Abstract—In the current age of increased air travel, safe airport security is imperative. And while correctly detecting potential threats is a top priority, it is also important that these security checks are performed as efficiently as possible to maintain the flow of travel. Current security methods utilize machine learning algorithms to analyze and detect threats on millimeter wave scans of individuals. This paper implemented and compared various classification algorithms on a dataset of millimeter waves scans to determine the most effective algorithm at detecting threats. This paper also examined different data preprocessing and model optimization techniques and their effect on the performance of the models. It was concluded that the convolutional neural network and long short-term memory network, trained on a specific zone of the body, were the most effective models for detecting potential threats.

I. INTRODUCTION

A. TSA Airport Scans

The Transportation Security Administration (TSA) was created to protect public transportation in the United States of America using a variety of different technologies [1]. One essential technology utilizes a comprehensive scan of the human body to reveal any threats. As those who pose a threat to national security become more advanced in their attempts to bypass these technologies, machine learning is an effective tool to ensure the safety of public transportation. The use of machine learning aims to automate the process of checking scans, reducing the need for manual human checks and enabling greater accuracy in detecting threats. The need for an accurate and efficient way to classify a scan as containing a threat or not is clear. A study by ProPublica showed that Germany (which uses the same millimeter-wave scanners as the TSA) had a false positive rate of 54 percent. This high rate was cited as being a result of irregularity in clothes and sweat [2]. Using a machine learning algorithm can account for these discrepancies. In addition, a better detection algorithm would more accurately identify threats, eliminating the additional resources spent on people who are falsely flagged.

This paper analyzes various ways of classifying TSA images as containing a security threat through various machine learning algorithms and optimization methods, and to ultimately find the most effective method of doing so.

II. BACKGROUND

A. Machine Learning

Machine learning is a category of artificial intelligence algorithms which have the ability to improve through data input. These algorithms process patterns in data and create corresponding alterations within their hidden workings to effectively “learn” from the data being inputted and become better at performing at a specified task [3]. Certain machine learning algorithms are built to mimic the human mind, such as neural networks, which are known for processing information with neurons like the brain. Regardless of what the algorithm is used for and its complexity, it uses a learning method that can typically fall into one of the following categories [4]:

1) Supervised machine learning uses a training dataset to learn and apply information to a testing set. The algorithm is able to learn by finding patterns and repetitions in the data that it looks for in future datasets. These algorithms are used to make predictions and when trained with labels can adjust its processes to become more accurate [5].
2) Unsupervised machine learning algorithms use unclassified and unlabeled data to learn how to process hidden patterns in data. Rather than choosing one correct output, these algorithms are used to find and describe these patterns [5].

3) Reinforcement machine learning algorithms have a direct link to their environment, allowing them to collect data based on their interactions. Through a process of trial and error, the algorithms learn the ideal behavior for certain situations through a “reinforcement signal” [4].

The advancement of machine learning has allowed for various software innovations to become possible, most of which are used to mimic human abilities such as image recognition or text recognition. Various forms of machine learning are used collectively to implement these technologies into education, safety, and everyday lives.

B. Classification Models

1) Logistic Regression: Logistic Regression is a classifying algorithm that is used to take certain input values and set the output as a binary value [6]. Therefore, this algorithm is suitable for identifying whether a body scan contains a threat. The logistic regression algorithm works by first flattening the input by converting the image from a 3 dimensional array to a single dimension array. The array is then passed into the activation function, commonly a sigmoid function. The function then outputs a value between 0 and 1, which represents the probability that the input would be classified positively.

\[ g(z) = \frac{e^{b_0 + b_1 \times z}}{1 + e^{b_0 + b_1 \times z}} \] (1)

The equation above is a sigmoid function and once the coefficients of \( b_0 \) and \( b_1 \) are determined through algorithmic optimizations, the probability of \( z \) can be found from said equation [7]. The output from this equation classifies the input where if the value of \( g(z) \) is greater than 0.5, it belongs in the positive class. Otherwise, it belongs in the negative class. The closer the output of the function is to either one or zero, the more confident the model is in its classification.

2) Neural Networks: Neural networks are the basis for a majority of machine learning algorithms. This specific type of algorithm aids a computer in analyzing training examples, which helps systems classify their data through pattern recognition. Neural networks contain layers of nodes, or neurons, that are connected to other layers with data passing between them. The data that each node receives is coupled with coefficients known as weights, and then a bias is added to the said nodes. This works to change the significance of the input by either increasing or decreasing it [9]. The resulting values are passed through an activation function to determine what information should be transferred further into the network of nodes. The neural network as a whole contains nodes that make up an input layer, hidden layers, and an output layer, as shown in Fig. 2. While the input layer takes in the initial data for the network, the output layer gives results of the processed data and transfers that information out of the network. The hidden layers contain the nodes that perform the computations that process the data from the input layer to the output layer [10]. In this research, the neural networks considered will be feed-forward, meaning that the information only moves in one direction from the input layer, the hidden layers, and the output layer [9].

