# Application of Deep Transfer Learning for Threat Detection in mm Wave Scan Images

Machine Learning Engineer Nanodegree: Capstone Project

Brayan Jaramillo
November 12, 2017

## 1 DEFINITION

### 1.1 PROJECT OVERVIEW

Numerous factors conspire to make flying a most uncomfortable experience for the nearly 2.6 million passengers who fly in and out of U.S airports every day.[1] Chief among these factors are the delays caused by the high false positive rates of the TSA's mm wave scanner screening technology.[2] Although the DHS does not publish information on these rates, several estimates do exist. Journalists at ProPublica covering "false-alarm rates" (false positive rates) at home and abroad found estimates ranging between 23-54% in Europe and 17%-38.5% in the U.S.[3]

In an effort to improve screening processes and technologies, the DHS has launched the Apex Screening at Speed (SaS) program. This program aims to develop technology that would "enable the scanning of walking passengers, acquiring data through most garments and reliably detecting a wider range of prohibited garments regardless of concealment".[4] After research done through this program identified false-positive rates as a significant contributor to bottlenecks at security checkpoints, the DHS created a Kaggle competition that challenges contestants to engineer better, and manufacturer-independent, algorithms.

The organizers of the competition provided a few different forms of the same data set, some with higher resolution than others. Generally, the datasets consist of ~1200 binary files containing headers, which provide technical details about the axes and increments of the scan, and the data itself, which consists of signal responses (in units of volts) at predefined spatial increments.[5] By considering small portions of the scans at a time, one can construct 2D slices that can be viewed as images of a subject from different points of view. Using collections of these images, one can employ machine vision techniques that detect when a threat is present.

### 1.2 PROBLEM STATEMENT

The DHS Kaggle challenge requires an estimate of the probability of a threat for each of 17 predefined zones on the body of each subject (see Figure 1). In this project, however, we

[1] (United States Department of Transportation 2017)
[2] (Department of Homeland Security 2017)
[3] (Grabell and Salewski 2011)
[4] (Department of Homeland Security n.d.)
[5] (Department of Homeland Security 2017)

will investigate the synthesis of deep transfer learning and classical machine learning techniques to solve a limited subset of this problem: *the classification of each subject as possessing **any** threat in the lower body (zones 8-16) or not*.

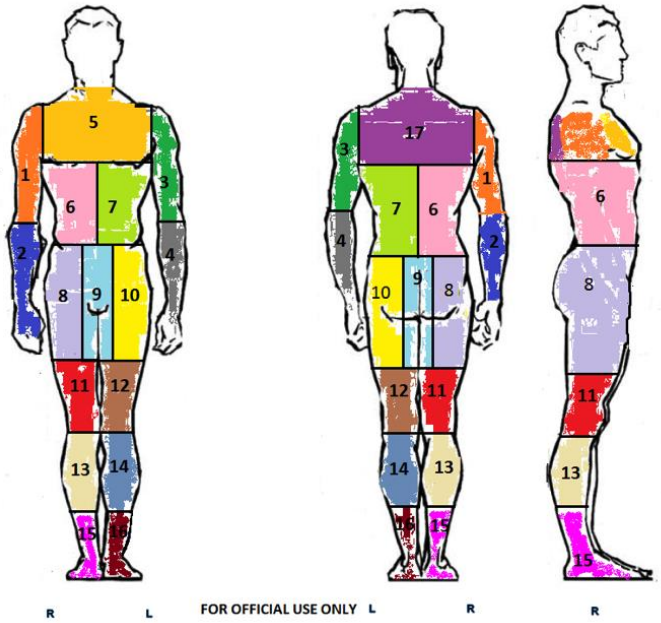


*Figure 1: Fourteen Threat Zones*[6]

More specifically, we would like to train a model that assigns a probability, $p_i$, of there being at least 1 threat present in each scan, $s_i$, of shape (400,360,16) such that the average log-loss over the set of scans, $S$, is minimized. The target label is constructed from the original label vector by applying the OR function to the subset of zones in the lower body such that:

$$y_i = OR(y_{i_z} \forall z \in \{8,9,10,11,12,13,14,15,16\})$$

The problem of classifying individual portions of the scan is vastly more complex than the binary classification of entire scans for several reasons. The most obvious is that the former is a multi-label problem while the latter is simply a binary classification. Multi-label problems required specialized approaches or the use of only certain algorithms, such as neural nets and decision trees, because, unlike multi-class problems, they can't be handled by the simple application of the Soft-Max function.

Another complication comes from the definition of the 17 zones. The zones are defined in terms of our conceptual understanding of body parts such as the "hand and forearm" or "below the knee." If the goal is to produce a probability for each zone, then one would have to either partition the zones programmatically or hope that the algorithm learns each zone as part of the training process. Pursuing a programmatic approach is difficult because the subjects are of different heights, genders, and weights. Most importantly, the classification of zones would have to be done from different angles where some zones are distorted or occluded all together. Feeding the entire scan to an algorithm and expecting it to learn not only what a threat looks

[6] (Department of Homeland Security 2017)

like but what each zone looks like with, and without, a threat increases the number of classes to be learned from 2 to 34; this must all be done with no supervised label of the zones and only 1115 samples.

To respect resource and time constraints, this project is limited to assessing the performance of transfer learning and classical machine learning approaches solely on the binary classification problem of detecting any threat in zones 8-16. After cleaning, cropping to the region of interest, and preprocessing, we will run 16 of 64 available angular slices (again, due to computational resource limits) through an Inception-V3 model pre-trained on ImageNet data and then concatenate and record the activations at the bottom 10 layers of the model to create a feature-rich dataset. We will split the feature-rich dataset of scans into an 80% training and 20% test set and train a Gradient Boosted Tree classifier on the training set. We will then compare the false-positive rate of the classifier to the "false-alarm" rates published by ProPublica. Finally, we will examine the importance of the extracted features and determine which layers of the Inception-V3 model are most relevant to this classification task.

## 1.3 METRICS

### 1.3.1 Log-Loss

The organizers of the competition chose the average Log-Loss as the metric to assess submissions. We will measure the log-loss of the Gradient Boosted Classifier to get a sense of its correctness and the degree of its conviction when making predictions. Ideally, we would want an algorithm that is correctly classifying scans with a high degree of conviction (e.g. 90%), rather than assuming the lazy epistemic position of near-random guessing (e.g. 51%).

