End Semester Report
on

Automatic Target Recognition in SAR Images Using Deep Learning

by

ADITYA SHARMA 2018A7PS0367P
and
AAYUSH ATUL VERMA 2017A7PS0061P

for the course
CS F376

Under the guidance of

J. JENNIFER RANJANI

At

Birla Institute of Technology and Science, Pilani For the Session 2020-2021, 2nd Semester

Table Of Contents

Introduction 3

Background Study 4
- Synthetic Aperture Radar ATR Framework 4
- Object Detection Techniques 5
- YOLO 6
- Faster RCNN 9
  - Part 1 : Convolution layers 9
  - Part 2: Region Proposal Network (RPN) 10
  - Part 3: Classes and Bounding Boxes prediction 10
  - Training 10
- Retinanet 10
  - Architecture 10
  - Feature Pyramid Network 11
  - Focal Loss 11

Literature review 12

Dataset 16

Work Done 17
- Research papers 17
- Image Classification 18
  - AlexNet 18
- Object Detection 19
  - Bounding boxes using OpenCV 19
  - Bounding boxes using OTSU for threshold 20
  - Issues 21
- Models for Object Detection 22
  - Using RetinaNet 22
  - Tiny-YOLOv4 23
  - Faster RCNN 25
  - Preprocessing 29
  - YOLOv4 on Preprocessed data 30

Conclusion 31

Future Scope 32

References 33

Acknowledgements

We would like to thank **Dr J. Jennifer Ranjani** for giving us the opportunity to work in this intriguing field. Her constant guidance and vast experience helped us throughout this project.

We would also like to use this opportunity to thank the **Computer Science Department of BITS Pilani** for granting us great opportunities and the platform of project-oriented courses to work on topics beyond the curriculum.

Introduction

Synthetic aperture radar (SAR) is an active ground observation system that can be installed on aircraft, satellites, spacecraft, and other flight platforms [1]. One of the most significant advantages of SAR Imaging is that it can return high-quality images at all times of the day, irrespective of the weather or altitude. Hence it has excellent applications for day to day monitoring as well as military purposes. SAR images are not like optical images, and the regions with high pixel intensity denote the possibility of a target. In the last decade, due to increased computation power and large amounts of data, various deep learning methods have gained traction in the domain of automatic target recognition (ATR). Specifically for ground targets, SAR images have been used due to their all-weather suitability.

In various fields of image classification, there has been a tremendous amount of research done using convolutional neural networks [2]. The problems faced while tackling SAR images is two-fold. Due to their small size and lack of optical colour range, they are very prone to overfitting. Another factor that plays is the amount of noise present in the image. This, along with the fact that they have a small optical range, makes it challenging to recognise the target image easily.

Due to the small number of features available to extract from these SAR images, often using the latest models lead to overfitting due to their large width and height in the architecture [3]. We aim to boost the accuracy of state-of-the-art models by performing some preprocessing steps to help better distinguish the target from its background. We also provide a way to create bounding boxes from the MSTAR dataset to perform the ATR step.

Background Study

A. Synthetic Aperture Radar ATR Framework

The entire framework contains two stages - the training and testing stage. Both these stages require a preprocessing step initially in order to maximize the distinguishable features of the target in the image. The main aim is to create a contrast between the background (noise included) and the target. The following feature extraction and training stages together constitute the deep learning models that we use.

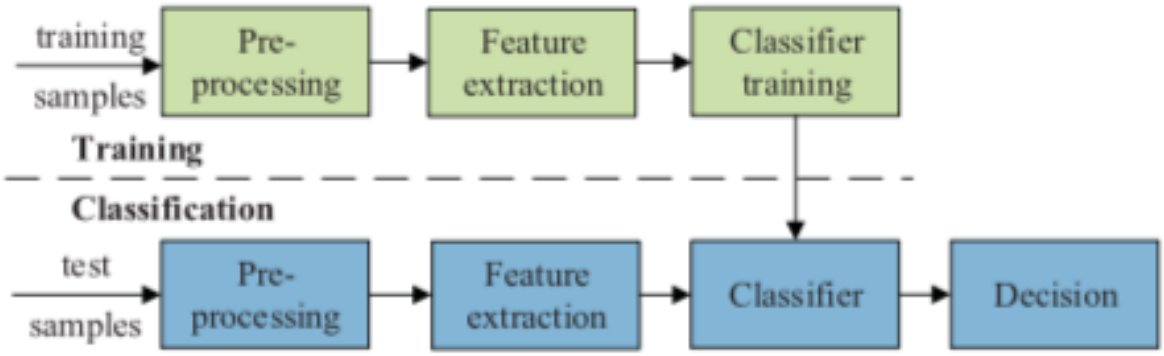

Figure. Complete pipeline of SAR ATR for the training and classification stage

B. Object Detection Techniques

An Image Classification model aims to tag the entire picture as one single class type. E.g. If there is a cat in an image, the model can only claim that the image has a cat. When these models encounter multiple classes in a single image, due to their restriction of labelling the entire image as a single class, there are problems faced. It considers the background of the image to be a part of the class type too and extracts features accordingly.

In such cases, Object Detection Models are handy. They have two primary objectives. First, it needs to recognise the target region of interest that contains the required object that has to be detected. Next, it needs to also act as a classifier to deduce to which class the object belongs.

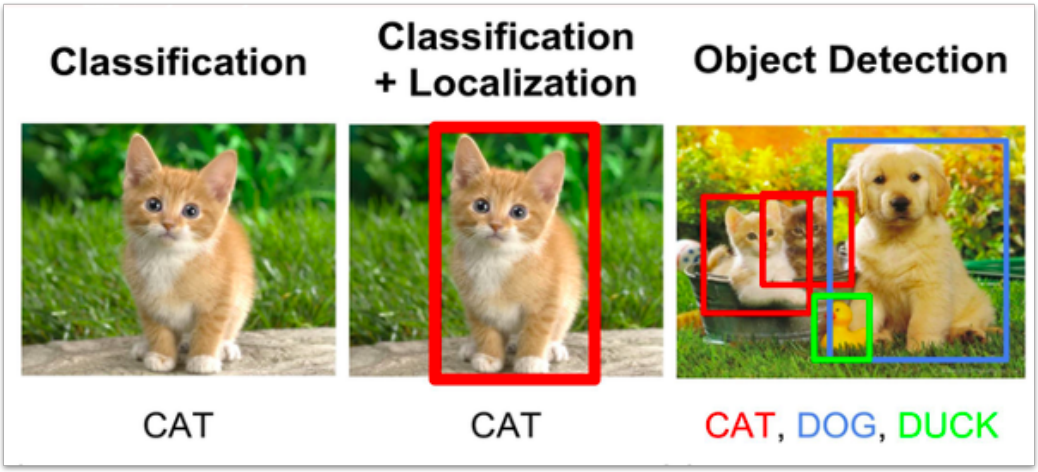

Figure. Understanding Object detection (Lecture 11: Detection and Segmentation)

Some state-of-the-art models in the domain of Object Detection are YOLO, FasterRCNN and SSD.

C. YOLO

YOLO is considered to be an object detection with one of the best speed-performance trade-offs. YOLO came on to the computer vision scene after the ground-breaking 2015 paper by Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection". Other region-proposal classification networks (fast RCNN) perform detection on various region proposals and perform prediction multiple times for multiple regions in an image. YOLO architecture, on the other hand, is more like FCNN (fully convolutional

neural network) and passes the image (nxn) once through the FCNN, and output is (mxm) prediction.