Since the neural network has no prior information on the input data, the weights and biases that are applied at each node begin as random values. The training algorithm tests these random values with training data and computes a loss value that reflects how well the neural network model represents the data. A larger loss means a worse model, so training works to tune the weights and biases to minimize loss. Then, when new images are fed into the model, the adjusted weights and biases will be applied in an attempt to properly classify the images [9].

3) Convolutional Neural Networks: Convolutional neural networks (CNN) are a neural network subtype designed for image processing. They are based on the scientific understanding of human neurons in the visual cortex, where each neuron is only responsible for a specific part of the input image [12]. The algorithm is able to learn the features of an image by identifying and assigning importance to various objects and image aspects. The identification of features is accomplished
by sliding a kernel (matrix of learnable weights) across the image and performing matrix multiplication with the pixel values to produce a new value for the resulting matrix. The image undergoes many such kernels, each producing a result that is passed under more kernels [13]. Convolving an image results in reduced dimensionality when the kernel is larger than a pixel. To keep the dimensions of the matrix constant, some models employ same padding, surrounding the image in layers of zeros so that the convolved image has the same dimensions as the input. Not using a layer of zeros is called valid padding [14]. Another parameter of convolution is the stride size. The stride is the number of pixels the kernel travels horizontally and vertically when sliding over. Larger strides result in smaller outputs.

In between each layer of kernels is a pooling layer. This project’s CNN algorithm used max pooling, which passes a kernel over the image and takes only the highest value inside. Max pooling boosts the model’s performance by extracting prominent features, eliminating unimportant noise, and reducing necessary computational power. Another type of pooling is average pooling, which takes the average of the values inside the kernel. Because average pooling merely suppresses noise and does not extract the most important features, max pooling more often leads to better results [12]. After several layers of convolution and max pooling, the model has learned the features by producing many filtered images, each with a different aspect highlighted by different combinations of kernels. These images are flattened into a string of values and passed through a standard neural network that classifies the image.

CNNs perform better than flattening the image into a long string of pixel values because it is able to capture spatial dependencies in the image. Flattening the pixel values stops the model from recognizing lines and edges, whereas the CNNs layers of kernels are able to train the model to recognize and assign importance to edges and shapes in the image. Because it is able to identify important features independently, a CNN eliminates a lot of the preprocessing necessary in other types of models, making it ideal for image analysis [12].

C. Long Short-Term Memory Networks

CNNs analyze one image at a time, applying convolutions and tuning weights and biases accordingly. However, when a sequence of images is provided, there may be certain relationships between consecutive images that CNNs are not designed to detect. For example, several images showing the progression of a jump always have a specific order. The person starts on the ground, moves upward, and then drops back to the ground. CNNs would analyze each frame individually and fail to consider the progressive movement of the human holistically. To allow neural networks to remember features from past images while working through training images, recurrent neural networks (RNN) were developed. Since then, a specific type of RNN called a long short-term memory (LSTM) network has been developed, and it has seen incredible success in its multitude of applications, including movement recognition, finance, and more [16].

The foundation of a basic LSTM are three gates that allow it to learn from both previous states and the current image the training algorithm is analyzing. It uses the information from these two items and appropriately modifies a cell state that is passed from iteration to iteration of training. This cell state is a matrix that acts as the LSTM’s “memory,” in that the LSTM uses the cell state to forget, learn, and remember all information needed for proper image classification [16].

The first gate determines what should be removed from the cell state, in other words, what should be forgotten. The second gate is made of two layers and determines what the cell state should learn. The third gate returns what cell state
information should be modified, and the second layer returns
the information to update the cell state with. The third gate
is responsible for filtering the current cell state to pass into
the next cell state, allowing future iterations to account for
the past. Each gate for each iteration of training uses this
filtered cell state and the corresponding training image as input
and applies each gate’s set of weights, biases, and activation
functions to produce the proper output [16].

Fig. 5 show three iterations of training, where the cell state
is passed from iteration to iteration as x and h. The four yellow
rectangles are four activation functions that make up the three
gates, with the middle two rectangles representing the two
layers of the second gate [16].

D. Data Preprocessing

The success of training depends on the design of the neural
network, but also the quality of the training data. Higher
quality training data, such as clearer pictures and centered
features, leads to a better model. Therefore, there exist many
tools to optimize the given data for more effective
training.

1) Data Augmentation: Data augmentation is a technique
implemented within machine learning to increase the amount
of data available to a model. Data augmentation encompasses
a variety of methods to alter original data. These alterations
may include rotating, cropping, flipping, scaling, and blurring
of images. Data augmentation is able to transform original
data for the model to be trained and tested. This creates
a diverse dataset which gives the model more data to train
upon, bettering its accuracy. The ability to generate a more
expansive dataset, especially when a wide selection of data is
not available, highlights data augmentation’s significance. Data
augmentation not only increases the amount of data, but also
provides varied data which will assist the model in completing
its task with data which is different from the original set.
Applying data augmentation to the machine learning models
theoretically should increase the performance as they are
exposed to these modified samples [9].

