Passenger Screening Model to Improve TSA's Threat Recognition in Airports.

Capstone Project

Shivaraju Gowda
January 6th, 2018

- Kaggle Competition Website
- Kaggle Result
- Code on Github

I. Definition

Project Overview

Airport security is critical and necessary part of safe travel. As anyone who has traveled on an airplane knows, this comes at the cost of long lines and wait times at the airport security checkpoint. The U.S. Transportation Security Administration (TSA) is responsible for U.S. airport security, they screen more than two million passengers a day.

TSA is very aware of their conflicting goals of thorough security screenings and shorter wait times to keep their customers safe and happy at the same time. When the sensors predict a threat, they need to engage in a secondary, time consuming manual screening process. The high false alarm rates with the current screening models creates significant wait times at the airports. They also anticipate increase in the number of travelers and new threats every year, so the prediction models need to be more accurate and continually improve their threat detection model.

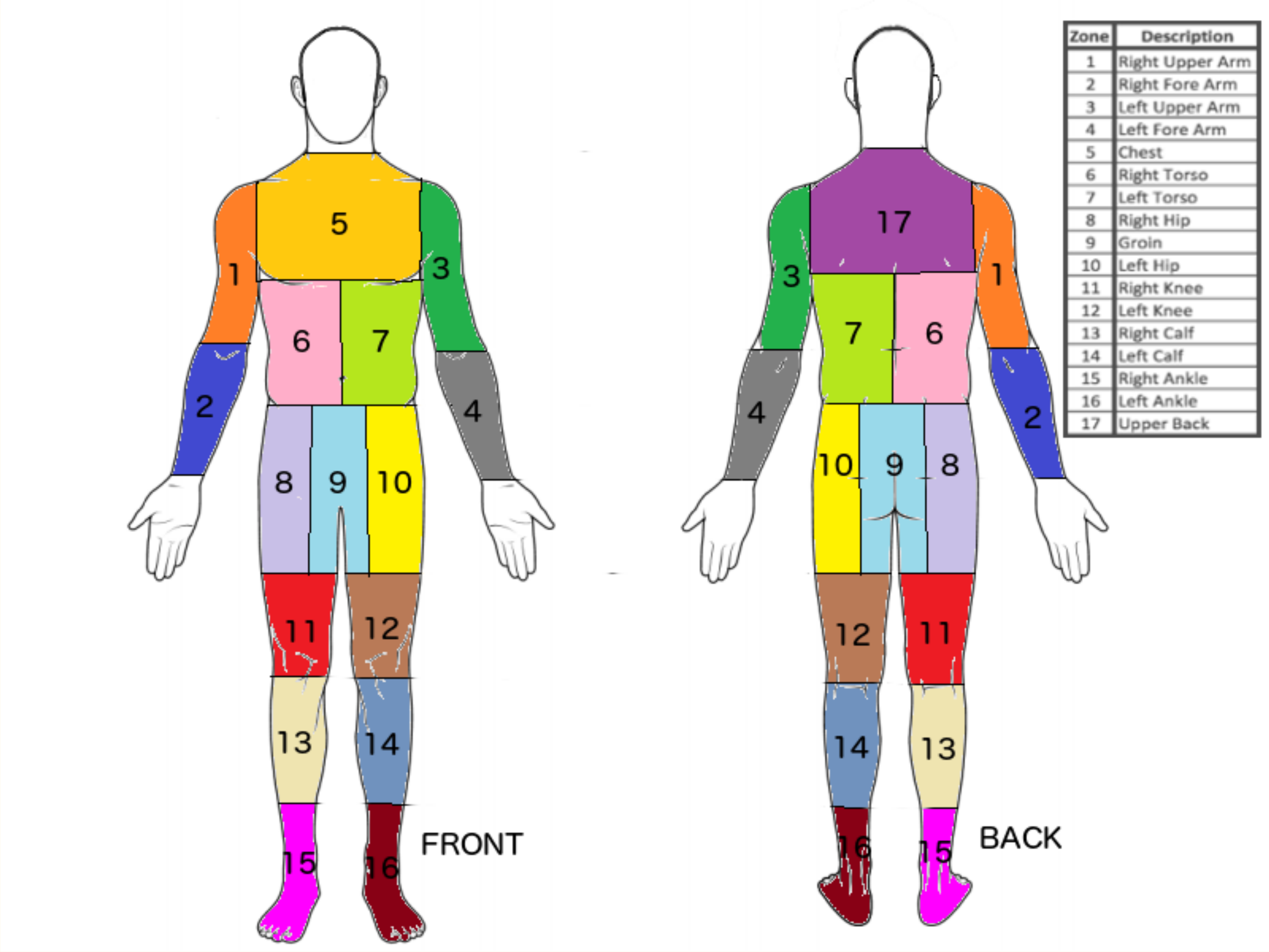
As of now, TSA purchases algorithms exclusively from the manufacturers of the scanning equipment. Like other enterprise software, these algorithms are proprietary, expensive, and have long release cycles. Because of the importance of accuracy and improved threat detection model to their bottom-line and the current success of Machine Learning in similar tasks, TSA has organized a Kaggle competition to see if the broader Machine Learning community can come up with a better model to detect threats during passenger screening. This competition has the biggest dataset and the highest prize($1.5 Million) of all the Kaggle competitions hosted till now.

Problem Statement

The main task of the competition is to use the output of the current sensors and develop algorithms/models to predict threats with increased accuracy. The model should also be nimble to adapt to new threats.

To train and evaluate the models/algorithms in this competition, sensor dataset for training and testing are provided for several subjects. The dataset is designed to capture real scanning conditions. They are comprised of volunteers wearing different clothing types for different seasons, different body mass indices, and different genders.

The body zones are defined as in this image. There are 17 zones in total.



Zone	Description
1	Right Upper Arm
2	Right Fore Arm
3	Left Upper Arm
4	Left Fore Arm
5	Chest
6	Right Torso
7	Left Torso
8	Right Hip
9	Groin
10	Left Hip
11	Right Knee
12	Left Knee
13	Right Calf
14	Left Calf
15	Right Ankle
16	Left Ankle
17	Upper Back

The threats come in different forms and shapes. They can be small handguns, knives, electronics, smuggled goods, etc. The model has to differentiate the threats from the clothing accessories.

The end goal is for the model to predict the probability of a threat present in each of the body zones of a given subject. A subject can have multiple threats or no threats at all.