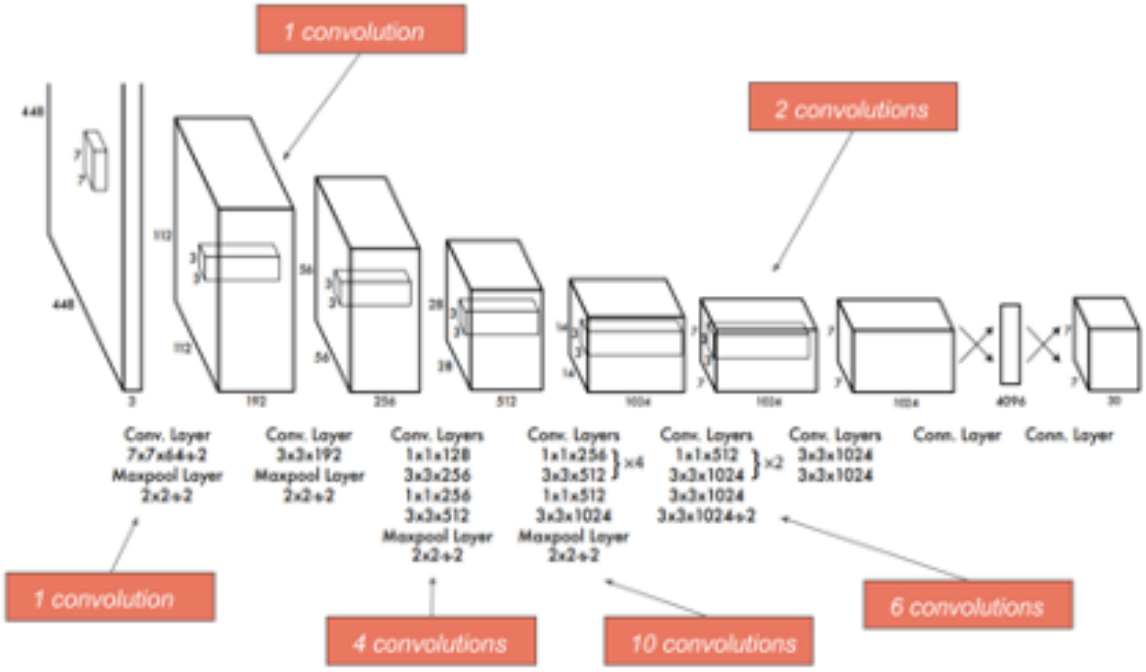

Figure. Original YOLO architecture (https://arxiv.org/pdf/1506.02640.pdf)

The algorithm works by dividing an image into multiple grids and checking whether any grid contains the object. The grid containing the middle point of the object has the target assigned to it using a multidimensional vector.

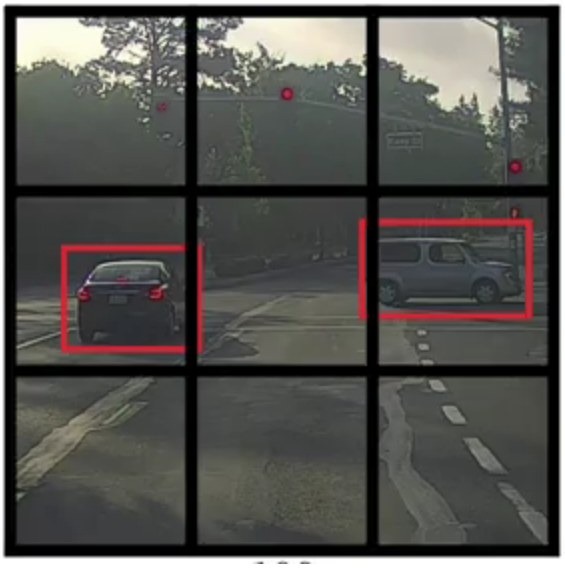

Figure. An image divided into a grid with two objects detected.

This vector contains information relating to the bounding box of the object. These bounding boxes for each grid are called anchor boxes used for the localisation of the targets. This vector contains information about the class in a one-hot encoded fashion along with the coordinates of the bounding boxes.

D. Faster RCNN

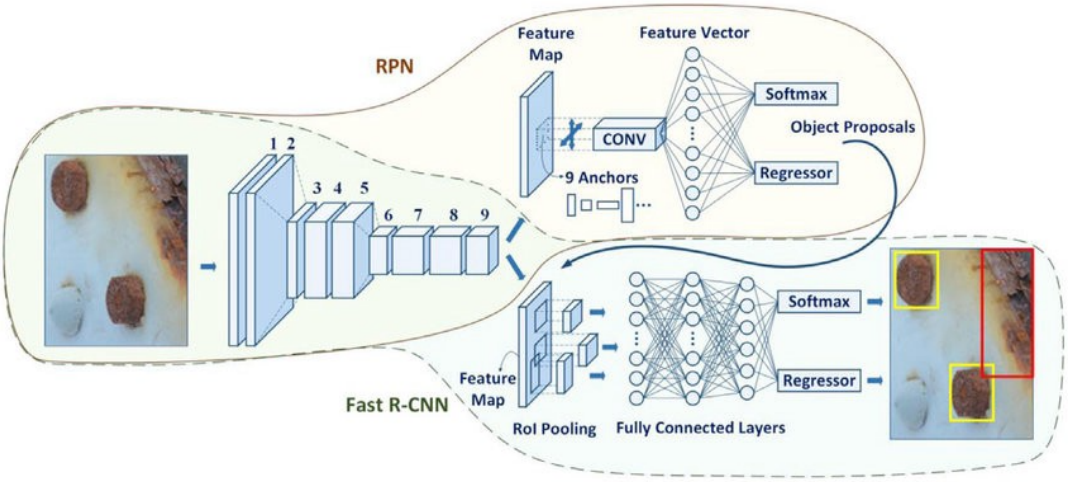

Figure. Faster RCNN architecture

Faster RCNN is composed of 3 parts:

Part 1 : Convolution layers

In these layers, we train filters to extract the appropriate features from the image. We compute convolution by sliding filter all along our input image and the result is a two-dimension matrix called feature map.

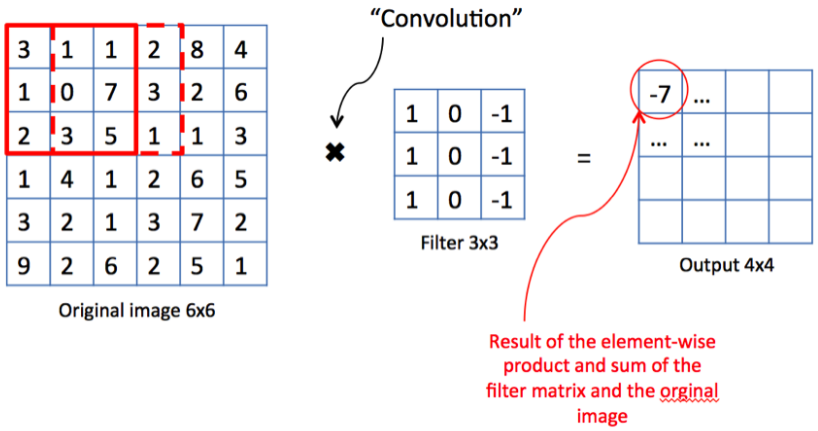

Figure: Working of convolutional filter

Part 2: Region Proposal Network (RPN)

RPN is a small neural network sliding on the last feature map of the convolution layers and predicts whether there is an object or not and also predict the bounding box of those objects.

Part 3: Classes and Bounding Boxes prediction

Now we use another Fully connected neural networks that take as an input the regions proposed by the RPN and predict object class (classification) and Bounding boxes (Regression).

Training

To train this architecture, we use SGD to optimize convolution layers filters, RPN weights and the last fully connected layer weights.