E. Model Optimization

When training a supervised learning model, the process will
yield a training accuracy and a testing accuracy. The testing
accuracy shows the model’s predictive ability on novel testing
data, and it is generally the target value for optimization. To
train a neural network, several parameters that must be set,
including the number of layers in the neural network, the
number of neurons in each layer, and the number of training
epochs [17]. It is difficult to predetermine the optimal values
for these parameters, so the values of these parameters are
mostly estimates for the first round of training. Thus, the
first training attempt commonly yields a low training accuracy
and/or testing accuracy. To optimize the model, the appropriate
strategy must be chosen, tested, and adjusted based on the
results of previous training iterations [18]. Poor training results
can be classified into two main categories, underfitting and
overfitting, and the appropriate optimization strategy can be
selected based on which category the training results fall under
[19].

1) Underfitting: Underfitting occurs when the structure of
the neural network is not complex enough or trains for too
few epochs to properly learn the patterns presented in the
training data. It can be identified when training results in both
a low training accuracy and a low testing accuracy. There is
no defined way to calculate the exact solution for underfitting,
but there are several approaches to work toward optimal results
[19].

Increasing the training epochs is one of the most simple
solutions to underfitting. The number of epochs refers to the
number of times the training will iterate over the training data.
A low training and testing accuracy can occur simply because
the model did not have enough epochs to find an optimal loss.
If the training accuracy stops increasing early on, however,
increasing epochs often will not help [19].

Supplying more data for model training is another potential
and rather intuitive fix for underfitting. However, this solution
is not always viable because more training data may not
always be available and it may not fix the root of the problem.
Additionally, increasing the data so that there is an extreme
imbalance between cases might harm the model. For example,
providing a large amount of dog pictures to a model that
differentiates between dogs and cats may result in a poor
model, as the dataset does not have enough images of cats
to create an unbiased model [20].

Increasing the number of layers or neurons in each layer
will most likely be the solution for underfitting. As mentioned
before, underfitting can occur when the model is not complex
even to learn the patterns presented in the training data.
Thus, increasing the number of layers or the number of
neurons for every layer increases the complexity of the model
and the likelihood that the model will be able to learn the
patterns presented in the data. As of right now, there are no
concrete ways to predetermine either of these quantities, so the
best solution is often just to train many models with different
values and see which model performs the best [19].

When working with CNNs, the kernel size is another
parameter to be taken into account. Just as it is difficult to
predetermine the number of layers and number of neurons
in each layer, the optimal kernel size is also difficult to
predetermine. However, it is common that the kernel sizes
of earlier convolutional layers are greater than those of later
convolutional layers [21].

2) Overfitting: Overfitting can occur when the model is too
complex, or the training algorithm runs for too many epochs.
The algorithm would learn the random noise in the training
data that do not contribute to the pattern the model is supposed
to learn. Then, the algorithm would use the extraneous noise to
classify new images and return poor results. Overfitting can
be identified by a high training accuracy but a low testing
accuracy [17].

Overfitting occurs at an unnecessarily large number of
epochs, so the most sensible solution would be to lower the
amount of epochs training runs for. Lowering the numbers
of epochs is called early stopping, and is similar to how increasing the number of epochs can help with underfitting. If the testing accuracy peaks in the middle of training and ends lower than the peak, early stopping will often solve the problem. While this solution is sensible, it will not fix all overfitting problems, for which there are other potential solutions [22].

A set of weights that are significantly larger than the rest is a common sign of overfitting. This is because these weights have a disproportionately large effect on the final classification, meaning the model is too reliant on these few aspects of the training data and will not perform well on new images. Dropout is performed by turning off certain chains of neurons at random throughout training. By ignoring weights at random, the network will have difficulty relying on any weight too much and avoid overtraining. Out of all the strategies of fighting overfitting, dropout is the most advocated for [23]. Regularization has a similar goal as dropout in that it aims to prevent any weights from becoming too large. It does this by placing larger costs on larger weights. In turn, the training algorithm will gravitate away from large weights because its purpose is to minimize the cost. The two forms of regularization are L1 regularization and L2 regularization. The main difference between these two methodologies is that L1 regularization places a cost that is proportional to the weights and L2 regularization places a cost that is proportional to the square of the weights [23]. Underfitting occurs when the model is not complex enough to learn the patterns, and overfitting can occur when the model is too complex. An overly complex model has too many parameters, and thus learns irrelevant details, as mentioned previously. As a result, lowering the number of layers or neurons in each layer for the model can help with overfitting. This strategy of adjusting the architecture of the neural network is often the solution when all other strategies fail [24].

F. Genetic Algorithms

Genetic algorithms are based on the fundamental ideas of evolution put forth by Charles Darwin. Using the concept of natural selection, genetic algorithms run through different combinations of parameters (neurons, layers, etc.) to optimize the run time of the learning process [8]. Genetic algorithms can be applied to the many different aspects of a neural network, but for the purposes of this research, the algorithm will be used to directly enhance the performance of the neural network in terms of its efficiency or results. There are 5 phases to the process of implementing genetic algorithms [25].