Log-Loss achieves this by penalizing strong incorrect convictions exponentially more than weaker, incorrect convictions. The mathematical formulation for our binary classification problem is as follows:

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

$$where\ y_i = correct\ label\ and\ p_i = predicted\ probability\ for\ sample\ i$$

### 1.3.2 AUC

We would like to compare the false positive rate of our algorithm to the false positive rates reported by ProPublica.[7] Since, we only have access to the false-positive rates and not the recall, then a good way of determining whether our algorithm is viable is by plotting the ROC curve, measuring the AUC, and identifying whether our algorithm has both a high AUC (>0.80) and whether it has a high true positive rate in the range of the false positive rates of the algorithms currently in use (0.17-0.50).

The ROC curve plots the true positive rate, or recall, against the false positive rate at various thresholds settings. Using the curve, it is possible to easily choose a setting that

[7] (Grabell and Salewski 2011)

balances recall and the false positive rate. AUC is the integral of this curve, and it yields a one number summary of the curve which ranges from 0 to 1, with 0.5 indicating an uninformative classifier. The AUC may be interpreted as the probability that a randomly chosen positive instance will be given a higher "score" (in this case, probability) than a negatively chosen negative instance.[8]

# 2 ANALYSIS

## 2.1 DATA EXPLORATION

### 2.1.1 Introduction

As mentioned in *section 1.*1, the dataset provided by the organizers of the Kaggle competition comes in the form of binary files with headers and a body containing a data array representing a spatial distribution of signal responses in volts. Specifically, the data array consists of 512 x 660 2D images distributed through 64 angles spaced at 5.625° (See Figure 2 for a visualization of one of these slices).

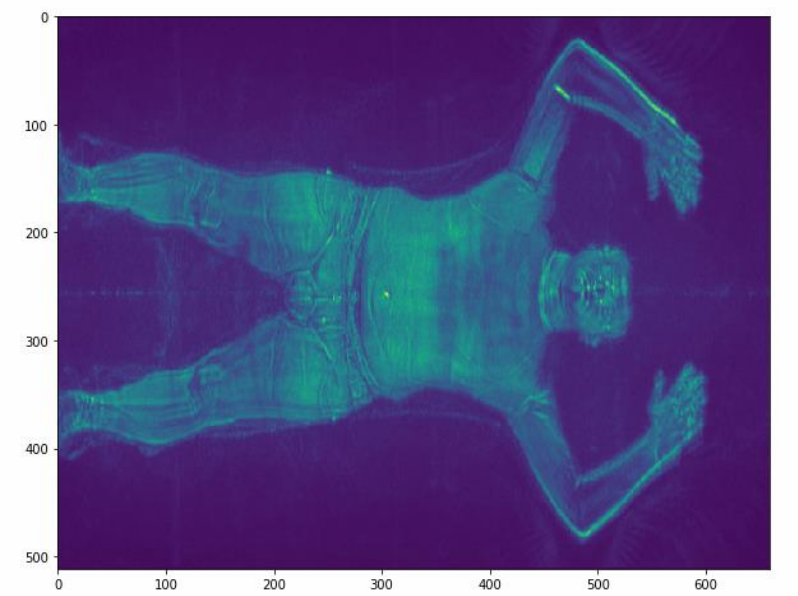


*Figure 2: Signal Response Plotted for Angle 0*

The labels for the task come in the form of a list of hashed ids with a zone number appended to the id and a binary flag indicating whether the file with the id contains a threat in the specified region (see Figure 3).

[8] (Fawcett 2005)

| | Id | Probability |
|---|---|---|
| 0 | 00360f79fd6e02781457eda48f85da90_Zone1 | 0 |
| 1 | 00360f79fd6e02781457eda48f85da90_Zone10 | 0 |
| 2 | 00360f79fd6e02781457eda48f85da90_Zone11 | 0 |
| 3 | 00360f79fd6e02781457eda48f85da90_Zone12 | 0 |
| 4 | 00360f79fd6e02781457eda48f85da90_Zone13 | 0 |
| 5 | 00360f79fd6e02781457eda48f85da90_Zone14 | 1 |
| 6 | 00360f79fd6e02781457eda48f85da90_Zone15 | 0 |
| 7 | 00360f79fd6e02781457eda48f85da90_Zone16 | 0 |
| 8 | 00360f79fd6e02781457eda48f85da90_Zone17 | 0 |
| 9 | 00360f79fd6e02781457eda48f85da90_Zone2 | 0 |

*Figure 3: Raw Data Labels*

To facilitate analysis of the data, we performed two transformations of the original data. First, we converted the long form labeled data to a wide format, where the index is the hashed id of the file and the labels for each of the zones are contained in an array of length 17. Second, we extracted the data arrays from the binary files and stored the data arrays, after some preprocessing we will explain in more detail below, into an "image" directory in an DHF5 file and we stored the label array from the previous step into a "labels" directory. In this way, our new dataset became a set of HDF5 files containing a 3D array of signal responses and a vector array of binary labels indicating the existence of a threat in each of 17 zones.

### 2.1.2 Descriptive Statistics

Two important observations may be gleaned by examining histograms of the signal responses of the scans (see Figure 4).

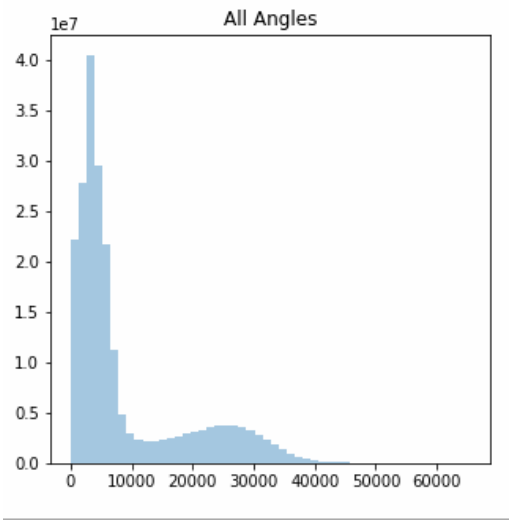


*Figure 4: Histogram of Signal for 9 Randomly Selected Scans*

We observe from Figure 2 and Figure 4 that most of the "pixels" in each image slice are part of the background and that the average signal response for the background lies between 0 and 10000. Our second observation is that the signal distribution is bimodal, with one peak corresponding to background noise and the other corresponding to the signal of interest, that is, the signal response around the subject in each image slice. This attribute of the data offers

an easy solution for reducing the noise: simply setting all pixels less than some threshold equal to the threshold. In our case, we set the threshold to 7000 and computed the histogram of the data once more (see Figure 5):