E. Retinanet

RetinaNet is one of the best one-stage object detection models that has proven to work well with dense and small scale objects. For this reason, it has become a popular object detection model to be used with aerial and satellite imagery.

RetinaNet has been formed by making two improvements over existing single-stage object detection models - Feature Pyramid Networks (FPN) and Focal Loss.

Architecture

There are four major components of a RetinaNet model architecture:

- Bottom-up Pathway

- Top-down pathway and Lateral connections
- Classification subnetwork
- Regression subnetwork

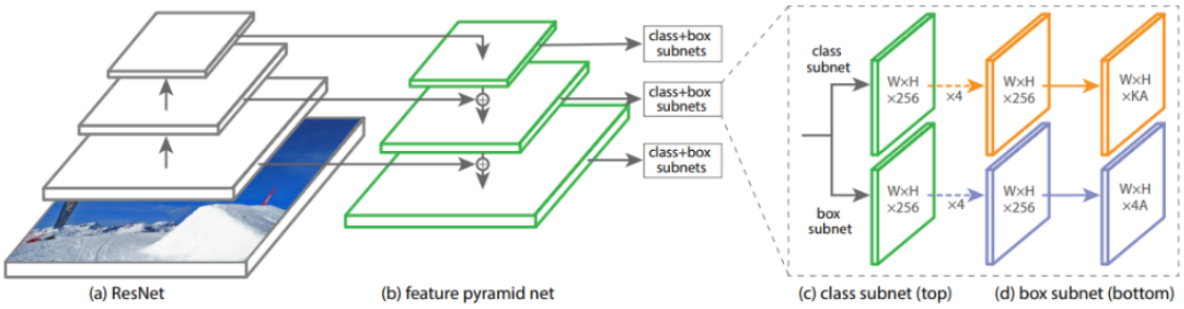

Figure. Retinanet Architecture

Feature Pyramid Network

The pyramid itself is derived from the inherent pyramidal hierarchical structure of the CNN. In a CNN architecture, the output size of feature maps decreases after each successive block of convolutional operations and forms a pyramidal structure.

Focal Loss

Focal Loss (FL) is an enhancement over Cross-Entropy Loss (CE) and is introduced to handle the class imbalance problem with single-stage object detection models.

$$FL(pt)=-(1-pt)\gamma\log(pt)$$

Literature review

Various approaches have been used for decades to implement the recognition of targets in SAR images. One of the most common and traditional approaches has been the feature-based approach. In this approach handcrafted features are designed and chosen in a way that they are able to distinguish the object from one another. These features are extracted from them and then used to train the classifiers [4]. The approach in [5] uses twelve features handpicked from the images, In such methods, the type of features had to be chosen by the researchers before itself. This led to a very high error rate when the model encountered something different. This led to the requirement of a more robust SAR-ATR model.

Figure. Traditional Machine Learning/Image classification dataflow

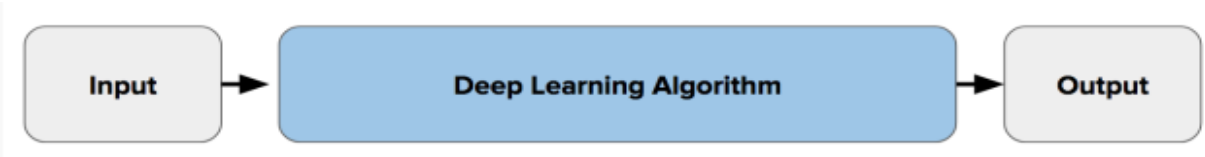

Figure. Neural Network/ Deep Learning dataflow

Problems with Three-Stage SAR ATR Architecture and the need for Neural Networks

To achieve automatic target recognition (ATR) systems for SAR interpretation, MIT Lincoln Laboratory put forward a standard SAR ATR architecture. The structure contains three stages: detection, discrimination and classification/recognition.

In the past few years, most researchers just focused on one part of these three stages, put forward theories such as scene segmentation, target discrimination, feature extraction, classifier design and so on. However, these theories and algorithms just out-perform in specific operating conditions, which makes them not able to be applied universally.

A reliable and universal system requires an effective connection between detection and recognition, so End-to-End models were proposed, and apply robust trainable classifiers, such as Adaboost and support vector machine (SVM) to realize SAR ATR. However, problems like size difference and target position mismatch between training samples and interpreting images still need to be solved.

Neural networks could extract features automatically and have obtained remarkable achievements in optical image detection. Regions with convolutional neural network (R-CNN), Fast R-CNN and Faster R-CNN were proposed, which can recognize different kinds of objects with different sizes in optical images with high accuracy. You Only Look Once (YOLO) was proposed, which achieved fast detection, but the accuracy is lower than Faster R-CNN.

Some of the works done in this domain are given ahead.

SAR Target Recognition in Large Scene Images via Region-Based Convolutional Neural Networks

https://www.mdpi.com/2072-4292/10/5/776/htm

Connecting the detection and recognition process to interpret large scene SAR images is difficult because of speckle noise and inefficient connection among these processes. Inspired by the great success of deep convolutional neural networks, methods of DCNN are applied to SAR image interpretation to extract features automatically, and a method to integrate detection and recognition of large scene SAR images based on non-maximum suppression between regions (NMSR) is proposed in this paper. The performance of this system is evaluated, and 94.67% of three-class recognition accuracy on the MSTAR data set proved that it is efficient with high accuracy.

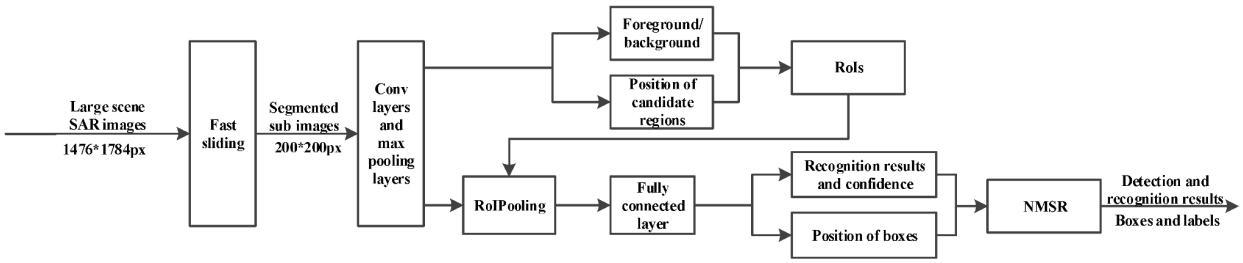

Figure: Flow chart of our model. (NMSR: non-maximum suppression among regions; SAR: synthetic aperture radar).

A Novel Convolutional Neural Network Architecture for SAR Target Recognition

https://www.hindawi.com/journals/js/2019/1246548/

Since the SAR image presents fewer features than the optical one, these CNN architectures with high performance in the optical image easily cause overfitting for the

SAR classification. Thus, in the work, based on the SAR characteristics, a novel CNN architecture (named umbrella) was constructed by minimizing the depth but extracting enough features at different levels so as to achieve rapid and accurate detection of the SAR targets.