1. Initial Population: Every possible combination, each of which is a potential solution to the algorithm, is listed as an individual that belongs to a larger population. The parameters of the individual are denoted as genes that are represented through a string of binary values.

2. Fitness Function: Fitness scores are assigned to each individual through a fitness function based on accuracy. These scores, essentially values between 0 and 1, designate the probability that the individual will be selected for the next generation of testing.

3. Selection: Based on the fitness score of the individuals, two pairs of individuals, known as parents, are created and are passed on for reproduction. The higher the fitness, the more likely an individual is to be selected to become a parent. Those that are not chosen will be killed off as non-potential solutions.

4. Crossover: The parents are mated at a random point in the genes. This point in known as the crossover point. The crossover process is the core of genetic algorithms as it ends up creating new combinations, or offspring, which can be added to the original population. The algorithm is able to sort through the variety of these combinations before settling on the most efficient one.

5. Mutation: Just as mutations are present in evolution, parts of the binary string may be flipped; however, the probability of this occurring is kept very low.

Several iterations of parents and offspring are created, introducing new generations. The algorithm terminates when it determines that the new population created is no longer changing with any new offspring produced. With this, the genetic algorithm can then output a solution that can be run with the training model.

G. Python Programming

To build the machine learning models, Python was chosen as the programming language. Although Python is often slower than C++ [26], another popular language for machine learning, it is suitable for image classification purposes because of its many open source libraries. These provide pre-written functions to aid with data analysis and machine learning. The NumPy library provides easy ways to create, manipulate, store, and access large matrices [27]. TensorFlow is an end-to-end, easy-to-use Python library for training neural networks [28], made even more user-friendly with Keras, an application programming interface (API) that uses TensorFlow as a back-end to aid in building models [29]. These libraries make the implementation of machine learning algorithms much easier and allow for rapid prototyping.

III. EXPERIMENTAL PROCEDURE

A. Dataset

The dataset comes from an online competition conducted by Kaggle in partnership with the TSA [30]. It contains 1247 .aps files, which show full body millimeter wave scans from 16 different angles. Each of the 16 frames is 512 pixels wide and 660 pixels tall. The majority of the examples contain a threat in one of 17 sets, which were predefined by the competition. The competition dataset also includes a .csv file containing the identification numbers for all the .aps files and labels for each of the 17 zones. A one indicates that there is a threat present in the zone, and a zero indicates that there is not [31].

When training the models, different size training sets with different ratios of positive to negative examples were used, depending on which produced the best results. These are further discussed in each algorithm’s respective section. However,
for the purposes of evaluating the models, it was important to control for testing. Therefore, each model was tested on the same set of 100 images, which were kept separate from the other 1147 training images.

B. Evaluation Metrics

When evaluating the data, it was important to note the severe consequences of the algorithm missing a threat. As such, it was important to use metrics other than accuracy for evaluation. Other metrics utilized were precision, recall, and F-1 score.

Precision is the ratio of true positive to predicted positives, or the proportion of predicted threats that were correct [33]. This was important because too many false flags defeats the purpose of implementing a machine learning algorithm for detecting threats.

\[
\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \quad (2)
\]

Recall is the ratio of true positives to actual positives, or the proportion of existing threats that were detected [33]. This was important because missing a potential threat could have disastrous results in the real world.

\[
\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \quad (3)
\]

Finally, the F-1 score is a metric that combines precision and recall into one number between zero and one. It is the harmonic mean of precision and recall, which is better than a plain average because it heavily penalizes low values in either metric. If either precision or recall has a value of zero, then the F-1 score will be zero no matter how high the other is [33].

\[
F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (4)
\]

C. Preprocessing

1) Zone and Perspective Analysis: The data files for the machine learning algorithms were given in an .aps format, which, in this case, is an animation of 16 different angles of a TSA body scan (reference Fig. 7). In order to display the images by themselves, a few separate functions are required. The competition discloses distinct regions of the body, as shown below.

![Body zones, as provided by competition](image)

Different methods were utilized when deciding how to train the model to detect threats on the body; algorithms could focus on the whole body or distinct zones. This research also accounted for the different perspectives of the file, which do not encompass all of the same zones. The algorithms presented in this paper differ based on the method used to approach the data.

2) Zone and Perspective Selection: To focus on specific perspectives and regions of the body, it was necessary to splice the image data. A .csv file was created that contained pixel coordinates which bounded each of the 17 regions. After the program was given the target zones, it used the coordinates in the .csv file to find the coordinates of a bounding region that encompassed all the target zones. Then, each image was read in from the .aps files and saved in a NumPy array. The body's regions of interest were obtained by splicing the NumPy array according to the all-encompassing bounding region. For all the algorithms except the LSTM, different perspectives of the same image were put side by side as one large image, and the combined image was fed into model as one training input.

3) Applied Data Augmentation: Data augmentation was implemented with the ImageDataGenerator class. The ImageDataGenerator class, provided by the Keras library, is compatible with TensorFlow. The ImageDataGenerator class takes images within the dataset and randomly applies modifications to the images based on the parameters specified [34]. The data augmentation parameters that were decided upon were image rotation and shifting of the width and height. These newly modified images were fitted to the models and tested with the evaluation metrics. Considering the fact that the images the model uses are generally standardized, only slight data augmentation parameters were used in order to prevent adverse effects on the results [35]. The rotation values ranged from 5 to 20 degrees and the shift values ranged from 5% to 20% of the original image.

