in different columns.

We would like to show a below sample image to get an idea how the image looks like after applying bounding box on the stall number. You can observe a green bounding box over around the cowstall number 29



The trained model was applied on 380 unseen images to predict their classes.

Data set	Train data set	Validation data set	Test data set
Number of images	916	126	261

5. Discussion

As we have decided to use pretrained models for this project, initial attempt was made using resnet50 model but we did not get satisfactory results for this simple dataset. Then, we tried with GoogleNet model whose performance is better than the former one for this dataset.

Regarding loss, I have used Cross Entropy loss function for cowstallnumbers and Regression loss for bounding box coordinates. As the regression loss value was far larger than the cross entopy loss, we have rescaled the regression loss to be in the same range like that of cross entropy loss

Regarding optimizer, I preferred to use Stochastic gradient descent to Adam optimizer with initial learning rate of 0.005. For every 20 epochs, we have changed the learning rate by 0.001 for faster convergence and to improve accuracy. To make the training faster, I have normalized the images.

6. Conclusion

This project has been started to achieve 90 percent accuracy but could achieve just 45 percent when tested to predict cowstall numbers of unseen test images. I have tried this using pretrained models such as resnet50 and GoogleNet. In this case, GooleNet performance is better than that of re-

sent50. By performing hyper parameter tuning, by data augmentation techniques and by using other pretrained models, we may achieve better accuracy. Going forward, I would like to extend this object detection project to identify objects in video files as well.

References

[1] https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/. 1

[2] https://pytorch.org/tutorials/beginner/basics/data$_t utorial.htmlk$.2

[3] https://www.kaggle.com/code/aryaprince/getting-started-with-object-detection-with-pytorch/notebook. 1