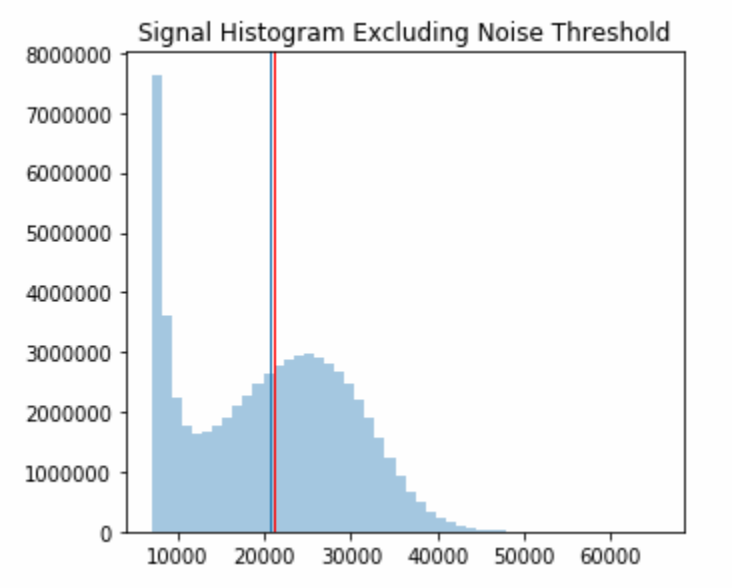


*Figure 5: Histogram of Signal Excluding Pixels At or Below Threshold*

The mean response for this random subset is 20675 and the median response is 21229.

With the use of this simple technique, we can make the images crisper and eliminate uninformative background variation in the signal response. Examine the difference when we apply the technique to the image below (see Figure 6).

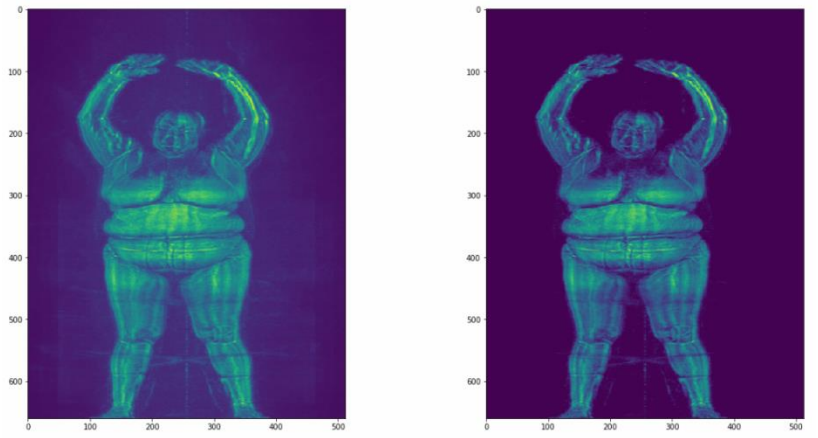

*Figure 6: Before (left) and After (right) Application of Noise Threshold*

We also examined the data labels to get an idea of the incidence of threats in the dataset. When we computed the positive rate of each of the 17 zones, we observed that it varied between ~8% and ~11.6%, suggesting that each sample could contain more than 1 threat (See Figure 7).

| Zone | sum | count | Pct |
|---|---|---|---|
| **Sum** | 1871 | 1147 | 163.121186 |
| **1** | 133 | 1147 | 11.595466 |
| **2** | 126 | 1147 | 10.985179 |
| **8** | 124 | 1147 | 10.810811 |
| **14** | 122 | 1147 | 10.636443 |
| **15** | 118 | 1147 | 10.287707 |
| **6** | 116 | 1147 | 10.113339 |
| **11** | 116 | 1147 | 10.113339 |
| **13** | 110 | 1147 | 9.590235 |
| **16** | 109 | 1147 | 9.503051 |
| **4** | 108 | 1147 | 9.415867 |
| **5** | 106 | 1147 | 9.241500 |
| **3** | 104 | 1147 | 9.067132 |
| **12** | 101 | 1147 | 8.805580 |
| **10** | 100 | 1147 | 8.718396 |
| **17** | 95 | 1147 | 8.282476 |
| **7** | 93 | 1147 | 8.108108 |
| **9** | 90 | 1147 | 7.846556 |

*Figure 7: Percentage of Sample Zones Containing Threats*

Looking more closely at the overlap, we noticed that each sample could contain threats in a total of 0 to 3 zones. The distribution of samples with each number of threats was approximately uniform (see Figure 8).

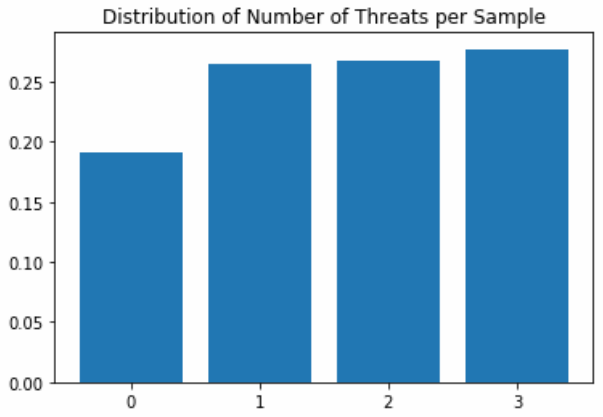


*Figure 8: Distribution of Number of Threats per Sample*

### 2.1.3 Solving a Simplified Version of the Problem

This problem is highly complex when tackled in the version it is presented in the Kaggle competition. In the interest of time and limited computational resources, we simplified the problem we sought to solve with this project. We reduced the multi-label problem of classifying threats in each of 17 zones to the binary classification problem of identifying threats in **any** of 9 zones of the lower body. Further, we only used 16 of the 64 angle-slices and only

the bottom two thirds of the scans. The changes to the input dimensions (after cropping, see *section 3.1.1)* are summarized as follows:

1. Only the bottom 3/5 of each scan is used.
   1.1. Array shape changes from (400,600,64) to (400,360,64).
2. Only 16 equidistant angles are used.
   2.1. Array shape changes from (400,360,64) to (400,360,16)
3. Target label changes from classification of threat at each of 17 zones to OR function applied to 9 zones of the lower body.
   3.1. $Old\ Target: \overrightarrow{\boldsymbol{y_i}}\ where\ |\boldsymbol{y_i}| = 17$
   3.2. $New\ Target: y_i\ where\ y_i = OR(y_{i_z}\ \forall z\ \in \{8,9,10,11,12,13,14,15,16\})$

## 2.2 Algorithms and Techniques

### 2.2.1 Dimensionality Reduction and Feature Extraction through Deep Transfer Learning

Even after the problem is reduced to a binary classification, there remains the issue of the exorbitantly high dimensionality of the data set. After cropping, each scan contains 400 x 360 x 16 = 2,304,000 dimensions. Further, we only have 1115 samples to explore the feature space. The first step in the project was to reduce the dimensionality of the dataset by running the images through an Inception-V3 network pre-trained on ImageNet data.