Metrics

Log loss is used as an evaluation metric for this competition. If there are N subjects in a test set, the model should make 17N predictions(17 body zones x N). Submissions are scored based on the formula:

$$-\frac{1}{17N}\sum_{i=1}^{17N}[y_i log(\hat{y_i}) + (1-y_i)log(1-\hat{y_i})],$$

where:

- N is the number of subjects in the test set
- $\hat{y}$ is our predicted probability of a subject having a threat in a given body zone.
- y_i is the true label for a given subject and body zone. It is 1 if a threat is present, 0 otherwise
- $log()$ is the natural (base e) logarithm

The probabilities are truncated at $max(min(p, 1 - 10^{-15}), 10^{-15})$. A smaller log loss is better.

Because of the nature of the log loss evaluation metric, it is not just accuracy of our model that matters but also the ability to correctly predict the confidence we have in our prediction. Predicting with high confidence a false prediction is heavily penalized.

Solution Approach and Prior Research.

At it is core this project is a computer vision problem of detecting an object in a video and then classifying it. In this case the frames of the video are also well defined since they are evenly spaced around a subject. Here is some prior research and approaches I came across while researching the solution for this problem.

- Multi-view Convolutional Neural Networks for 3D Shape Recognition[1]
- Object Detection from Video Tubelets with Convolutional Neural Networks[2]
- Tensorflow Object Detection API[3]

Prior to 2014, methods like Haar Cascade using OpenCV were used to solve such problems however lately Neural Net methods(CNN) have shown better results and are predominantly used to solve these kinds or problems. The TensorFlow Object Detection API came out early in 2017 and is very well documented and comes with lots of examples. I decided to use TensorFlow Object Detection API for my approach to this project.

II. Analysis

Datasets and Inputs

The input dataset contains a large number of body scans of a number of subjects at different granularity levels from a millimeter wave scanner. The scan for each subject consists of the following four binary files:

- _.ahi = calibrated object raw data file (2.26GB per file)
- _.aps = projected image angle sequence file (10.3MB per file)
- _.a3d = combined image 3D file (330MB per file)
- _.a3daps = combined image angle sequence file (41.2MB per file)

We could choose to either use the most detailed scan or the most coarse one to make our prediction. The trade-off is the time taken to process the images for training and inference, the most detailed format is 2.26 GB vs. 10.3 MB for the most coarse one. The aps format is the least detailed scan with 16 frames of images equally spaced radially. Each frame is a monochromatic array of size 512 x 660.

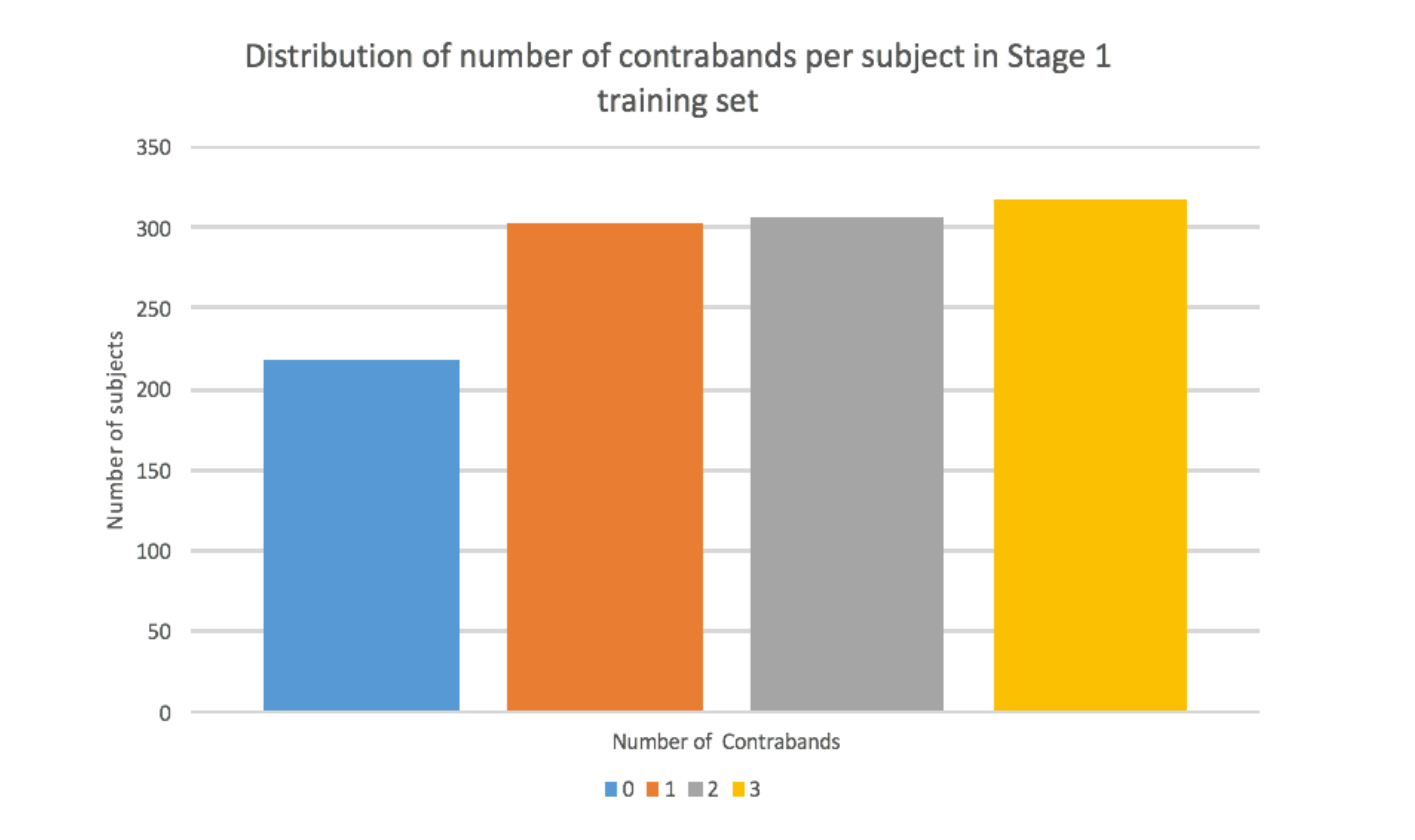
The data is sensitive because of the nature of the images. Due to the confidentiality agreement I cannot display them here. It is also quite large in size and could be found on the Competition Website.