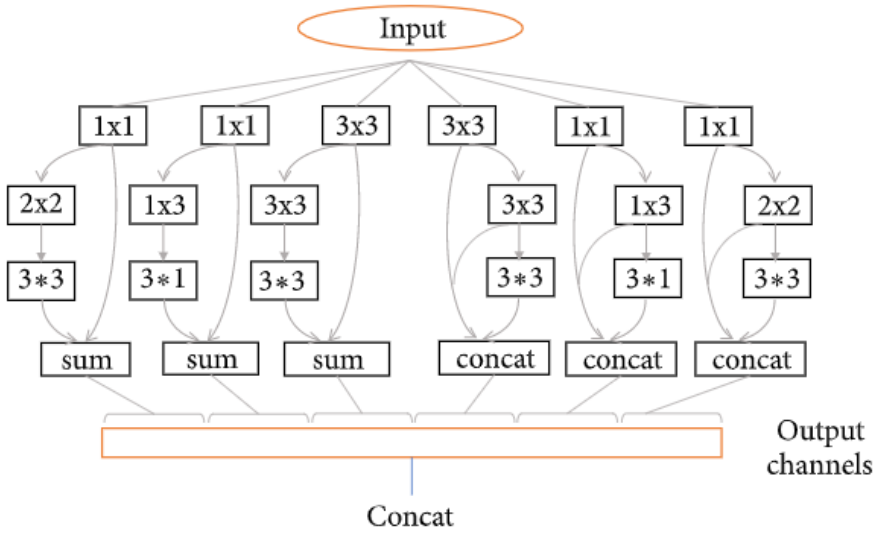

Figure. The umbrella module of the basic component of the proposed architecture.

In all the cases under consideration, the umbrella model can achieve more than 99% accuracy for the classification of 10-class targets and higher than 96% accuracy for the 8 variants of the T72 tank.

Dataset

The images in the dataset are from the publicly released MSTAR dataset that includes categories of tanks based on the ground target. There were two types of SAR images present for each class - images with a 17º depression angle and images with a 15º depression angle. All images with a 17º depression angle are used in training while the remaining images are used for testing purposes and are not included in the training process.

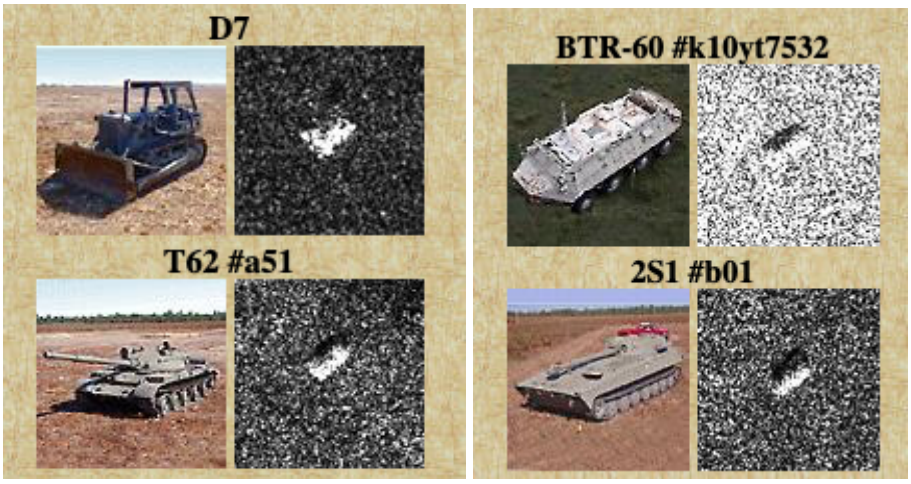

Figure. Images of the original image beside their SAR image

Class	2S1	D7	ZIL131	BTR60	BTR70	ZSU_23_4	T62	T72	BMP2	BRDM_2
Train	300	300	300	257	234	300	299	233	234	257
Test	275	275	275	196	197	275	274	197	196	275

Table: Number of images in the respective train and test sets

The dataset is slightly imbalanced with a varying degree of noise present in each class. The image dimensions and size also varies from class to class.

Work Done

Research papers

Studied and analysed the following research papers on automatic target recognition in SAR images with their link as follows.

- Automatic target recognition in SAR images: Comparison between pre-trained CNNs in a transfer learning-based approach
  https://www.researchgate.net/publication/326051310_Automatic_target_recognition_in_SAR_images_Comparison_between_pre-trained_CNNs_in_a_tranfer_learning_based_approach
- Automatic Target Recognition in SAR Images Based on Combination of CNN and SVM
  https://ieeexplore.ieee.org/document/9237422
- Classification of Large-Scale High-Resolution SAR Images with Deep Transfer Learning
  https://arxiv.org/abs/2001.01425
- A Novel Convolutional Neural Network Architecture for SAR Target Recognition
  https://www.hindawi.com/journals/js/2019/1246548/
- SAR Target Recognition in Large Scene Images via Region-Based Convolutional Neural Networks
  https://www.mdpi.com/2072-4292/10/5/776

Image Classification

The aim was to create a baseline classification model on the MSTAR dataset. The results gave an idea about the variations and quality of the dataset.

AlexNet

AlexNet is a convolutional neural network architecture conceptualized in 2012. It contains 5 convolution layers and 3 fully connected layers. For our purpose, we initialised the model using the vision:v0.6.0 version pretrained weights available on PyTorch.

The model was trained for 35 epochs and the results observed were as follows:

```
#Testing classification accuracy for individual classes.
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in test_data_loader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = AlexNet_model(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        class_names[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of   2S1 : 94 %
Accuracy of  BMP2 : 97 %
Accuracy of BRDM_2 : 98 %
Accuracy of BTR60 : 97 %
Accuracy of BTR70 : 100 %
Accuracy of    D7 : 99 %
Accuracy of   T62 : 100 %
Accuracy of   T72 : 98 %
Accuracy of ZIL131 : 99 %
Accuracy of ZSU_23_4 : 100 %
```

Figure. Results of training on AlexNet

The average accuracy observed overall classes was 97.4%

Since AlexNet is one of the most primitive CNN architectures, we expect better results from larger networks. But based on this performance, we can conclude that the dataset is good enough to provide acceptable generalizations and we can move forward to working on object detection algorithms.

Object Detection

The first step for training an Object Detection model is to create a dataset with images and corresponding annotations for the target. Since these annotations weren't available publicly, we had to draw bounding boxes on our own.

Bounding boxes using OpenCV

Object localisation was done via an OpenCV library using the cv2.boundingRect() and cv2.minAreaRect() functions.

```
# threshold image
ret, threshed_img = cv2.threshold(img,127, 255, cv2.THRESH_BINARY)

# find contours and get the external one

contours, hier = cv2.findContours(threshed_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)



#image, contours, hier = cv2.findContours(threshed_img, cv2.RETR_TREE,
#                cv2.CHAIN_APPROX_SIMPLE)

# with each contour, draw boundingRect in green
# a minAreaRect in red and
for c in contours:
    # get the bounding rect
    x, y, w, h = cv2.boundingRect(c)
    # draw a green rectangle to visualize the bounding rect
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # get the min area rect
    rect = cv2.minAreaRect(c)
    box = cv2.boxPoints(rect)
    # convert all coordinates floating point values to int
    box = np.int0(box)
    # draw a red 'nghien' rectangle
    cv2.drawContours(img, [box], 0, (0, 0, 255))
```

Figure: Implementation

```
print(len(contours))
cv2.drawContours(img, contours, -1, (255, 255, 0), 1)
```

```
2
array([[22, 44, 66, ..., 33, 33, 22],
       [20, 38, 44, ..., 34, 31, 20],
       [13, 18, 23, ..., 35, 32, 22],
       ...,
       [22, 27, 29, ..., 29, 19, 20],
       [27, 26, 23, ..., 23, 24, 31],
       [24, 26, 33, ..., 21, 30, 34]], dtype=uint8)
```

Figure: Matrix of the image including bounding boxes

```
img_bd_box = cv2.resize(img, (395, 395))
cv2_imshow(img_bd_box)
```

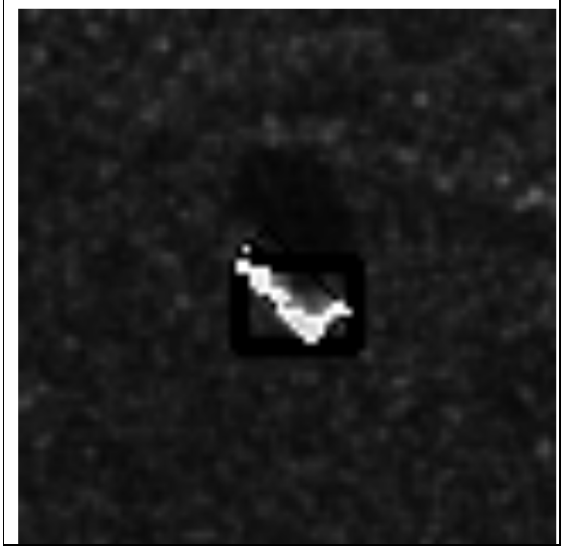

Figure: Resultant Image

Some issues were observed in a few images. The bounding boxes were not completely getting encompassed in some images.