It is a misconception that deep learning is only appropriate for very large datasets. One may enjoy many of the advantages of deep neural networks without a large data set by employing transfer learning techniques. See Figure 9 for a chart on the various scenarios in which transfer learning may be applied.

| | **very similar dataset** | **very different dataset** |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer | You're in trouble… Try linear classifier from different stages |
| **quite a lot of data** | Finetune a few layers | Finetune a larger number of layers |

*Figure 9: Transfer Learning Scenarios*[9]

For this project, we extracted the activations at 10 different layers of a pre-trained Inception-V3 model using Keras with a TensorFlow backend. We chose deep transfer learning as a strategy for dimensionality reduction because since the breakthrough achieved by AlexNet in 2012, Deep Neural Networks (DNN) have been consistently outperforming other models for image classification challenges such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).[10] The success of these models is often attributed to the facility of deep architectures

[9] (Li, Johnson and Young 2017)
[10] (Russakovsky, et al. 2015)

for hierarchically extracting features of greater and greater abstraction. We chose Inception-V3 because it has a small memory footprint, requires a relatively smaller number of operations, and has achieved high accuracy in the ILSVRC (see Figure 10).

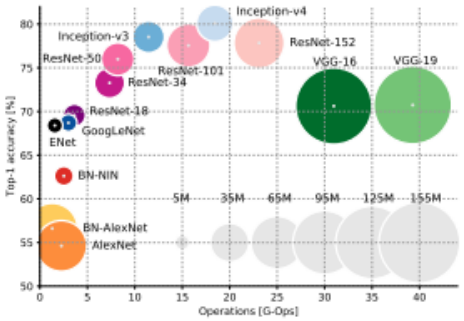


Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from $5 \times 10^6$ to $155 \times 10^6$ params. Both these figures share the same $y$-axis, and the grey dots highlight the centre of the blobs.

*Figure 10: Comparison of Various DNN Architectures*[11]

Furthermore, there is evidence that Inception-V3 can adapt well to input of low input resolution, which may be useful in "systems for detecting relatively small objects."[12] This flexibility with respect to the input resolution is favorable because not only are our images of different dimensions than those used to train the model (299 x 299) but the threats comprise only a small portion (~400/160K pixels) of the entire scan-slice.

The Inception-V3 model distinguishes itself from other convolutional neural networks (CNNs) by employing "inception" modules in its architecture. These modules consist of not one, but numerous convolutions, which may have varying kernel sizes and filter sizes, and pooling layers that are all applied to the preceding layer, and whose outputs are concatenated at the end (see Figure 11). Inception modules improve the modeling performance of the network by creating more activations per tile, which in turn facilitates higher dimensional representation of the data.

[11] (Canziani, Culurciello and Paszke 2016)
[12] (Szegedy, et al. 2015)

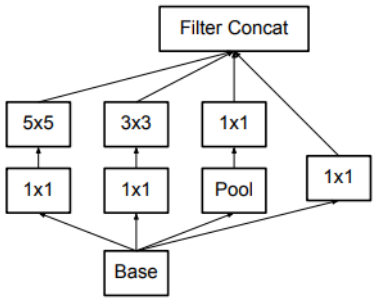


*Figure 11: Original Inception Module*

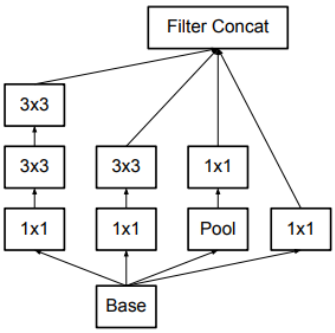


*Figure 12: Type 1 Inception Module*

The Inception-V3 model improves the implementation of inception modules by employing factorizations of larger spatial filters, such as the 5 x 5 filter in Figure 11. Filters like the 5 x 5 filter can be factorized into a 3 x 3 filter followed by a fully connected layer. Factorized representations have the benefit or reducing the computational expense of the filters, which can be used instead to increase the filter depths (compare Figure 11 and Figure 12). By factorizing the filters, the authors of Inception-V3 manage to create an exceptionally deep model, while keeping the number of parameters small. For comparison, VGG19 has a depth of 23 with 138.4 million parameters, while Inception-V3 has a depth of 159 with 23.9 million parameters.[13]

Like many CNN's, Inception-V3 was trained through back propagation using stochastic gradient descent (SGD) with a batch size of 32. In back propagation with SGD, the negative gradient of the loss function—evaluated for the batch, since evaluating the loss for the entire dataset is computationally infeasible— with respect to each weight in the network is calculated iteratively and the weights are updated according to the step-size, which is the negative partial derivative scaled by the learning rate, $\alpha$.

The Inception-V3 network we used for feature extraction was trained on ImageNet data, which contains 1000 classes, using a regularized form of the cross-entropy loss function, of which log-loss is a corollary for binary classification.

$$Cross - Entropy = -\sum_{k=1}^{K} \mathrm{y_i} \log(p_k)$$

$$where\ y_i is\ ground\ truth\ and\ p_k\ is\ predicted\ probablity\ of\ class\ k$$

A summary of the model architecture is provided in Figure 13. For our project, we extracted the global maximum of each feature in each of the 10 inception modules in the network.