Data Exploration

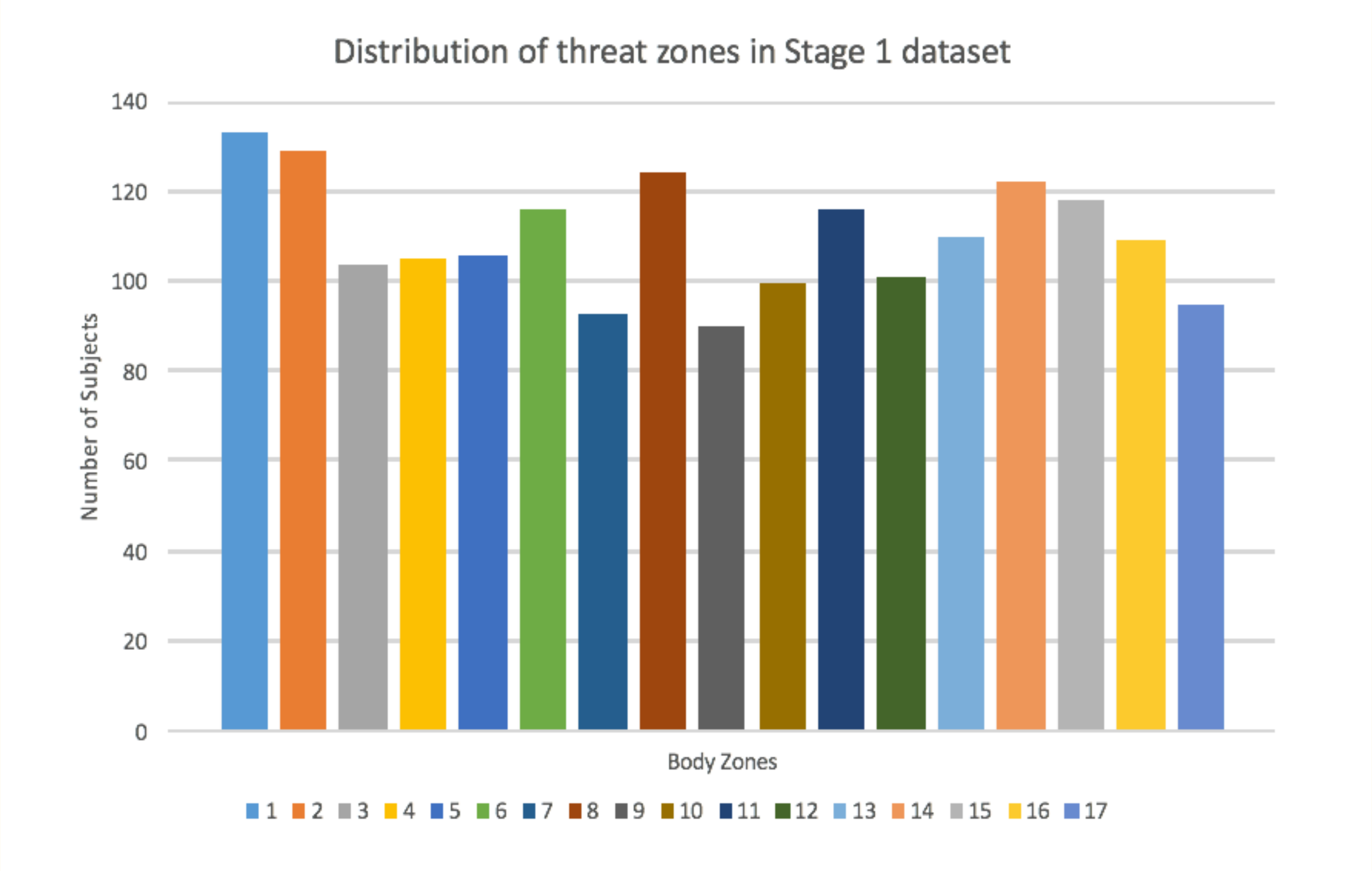
This is a 2 stage competition with the following number of subjects in each stage.

- Stage 1
  - Training : 1147 subjects.
  - Testing : 100 subjects.
- Stage 2
  - Testing : 1388 subjects.

In Stage 1, there were a total of 24 unique individuals. There were 14 threats which were incorrectly labelled in the training set and I corrected them for my analysis. Stage 2 individuals were different from Stage 1. Stage 2 data was provided towards the end of the competition for scoring in the private leaderboard. So only Stage 1 data was used for training.



The above graph shows the distribution of the number of threats in a subject in the Stage 1 dataset. We can see that the maximum number of threats per subject was 3. The number of subjects having 0 to 3 threats were more or less equally distributed.



The above graph shows the distribution threats in different body zones of the subjects in Stage 1 dataset. Again, the threats seem to be equally distributed in different body zones in the dataset.

Algorithms and Techniques

To come up with a model for this project, I used these two pieces of information.

1. Recent success in object detection in an image using Machine Learning.
2. Relatively consistent human body ratios on the vertical axis.

The first will help us detect the location of a threat in an image. It will also be used to detect the location of head/hands and groin. The second will help in figuring out the body zone the threat belongs to using the relative position of the threat with respect to head, hands and groin. These two operations will be performed for each of the frames in a scan. The final logic will aggregate the predictions of all the frames to make a final prediction of the probability of threats in 17 body zones of the subject.

Object Detection

In this section, I am going to briefly cover the technology underlying the first stage of the proposed model- Object Detection in an image. Given an image, Object detection is an algorithm to detect the class of an object and its bounding boxes(usually a tuple containing the top-left top and bottom-right coordinates) in the image, there can be more than one object in an image. The bounding boxes are the added complexiety in Object Detection over the traditional image classification problem.

Object Detection algorithms build on top of the image classification components. In the simplest Object Detection algorithm(R-CNN) a neural net based image classifier is run over an externally computed crops(bounding boxes) of the input image. More sophisticated algorithms refine the way the bounding boxes are computed, they reformulate the detection of bounding boxes as a learning of offsets over a predefined anchors for the image. The following are the two architectures based of these approach.

1. Single Shot Detector (SSD)
2. Faster R-CNN

SSD based approaches use single feed-forward convolutional network to predict classes and the anchor offset in one pass through the network. Because of the single pass, they are faster for inference but lack in precision when compared to other architectures.

