Introduction

Detecting prohibited items in X-ray images is a crucial task for security maintenance. We use a 1% subset of the **SIXray dataset**, consisting of **192 X-ray images** (.jpg), each paired with an **XML annotation** detailing object labels (e.g., guns, knives), bounding box coordinates, and image dimensions.
While CNNs have made great developments in object detection, they fail to perform with occlusions. STNs offer the capability of adding a layer of spatial invariance for the detection model to localise the images for concealed objects. Thus the research question: **Can STNs, combined with different neural networks, be adapted to improve the detection and localization of prohibited items in X-ray images with varying occlusion levels?** While STNs show promise in improving detection through adaptive transformations, their performance across varying occlusion levels needs further investigation.
Notations: **CNN** - Convolutional Neural Networks, **STN** - Spatial Transformer Networks, **DF** - Dataframe, **RFB** - Receptive Field Block Networks, **BB** - Bounding Boxes, **ROI** - Region of Interest. **h/v/hvflips** - Horizontal/Vertical/Horizontal-Vertical flips. **NMS -** Non-maximum suppression (controls overlap of detected boxes). **mAP** - Mean Average Precision **IoU** - Intersection of Union. **HP** - Hyperparameter.

Literature Review

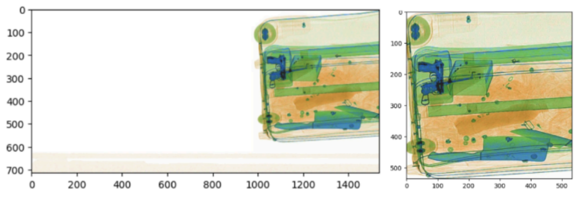
We select the SIXray dataset. Standard preprocessing steps like resizing, whitespace removal, and feature construction enhance model robustness. Additionally, curated augmentations such as erasing, blocking, and blurring help adapt the model to real-life object detection scenarios [1]. We take inspiration from STNs and CNNs and apply them to customise for STN +FasterRCNN model [3]. We studied the different types of STNs - affine, projective and TPS from implemented tutorials [5] which informed our decision to use "affine" transformation for the STN model. We also incorporate RFBNet [4] into our approach. RFBNet extends the receptive field of convolutional layers in a way that allows the model to better capture multi-scale context. This makes it particularly suited for detecting objects with varying sizes. For example, a gun is much smaller in size than a plier.

Methods

a. Preprocessing

We converted XML annotations to CSV and image data into tensors for efficient processing in Pytorch.

Image resizing and white space removal: We observed that the images contained a significant amount of background whitespace, which detracted from the focus on the objects of interest. To address this, we developed a **custom algorithm that dynamically identifies and removes the white background** from each image. This process effectively zooms in on the area containing the object, ensuring that only the relevant portion—such as the luggage—remains visible. This adjustment enhances the prominence of the objects being detected, improving the model's ability to focus on them. **It can adapt to any input image, making it versatile and robust for varying image types**. First, the image is divided into segments based on a specified width (seg_wid). For each segment, the get_whiteness function calculates the percentage of near-white pixels. Segments exceeding a defined threshold (v_whiteness_threshold for vertical, h_whiteness_threshold for horizontal) are considered background and removed. The remaining segments are retained, ensuring only the relevant portions of the image remain. The get_whiteness function adjusts for lighting conditions and calculates the proportion of white-like pixels using a tolerance level. Additionally, the BaseTransform class resizes images and adjusts brightness by adding or subtracting mean RGB values based on the desired transformation ("light" or "dark"). This algorithm effectively focuses on objects by removing unnecessary white spaces.


Before and After implementing whitespace removal on an image

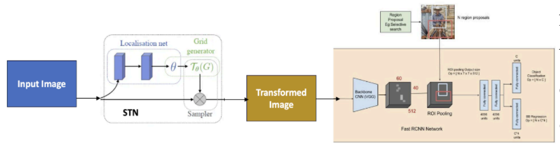
Augmentation: *Colour Jitter*, randomly alters brightness, contrast, saturation and hue introducing variability, and making the model more robust to different lighting conditions across X-ray machines. This helps to standardise the colour and brightness variations. *Gaussian Blur*, applied to smooth image noise, introduces a controlled level of image blur. X-ray images may appear with varying sharpness depending on focus and resolution, so adding controlled blur encourages the model to adapt to these natural variances. *Flipping*, hflips, vflips and hvflips have been performed. This enabled the model to learn rotational invariances, making it more effective in recognising prohibited items oriented differently. *Gaussian Noise* is a technique that boosts the model's robustness by helping it learn to filter out noise. Research shows that noise injection improves performance by making neural networks resilient to minor perturbations [1]. *Random Erasing*, selectively removes part of an image randomly by altering the pixel values of the target area. As noted in the research, this technique significantly enhances adaptability in classification tasks [1]. *Random Blocking,* a customised *Hide and Seek* [1] algorithm where we randomly block an image with a mask having a random