D. Algorithms

1) Logistic Regression: A logistic regression algorithm was implemented using a Sequential model in the Keras library.
Logistic regression outputs the value as a binary value of 1 or 0 which makes it suitable for this type of classification of containing a threat or not. The image was flattened into a one dimensional array and passed through one Dense layer consisting of a Sigmoid Activation layer. The algorithm was run using 40 epochs, trained on 218 images, and tested on 100 images.

Listing 1. Snippet of Baseline Logistic Regression Code

```python
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

model = models.Sequential()
model.add(keras.layers.Flatten( input_shape=(maxX-minX, maxY-minY, numPerspectives)))
model.add(keras.layers.Dense(1, activation='sigmoid'))

# Data augmentation was used on a similar logistic regression algorithm. There were several basic parameters that could be used to alter the dataset. The parameters that were selected were rotation, width, and height shift. The precise values for each of these and the process of selecting these parameters required guessing and checking to find the most optimal changes to the data and their values. The logistic regression with data augmentation was run using 40 epochs, trained on 218 images, and tested on 100 images.

Listing 2. Snippet of Baseline Logistic Regression Code with Data Augmentation

```python
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

model = models.Sequential()
model.add(keras.layers.Flatten( input_shape=(maxX-minX, maxY-minY, numPerspectives)))
model.add(keras.layers.Dense(1, activation='sigmoid'))

# Data augmentation was used on a similar logistic regression algorithm. There were several basic parameters that could be used to alter the dataset. The parameters that were selected were rotation, width, and height shift. The precise values for each of these and the process of selecting these parameters required guessing and checking to find the most optimal changes to the data and their values. The logistic regression with data augmentation was run using 40 epochs, trained on 218 images, and tested on 100 images.

Listing 3. Snippet of Baseline Zone-Specific CNN Code

```python
import tensorflow as tf
from tensorflow import keras

# Perspectives 11−15, 0
sides = 6

# Image dimensions in pixels
width = 130
height = 125

# Building the model
model = models.Sequential()

# Convolutional Layers
model.add(layers.Conv2D(32, (4, 4), activation='relu', kernel_regularizer = tf.keras.regularizers.l2(1=0.01), input_shape=(width*sides, height, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer = tf.keras.regularizers.l2(1=0.01)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_regularizer = tf.keras.regularizers.l2(1=0.01)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())

# Decision Network
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(.5))
```

Because max pooling reduced the dimensions of the image, it was possible to increase the number of output channels with each subsequent layer. From first to last, the number of output channels were 32, 64, and 128. In addition, each layer of convolution went through batch normalization after convolving. This divided each batch by the largest value in the batch, ensuring that all values were between zero and one, with the largest value being one [36]. This resulted in improved performance compared to trials without batch normalization.
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
loss=tf.keras.losses.BinaryCrossentropy(),
metrics=['accuracy'])

history = model.fit(cnn_train, train_labels,
batch_size=2, epochs=40,
validation_batch_size=2)

All convolution and dense layers contained a rectified linear unit ("ReLU") activation function. The ReLU is a piecewise function that returns the input if it is positive, otherwise it returns zero. ReLU activation functions are popular for neural networks because they often lead to better performance [37] [38]. The final, single-node, dense layer had a sigmoid activation function, which returns values between zero and one. It effectively performed logistic regression on the values in the hidden dense layer to output a probability.

To combat overfitting, the model utilized L2 regularization. The proportional constant in the model was 0.01. Another measure taken against overfitting was adding dropout layers with a rate of 0.5 between the dense layers in the decision network. While training the model, it was predicted that recall could be improved by reducing overfitting. To accomplish this, the L2 regularization term was increased by factors of five and ten. However, this had an adverse effect on testing results and no significant effect on training results. Because of this, the model with the regularization term of 0.01 was found to have the optimal results.

The model was trained on a set of 348 images of zone 11 with a 2:1 ratio of negative to positive examples. Only 348 images were used because the entire training set of 1147 contained only 116 examples where a threat was present in Zone 11. After initial training on a set of 232 images with a 1:1 positive to negative split, the model performed poorly. After training on 348 images with a 2:1 split, the algorithm performed much better.

The CNN was trained for 40 epochs with Binary Crossentropy as its cost function and the "adam" optimizer. Both these functions came from the Keras library. Initially, when training the model on the default learning rate of 0.001, the cost did not converge and fluctuated back and forth between various values. To address this, the learning rate was lowered to 0.0001, which resulted in convergence and good testing results.