In Faster R-CNN, the detection is done in three stages. First, the input image is sent half-way through a image classification module. Next, the features at this intermediate level are used to construct class agnostic box proposals(tied to some learning algorithm). At last, the cropped images using box proposals are fed through the remainder of the image classification module to output the classes.

To accommodate different precision, domain, training time and inference time requirements, TensorFlow provides Object Detection algorithms based on the combination of the above architectures, the underlying image classification neural nets(VGG, RESNET, INCEPTION) and pre-trained datasets (ILSVRC, COCO). I plan to benchmark the following algorithms for my use-case.

- ssd_mobilenet_v1_coco
- faster_rcnn_resnet101_coco
- faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco

Once implemented, this module will provide the bounding boxes for head, hands, groin and threats in an image

Zone Prediction using Supervised Learning

Based on the output of the Object Detection module, if there are more than one threat in an image then the model has to predict the body zones of those threats. This task will be done using a supervised learning algorithm and I plan to use XGBOOST for this purpose.

XGBOOST[4] “Extreme Gradient Boosting” is an implementation of gradient boosted decision trees. It is known for its speed and scalability. The input to the XGBOOST will be :

- Frame Number.
- Bounding box for the head.
- Bounding boxes of hands.
- Bounding boxes of groin.
- Bounding boxes of threat.

Based on the above inputs, the XGBOOST model will be trained to predict the body zone(1-17) of the threat.

Benchmark

As discussed in the above section, my model will have two steps for detecting the threats in body zones of a subject.

1. Detect Head, Hands, Groin and Threat bounding boxes using Object Detection API
2. Body Zone prediction of a threat using the bounding boxes of threat, Head, Hands and Groin obtained in

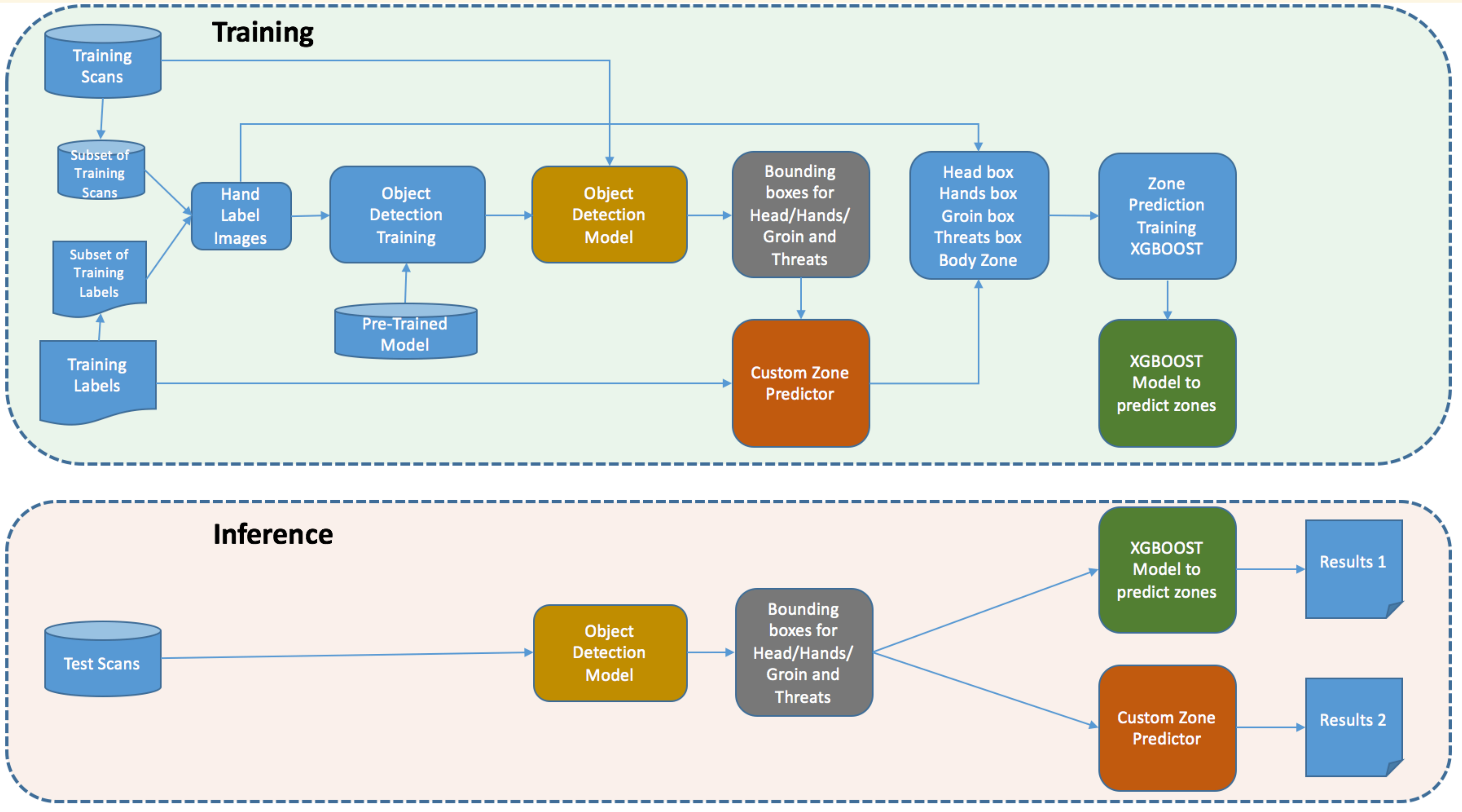
the above step.

In order to accomplish 1, I will need a very accurate object detection model. The recent release of TensorFlow Object Detection API shows an accuracy of 80%, I plan to leverage it. Since there are 16 or more frames of a subject at different angles, I should be able to get more accuracy than this baseline.

There might be a large variation in human body parts ratios horizontally, but human body parts ratios vertically are reasonably consistent. Professional artists usually depend on the size of the head to draw a proportional human figure. I plan to leverage this information to detect the zones a threat bounding box belongs to based on the locations of Head, Hands and Groin and the frame number. I will need to experiment with the exact ratios, but because of the relative sizes of objects involved, I should be able to get a good prediction even with reasonable estimates for the ratios. I also plan to experiment with training an XGBOOST based ML model to predict the zone in this step.

A naive predictor which predicts the mean probabilities of threats for all zones in Stage 1 training set results in a log loss of 0.29089 for Stage 1 test set, so my model should aim to get (way) better than this result.

