Object Recognition using Involution

Rishav Singh
cs21mtech11005@iith.ac.in

Shubham Gupta
cs21mtech14001@iith.ac.in

Srivathsa L Rao
cs21mtech12007@iith.ac.in

Ravi Chandra Duvvuri
cs21mtech11001@iith.ac.in

Raguru Sai Sandeep
cs21mtech11008@iith.ac.in

Abstract

Accurate object Recognition has been an important topic in the of computer vision systems. With the development of deep learning techniques, the accuracy for object recognition has increased exponentially. Convolution has been the core ingredient of modern neural networks, triggering the surge of deep learning in vision specially for object recognition. Involution is atomic operation for deep neural networks by inverting the aforementioned design principles of convolution. This project aims to incorporate state-of-the-art technique for object recognition and the Involution Technology with the goal of achieving high accuracy. The real challenge in the previous object recognition mechanisms was the dependency on other computer vision techniques for the current deep learning based approach, which ultimately makes the performance slow and non-optimal. In this project, we try to incorporate the involution operation over the previously state-of-the-art Faster region convolutional neural network (Faster RCNN) or You Only Look Once(YOLO) to get better results as Involution is proved to be experimentally better. In this project, we use a completely deep learning based approach to solve the problem of object recognition in an end-to end fashion. The network is trained on the most challenging publicly available dataset (PASCAL VOC), on which an object recognition challenge is conducted annually. The resulting system is fast and accurate, thus aiding those applications which require object recognition.

1. Introduction

There are fascinating problems with computer vision, such as object recognition, which is a part of an area called object recognition. For this type of issues, there has been a robust development in recent few years, mainly due to the advances of convolutional neural networks, deep learning techniques, and the increase of the parallelism processing power offered by the graphics processing units (GPUs). The image classification problem is the task of assigning to an input image one label from a fixed set of categories.

Convolution has been one of the most popular techniques used for deep learning in vision. Convolution kernels have 2 important properties:
1. Spatial Agnostic
2. Channel Specific

But these properties deprive convolution kernels of the ability to adapt to diverse visual patterns with respect to different spatial positions.

1.1. Involution

What is Involution ?

Like in convolution, in involution too, we have kernels which is moved across the whole input image based on the specified stride value and an output patch is calculated after multiplication operation followed by addition operation. In this diagram we see the difference between a convolution kernel and an involution kernel.

First we multiply each value in the corresponding dimension in the kernel with the input patch.

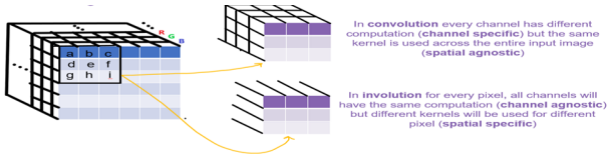


Figure 1. Convolution v/s Involution(Multiplication Op.)

Figure 2. Convolution v/s Involution(Summation Op.)

$$\mathcal{H}_{i,j} = \phi(\mathbf{X}_{i,j}) = \mathbf{W}_1\sigma(\mathbf{W}_0\mathbf{X}_{i,j}).$$

Figure 3. ϕ Function

Here σ signifies activation function and batch normalisation. W_0 and W_1 are weights. X_{ij} is the pixel at dimension (i,j).

In involution based on the centre pixel of the input patch the kernel for that input patch is calculated through the ϕ function. Its basically a double layered neural network which gives us a kernel of dimension $1 * 1 * K^2$. We then reshape it to the 2d kernel i.e $K * K * 1$.

Next we do the multiply operation like convolution, but unlike convolution we broadcast the computations on all the channels of the input patch as shown in the figure. Next we perform the add operation to get the centre pixel as shown in the figure. In this way we continue calculating each and every pixel throughout the image.