Bounding boxes using OTSU for threshold

To have a better contrast in the recognition between the target and its background, we applied OTSU thresholding to each image.

```
# threshold image
ret, threshed_img = cv2.threshold(img,127, 255, cv2.THRESH_OTSU)
```

```
# find contours and get the external one
contours, hier = cv2.findContours(threshed_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
for c in contours:
    # get the bounding rect
    x, y, w, h = cv2.boundingRect(c)
    # draw a green rectangle to visualize the bounding rect
    print(x,y,w,h)
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 255, 255), 1)
```

Figure: Implementation

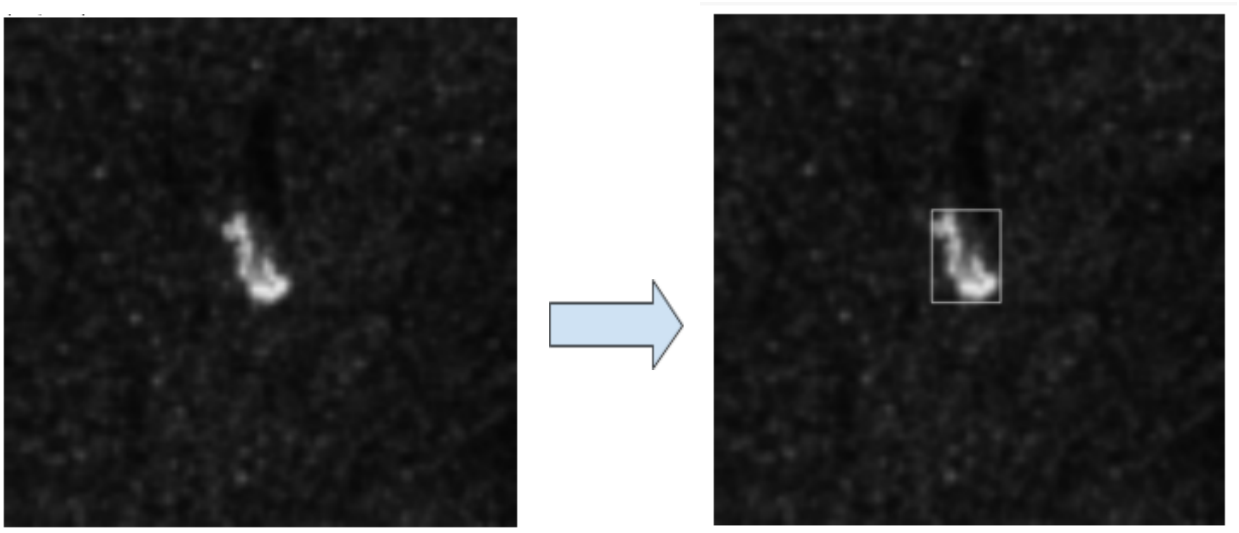

Figure: Results

Issues

Problem: Entire target object is not getting contained within the bounding box

Proposed solution: Increase the size of the bounding box by a factor proportional to the dimensions of the image

Problem: There are multiple bounding boxes detected even though the dataset has only one object per image

Proposed solution: Choose the box with the largest area

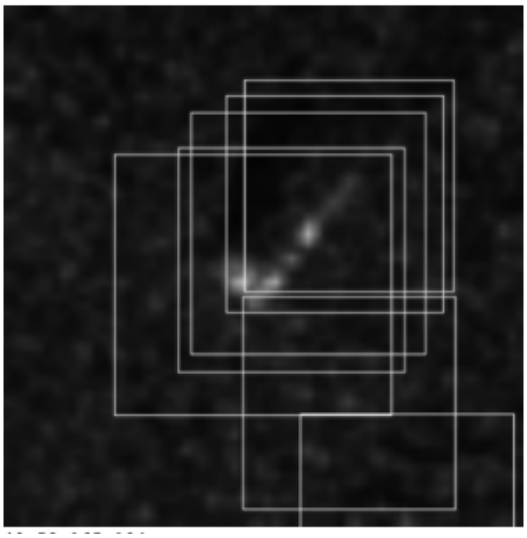

Figure: Increasing the size of each detected box

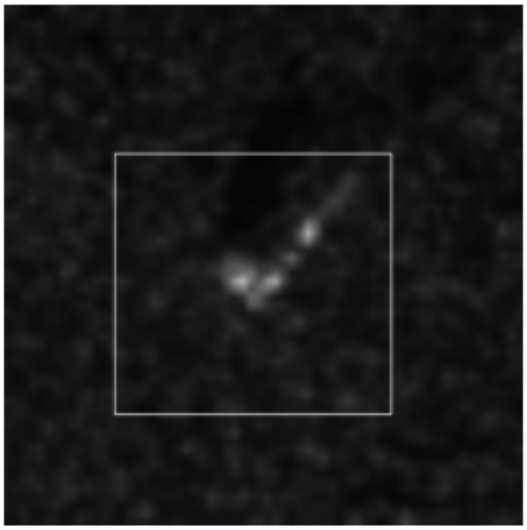

Figure: Choosing the box with the largest area

In spite of this, when the accuracy of the model did not increase for some classes we curated the entire dataset. In doing so we realised that there were some images in which bounding boxes partially contained the object and in some rare cases there was no object in the image.

For such cases, labelImg tool was used in which bounding boxes were corrected manually by going through the whole test and training dataset.

Models for Object Detection

Using RetinaNet

Implementation:https://colab.research.google.com/drive/1HtVfIfCN5Czh-iumEHGSHJXof8mxxX63?usp=sharing

```
2021-03-22 17:36:06.178699: I tensorflow/stream_executor/platform/default/dso_load
2021-03-22 17:36:06.395707: I tensorflow/stream_executor/platform/default/dso_load
Running network: 100% (2746 of 2746) |####| Elapsed Time: 0:11:49 Time:  0:11:49
Parsing annotations: 100% (2746 of 2746) || Elapsed Time: 0:00:00 Time:  0:00:00
232 instances of class 1 with average precision: 0.3292
233 instances of class 2 with average precision: 0.2989
299 instances of class 3 with average precision: 0.4372
299 instances of class 4 with average precision: 0.4929
299 instances of class 5 with average precision: 0.3951
299 instances of class 6 with average precision: 0.8641
298 instances of class 7 with average precision: 0.3194
298 instances of class 8 with average precision: 0.7093
233 instances of class 9 with average precision: 0.2111
256 instances of class 10 with average precision: 0.3958
Inference time for 2746 images: 0.2419
mAP using the weighted average of precisions among classes: 0.4580
mAP: 0.4453
```

Figure: Training the model

```
2021-03-22 20:01:41.284452: I tensorflow/stream_executor/platform/default/dso_loader
Running network: 100% (2435 of 2435) |####| Elapsed Time: 0:10:23 Time:  0:10:23
Parsing annotations: 100% (2435 of 2435) || Elapsed Time: 0:00:00 Time:  0:00:00
196 instances of class 1 with average precision: 0.3616
196 instances of class 2 with average precision: 0.2544
274 instances of class 3 with average precision: 0.4124
274 instances of class 4 with average precision: 0.5298
284 instances of class 5 with average precision: 0.4120
274 instances of class 6 with average precision: 0.8554
273 instances of class 7 with average precision: 0.3034
274 instances of class 8 with average precision: 0.6982
195 instances of class 9 with average precision: 0.2004
195 instances of class 10 with average precision: 0.3531
Inference time for 2435 images: 0.2404
mAP using the weighted average of precisions among classes: 0.4568
mAP: 0.4381
```

Figure: Prediction on testing data

Retinanet gave very low accuracy. One of the possible reasons for this can be setting the batch size very low(4). Upon increasing the batch size, the model was not getting trained due to memory and GPU limitations.