III. Methodology


Training
Training Scans
Subset of Training Scans
Subset of Training Labels
Training Labels
Hand Label Images
Object Detection Training
Pre-Trained Model
Object Detection Model
Bounding boxes for Head/Hands/ Groin and Threats
Custom Zone Predictor
Head box Hands box Groin box Threats box Body Zone
Zone Prediction Training XGBOOST
XGBOOST Model to predict zones
Inference
Test Scans
Object Detection Model
Bounding boxes for Head/Hands/ Groin and Threats
XGBOOST Model to predict zones
Custom Zone Predictor
Results 1
Results 2

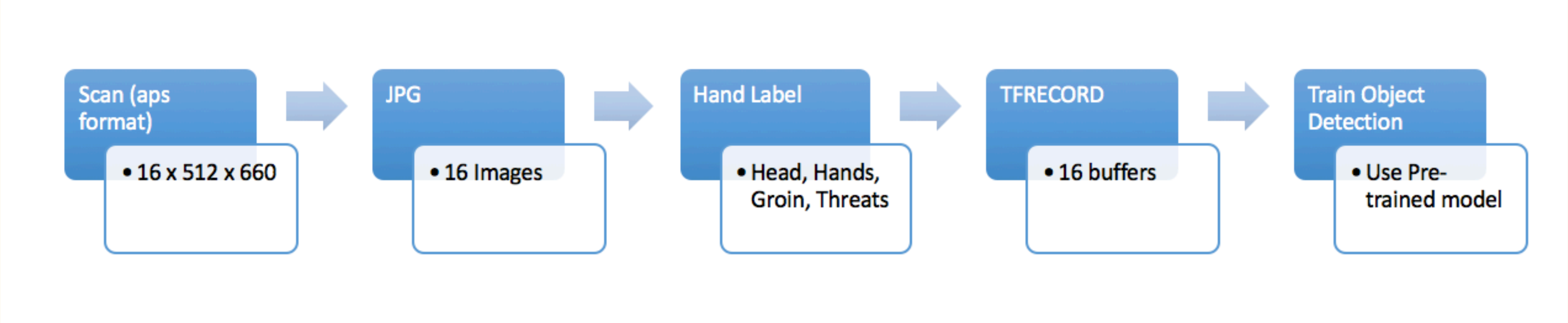
Machine Learning Pipeline

The above diagram illustrates the machine learning pipleline used for this project. The training step involves training two machine learning models: Object Detection and Zone Prediction. For inference we first find the bounding boxes for head/hands/groin and threats and use it to predict the body zones the threats are present in.

Object Detection.

Data Preprocessing

Here is the high-level flow of data preprocessing needed for object detection.



As discussed in the "Datasets and Inputs" section, the input scan comes in different granularity levels. I visualized the scans at different levels and I found that in the aps format(the coarsest one) I could see most of the threats. Only in approximately 1-2% of the cases I couldn't see the threats especially in the groin area. The computational needs of more detailed scans were also way too much to handle for my hardware budget. Furthermore, the increased time for analyzing such detailed scans would hinder more experimentation, so I made a decision to work with aps input format only.

The TensorFlow Object Detection API takes input images in 3 channels(RGB). Our input is monochromatic. Even though converting the input to color is adding no more information, I decided not to experiment in this area and converted the monochromatic array to jpg images using matplotlib viridis color map.

The jpg images were hand labelled using RectLabel Software. During this stage the following body parts were labelled:

- Head
- Hands
- Groin
- Threat (Infact, threats were labelled as body zones the threat belongs to)

The threats were labelled as body zones they belong to so that the same data could also be used in XGBOOST Zone Predction model described in the follow on sections.

The hand labelled images and annotations were later parsed and converted to TFRecord format. The body zone labels for threats were programatically converted to one label called "threat". The image/object detection libraries usually detect well defined single objects, like a cat or a dog for example. In our case, the threat can come in different shapes and sizes. It is infeasible to come up with a list of all threats upfront. So it will be interesting to see if the models can capture different threats under one label. My initial experiments showed this should be possible and I used the same label for all threats.

Implementation

My experiments showed CPU only local desktop was 10x slower than a GPU for training the Object Detection even after compiling TensorFlow with all the optimizations enabled. To use a GPU, I could either buy a new desktop or use Cloud infrastructure. The Google Cloud is well suited for TensorFlow Machine Learning workload. The Google Cloud Engine could also run this training on multiple GPUs. So even though it was a learning curve for me I choose to leverage $300 Google Trial Credit and use the Google Cloud ML Engine for training object detection and do its inference.

The TensorFlow Object Detection module was used as a black box in this stage. I did not implement the algorithms, rather I used the pre-trained models(which can be downloaded online) with the relevant configurations to enable a particular algorithm. The details of how to use the object detection module can be found here. We had to provide the following inputs.

- Hand labeled images in TFRecord format for training and validation.
- Configuration file which specifies the algorithm, location of the input images, hyper parameters for the algorithm, total number of iterations and few other other related variables.
- Pre-trained model

The TensorFlow Object Detection API is well documented and a related paper[5] discusses the tradeoff of accuracy vs. speed for many of the models it supports. I benchmarked the following algorithms for my use case and picked "Faster RCNN RESNET 101" based on the following results.

- Number of training images = 2230
- Number of test images = 669
- Hardware : Nvidia Tesla K80 on Google Cloud

	SSD MobileNet	**Faster RCNN RESNET 101**	**Faster RCNN Inception RESNET**
Training Time (20K Iterations)	4hr 39 mins	3hr 2mins	9hr 36 mins
Precision for Threats	0.5669	0.611	0.511
Overall Precision	0.877	0.893	0.796
Inference (for 100x16 Images)	5.75 mins	56.76 mins	NA

Refinement

I experimented with the following hyper-parameters for Faster RCNN RESNET 101 and choose the optimal parameters.