[13] (Keras Applications 2017)

| type | patch size/stride or remarks | input size |
|---|---|---|
| conv | 3×3/2 | 299×299×3 |
| conv | 3×3/1 | 149×149×32 |
| conv padded | 3×3/1 | 147×147×32 |
| pool | 3×3/2 | 147×147×64 |
| conv | 3×3/1 | 73×73×64 |
| conv | 3×3/2 | 71×71×80 |
| conv | 3×3/1 | 35×35×192 |
| 3×Inception | As in figure 12 | 35×35×288 |
| 5×Inception | As in figure 14 | 17×17×768 |
| 2×Inception | As in figure 15 | 8×8×1280 |
| pool | 8 × 8 | 8 × 8 × 2048 |
| linear | logits | 1 × 1 × 2048 |
| softmax | classifier | 1 × 1 × 1000 |

*Figure 13: Inception-V3 Architecture*[14]

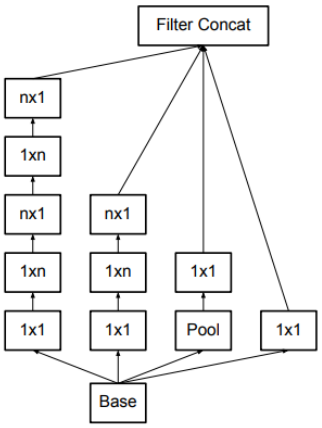


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the $17 \times 17$ grid. (The filter sizes are picked using principle 3)

*Figure 14: Type 2 Inception Module*

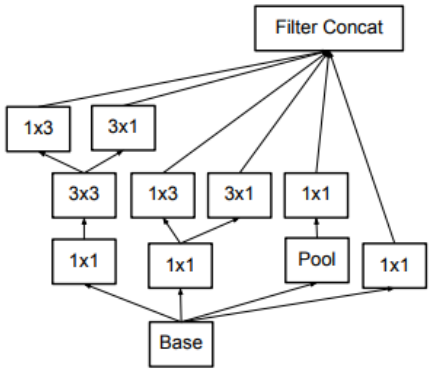


*Figure 15: Type 3 Inception Module*

### 2.2.2 Gradient Boosted Tree Classifier

After extracting and reducing the dimensionality of the data from 2.6 million to 128 thousand, we chose to train a Gradient Boosted Tree classifier on the feature-rich data set for a few reasons. First, we expected we would have to capture non-linear relationships among the extracted features to obtain satisfying results. Secondly, since the ImageNet dataset, which contains color images of a thousand different classes of objects, is so different from our target dataset, we expected only a small subset of the features extracted to be important for determining the existence of a threat. Even after significant dimensionality reduction, there are still many more features than samples, so we needed an algorithm that could both identify what features, and from what layers of the Inception-V3 model, were most predictive, and be robust to overfitting.[15] Gradient boosted trees fulfilled both criteria.

It has been proposed that Gradient Tree Boosting, like other boosting and bagging algorithms, tend to be robust to overfitting, sometimes even reducing training error to zero

[14] (Szegedy, et al. 2015)
[15] (Schapire, et al. 1997)

while continuing to reduce generalization error, because they maximize the "margin", or difference, between their confidence in the top-1 class prediction and their confidence in the other possible classes.[16] Since Gradient Tree Boosting uses decision trees as its weak learner, it is possible to estimate the importance of each feature by calculating the average proportion of samples whose classification each feature contributes to; the features that maximize information gain are more likely to be at the top of the various weak learners.

Gradient Tree Boosting may be thought of as a greedy function approximation algorithm where in each successive iteration, a new tree of fixed size, in our case given by the maximum depth, is constructed so that the weighted predictions of the new model minimize the loss function. Since it would be computationally infeasible to search through all the possible trees that minimize the loss function, the algorithm employs steepest descent minimization by training the next tree learner on the "pseudo-residuals" of the last model, given by the negative of the gradient. Once the tree candidate has been chosen, the weight for that learner is calculated by finding the weight that minimizes the loss function. This last optimization is more feasible because it involves only one dimension, i.e. the weight itself. Friedman, summarizes the algorithm as follows in the paper where he introduces this gradient boosting strategy:

**Algorithm 1: Gradient_Boost**

1 $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$

2 For $m = 1$ to $M$ do:

3 $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$

4 $\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$

5 $\rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$

6 $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$

7 endFor

end Algorithm

*Figure 16: Summary of Gradient Boosting Algorithm[17]*

In step one, an initial learner is created, which guesses the same value, $\rho$, for all samples. Then, the negative gradient with respect to the loss function is calculated, evaluated at the last model. The parameters for the new learner are calculated in step 4. In our case, a new tree is fitted on the "pseudo-residuals" calculated in step 3. In step 5, a weight for the learner, given by $\rho_m$, is calculated by solving the one-dimensional minimization mentioned above. Finally, the model is updated with the new learner, $h(x; a_m)$, and its corresponding weight, $\rho_m$.

In *sklearn*'s implementation, trees are built according to the CART algorithm, wherein, at each node, each predictor's best split is evaluated according to the appropriate impurity measure—Gini criterion in our implementation—and the best split is applied in a greedy

[16] (Schapire, et al. 1997)
[17] (Friedman 1999)

fashion until the stopping criteria have been met, most likely until the maximum depth has been attained.[18]

$$Gini = 1 - \sum_j p_j^2$$

$$where\ p_j is\ the\ probability\ of\ class\ \text{j}$$

### 2.3 BENCHMARK

The clearest benchmark we have for the performance of threat detection algorithms are estimates of the performance of algorithms already deployed on the field. As mentioned in *section 1.1*, according to reporting by ProPublica, false-positive rates in the U.S are estimated to fall between 17% and 38.5%.[19] It is important to note, however, that the 17% value was obtained under conditions of minimal recall. Again, since we do not know what the recall of the benchmark algorithms is, even if we obtain a similar false positive rate our approach may still be superior. That said, a false-positive rate below 40%, while maintaining a recall greater than 80%, would indicate that the use of transfer learning in conjunction with traditional machine learning algorithms can deliver results comparable to the specialized, manufacturer specific algorithms being deployed today.

Concerning the benchmark for the log-loss metric, we may use an uninformative classifier as the benchmark. If our classifier were to predict 0.5 probability for every single sample, then the average log loss we would expect is 0.69. A log-loss below this, then, would demonstrate that the probability outputs of our classifier are correct more often, and with greater conviction, than a random classifier would be.

## 3 METHODOLOGY

### 3.1 DATA PREPROCESSING

Of the preprocessing steps outlined below, only the first, 3.1.1, was applied at once to the whole data set. The remaining preprocessing steps were integrated into the Inception-V3 model in Keras to take advantage of faster GPU processing (see notebook "MultilayerFeatureExtraction").

#### 3.1.1 Cropping and Noise Reduction

The first step in preparing the raw data was reducing the cloud of noise we noticed around the subject in the data exploration phase. To achieve this, we followed the simple procedure outlined in *section 2.1.2:* any pixel where the signal response was lower than the

[18] (Breiman, et al. 1984)
[19] (Grabell and Salewski 2011)

threshold of 7000, we set to 7000 (see Figure 6 for a side to side comparison). Realizing that in many of the image-slices, the background comprised most of the pixels, we extracted 400 x 600 crops of each image slice, originally 512 x 660, centered around the subject. Notice that for this project, where we limited ourselves to the lower body, only 400 x 360 pixels of the 400 x 600 crops were fed to the next preprocessing step (see *section 2.1.3).*

### 3.1.2 Conversion to RGB Scale

Deep models pre-trained on the ImageNet dataset are, of course, trained on images with the usual RGB encoding, where each pixel is given a value for each of its 3 channels in the set of integers between 0 and 255. In order to make use of the Inception-V3 model, we first rescaled our data to fit between integers 127 and 255 and copied the data array two times in order to simulate a grey-scale image. The final shape of each sample was (400,360,16,3).