3) Long Short Term Memory Network: To train an LSTM, the model had to be able to take in a series of images as a single training item. This differed from the CNN, which only took one image for one training item. A time distributed layer allowed several different perspectives of the same scan to be passed into the convolutional layers as one input item, representing the rotation of a scan. The final model consisted of three convolutional layers, each followed by a batch normalization layer and a max pooling layer. The data was then flattened and fed into an LSTM layer with 32 units to account for previous cell states. For the final image classification, the result from the LSTM layer was passed through three hidden layers with dropout layers following the first two. The output layer, just as with the CNN, consisted of one neuron with a sigmoid activation function to characterize the binary classification [39].

Listing 4. Building the Zone-Specific LSTM

def buildCNN(shape):
    momentum = .9
    model = Sequential()

    #first convolutional layer
    model.add(layers.Conv2D(32, (3,3), input_shape=shape,
                            padding='same',
                            activation='relu'))
    model.add(layers.BatchNormalization(momentum=momentum))
    model.add(layers.MaxPool2D())

    #second convolutional layer
    model.add(layers.Conv2D(64, (3,3),
                            padding='same',
                            activation='relu'))
    model.add(layers.BatchNormalization(momentum=momentum))
    model.add(layers.MaxPool2D())

    #third convolutional layer
    model.add(layers.Conv2D(128, (3,3),
                            padding='same',
                            activation='relu'))
    model.add(layers.BatchNormalization(momentum=momentum))
    model.add(layers.MaxPool2D())

    #flatten data for LSTM layer
    model.add(layers.Flatten())
    return model

def action_model(numOutputNeurons = 1):
    shape = (numPerspectives,
             maxX - minX, maxY - minY, 1)
    CNN = buildCNN(shape[1:])
    model = Sequential()
    model.add(layers.TimeDistributed(CNN,
The initial architecture of the LSTM differed significantly from the final architecture and yielded much poorer results. The model had difficulty increasing the training accuracy, signifying underfitting. Since the loss reached a minimum at which it stopped decreasing at an early epoch, the only solutions were to simplify input data and begin guessing at different architectures. To simplify the input data to reduce the required complexity of the LSTM, only the thigh region was fed as training input. This meant the LSTM only had to analyze the leg instead of the entire body, reducing the input data’s complexity to alleviate underfitting.

The underfitting persisted after modifying the input data, so the number of hidden layers and number of neurons in each hidden layer were increased to increase complexity. The focus in a specific region of the body meant the structure of the neural network did not have to be modified drastically, allowing for reasonable guessing. Eventually, an appropriate complexity was found, upon which the training accuracy increased consistently throughout epochs. The testing accuracy, however, saw less success, meaning overfitting was occurring. To combat the overfitting, two dropout layers were added after the first two hidden layers, arriving at the final architecture.

4) Genetic Algorithm: Running the genetic algorithm requires a series of parents and children to create a population of different combinations. The algorithm is given a set of variables assigned to probabilities that will be applied to the population through different generations. The probability that a population would be kept after a generation was set to 40%, the probability that a combination would be rejected was set to 10%, and the probability that an individual would mutate was set to 20%. These percentages are determined by previous research that has tested out these values. A function was used to create a population and generate the fitness function, which was based on the accuracy of the individual being tested. Separate functions were also created to breed, mutate, and evolve the population. As the genetic algorithm passed through different generations, the different metrics were logged in a file to be able to reference later. The parameters that were tested in this genetic algorithm were the number of neurons, the number of layers, the activation function, and the optimizer, and then these parameters were passed into the parameters that helped create the model for the neural network and its hidden layers. These were the parameters chosen because they are the foundation for the hidden layers that train and test the data.

Listing 5. Creating neural network model with genetic algorithm parameters

```python
def compile_model(network, nb_classes, input_shape):
    # access network parameters
    nb_layers = network['nb_layers']
    nb_neurons = network['nb_neurons']
    activation = network['activation']
    optimizer = network['optimizer']

    # add layers
    model.add(Flatten(input_shape=input_shape))
    for i in range(nb_layers):
        model.add(Dense(nb_neurons, activation=activation))

    # output layer
    model.add(Dense(1, activation='sigmoid'))
    model.compile(
        loss='binary_crossentropy',
        optimizer=optimizer,
        metrics=['accuracy', 'precision_m', 'recall_m', 'f1_m'])
```

In testing the combinations, the parameters were given base choices to work with. The possible number of neurons were set to 32, 64, and 128. The number of layers were set between 1 and 4. The activation layers were assigned as either ReLU, ELU, or tanh. The Exponential Linear Unit (ELU) works the same as the ReLU with positive numbers but creates an exponential output for the negative numbers [41]. The tanh function, which is just a scaled sigmoid function, bounds its output between -1 and 1 while being differentiable [42]. The optimizer was left as only “adam” as it is able to achieve fast results by maintaining the learning rate as it applies to each weight in the neural network and parameter [43]. The code for the genetic algorithm itself was based on the algorithm proposed by Matt Harvey, the founder of Coastline Automation, and changes were made to incorporate the specific TSA data [44] (see listing 6 in the appendix). The validity of the genetic algorithm was then tested based on the metrics that the neural network outputted. The neural network itself was only trained on zone 11 just to keep the simplicity.
of the neural network while placing emphasis on testing the results of the genetic algorithm.