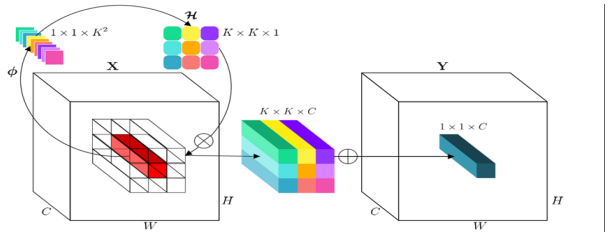


Figure 4. Involution Methodology

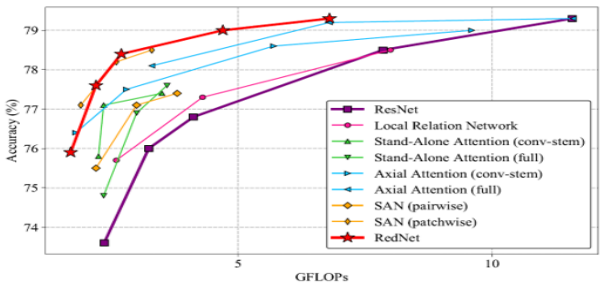
Architecture	#Params (M)	FLOPs (G)	Top-1 Acc. (%)
ResNet-26 [18]	13.7	2.4	73.6
LR-Net-26 [20]	14.7	2.6	75.7
Stand-Alone ResNet-26 [39]	10.3	2.4	74.8
SAN10† [64]	10.5	2.2	75.5
RedNet-26	**9.2**	**1.7**	**75.9**
ResNet-38 [18]	19.6	3.2	76.0
Stand-Alone ResNet-38 [39]	14.1	3.0	76.9
SAN15† [64]	14.1	3.0	77.1
RedNet-38	**12.4**	**2.2**	**77.6**
ResNet-50 [18]	25.6	4.1	76.8
LR-Net-50 [20]	23.3	4.3	77.3
AA-ResNet-50 [2]	25.8	4.2	77.7
Stand-Alone ResNet-50 [39]	18.0	3.6	77.6
SAN19† [64]	17.6	3.8	77.4
Axial ResNet-S‡ [50]	**12.5**	3.3	78.1
RedNet-50	15.5	**2.7**	**78.4**
ResNet-101 [18]	44.6	7.9	78.5
LR-Net-101 [20]	42.0	8.0	78.5
AA-ResNet-101 [2]	45.4	8.1	78.7
RedNet-101	**25.6**	**4.7**	**79.1**
ResNet-152 [18]	60.2	11.6	**79.3**
AA-ResNet-152 [2]	61.6	11.9	79.1
Axial ResNet-M‡ [50]	**26.5**	**6.8**	79.2
Axial ResNet-L‡ [50]	45.8	11.6	**79.3**
RedNet-152	34.0	**6.8**	**79.3**

Table-1: The architecture profiles on ImageNet val set. Single-crop testing with 224*224 crop size is adopted. As we can see from the above observations, even at lower number of parameters and lower FLOPs RedNet(Involution network) performs consistently better than its convolution counterparts.

Architecture	GPU time (ms)	CPU time (ms)	Top-1 Acc. (%)
ResNet-50 [18]	11.4	895.4	76.8
ResNet-101 [18]	18.9	967.4	78.5
SAN19 [64]	33.2	N/A	77.4
Axial ResNet-S [50]	35.9	377.0	78.1
RedNet-38	11.4	156.3	77.6
RedNet-50	14.3	211.2	78.4

Table-2: Runtime analysis for representative networks. The speed benchmark is on a single NVIDIA TITAN Xp GPU and Intel Xeon CPU E5-2660 v4@2.00GHz

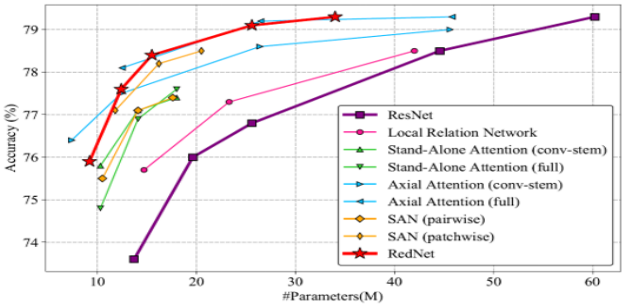
Again the above observation shows RedNet performs in considerably lower CPU times and give on par or better accuracies than convolution couterparts



(a) The accuracy-complexity envelope on ImageNet.

Figure 5. Convolution v/s Involution(Summation Op.)

In the low resource region RedNet performs significantly better



(b) The accuracy-parameter envelope on ImageNet.

Figure 6. Convolution v/s Involution(Summation Op.)

Several ablation studies designed to understand the contributions of individual components, taking RedNet-50 as an example.

Stem
Placing a 3 * 3 involution at the bottleneck position of the stem improves the accuracy from 77.7% to 78.4% with marginal cost

Kernel Size
Steady improvement is observed when increasing the spatial extent up to 7*7 with negligible computational overheads. The improvement somewhat plateaus when further expanding the spatial extent.