size (provided no detectable item is hidden) and blend the mask into the image for it to be perceived as a seamless distortion in the image. We blend this mask by incorporating the mean pixel values of the image and adding Gaussian blur to blend the edges of the block.
Each of these augmentations is selected based on its potential to mirror challenges in real-world X-ray imaging. More instances of techniques like Random Erasing and Hide and Seek were added than the others because they actively force the model to adapt to various occlusion levels, which is a crucial element of our research question. Furthermore as shown in studies (see [1] and Table 2), these augments have one of the highest accuracy, especially on the ResNet-50 backbone models.
Class Balancing: We applied weights inversely proportional to class frequency, which makes sure that less frequent classes have higher weights. This approach is chosen to prevent bias in the model and enable it to learn from minority classes ("scissors" and "knife" classes in our dataset).
Normalising and One Hot Encoding: Min-Max normalisation is conducted to ensure pixel values range between 0-1 thereby helping the models to perform better since only one consistent tensor range is to be focused on. Test and train data have similar mean and variances therefore consistent methodology is applied. We apply one-hot encoding to the "class" feature where it is specified which prohibited item is present in the image. This feature contained 5 values [‘scissor’, ‘plier’, ‘knife’, ‘wrench’, ‘gun’].
Data Cleaning and Conversion: We dropped unnecessary columns that contain metadata like ‘depth’, ‘height’, ‘width’, ‘pose’, ‘truncated' and ‘difficult’ from the annotations DF and columns like ‘mode’ from the image DF. These attributes are irrelevant to our detection task. The models focus solely on BB locations and class labels. We also merge and convert these DFs to tensors with a custom data loader. This is necessary to ensure compatibility with Pytorch’s tensor-based pre-processing.
Training Set occlusion levels (feature generation): To evaluate model performance across varying levels of occlusion, we developed a custom algorithm to identify overlaps within each image. The algorithm calculated the areas covered by 1, 2, 3, or more overlapping objects for every image. Using this information, we computed a weighted average to derive an occlusion index for each image, where a higher occlusion index indicates a greater degree of overlap among objects. This process was implemented dynamically, allowing the algorithm to calculate this metric for any given input image. Our custom made algorithm computes occlusion levels by building a graph of overlapping bounding boxes for each image. Each object is represented as a node, with its bounding box as an attribute. The algorithm iteratively checks for overlaps between pairs of nodes using the calculate_intersection_area function. A DF is created, listing each polygon, its area, and the number of overlaps it represents. This DF is used to quantify the occlusion levels, providing insights into the extent of overlaps in each image. Finally, we take the weighted sum of squares of the polygon area with the number of occlusions to calculate the final occlusion level per image.

b. Three Algorithms

FasterR-CNN (baseline): This model was chosen due to its powerful architecture that integrates feature extraction, regional proposal generation, and classification in a unified end-to-end framework. The model backbone is **ResNet-50**, which was selected for its depth and computational efficiency. It supports feature extraction at multiple levels making it suitable for complex patterns in X-ray imagery. The **RPN** produces high-quality regions of proposals faster than traditional methods. We defined custom anchor sizes, aspect ratios and **NMS thresholds** that help configure the RPN, helping in detecting objects of various shapes and scales within the image. Modifications like the use of **ROI pooling** help streamline the detection of smaller objects. A multi-task loss function is implemented which uses **cross-entropy** and **Huber loss** (classification and BB regression) to penalise inaccuracies adjusted to the class weights calculated.

STN-Faster RCNN (medium): Traditionally STN is used along with CNN as the object detection head; however since our problem statement deals with multiple classes and bounding boxes per image, Faster-RCNN was deployed for the benefits it offers for multi-class object detection as mentioned above. We apply the STN layer along with the Faster-RCNN model here. STN has a **localization network** with a simple CNN with ReLU activations and max-pooling layers to extract features from the input image which will inform the subsequent transformation to be applied. Then there is an **affine predictor** created by two fully connected layers with ReLU activations; we initialise weights and bias for this matrix in the first transformation. From this part, we get the **affine matrix** which helps transform the image coordinates. Now, STN with affine transformation first utilises several types of transformations - rotate, scale, translate and shear. Post this, an ‘**affine grid**’ is created which contains a grid of transformed coordinates and ‘**grid sampling**’ which contains the transformed image. This transformed image is then **sent to the Faster -RCNN** model (similar implementation as algo 1) for object detection. STN learns and updates its parameters (weights and biases for affine matrix) based on the loss calculated by the Faster R-CNN during the backpropagation process therefore ensuring an end-to-end training model. We chose this implementation as it



Demonstrative example for STN FastRCNN[4] assists the detection model by feeding it transformed images with slight variations (affine selected for slight variations).