Tiny-YOLOv4

Implementation: https://github.com/aayush-v/SAR-ATR/tree/main/YOLOv4_SAR

Initially, the dataset was trained with no preprocessing

```
 detections_count = 3218, unique_truth_count = 2435
class_id = 0, name = T72, ap = 99.98%           (TP = 196, FP = 7)
class_id = 1, name = BTR60, ap = 94.24%         (TP = 182, FP = 21)
class_id = 2, name = D7, ap = 100.00%           (TP = 274, FP = 4)
class_id = 3, name = 2S1, ap = 99.99%           (TP = 284, FP = 6)
class_id = 4, name = BRDM_2, ap = 99.27%        (TP = 265, FP = 10)
class_id = 5, name = T62, ap = 99.98%           (TP = 264, FP = 1)
class_id = 6, name = ZIL131, ap = 99.90%        (TP = 274, FP = 10)
class_id = 7, name = BMP2, ap = 99.94%          (TP = 195, FP = 6)
class_id = 8, name = ZSU_23_4, ap = 99.99%      (TP = 274, FP = 3)
class_id = 9, name = BTR70, ap = 97.71%         (TP = 187, FP = 16)

 for conf_thresh = 0.25, precision = 0.97, recall = 0.98, F1-score = 0.97
 for conf_thresh = 0.25, TP = 2395, FP = 84, FN = 40, average IoU = 79.70 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.990993, or 99.10 %
Total Detection Time: 9 Seconds
```

Figure: Class wise mean average precision observed on training using YOLOv4

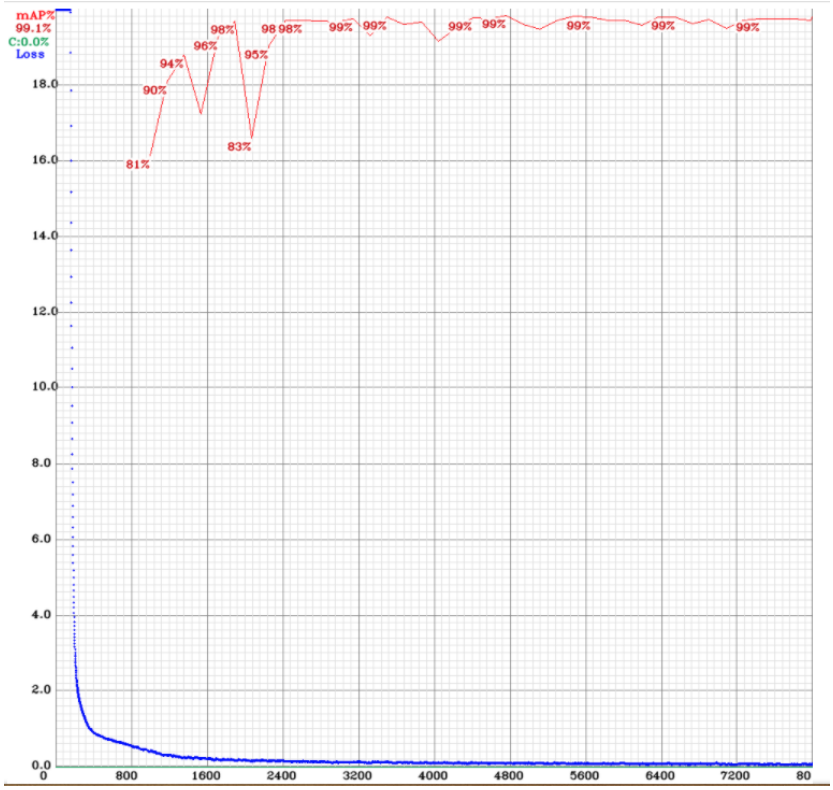

Figure: Change in accuracy and loss with increasing number of iterations

It was observed that on training the MSTAR dataset with tiny-YOLOv4, we achieved high accuracy on average. Testing the entire test set took approximately 20 seconds. However, there were a few classes that highly underperformed as compared to the other classes- namely BTR60 and BTR70.

At this point, we curated the entire dataset on labelImg and performed some data augmentation techniques to the classes that weren't evenly distributed.

The augmentation techniques used were Rotation by an angle, flipping the images and translation. There were improvements observed but they were only marginal.

```
 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
2436
 detections_count = 3218, unique_truth_count = 2435
class_id = 0, name = T72, ap = 99.98%           (TP = 196, FP = 7)
class_id = 1, name = BTR60, ap = 94.24%         (TP = 182, FP = 21)
class_id = 2, name = D7, ap = 100.00%           (TP = 274, FP = 4)
class_id = 3, name = 2S1, ap = 99.99%           (TP = 284, FP = 6)
class_id = 4, name = BRDM_2, ap = 99.27%        (TP = 265, FP = 10)
class_id = 5, name = T62, ap = 99.98%           (TP = 264, FP = 1)
class_id = 6, name = ZIL131, ap = 99.90%        (TP = 274, FP = 10)
class_id = 7, name = BMP2, ap = 99.94%          (TP = 195, FP = 6)
class_id = 8, name = ZSU_23_4, ap = 99.99%      (TP = 274, FP = 3)
class_id = 9, name = BTR70, ap = 97.71%         (TP = 187, FP = 16)

 for conf_thresh = 0.25, precision = 0.97, recall = 0.98, F1-score = 0.97
 for conf_thresh = 0.25, TP = 2395, FP = 84, FN = 40, average IoU = 79.70 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.990993, or 99.10 %
Total Detection Time: 9 Seconds
```

Figure: Class wise mean average precision observed on training using YOLOv4

The YOLO model failed to show any major improvements in spite of the data augmentation.

Faster RCNN

Since FasterRCNN is known to outperform YOLO in terms of accuracy, we tried using FasterRCNN with the same dataset to get a better idea about the quality of the dataset and the labelling.

The following graphs show the progress while training for 50 epochs.