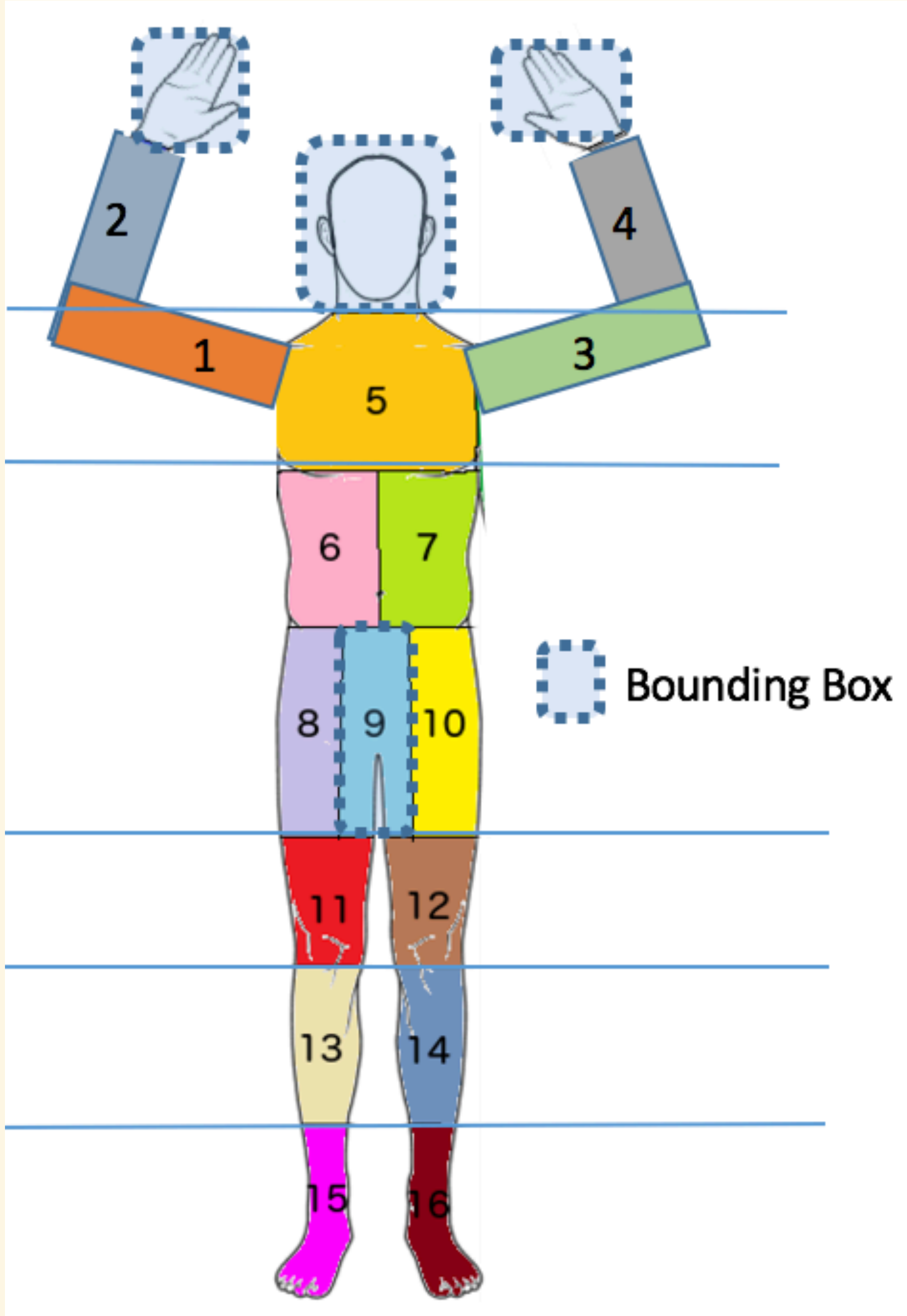
- Stride (8 and 16)
- Max Proposals (10, 20, 50, 100, 300)
- Image Resolution (312 x 312, 512 x 610)
- Data augmentation (random_horizontal_flip)

I started with 1600 images(100 subjects) and checked the results every 20K iterations and added more images to the test set based on the results. I ended up stopping at a total of 220K iterations with 3000 training images. In the end the Head/Hands/Groin were detected with a precision of more than 98%. The threats were detected with a precision of 80%.

Zone Prediction.

Once the bounding boxes for head/hands/groin and the threat are detected, the next task is to predict the zone the threat belongs to.

Custom Body Zone Predictor.



As we can see in the image above, the vertical zone a threat belongs to can be deciphered to a reasonable degree using the following formulas with assumptions on only four ratios(ForeArm_ratio, Chest_ratio, Knee_ratio, Ankle_ratio).

- Upper Body = Groin_Start - Head_End
- Lower Body = Image_End - Groin_Start
- Upper Arm = Between Hands and Head_End
- Fore Arm = Between Head_End + Upper Body * ForeArm_ratio
- Chest = Between Head_End and Head_End + Upper Body * Chest_ratio
- Torso: Between Chest and Groin.
- Hip: Groin.
- Knee: Groin_End + Lower Body * Knee_ratio.
- Calf: Knee + Lower Body * Ankle_ratio.
- Ankle: Between Calf and Image_End.

The ratios can be tweaked during the training stage to have a minimal error for all the training data.

Once the vertical zone is detected, the side on which a threat is present can be deciphered using the centerline of head or groin and the frame number. Based on the vertical zone and the side we can come up with the exact body zone a threat bounding box belongs to.

I needed a few iterations to get the vertical ratios right, but once I fine tuned them, the model predicated zones with greater than 95% accuracy for mirrored zones and in forward and backward facing frames. The accuracy was lower for side(or near-side) frames and for zones 5, 17 and 9(chest, upper back and groin). I did some hand tuning for zone 5, 17 and 9 with some success. I also marked a threat across frames by comparing the vertical points of the bounding boxes and put them in the same bucket, this helped improve the accuracy a bit but I still had some wrong predictions.

XGBOOST model to predict zone.

The above custom algorithm approach gave a decent solution to the problem of zone detection. However, there were still some errors in zones 5, 17 and 9. So I tried a supervised learning algorithm using XGBOOST.[6] XGBOOST is a distributed gradient boosting library. The input for the XGBOOST model was generated through the following ways:

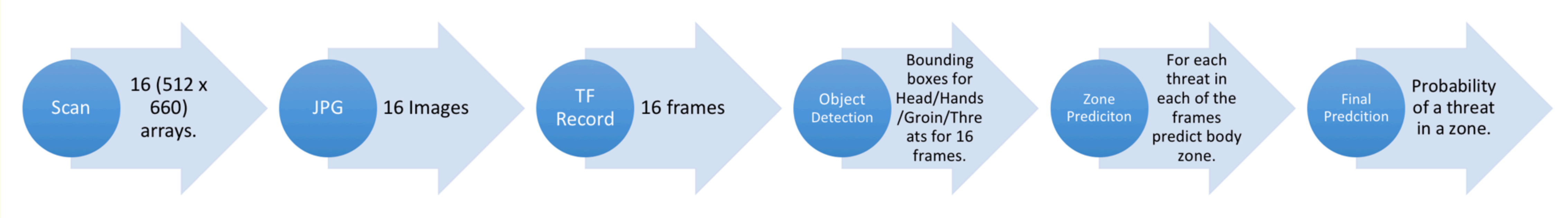
1. The Labeled input data in "Training Object Detection" where the actual threats are hand labeled as body zones and swapped out with one label as "threat" just before inputting to TFRecord.
2. Use the above custom zone predictor to predict body zone for all the training data and use only the correctly predict zones as input to XGBOOST.