RFBNet (difficult): RFBNet (Receptive field Block Network) was chosen due to its ability to balance accuracy and speed and was published after the first two papers. RFBNet's architecture enhances feature extraction by using Receptive Field Blocks (RFBs). This mimics the human visual system's varying receptive fields, thereby allowing the network to capture multi-scale features and context effectively. The core of the architecture includes a backbone like VGG or MobileNet for initial feature extraction. This is followed by RFB modules that enrich spatial information before feeding into detection heads for predicting bounding boxes and class probabilities. The RFB modules are designed to better capture object details at different scales, which is particularly valuable for detecting objects of varying sizes in a dataset. This aspect of the RFB net was vital for us since we were detecting weapons that had varying sizes. The primary challenges faced when applying RFBNet to non-square images. So we resize all images to squares of 512 x 512 for this model. Customisations included adjusting the anchor sizes and aspect ratios to better match the objects in the dataset, as well as fine-tuning the learning rate and loss function to improve convergence. These adjustments helped RFBNet maintain a balance between accurately detecting objects and minimising computational overhead, making it well-suited for the given problem.

c. 20-Fold Cross Validation

We implemented 20-fold cross-validation by splitting the dataset into 20 distinct subsets and iteratively training the model on 19 subsets while validating the remaining one. This provides a high-resolution performance assessment than methods like bootstrapping. However, this approach is very computationally intensive but offers better model consistency and reduced bias in performance estimates. In each fold, we identify the optimal epoch by evaluating validation loss, resulting in 20 distinct prediction sets that allow us to thoroughly assess the performance metric for each model.

d. Hyperparameter Tuning

We implemented a stratified k-fold approach for HP tuning, ensuring that each fold maintains a balanced representation of all classes across the dataset. This stratified approach is crucial for object detection tasks, as it prevents class imbalance within individual folds, leading to more reliable and generalisable HP optimisation. The nested (inner) loop uses a grid search mechanism, where we tune based on each combination of HP of the model. Here we utilise the mAP as the primary metric to identify the configuration of the best HPs. This is a widely used evaluation metric in object detection, as it captures both precision and recall across all detected objects. The outer loop evaluates the tuned model using Recall@IoU (secondary metric), which focuses on the recall aspect at an IoU threshold of 0.5. We use Recall@IoU to refine our model due to the critical nature of detecting all potential threats; minimising the risk of missing any prohibited items. This focus helps reduce false negatives.

HPs tuned: FasterRCNN - NMS Threshold, anchor sizes. STN Faster RCNN - anchor sizes. RFBNet - anchor sizes, aspect ratios, and NMS threshold.

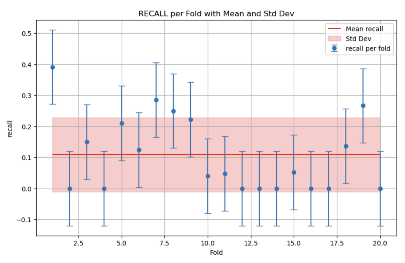
<u>**Results and Discussion**</u>

Our evaluation metrics include the mean and variance mAP, precision, recall and f1 score from all folds. Alongside these, we have also compared the training and inference time for these models. We set an epoch count below 10 and used a batch size of 3, balancing between training duration and model convergence. These settings were chosen to accommodate the computational constraints that were faced throughout this project. For each fold, we chose the best epoch by validation loss. We tested our test set on the best models and configurations that were yielded during HP tuning.

a. Performance Analysis for Each Algorithm

Faster RCNN - Mean mAP:0.0343; Recall:0.1089; Precision:0.1291; F1Score:0.0981; Training time/fold:58.54 s (Paid GPU);

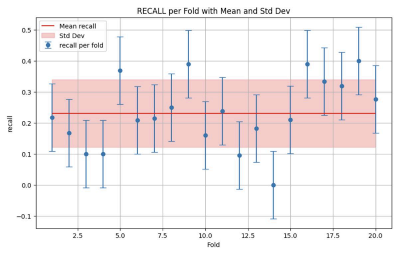
The low mAP value indicates that the model struggled to generalise and detect objects accurately. Due to insufficient exposure to varied objects within the small number of images. After further analysis of the test images, we found out that the object "plier" was detected comparatively more accurately than the other classes, which made sense due to the "plier" class being overrepresented in the training data. This behaviour of the model decreases the precision and slightly inflates recall value (relative to mAP) for certain classes suggesting that the model occasionally managed to detect objects, but not very precisely. For the false positives, we observe that the BBs are somewhat accurately localised within the regions of the image containing the objects, although they struggle with classification. The figure shows the error bars for recall per fold in this model.