We chose to limit the range of our data to the set of integers between 127-255, inclusive, because we expect activations in a neural network to be normally distributed around the average value in the training data. We therefore want to align the background signal response in our data with the average value in the training data, which for images is expected to lie in the middle of the available range, namely 127.

### 3.1.3 Inception-V3 Preprocessing; Normalization Between -1 and 1

Different deep network architectures use different preprocessing techniques for their training data. In the case of Inception-V3, the authors transformed the images from the RGB space of 0-255 to the region between -1 and 1. We applied this preprocessing step as well, since the model weights were optimized using this preprocessing step on the ImageNet data set.

### 3.1.4 Rotation

When the image-slices are visualized, it becomes apparent that they are rotated 90° such that the “floor” in the image lies on the left side of the image-slice. Since image classification in neural networks is not rotation-invariant, we rotated the image so that when visualized, the subject appears to be standing (see Figure 17 for a visualization of preprocessing steps up to this point)

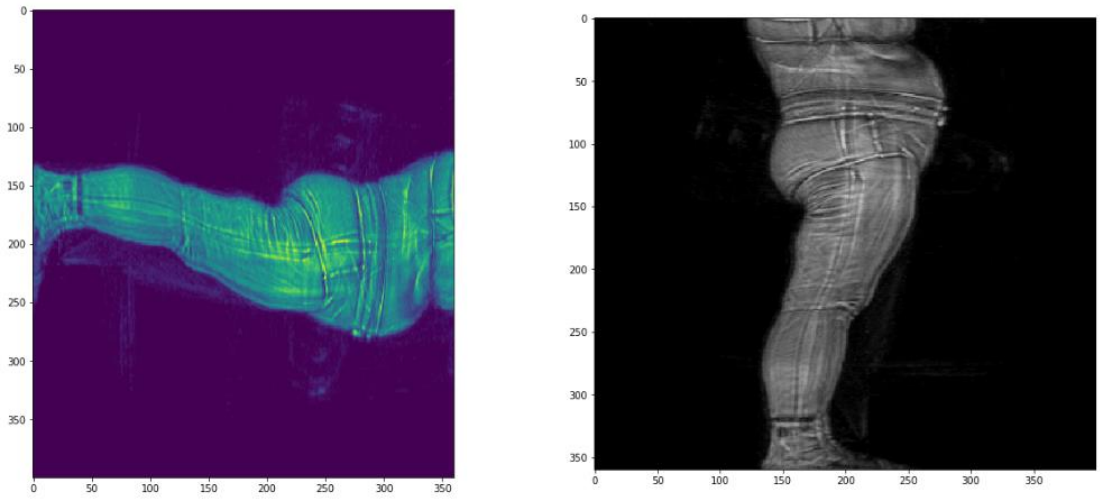

*Figure 17: Raw Image (left) vs. Preprocessed Image (right)*

### 3.1.5 Feature Extraction

The feature extraction step may be itself considered the last preprocessing step. The maximum activations for each feature, also known as global 2D max pooling, in the first 10 layers of the Inception-V3 network were concatenated, flattened, and stored as the new feature vector for each sample. The activations are difficult to interpret exactly, but they represent features of increasing degree of abstraction, especially as a function of the depth of the layer in which they are found (features in deeper layers are expected to be more abstract and more specific to the original training task).

### 3.1.6 Balancing Data

Although boosting tends to be less sensitive to imbalanced data, we balanced our training dataset to establish a standardized pipeline for future iterations of this project. Our training dataset contained 891 samples and 544 positives, corresponding to a 61.1% positive rate. We balanced the data using random oversampling of the negative class, which brought our final training set to 544 positive samples and 544 negative samples.

## 3.2 Implementation

Gradient Boosted Tree classifiers cannot be run in parallel, much less out-of-core, because they require all the data to be in memory to iteratively build each weak learner. The greatest implementation challenge in this project was reducing the data set to a tractable size while extracting high-quality features. To preprocess the large data set into this tractable form, we downloaded the Kaggle dataset from a Google bucket to an AWS S3 bucket and applied the Boto3 package to communicate with the S3 bucket from our environment.

For preprocessing step 3.1.1, where we crop and reduce the noise of our data set, we applied a 80-20 train-test split, applied the preprocessing functions, and stored the data into another directory in our S3 bucket (see notebook “CreateCleanDataSet”).

For the rest of the preprocessing and feature extraction, we built custom layers in Keras that transformed the images as they were fed to the Inception-V3 architecture. This was done to take advantage of the GPU capabilities of TensorFlow. The layer activations yielded by the entire data set were then stored in single ".hdf5" files, one for the training set, and another for the test set (see notebook "MultilayerFeatureExtraction").

Finally, once the compressed, but feature-rich, representation of the data could be easily loaded onto memory, we balanced the data set using random oversampling of the negative class with the package *imbalanced-learn*. By examining the splits of a Gradient Boosted Tree classifier with 2000 weak learners, we examined how the utility of the extracted features varied with angle and layer of provenance. Then, we used *scikit-learn's* implementation of grid search and the Gradient Boosted Tree Classifier to train an optimized classifier.

## 3.3 Refinement

On our first attempt, it became clear that trying to optimize a Gradient Boosted Classifier using 128,000 features would take much too long. Furthermore, it was likely that many, if not most, of these features would not be relevant to the classification task at hand. We therefore used our first solution, to identify the features we proceed with in the next iterations of our hyperparameter optimization. The first model, which considered all the features at every step and selected a random subset of 80% of the data for each of 100 weak learners, yielded a log-loss of 0.40 and an AUC of 0.86 on the test set. More importantly, it identified the 615 features that had a non-zero importance score, as measured by the average proportion of the samples that each feature helped classify.