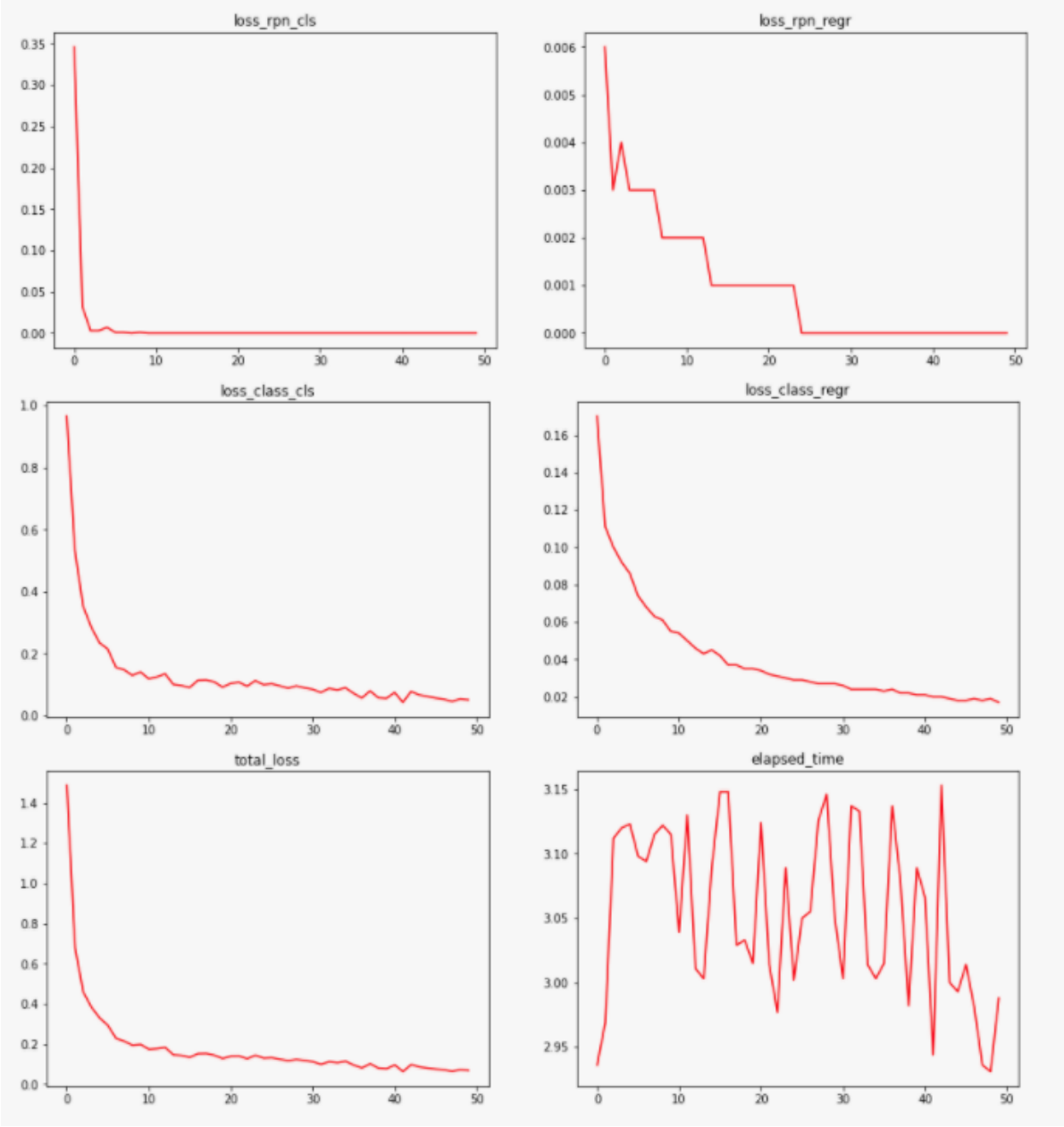

Figure: Progress graphs for Faster RCNN while training

Some observations that can be made from the above graphs is the difference in the fall of loss between the region proposal network's classification (whether background or not) and regression(the IoU of the bounding box). The classification loss smoothly reduces, while the regression loss reduces in steps. We believe this is due to a large amount of noise in the dataset.

After training for 50 epochs we got the following results:

```
Elapsed time = 1.6154000759124756
BTR70 AP: 0.998211946624139
BRDM_2 AP: 0.9912004300023132
BTR_60 AP: 0.9818174166398848
ZSU_23_4 AP: 0.9926441309282013
D7 AP: 0.9887679601222616
T62 AP: 0.9930902596926581
2S1 AP: 0.9988155620520416
ZIL131 AP: 0.9827157415557404
mAP = 0.990907930952155

mean average precision: 0.9905805614662272
```

Figure: Class wise mean average precision observed on training using FasterRCNN

Observations:

- The average precision increased to 0.9905
- The performance of both BTR60 and BTR70 has significantly increased and is comparable to other classes.
- The time to test each image is approximately 1.6 seconds and approximately takes an hour to iterate through the entire test set.

These observations indicate that it is possible to achieve much higher accuracy than was being achieved by the YOLO model. But the biggest drawback is that it takes nearly an hour to iterate through the entire dataset as compared to 20 seconds for YOLO. Hence the speed tradeoff is very large. Our aim was to achieve a precision similar to the FasterRCNN model while not compromising the speed.

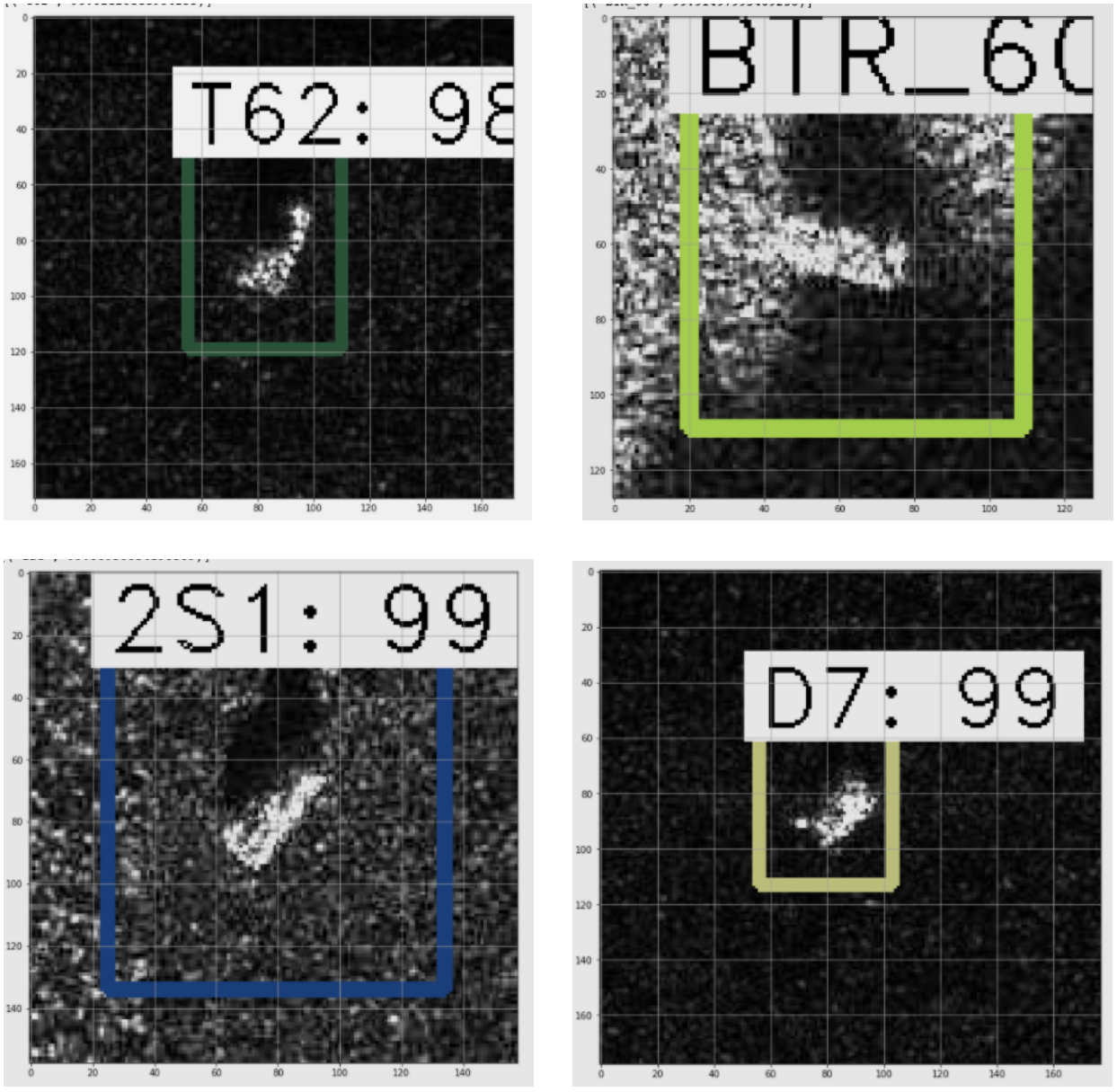

Figure: Sample predictions made by FasterRCNN model

Preprocessing

As we observed that due to a large amount of noise in some images, the YOLO model did not converge as well as FasterRCNN. The FasterRCNN model faced some issues too as can be observed in the generalisation of the RPN for regression.