STN + Faster RCNN - Mean mAP: 0.1162; Recall:0.2308; Precision: 0.067, F1 Score: 0.0986; Training time/fold: 319.70 s (Paid GPU);

The mAP of ~0.11 suggests the model is facing difficulty in localising the objects leading to inaccuracy in detecting overlap of interested regions and incorrect classifications

The model seems to only predict ~23% of the actual objects which implies there are many misses in detection and a need for improvement in generating bounding boxes. While recall is higher than mAP here i.e. since the model is producing a large number of predictions, some are correct (relatively high recall) but many are false positives (low precision). On further investigation, we observe that mostly, STN is not able to generalise on transformations and many a time converges to the identity transformation matrix suggesting that the STN layer is not able to learn properly to feed to the Faster RCNN model due to data insufficiency.



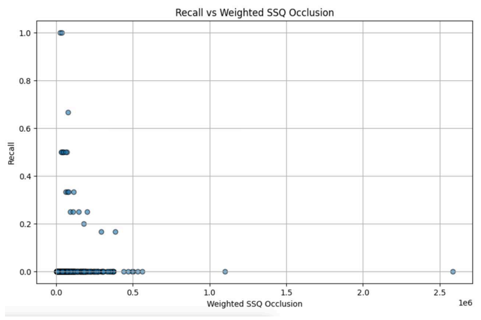
RFB net yielded low (~0.0) results in all metrics. This is primarily due to the small dataset size of only 192 images used for training. This limitation arose from the computational constraints, which forced us to use only a small subset of all the images in the training dataset. With such a limited number of images, there was insufficient data to effectively train the model, especially for object detection tasks that require a variety of samples to learn object features robustly. This led to poor generalisation, which caused it to miss the objects. For RFB net to yield more meaningful results we need to train the model with larger datasets which require much larger computational power.

b. Experimental Results

Performance metric vs Occlusion Index (Scatter plot)

In the experiment conducted, we study the recall of the model vs weighted occlusion levels present in each image (refer to feature generation) and we run this experiment for the test dataset.

From the plot, we can see that there is some trend; recall decreases as occlusion increases (for positive recall) suggesting that even complex deep learning models will face difficulty in making predictions in object detection in images as the level of occlusion increases.



<u>Conclusion</u>

We see that by comparing the performance of the STN model to the non-STN model, the STN model has higher recall which can be indicative of the transformations aiding the detection head in making better predictions and localization. However, since the dataset is small, we would need to train our models on a larger dataset to accurately answer this question. Moreover, we observe in the STN model that many transformations are trying to converge to identity matrices which indicates that since STN transforms the whole image and we are dealing with a multi-class problem with occlusions, we would require an even more localised approach.

Alternative approaches considered for the project.

Spatial Transform Decoupling in conjunction with Faster RCNN could be an alternative approach where we use STN methodology but transform only the area of interest in the image instead of transforming the entire image [7].

As an alternative to grid search, bayesian optimisation could have proven to be a more efficient alternative, especially for our project's high computational costs. By focusing on promising areas in the search space it reduces the number of evaluations needed to find the optimal HP.

Github - https://github.com/pranavpai/Statisitcal_Machine_Learning_Project

Pkl files (data) link - https://drive.google.com/drive/folders/1RRFwa6JHmrV9S3zGDXhhDLK1yzVvrUU3?usp=sharing

<u>References</u>

[1] Kumar, T., Mileo, A., Brennan, R., & Bendechache, M. (2023, January 7). *Image Data Augmentation Approaches: A Comprehensive Survey and Future Directions*. arXiv.org. https://arxiv.org/abs/2301.02830

[2] *Study of object detection based on Faster R-CNN*. (2017, October 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/8243900

[3] Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015, June 5). *Spatial transformer networks*. arXiv.org. https://arxiv.org/abs/1506.02025

[4] Liu, S., Huang, D., & Wang, Y. (2017, November 21). *Receptive field block net for accurate and fast object detection*. arXiv.org. https://arxiv.org/abs/1711.07767

[5] STN implementation https://pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html

[6] Hongtian Yu1, Yunjie Tian1, Qixiang Ye, Yunfan Liu (2024, Feb 22), *Spatial Transform Decoupling for Oriented Object Detection* https://arxiv.org/html/2308.10561v2