The input consisted of frame number, bounding boxes for head, hands, groin and the threat. The label was the body zone the threat belong to. The following hyperparameters were tried for the XGBOOST model.

- n_estimators = [200, 300, 400, 500, 600]
- max_depth = [4, 6, 8, 10]

The XGBOOST model gave very high accuracy of more than 98% on the cross validation set.

Inference


Scan
16 (512 x 660) arrays.
JPG
16 Images
TF Record
16 frames
Object Detection
Bounding boxes for Head/Hands /Groin/Threats for 16 frames.
Zone Prediciton
For each threat in each of the frames predict body zone.
Final Predcition
Probability of a threat in a zone.

The overall inference model is shown in the above image. The inference was done in batches. The final stage aggregated the probabilities of all threats in all the frames and applied some heuristics to make a prediction on the probabilities of threats being present in each of the body zones of the test subjects.

I decided to use both Custom Zone Predictor and XGBOOST model for my overall predictions since Stage 2 allowed the participants to submit 2 entries.

Confidence level.

The model sets the confidence level based on the following factors.

- Probability of threat bounding box given by the Object Detection API.
- Number of frames the object appeared in (out of the total 16).

```
def getProbability(nFrames) :
    if nFrames >= 4:
        return 0.9908
    elif nFrames == 3:
        return 0.9174
    elif nFrames == 2:
        return 0.6157
    elif nFrames == 1:
        return 0.2135
    else:
        return 0.00184
```

IV. Results

The following results show the predictions of the model for test and validation set using the XGBOOST zone prediction. The "Correct predictions" column corresponds to the number of threats correctly predicted by the model. The "Incorrect predictions" are the number of body zones the model incorrectly predicted has having threats, usually this happens when the model incorrectly predicts a clothing accessory as a threat or when the threats on zone 5, 17 and 9 are misclassified as some other zone. The "Missed Threats" are the number of threats not detected by the model.

Results of the model on the test Set with the object detection probability > 0.05

Number of frames	Total number of threats	Correct predictions	Incorrect predictions	Missed threats
>= 5	1571	988	0	593
>= 4	1571	1266	1	305
>= 3	1571	1451	5	120
>= 2	1571	1538	32	33
>= 1	1571	1551	109	20

Results of the model on the test Set with the object detection probability > 0.5

Number of frames	Total number of threats	Correct predictions	Incorrect predictions	Missed threats
>= 5	1571	853	0	718
>= 4	1571	1143	1	428
>= 3	1571	1371	2	200
>= 2	1571	1496	11	75
>= 1	1571	1546	99	25

Results of the model on the validation Set with the object detection probability > 0.05

Number of frames	Total number of threats	Correct predictions	Incorrect predictions	Missed threats
>= 5	300	185	4	115
>= 4	300	235	5	65
>= 3	300	277	7	23
>= 2	300	288	15	12
>= 1	300	290	33	10

Log Loss = 0.0353

As can be seen in the result, the more the number of frames a threat appears in, the more likely it is being correctly predicted. The probability of the Object Detection module also can be used to improve the prediction but at the cost of missing some threats. So the confidence level of the model's prediction is set based on the these two factors.

For the validation set I got a log loss of 0.0353.

Kaggle Results

With the above the setup I got the following results(log loss) in the competition:

1. Stage 1
   - Using Custom Zone Predictor : 0.052
   - Using XGBOOST : 0.045
2. Stage 2
   - Using Custom Zone Predictor : 0.08630
   - Using XGBOOST : 0.10631

Overall ranking : 20th out of 518 teams, Silver Medal

The XGBOOST model probably overfit to the test data. In retrospect this is understandable since there were only 24 unique subjects in the the training data and those individuals also participated in the Stage 1 test data. Stage 2 consisted of entirely different individuals. So in Stage 2, the Custom Zone Predictor performed better than XGBOOST.

Since I based my confidence levels on the test data I was slightly more optimistic of the model detecting threats. If I had set the confidence level of 0 frames as 0.05 instead of 0.00184, I would have got a log loss score of 0.08278 with a ranking of 17 instead of 20.

When compared to the benchmark model of mean probabilities of threats(log loss 0.29) my model does a good job of predicting the threats on a bigger(than the training set) and unseen dataset. The training is done only on 24 unique individuals, I expect it to generalize even better when trained on more individuals and threats.

V. Approaches by Other Teams

The competition completed on December 15th, 2017, since then many of the participants have shared their solution approach. The successful approaches boiled down to detecting the threats and mapping them to the zones(consistent with my approach). I was surprised to find that (just like myself) most of them did not find value in using higher granularity data input and used only the aps format files. The variations in the approaches differed in the following ways(in the order of priority).

- Single model for object detection and zone prediction (Multi-view Convolutional Neural Networks MVCNN)[7].
- Unique individuals in Cross Validation Set.
- Data augmentation.
- RGB color channels.
- Segmentation.
- Ensemble methods.

The MVCNN type approaches where a single model was used to detect the threat and predict the zone showed up higher in the rankings. This is understandable since such approaches not only reduced the error rate because of the single model but also required less data preprocessing, allowing participants more time to experiment in other areas.

As I mentioned in Data Inputs section, there were just 24 unique individuals in the test dataset. My validation set consisted of randomly picking the images from the entire set. Participants who were careful to place scans of unique individuals in the same set for cross validation had better confidence predictions for their results.

Data augmentation approaches involved using a horizontal flip, stretching of the images. One approach (competition winner) went even further to create 800 more datapoints using a combination of images and threats in image manipulation software(GIMP).