IV. RESULTS AND ANALYSIS

Fig. 8. Bar graph with results from all algorithms.

A. Logistic Regression Results

As shown in Fig. 8, the logistic regression algorithm without data augmentation had a test accuracy of 71.00%, a precision of 100.00%, a recall value of 63.75%, and an F-1 score of 77.86%. The logistic regression algorithm when used with data augmentation returned better results with a test accuracy of 86.00%, a precision of 97.14%, recall of 85.00%, and an F-1 score of 90.67%. The logistic regression with data augmentation performed significantly better than the algorithm that used data augmentation which can be explained by the data that the algorithms receive. This is because data augmentation increased the diversity of the training data, which led to an increase its accuracy on test data. In this case, the image scans were very consistent, however, the data augmentation provided enough diversity in the scans of the body for it to result in a better precision, recall and F-1 score than the logistic regression without data augmentation.

B. Convolutional Neural Network Results

The CNN trained for 40 epochs on a training set of 348 images with a 2:1 ratio of negative to positive examples. As shown in Fig. 6, the CNN had a test accuracy of 99.0%, a precision of 100%, a recall of 88.89%, and an F-1 score of 94.12%. These results confirm the prediction that the CNN would perform better than the logistic regression model. As mentioned in the Background section, logistic regression is unable to capture the spatial dependencies (shapes and lines in the image) that CNNs can. However, it was unexpected that the CNN performed just as well as the LSTM since, theoretically, the LSTM would be able to capture the temporal dependencies of the video, which the CNN cannot. This is discussed further in the LSTM results.

C. Long Short-Term Memory Network Results

The LSTM performed with an accuracy of 99%, precision of 100%, recall of 88.89%, and F-1 score of 94.12%. This exactly matches the result of the CNN, despite the LSTM’s ability to analyze sequences of images. The lack of difference between the results of the two models can be attributed to the small testing set. To keep results consistent, the TSA’s testing set of 100 images was used for all algorithms. Both the CNN and LSTM missed exactly one image, meaning the only better result would be perfect percentages. Thus, better results with the LSTM would be near impossible, so the difference between the CNN and LSTM was not highlighted with merely 100 images. To truly show the superiority of the LSTM, a larger testing set would be needed.

D. Genetic Algorithm Results

The genetic algorithm ran through many populations before settling on the very last individual tested with the following results for the parameters: 128 neurons, 4 layers, tanh activation, “adam” optimizer. The reasoning behind why this combination was predicted by the algorithm can be explained with previous research. Out of the options passed in for the number of neurons, 128 was reported to give the best results for the neural network. It has been reported that not enough neurons can prevent the model from learning the classification patterns while an increasing number of neurons leads to a logarithmic growth in accuracy. At 128 neurons, the growth starts to subside and the accuracy does not change much with the increase in neurons [44]. Increasing the number of layers is able to increase the depth of the neural network, which explains why 4 layers was chosen- it is the highest number of layers proposed, which allows for the neural network to compute its results more efficiently [45]. The tanh activation function was chosen as the tanh function works best when classifying between two outputs as opposed to having an output with no bounds, which would be created with the ReLU and the ELU [46]. With these parameters passed into the normal neural networks the accuracy was reported to be 81%, the precision was reported to be 43.3%, the recall was 65%, and the F-1 score was reported 52%. Compared to the other models ran the results were low as the neural network was not specialized to deal with images such as the CNN. While the genetic algorithm gave its best output, the neural network was optimized to give results to the best of its ability. The 81% accuracy proves that the genetic algorithm provided good results and that it worked as intended.

V. CONCLUSIONS

The goal of the project was to determine the optimal algorithm for detecting threats in airport scans. This research shows that the zone specific CNN and LSTM would be the best choices for implementation. These algorithms performed better than logistic regression and genetic algorithms and were extremely successful on the testing set, as seen in Fig. 8. Each model only misclassified one image out of a test set of 100. Most notably, they both had a precision of 100%, meaning...
no false positives. If implemented, these would be a drastic improvement over the current false positive rate of 54%. Based on these results, it is recommended that the TSA implement a CNN or LSTM trained on specific zones in order to improve the efficiency and efficacy of passenger screening.

A. Future Work

1) Greater Data Acquisition: More data is necessary to properly evaluate and compare the algorithms. As discussed earlier, it is believed that the lack of difference in performance between the CNN and LSTM is due to the small testing size, but more data would be necessary to verify this hypothesis. More data would also be necessary to better evaluate the recall of these two algorithms, since the testing set only included nine positive examples. This is especially important because of the severe consequences of missing potential threats in airport security. Although the purpose of the project was fulfilled in that the best models were identified, further work is necessary to produce conclusive and meaningful results.

2) Extended Zone and Perspective Analysis: Considering the limited time frame available to complete the research, it was not possible to implement as many approaches hoped for. However, in the future, the dataset would be analyzed using different body regions and zones. The logistic regression models were trained and tested based on a full body view, primarily from the angle 0, a front-facing view. However, the convolutional neural network, long short-term memory network and genetic algorithms focused on zone 11 (reference Fig. 5). This project was more of a proof of concept, as none of the models were able to identify the specific zone of a threat when analyzing the entire body. Future analysis would include a full analysis of the body by region by all the algorithms. These regions could be the given 17 or there could be three regions consisting of torso, arms, and legs. This would be accomplished by dividing the image into its zones and training a separate model for each region.