Kernel Size	#Params (M)	FLOPs (G)	Top-1 Acc. (%)
3×3	14.7	2.4	76.9
5×5	15.1	2.5	77.4
7×7	15.5	2.6	77.7
9×9	16.2	2.7	77.8

(a) Accuracy saturates with **kernel size** increasing.

Group Channel
Sharing a kernel per 16 channels halves the parameters and computational cost compared to the non-shared one, only sacrificing 0.2% accuracy. However, sharing a single kernel across all the C channels obviously under performs in accuracy.

#Group Channel	#Params (M)	FLOPs (G)	Top-1 Acc. (%)
1	30.2	5.0	77.9
4	18.5	3.0	77.7
16	15.5	2.6	77.7
C	14.6	2.4	76.5

(b) Appropriate **grouping channels** improves efficiency.

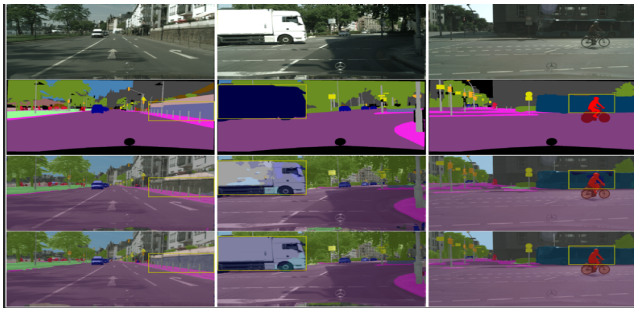


Figure 7. Visualization of Segmentation on Cityscapes Validation Set

1^{st} row shows the actual input image
2^{nd} row shows the ground truth
3^{rd} row shows the image segmentation for Res-Nets
4^{th} row shows the image segmentation of Red-Nets

2. Literature Survey

Involution based Papers:
The idea of Involution has evolved from both Self-Attention and Convolution. So, we are going to discuss important aspects of both type of variants here.

- Self-Attention based variants:
  1. Exploring Self-attention for Image Recognition (2020): This paper considered 2 types of self-attention techniques:
     - Pair wise self-attention: It generalizes the standard dot-product attention used in natural language processing match/outperform convolution counterparts.
     - Patch wise self-attention:These operators, like the convolution, have the ability to uniquely identify specific locations within their footprint. It substantially outperformed convolution counterparts.
  2. Stand-Alone Self-Attention in Vision Models (2019): A simple procedure of replacing all instances of spatial convolutions with a form of self-attention applied to Res-Net model produced a fully self-attention model that outperforms the baseline
- Convolution and Variants:
  1. o David Ha, Andrew Dai, and Quoc Le. Hypernetworks. In ICLR, 2017
  2. o Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In NIPS, 2016.
  3. o Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In NeurIPS, 2019

  Recently, dynamic convolutions have emerged as powerful variants of the stationary ones. In the first 3 papers These approaches straightforwardly generate the entire convolution filters their dynamically generated convolution filters still conform to the two properties of standard convolution, thus incurring significant memory or computation consumption for filter generation.

  1. • Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In ICCV, 2017
  2. • Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. In CVPR, 2017.

3. • Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In CVPR, 2019.

In the above 3 papers, they parameterize the sampling grid associated with each kernel. Only certain attributes e.g. the footprint of convolution kernels are determined in an adaptive fashion.

1. Hang Su, Varun Jampani, Deqing Sun, Orazio Gallo, Erik Learned-Miller, and Jan Kautz. Pixel-adaptive convolutional neural networks. In CVPR, 2019
2. JiaqiWang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. Carafe: Content-aware reassembly of features. In ICCV, 2019

There also exist previous work that adopt pixel-wise dynamic kernels for feature Aggregation as in the next 2 papers mentioned above, but they mainly capitalize on the context information for feature up-sampling and still rely on convolution for basic feature extraction.

- Julio Zamora Esquivel, Adan Cruz Vargas, Paulo Lopez Meyer, and Omesh Tickoo. Adaptive convolutional kernels. In ICCV Workshops, 2019

The most relevant work towards substituting convolution rather than up sampling Is the last paper, but in that paper also , the pixel-wise generated filters still inherit one original property of convolution, to perform feature aggregation in a distinct manner over each channel.

Object Recognition based Papers:

1. R-CNN: Regions with CNN Feature.

   Object Recognition mainly consists of three modules:

   - The first generates category-independent region proposals. These proposals define the set of candidate detection available to our detector.
   - Then using greedy like algorithms that then do recursion and try to bring the similar regions together.
   - Now from these similarly generated regions we get the final regions.