To tackle the noise, we performed the following preprocessing steps on MATLAB.

- Change in the brightness and contrast of the images
- Used the lee filter
- Weighted superimposition of the above two changes to form a new image
- Gaussian Filter

```
Y = uint8(Lee_Filter(I2,I2,[5 5]));
% figure, imshow(Y)

addedLee = imadd(Y, added/3);
% figure, imshow(addedLee)
aL = im2double(addedLee);
% figure, imshow(aL)
% whos aL

Iblurred = imgaussfilt(aL,2);
```

Figure: Code snippet for preprocessing

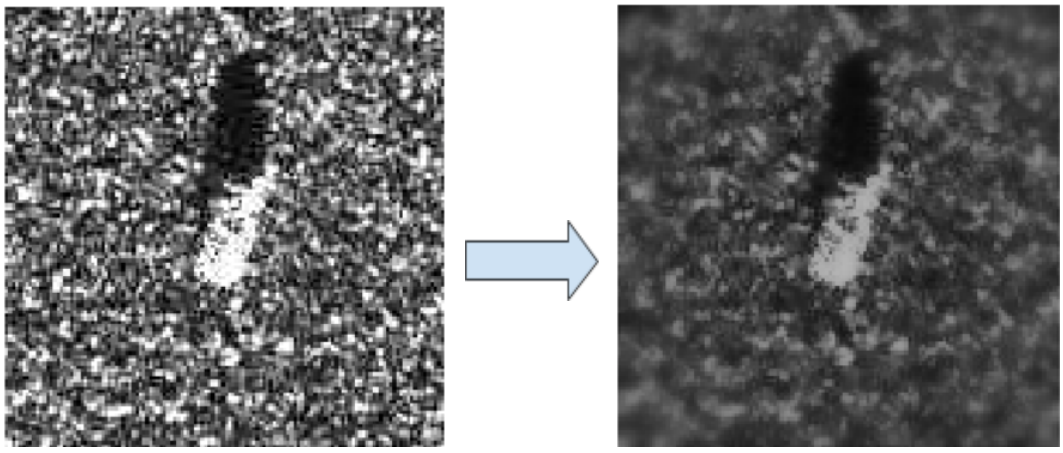

Figure: Preprocessing of an image

YOLOv4 on Preprocessed data

To work on the speed attained by FasterRCNN we tried to implement the YOLO algorithm on the new dataset.

The results of the training were as follows:

```
 detections_count = 2735, unique_truth_count = 2111
class_id = 0, name = 2S1, ap = 99.85%           (TP = 274, FP = 41)
class_id = 1, name = BRDM_2, ap = 99.99%        (TP = 273, FP = 2)
class_id = 2, name = BTR60, ap = 99.96%         (TP = 183, FP = 0)
class_id = 3, name = D7, ap = 100.00%           (TP = 274, FP = 9)
class_id = 4, name = BTR70, ap = 100.00%        (TP = 274, FP = 0)
class_id = 5, name = T62, ap = 99.95%           (TP = 270, FP = 5)
class_id = 6, name = ZIL131, ap = 99.99%        (TP = 274, FP = 12)
class_id = 7, name = ZSU_23_4, ap = 99.99%      (TP = 273, FP = 2)

 for conf_thresh = 0.25, precision = 0.97, recall = 0.99, F1-score = 0.98
 for conf_thresh = 0.25, TP = 2095, FP = 71, FN = 16, average IoU = 84.74 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.999664, or 99.97 %
Total Detection Time: 9 Seconds
```

Figure: Class wise mean average precision on training using YOLOv4

Observations

The average precision of classes BTR60 and BTR70 is much higher than before and is comparable to other classes too. The entire testing time took around 20 seconds on average as compared to an hour for FasterRCNN.

The accuracy achieved by YOLO is comparable to FasterRCNN too.

Conclusion

Faster RCNN gave better results in terms of average precision in comparison to YOLO on the original dataset. However, YOLO gave much faster predictions on the Test set.

Faster RCNN took almost an hour to calculate the complete mean average precision(mAP) value on the test image dataset of 2400 images, whereas YOLO took just 20 seconds for the same task. Hence initially there was a tradeoff between accuracy(measured in mAP) and time(measured in seconds) on shifting between the two models.

However, after applying the respective preprocessing techniques mentioned above in the report, and then using YOLO on the preprocessed dataset, we were able to increase the accuracy of YOLO to the extent of Faster RCNN along with the fast speed of YOLO(20 seconds).

Thus there can be improvements achieved in ATR for SAR images by performing specific preprocessing techniques to clean the images.

Future Scope

Object detection in the domain of SAR images is not explored much. Hence there is a lot of scope for improvements. This project can be further extended to include these preprocessing techniques in the original architecture of YOLO itself, specifically designed for SAR target recognition with better accuracy along with the fast speed of YOLO. This project was specifically directed towards the tank targets of MSTAR dataset.

Various other SAR datasets, such as the ship dataset, can be used to examine the applicability of these techniques further and whether there can be some more preprocessing techniques used.

A more detailed study can be done to see the effects of the preprocessing techniques on the bounding box creation function.

References

[1] C. Tison, N. Pourthie, and J.-C. Souyris, “Target recognition in SAR images with support vector machines (SVM),” in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, Barcelona, Spain, Jul. 2007, pp. 456–459.

[2] Q. Lv, Y. Dou, X. Niu, J. Xu, J. Xu, and F. Xia, “Urban land use and land cover classification using remotely sensed SAR data through deep belief networks,” *Journal of Sensors*, vol. 2015, 2015.

[3] Z. Lin, K. Ji, M. Kang, X. Leng, and H. Zou, “Deep convolutional highway unit network for SAR target classification with limited labeled training data,” *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 7, pp. 1091–1095, 2017.

[4] Z. Huang, Z. Pan and B. Lei, "Transfer Learning with Deep Convolutional Neural Network for SAR Target Classification with Limited Labeled Data," Remote Sensing, vol. 9, no. 9, p. 907, 2017.

[5] J. I. Park, S. H. Park and K. T. Kim, "New Discrimination Features for SAR Automatic Target Recognition," in IEEE Geoscience and Remote Sensing Letters, vol. 10, no. 3, pp. 476-480, May 2013.

[6] Geng, J.; Fan, J.; Wang, H. High-resolution SAR image classification via deep convolutional autoencoders. IEEE Geosci. Remote Sens. Lett. 2015, 12, 2351–2355.

[7] Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Trans. Pattern Anal. Mach. Intell. 2017, 39, 1137–1149.

[8] Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.\

[9] Ding, J.; Chen, B.; Liu, H.; Huang, M. Convolutional Neural Network With Data Augmentation for SAR Target Recognition. IEEE Geosci. Remote Sens. Lett. 2016, 13, 364–368.