3) Combination of Classification Techniques: While individually, the classification techniques worked well, it could be possible to utilize multiple in the same model to generate better results. A proposed idea for a combination of the different approaches would be to use a genetic algorithm to determine the optimal number of neurons, layers, and activation functions in a combined CNN and LSTM in addition to utilizing data augmentation for the most effective algorithm. One of the challenges of doing this, however, would be the data augmentation. Since the data is already very normalized, determining the right parameters that would have to be used would be difficult. However, if the proper parameters and values were found (such as rotation degrees, width shift percentages, and height shift percentages), the combination of these different algorithms would theoretically result in a more effective model.

4) Applications: As the TSA runs through security checks, it seeks to implement machine learning into its processes on a regular basis [46]. As many new systems implement machine learning into their tasks, they are able to emphasize the value of becoming efficient and effective. Understanding which type of model to use and any preprocessing techniques to adopt can help provide results that are more accurate. In a potentially life-threatening situation, the results have shown that applying the CNN to the security checks will be able to accurately and efficiently detect threats. This allows for the security check process to run faster and smoothly with less human error present. Implementing the CNN into security scans can bring the nation to a safer state as people can rely on the system that detects threats.

APPENDIX

Listing 6. Optimizer from genetic algorithm [47]

```python
from functools import reducerom operator import add
import random
from network import Network
class Optimizer():
def __init__(self, nn_param_choices, retain=0.4, random_select=0.1, mutate_chance=0.2):
    self.mutate_chance = mutate_chance
    self.random_select = random_select
    self.retain = retain
    self.nn_param_choices = nn_param_choices

def create_population(self, count):
    pop = []
    for _ in range(0, count):
        network = Network(self.nn_param_choices)
        network.create_random()
        pop.append(network)
    return pop

@staticmethod
def fitness(network):
    return network.accuracy

def grade(self, pop):
    summed = reduce(add, (self.fitness(network) for network in pop))
    return summed / float((len(pop)))

def breed(self, mother, father):
    children = []
    for _ in range(2):
        child = {}
        for param in self.nn_param_choices:
            child[param] =
```


random.choice([mother.network[param],
              father.network[param]])
network = Network(
self.nn_param_choices)
network.create_set(child)
if self.mutate_chance >
random.random():
  network =
self.mutate(network)
children.append(network)
return children
def mutate(self, network):
mutation = random.choice(list(self.nn_param_choices.keys()))
network.network[mutation] =
random.choice(self.nn_param_choices[mutation])
return network
def evolve(self, pop):
  graded = [x[1] for x in sorted((graded, key=lambda x: x[0], reverse=True))]
  retain_length =
  int(len(graded) * self.retain)
  parents = graded[:retain_length]
  for individual in graded[retain_length:]:
    if self.random_select >
      random.random():
        parents.append(individual)
  parents_length = len(parents)
  desired_length = len(pop) -
  parents_length
  children = []
  while len(children) <
    desired_length:
      male = random.randint(0,
                              parents_length - 1)
      female = random.randint(0,
                              parents_length - 1)
      if male != female:
        male = parents[male]
        female = parents[female]
        babies = self.breed(male, female)
        for baby in babies:
          if len(children) <
            desired_length:
            children.append(baby)
  parents.extend(children)
  return parents

ACKNOWLEDGEMENTS

The authors of this paper gratefully acknowledge the following: Project mentor Sonia Purohit, for her valuable knowledge of machine learning and image processing; Residential Teaching Assistant and Project Liaison Steven Chen for his invaluable assistance; Director Dean Jean Patrick Antoine and Director Emeritus Dean Ilene Rosen, for their management and guidance; Head counselor Rajas Karajgikar for his continual guidance and support; Research Coordinator Benjamin Lee for his assistance in conducting proper research; Rutgers University, Rutgers School of Engineering, and the State of New Jersey for the chance to advance knowledge, explore engineering, and open up new opportunities; Lockheed Martin and NJ Space Grant Consortium for funding of our scientific endeavours; and lastly NJ GSET Alumni, for their continued participation and support.

REFERENCES

“ECOSYSTEM.” NumPy.


“Passenger Screening Algorithm Challenge.” (Overview) Kaggle.

“Passenger Screening Algorithm Challenge.” (Data) Kaggle.


“tf.keras.preprocessing.image.ImageDataGenerator,” Tensorflow.

F. Chollet “Building powerful image classification models using very little data.” The Keras Blog, 05, June 2016.


“Alpha.” Exponential Linear Unit (ELU) Layer - MATLAB.


“How Does the Number of Hidden Neurons Affect a Neural Network’s Performance.” How Does the Number of Hidden Neurons Affect a Neural Network’s Performance, 3 Jan. 2016.