   But Still there are some problems with R-CNN like It takes a huge amount of time to train the network also taking a long time around 47 seconds for each test image and the selective search algorithm is a fixed algorithm. So other modifications like Fast R-CNN and Faster R-CNN are made.

2. Fast R-CNN : R-CNN with truncated SVD processes pictures 146 times faster than R-CNN without it and 213 times faster than R-CNN without it. Fast-RCNN will learn to classify these adversarial examples by adapting themselves. CNN gets fed with images rather than the region proposals to produce a convolutional feature map. We can identify the region of the proposals using the convolutional feature map and warp them into squares, which we then reshape using an RoI pooling layer into a fixed size that can be input into a fully connected layer.
   We can use a softmax layer to forecast the class of the proposed region, and the offset values for the bounding box using the RoI feature 19 vectors. Because we do not have to feed the convolutional neural network 2000 area proposals every time, Fast R-CNN is faster than R-CNN. Instead, the convolution procedure is performed only once per image, creating a feature map. Even though Fast R-CNN trains and tests on a single scale, fine-tuning the convolutional layers results in a significant increase in mAP (from 63.1

3. Faster R-CNN : To find region proposals, both algorithms (R-CNN and Fast R-CNN) use selective search. Selective search is a slow and time-consuming operation that degrades network performance. Faster - R CNN reduces the detection networks' operating time, exposing the region proposal calculation as a bottleneck. They make use of graphics processing units (GPUs). A convolutional network gets an image as input, which outputs a convolutional feature map, similar to Fast R-CNN.
   A separate network is employed to forecast region proposals rather than using the feature map's selective search technique to discover area proposals. An ROI pooling layer reshapes the anticipated region's proposals, classifies the image within the proposed region, and predicts offset bounding box values. Faster R-CNN is faster than its predecessors, as shown below. As a result, it can even detect objects in real-time.

3. What are we doing ?

The motive of the paper is to reproduce the results of the YOLO algorithm and prime goal being is to plug the concept of Involution in the architecture of YOLO and to make the model more efficient as we know Involution operation takes less time as compared to Convolution as shown in the Introduction itself, Involution is mathematical based and has less data dependencies compared to convolution and hence ultimately the efficiency is increased.

YOLO with Involution has theoretically the capability to increase the accuracy by using the combination of Involution and Convolution layers as proposed in the Involution

paper[1]. RedNets are combination of Involution and Convolution Layers that ultimately are proposed to give better results than ResNets that are used in YOLO.

Goals related to the project:

1. Implementation of an Involution operation that takes an image input and return the Involuted Images.
2. Mimic the SOTA YOLO results that uses generic Convolution and ResNets.
3. Plug Involution in place of Convolution for YOLO algorithm for some layers of ResNets, basically the implementation of RedNets.

4. Methodology and Working principle

Since our model was built from the ground up, the methodology is broken down into several pieces. We initially constructed Involution, then tested and attempted to duplicate the output of the YOLO technique using the generic convolution operation and ResNets, and then attempted to use Involution and a partial version of RedNets.

We explain the methodology of the mentioned mechanism below:-

- **Involution Methodology:-**
  Convolutions, though space invariant, have interchannel redundancy. Involution overcomes this. So, Involution can model long-range dependencies and relations.

  Working Principle:

  Given an image of W x H x C, we pick a 1 x 1 x C batch from the image. Then, we convert the batch using convolution into 1 x 1 x k2 features where k is the size of the kernel. Using convolution, the 1 x 1 x k2 features are converted into k x k x 1 features.

  We repeat the process for every patch in the image, maintaining the stride along the horizontal and vertical dimensions.

  We generate a kernel for every patch in the image. And instead of using one filter across all the channels, we can group the channels and use one filter per group.

  We have also used a reduction ratio to reduce the number of channels. If there are more channels, it will be computationally expensive, and some features may not be retained when passing through the involution layer.

- **YOLO Algorithm Methodology:-**
  You only look once is a cutting-edge object detection algorithm designed for real-time applications. Unlike some of its competitors, it is not a typical classifier masquerading as an object detector. It differs from other object detectors because it uses a Single Involution for both object categorization and localization.

Using logistic regression, YOLO calculates an object's confidence score in each bounding box. The architecture of the algorithm also incorporates numerous vital elements such as residual blocks, skip connections, and upsampling. In the YOLO framework, there are three processes for analyzing a frame. The input image is downsized first, then a single Involution is run on it, and finally, non-max suppression criteria are added to the resulting detections.