Using only the 615 features we identified using the first model we trained, we performed several rounds of a 3-fold cross-validation grid search, each time narrowing the parameter ranges. We chose as an objective function the minimization of the average cross-validation log-loss along the parameters of number of estimators, learning rate, subsampling ratio, and max depth. For the sake of time restrictions, we considered a random subset of 307 features for the creation of each weak learner. The iterations are summarized in Figure 18.

| | Num Estimators | Learning Rate | Subsampling Ratio | Max Depth | Best Params. | Train Log-Loss | Val Log-Loss |
|---|---|---|---|---|---|---|---|
| **Trial 1** | 100 | 0.1 | 0.8 | 3 | NA | 5.16E-02 | 4.09E-01 |
| **Trial 2** | 200, 500, 1000 | 0.01, 0.05, 0.1 | 0.25,0.5,0.75, 0.8,1 | 3,5 | 1000,0.05,0.25,3 | 0.000813 | 0.00896 |
| **Trial 3** | 750, 1000, 1200, 1500 | 0.025, 0.05, 0.75 | 0.1,0.25,0.8 | 3 | 1500,0.05,0.25,3 | 2.16E-04 | 8.50E-02 |
| **Trial 4** | 1400, 1500, 1600, 1800, 2000 | 0.05 | 0.2,0.25,0.30 | 3 | 1500,0.05,0.2,3 | 2.15E-04 | 7.61E-02 |
| **Trial 5** | 1400, 1500, 1700, 1900, 2100 | 0.05 | 0.1,0.15,0.2,0.25 | 3 | 1500,0.05,0.2,3 | 2.15E-04 | 7.61E-02 |

*Figure 18: Results of Cross-Validation Trials*

The best parameters were identified at the end of trial 5. With 1500 estimators, a learning rate of 0.05, a subsampling ratio of 0.2, and a max depth of 3, our Gradient Boosted Classifier yielded a log-loss of 0.62 and an AUC of 0.87 on the test set. It is evident from the significant increase in the loss that the model had overfit. One potential reason is the way in which the data was balanced. By using oversampling, we created the possibility of identical samples existing in both the validation set and the training set. Although is not generally good practice to continue training after a final measurement on the test data, we retrained the model with the same final parameters, but this time we employed early stopping using a random subset of 20% of the training data as a validation set. The final model yielded a log-loss of 0.40 and an AUC of 0.87.

# 4 RESULTS

## 4.1 MODEL EVALUATION AND VALIDATION

Our final Gradient Boosted Tree classifier was trained using a learning rate of 0.05 and a subsampling rate of 0.20, which we chose based on the results of several rounds of cross-validation (without early stopping). Once these final parameters had been chosen, we re-trained the model with early stopping after 10 iterations without an improvement in the loss of the test set, which yielded 323 estimators.

It was surprising that such a low subsampling rate emerged as the best value for the subsampling parameter. One would expect that the balance between introducing randomness in the samples used by each estimator and providing more data from which to learn would lie at least above a 50% subsampling rate. It can be seen, however, how such a low subsampling rate could improve the performance of the algorithm by considering that fitting a learner to

small subsets of the data amounts to performing a sort of bagging procedure where each learner is focused on correctly classifying different subsets of samples.

The optimal learning rate acquired from our cross-validation trials is within the expected range of recommended learning rates, as suggested by Tibshirani et al. (<=0.10).[20]

The model's solution is robust to changes in the training data. When 10-fold cross-validation was applied to the model using the final hyperparameters, the average Log-Loss was 0.16 with standard deviation of 0.06. The average AUC was 0.99, with a standard deviation of 0.01. The small standard deviations of the results suggest the model performs well even under small perturbations (10% of samples) in the training data. Finally, the model generalizes well even when it is presented with a test set of data that wasn't used during the 5 cross validation trials, yielding an AUC of 0.87 and a Log-Loss of 0.40.

## 4.2 JUSTIFICATION

The best way to compare the performance of our model to our false positive benchmark of 17% is by examining the ROC curve of our model in Figure 19. In the range between a false positive rate of 0.17 and 0.30, the true positive rate remains constant at around 0.78. The true positive rate does not rise above 0.9 until the false positive rate rises above 0.35. These results suggest a performance comparable to that of the screening algorithms currently in use, which have a false positive rate of 0.17 at their least sensitive, and a false positive rate of 0.38 at their most sensitive.[21]Further, these results show that an approach combining feature extraction using deep transfer learning and traditional boosting methods, such as a Gradient Boosted Tree regressor, can yield results comparable to those currently achieved by manufacturer-specific and domain-specific approaches.

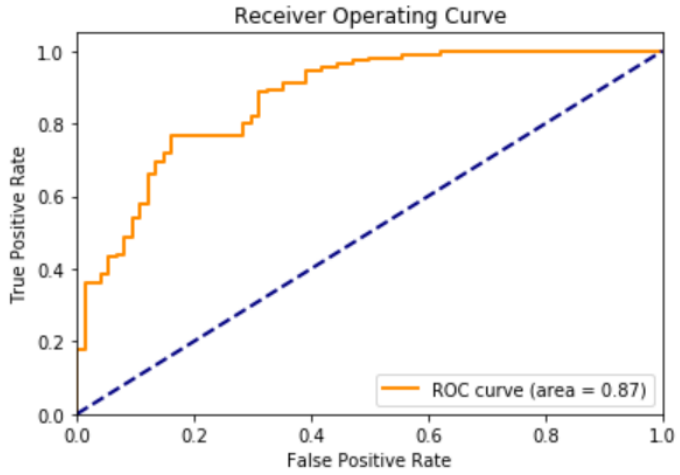


*Figure 19: Final Model ROC Curve*

[20] (Tibshirani, Hastie and Friedman 2009)
[21] (Grabell and Salewski 2011)

Regarding our Log-Loss benchmark, our model performs much better on the test set (0.40) than guessing "0.5" for every sample, which would lead to a Log-Loss of 0.69 in a balanced dataset.

It is important to recognize that this work only establishes the foundation for tackling the larger problem of producing zone-specific classifications. It is unclear what resolution of data extraction would be necessary to scale up the solution we have developed here for zone-specific classifications of the entire body.