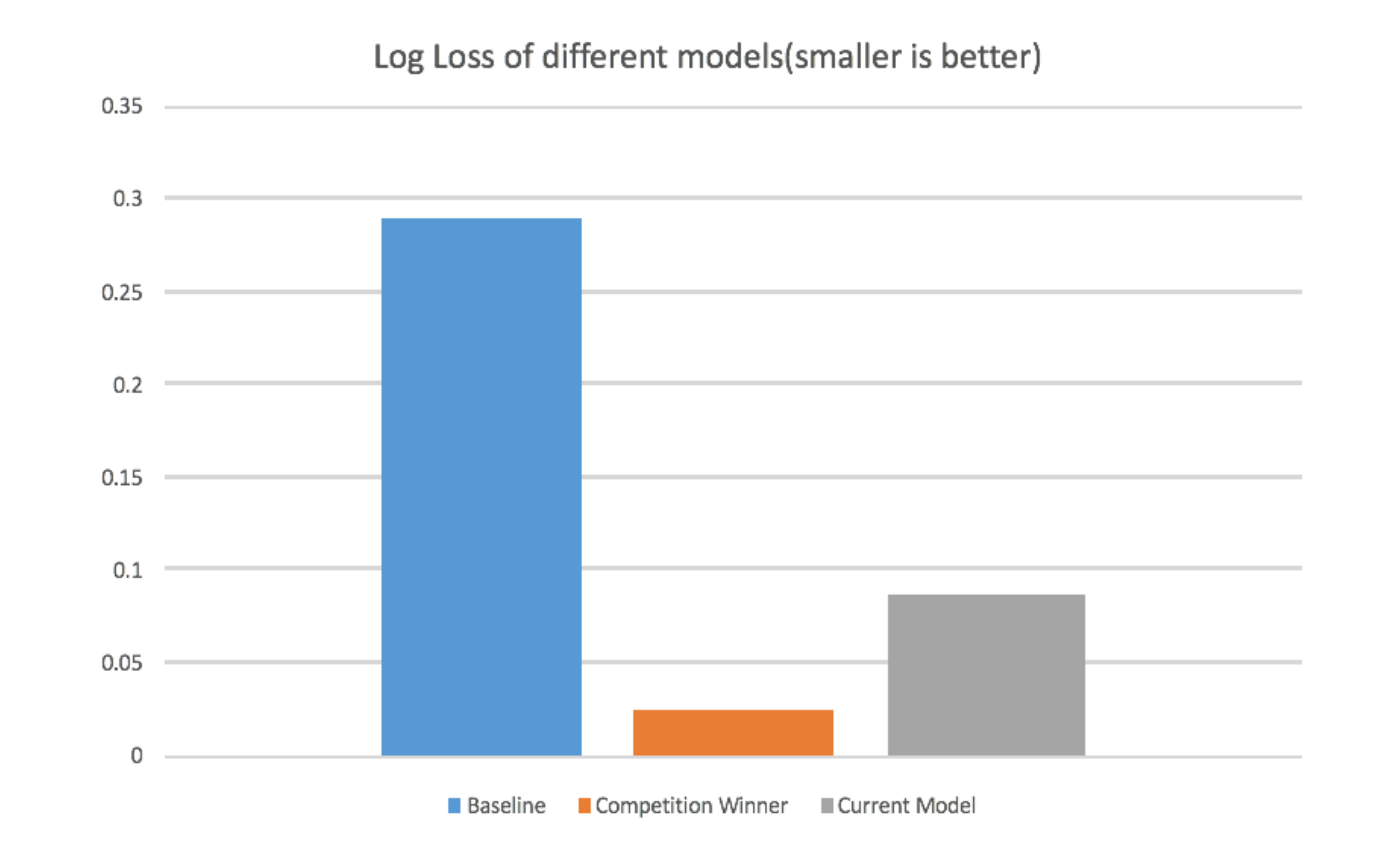
The input data is monochromatic, however, the object detection modules require 3 color channels as inputs. I mapped the monochromatic input to a viridis colormap using matplotlib and fed it to the Object Detection module. The winner of the competition had an ingenious way of using the 2 redundant channels, he took the average and standard deviation per pixel of the ten similar scans of the subject and placed them in the other two channels. The threats stood out in the red channel because of this.

Some participants used segmentation for their approach, such approaches yeilded reasonable results but it looked like this approach came with substantial increase in effort and time for data preparation.

A few other approaches used ensemble methods, they might not be using entirely different models but similar models with different set of parameters.

VI. Conclusion

In summary, the model used machine learning to detect bounding boxes for objects(threats, head, hands and groin) in a series of images. A supervised learning algorithm(XGBOOST) was trained and used for predicting the body zones a threat belongs to based on the relative positions of bounding boxes of a threat to the head, hands and groin.



This model is 4 times better then just predicting the average of the threats in the body zones(baseline). The winning result of the competition is 3.5 times better than my model as seen in the image above. I think the following enhancements would have helped my model bridge the gap between the winning result.

1. Place unique individuals in one set in cross validation setup during training. In my tests, I randomly picked the images for validation and hence there was leakage(because there are just 24 unique individuals) which resulted in an overconfident model.
2. Use one model like MVCNN instead of combining two different models like I did since the errors in each stage multiply.
3. Make use of the RGB channels to input more information.
4. Train on more data with different individuals and different types of threats in the dataset. The result are impressive considering the fact that the training set was limited to only 24 unique individuals.

It was interesting to find that a single label could be used to detect different forms and shapes of threats.

This was my first Kaggle competition and I am happy with the results I got in applying the concepts learnt in the Udacity ML NanoDegree program. The first 8 members of the competition were "in the money" and I was off by just 12 rankings. Furthermore, if I had correctly tweaked the confidence level I would have been higher by 3 ranks.

1. Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition", Proceedings of ICCV 2015, http://vis-www.cs.umass.edu/mvcnn/↩

2. Kai Kang, Wanli Ouyang, Hongsheng Li, Xiaogang Wang, "Object Detection from Video Tubelets with

Convolutional Neural Networks", https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Kang_Object_Detection_From_CVPR_2016_paper.pdf↩

3. Tensorflow Object Detection API https://github.com/tensorflow/models/tree/master/research/object_detection↩
4. Tianqi Chen, Carlos Guestrin, "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754↩
5. Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors", https://arxiv.org/abs/1611.10012↩
6. Tianqi Chen, Carlos Guestrin, "XGBoost: A Scalable Tree Boosting System", https://arxiv.org/abs/1603.02754↩
7. Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition", Proceedings of ICCV 2015, http://vis-www.cs.umass.edu/mvcnn/↩