You Only Look Once divides the input image into a grid of SxS cells, with each cell accountable for five bounding box predictions that describe the rectangle around the item. It also generates a confidence score, which measures the degree of assurance that an object was enclosed. As a result, the score has nothing to do with the type of object in the box, only with the shape of the box. A class is predicted for each predicted

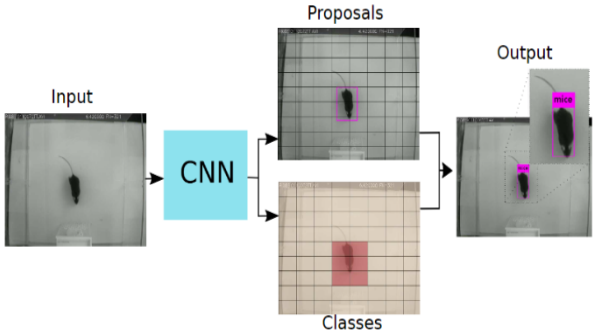


Figure 8. YOLO Pipeline

bounding box, just like a conventional classifier, resulting in a probability distribution across all possible classes. The class prediction and the confidence score for the bounding box are combined to form a single final score that specifies the likelihood that each box contains a specific type of object. Given these design decisions, the majority of the boxes will have low confidence scores, so only the boxes with final scores more significant than a certain threshold are retained.

Figure 9 represents the loss function minimized by the YOLO algorithm's training stage. 1_{ij}^{obj} indicates

$$\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]+\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]$$

$$+\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2+\lambda_{coord}\sum_{i=0}^{s^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2+\sum_{i=0}^{s^2}\sum_{c\in classes}\left(p_i(c)-\hat{p}_i(c)\right)^2$$

Figure 9. Loss function minimized by YOLO

whether or not an object occurs in cell i. and 1_{ij}^{obj} is the j^{th} bounding box predictor in the cell i that is accountable for that prediction; x, y, w, h, and C are the coordinates that reflect the box's centre relative to the grid cell's limits. The predicted width and height are based on the entire image. Finally, C stands for confidence prediction, defined as the IoU between the predicted and ground truth box.

The YOLO network's image processing is described in the following. Initially, the data is sent into an Involution Neural Network, which generates bounding boxes with confidence ratings for each perspective and a class probability map. The final forecasts are formed by combining the outcomes of the initial processes. As

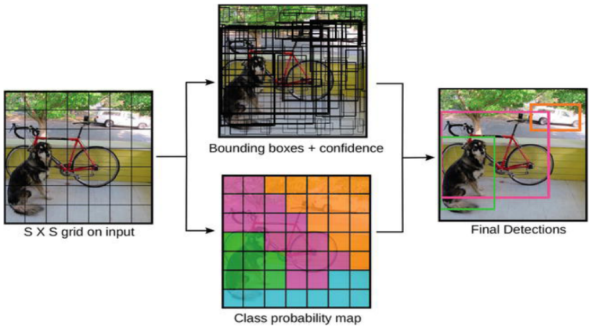


Figure 10. Detection of the YOLO model as a regression problem

a result, the input image is divided into an $S * S$ grid, with B bounding boxes, confidence for those boxes, and C class probabilities predicted for each grid cell.

- **Involution in YOLO:-**
  Input dimension is same as output dimension since the involution can be added before convolution without changing any dimension in the yolo architecture No need of pre trained weights in case of the involution layer as the kernels are made on the fly. If high resolution images are input then we resize it to the required dimension. The stride of the involution layer is set to 1 to be consistent with the following convolution dimensions Even if reduction ratio and loop size are used to reduce the computational efforts the output activation maps is again reshaped into the input dimensions maintaining the dimensional consistency.
  We introduced 3 involution layers into the existing architecture with the following specifications:

  1. 64 channels, kernel size = 3 and stride = 1
  2. 192 channels, kernel size = 3 and stride = 1
  3. 512 channels, kernel size = 3 and stride = 1

5. What work we have done?

- Our first goal was to implement the Involution layer and use it to classify image. We have tried to implement this on the CIFAR-10 dataset In this model, we have implemented the involution layer and plugged it in a very primitive neural network architecture. We have tested our model for image classification on images from CIFAR-10 dataset. As the model involves only two involution layers and linear layers, the model accuracy is very low.