# 5 CONCLUSION

## 5.1 FREE-FORM VISUALIZATION

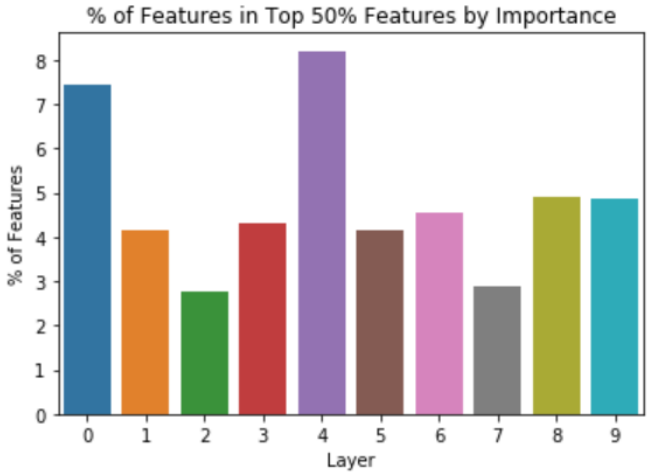


*Figure 20: % of Features Among Top 50% Important Features by Layer*

One important attribute of our approach is that the use of gradient boosted trees can help inform future iterations of feature extraction. We selected the top 50% of features with non-zero importance and calculated the percentage of features in each layer that appeared in this group. Some layers immediately stood out above the others. Layers 0 and 4 were almost twice as likely to produce a feature in the top 50% group than some of the other layers. This result validates our expectation that lower level layers tend to be more easily generalizable to different tasks. It is surprising, however, that there appears to be no monotonic relationship, indicating that there may be discontinuities in the degree of usefulness of features from adjacent layers. This observation may lead to strategies for improving future iterations of the feature extraction process (more on this subject in section 5.3).

## 5.2 Reflection

In this project, we have examined the feasibility of applying a combination of deep transfer learning and a Gradient Boosted Tree Classifier to a problem normally solved with dedicated manufacturer-specific solutions. We downloaded the dataset of mm wave scans from a Google bucket set up by the organizers of the TSA Kaggle competition that sparked the idea for this project into an AWS S3 bucket.[22] Then, we applied a simple algorithm for cleaning the data before running 16 images from each sample scan through an Inception-V3 model pre-trained on ImageNet data. We recorded the activations from the first 10 layers of the model into an HDF5 file and used this file as our new feature-rich dataset. Using this new dataset, we trained and optimized a Gradient Boosted Tree Classifier, yielding an AUC of 0.87 and a Log-Loss of 0.40 on a test set. The model performed well with a recall between 0.79 and 0.9 within the false positive rate region of 0.17-0.38, which is the operational false-positive rate of algorithms currently in use. These results validate the use of machine learning approaches, instead of relying on the specialized knowledge of electrical engineers for this task.

The most difficult aspects of this project involved the handling of the data itself. The clean dataset is 108 GB large, much bigger than the RAM of any readily available machines (for me, anyway). The size of the data necessitated the implementation of a pipeline that could perform the feature extraction process using out-of-core methods. The pipeline we developed here relies heavily on the use of the Boto3 package to communicate with the AWS S3 bucket where the data resides.

## 5.3 Improvement

For this project, we were focused on the problem of whether a traditional machine learning algorithm, aided by deep transfer learning, could be used to detect the existence of any threat in a generalized region (the lower body in this case). Solving the more complex problem of classifying threats in several localized regions within a single scan will require several changes to our approach.

Firstly, the resolution at which important features is extracted will have to increase. In our approach, we down-sampled by applying global max pooling of the activations from our Inception-V3 model. Using our analysis of feature importance, we may selectively increase the resolution of the most important features while down-sampling features with lower importance.

Secondly, we can increase and diversify our sample size by applying data augmentation methods. Data augmentation increases the generalizability of models by applying random transformations, such as shearing, to the samples. In our case, it would likely be advisable to

[22] (Department of Homeland Security 2017)

perform the same transformation on each of the various angle-slices that comprise each sample.

Lastly, we may increase the resolution of our raw dataset to begin with. While the raw ".a3daps" dataset is ~50GB, the full volumetric format, ".a3d", is ~430GB large. By using full volumetric data, we may extract transverse slices of the subject and eschew the problem of visual distortions as you view the subject from different angles. These images would contain more abstract shapes and curves than the frontal images we used in this project. For this reason, we would have to redo our feature importance analysis to determine which features to preserve at high resolutions, which to down-sample, and which to eliminate.

# 6 BIBLIOGRAPHY


Breiman, L., J. H. Friedman, R. Olshen, and C. J. Stone. 1984. *Classificatin and Regression Tree.* Wadsworth & Brooks/Cole Advanced Books & Software.

Canziani, Alfredo, Eugenio Culurciello, and Adam Paszke. 2016. "An Analysis of Deep Neural Network Models for Practical Applications." *arXiv:1605.07678.*

Department of Homeland Security. n.d. "Apex Screening at Speed." *Official Website of the Department of Homeland Security.* Accessed November 13, 2017. https://www.dhs.gov/science-and-technology/apex-screening-speed.

—. 2017. "Passenger Screening Algorithm Challenge Data." *Kaggle.* June 22. Accessed November 13, 2017. https://www.kaggle.com/c/passenger-screening-algorithm-challenge/data.

—. 2017. *Passenger Screening Algorithm Challenge Description.* 06 22. Accessed November 13, 2017. https://www.kaggle.com/c/passenger-screening-algorithm-challenge.

Fawcett, Tom. 2005. "An introduction to ROC analysis." *Pattern Recognition Letters.*

Friedman, Jerome H. 1999. *Greedy Function Approximation: A Gradient Boosting Machine.* Stanford: IMS 1999 Reitz Lecture.

Grabell, Michael, and Christian Salewski. 2011. "Sweating Bullets: Body Scanners Can See Perspiration as a Potential Weapon." *ProPublica.* 12 19. Accessed November 13, 2017. https://www.propublica.org/article/sweating-bullets-body-scanners-can-see-perspiration-as-a-potential-weapon.

2017. "Keras Applications." *Keras: The Python Deep Learning Library.* Accessed 11 28, 2017. https://keras.io/applications/.

Li, Fei-Fei, Justin Johnson, and Serena Young. 2017. "Lecture 7 | Training Neural Networks II | Stanford Univeristy." *Youtube.* 08 11. Accessed 11 17, 2017.

https://www.youtube.com/watch?v=_JB0AO7QxSA&index=7&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Adita Khosla, and Michael Bernstein. 2015. "Imagenet large scale visual recognition challenge." *International Journal of Computer Vision* 115(3):211–252.

Schapire, Robert E., Yoav Freund, Peter Bartlett, and Wee Sun Lee. 1997. "Boosting the margin: A new explanation for the effectiveness of voting methods." *Machine Learning: Proceedings of the Fourteenth International Conference.*

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. "Rethinking the Incpetion Architecture for Computer Vision." *2015arXiv151200567S.*

Tibshirani, R., T. Hastie, and J. Friedman. 2009. *Elements of Statistical Learning Ed. 2.* Springer.

United States Department of Transportation. 2017. *Air Traffic by The Numbers.* 11 7. Accessed November 13, 2017. https://www.faa.gov/air_traffic/by_the_numbers/.