  But the model performance will definitely improve after a lot of modifications that we have planned for the future. We will work on changing the architecture of the model, tweak the parameters and explore with other loss functions.

- We were able to successfully mimic the output of YOLO algorithm and were getting a decreased loss. Though we did check only for 1^{st} and 10^{th} epochs because of the resource intensive nature of the project. YOLO is instructed to be ran for 1000 epochs in the referenced GitHub $repository^{12}$ but for us each epoch took about 10 mins for a very small partial dataset and it will be even more for a full dataset, also the GPU and RAM requirement will go higher if we run the code for higher epochs and complexity. So, due to the constraints of not having enough time to train the model for about 21 days and unavailable hardware resources, we were not able to run the full fledged code with the whole data.

- We wrote the code for plugging the Involution operations for 3 extra involution layers in the model and did the necessary changes that is explained in the methodology section, again here we had the constraint of not having the all the required resources we need to run the code and as it was an additional goal that we included in the project we did not get much time to implement it but still we did what we could and we were getting a error of memory overflow as it was using an extensive temporary memory, as a consequence it ran only partially.

6. Results and Outputs

- For our first goal we were able to get a good accuracy of 10% with just 2 Involution layer as mentioned previously.

- For our 2^{nd} Goal we were getting decreased mean loss from 991.6 to 179.2 in just 10 epochs as shown in figure 11 and 12 respectively

- For our final goal as mentioned earlier we were getting error message of memory overflow due to shortage of

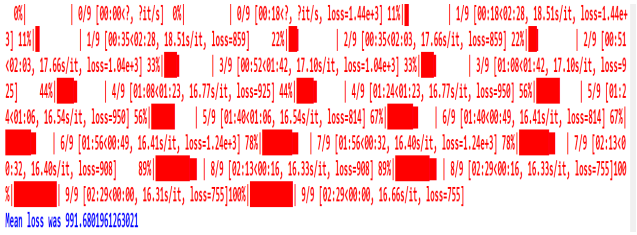


Figure 11. YOLO Algorithm for 1^{st} epoch

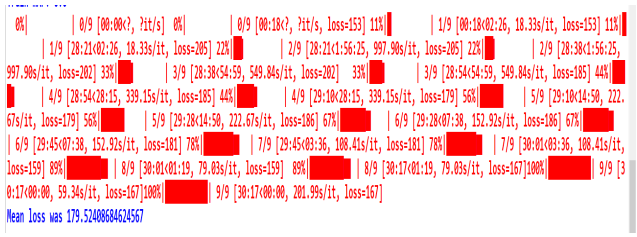


Figure 12. YOLO Algorithm for 10^{th} epoch

memory resources and un-optimized memory usage by the model. we can see the error message in figure 13

Figure 13. Error message for Involution in YOLO model

7. Future Scope

1. We can use pre trained weights trained on image classification dataset for object detection.
2. We want to implement YOLO for 1000 epochs in future. Currently we could only run it for 10 epochs as running for 1000 epochs takes more than 10 days and is not feasible.
3. Our model is using too much temporary memory (in order of GB) while running. In future we want to involve some memory optimisation techniques so that lesser temporary memory is used.

8. References

1. Duo Li, Jie Hu, Changhu Wang, Xiangtai Li, Qi She, Lei Zhu, Tong Zhang, Qifeng Chen; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 12321-12330
2. Hengshuang Zhao, Jiaya Jia, Vladlen Koltun; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 10076-10085
3. Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. NeurIPS, 2019
4. Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee, YOLOACT: Real-Time Instance Segmentation.
5. Julio Zamora Esquivel, Adan Cruz Vargas, Paulo Lopez Meyer, and Omesh Tickoo. Adaptive convolutional kernels. In ICCV Workshops, 2019
6. Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results, CVPR, 2019.
7. Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. ICCV, 2017
8. Xu Jia, Bert De Brabandere, Tinne Tuytelaars, Luc V. Gool, Dynamic Filter Networks, NIPS 2016
9. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi: You Look Only Once, Unified, Real Time Object recognition
10. R-CNN, Fast R-CNN and Faster R-RCNN , https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
11. PASCAL VOC Dataset https://paperswithcode.com/dataset/pascal-voc
12. Aladdinpersson. (n.d.). Machine-learning-collection/ml/pytorch/objectdetection/yolo at master · Aladdinpersson/machine-learning-collection. GitHub. Retrieved May 2, 2022, from https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/$object_detection$/YOLO