Technical Report

THE APPLICABILITY OF MACHINE LEARNING METHODS ON INFRARED VIDEO DATA

Victoria Gerardi

September 2020

U.S. ARMY COMBAT CAPABILITIES DEVELOPMENT COMMAND ARMAMENTS CENTER

Enterprise and Systems Integration Center

Picatinny Arsenal, New Jersey

CONTENTS

LIST OF FIGURES AND TABLES

PREFACE

Situational awareness is a necessity for the warfighter. A common surveillance method is making use of sensors. Electro-Optical/Infrared (EOIR) Sensors use both visible and infrared sensors, allowing them to be used in both light and dark (day/night) scenarios. These systems are often used to detect Unmanned Aircraft Systems (UAS). Recognizing these objects in the sky requires diligence from the human that is monitoring the system. The purpose of this report is to investigate the feasibility of using convolutional neural networks on Infrared data to identify images of UAS in the sky. The data used for this project was provided by the - Precision Targeting and Integration group in the Combat Capabilities Development Command – Armaments Center.

SUMMARY

The report considers image data from IR sensors. These images are fed into a feed forward convolutional neural network which classifies images as either having a UAS or not. The convolutional model proved to be effective first attempt at working with this data. This report provides a future direction to expand upon in the future. Recommendations include fine tuning this model, as well as using other machine learning methods on this data set such as object detection and the "You only look once" (YOLO) algorithm.

INTRODUCTION

Background

This report represents an investigation into using a machine learning algorithm to analyze Infrared video data, with a possible outcome of this effort being the improved performance of Counter-Unmanned Aircraft Systems (C-UAS) developed by the Combat Capability Development Command's Armament Center (CCDC AC).

Problem

Infrared (IR) Sensors are sensors used in military applications which have both visible and infrared sensors. These sensors provide situational awareness for both day and night scenarios. The general operation of these sensors is below:

1. Detect air movement (UAS, planes, birds, tree movement, etc.) and signal to human in the loop of the detection
2. The human in the loop decides if this movement is a UAS
3. The human uses the optical tracker to lock in on the target
4. A fire mission is created on the weapon system

The data used for this project consisted of 10 videos. These videos ranged between 0:25 minutes and 4:59 minutes. The videos contain footage of drones in the sky. At times, clouds are also visible in the sky.

Currently, a human is used to identify when a drone appears in the camera. The goal of this project/investigation is not to replace the human in the loop. This project aims to investigate the initial feasibility of a machine learning algorithm to identify UAS in the videos. This investigation may lead to other applications of machine learning related to this application. This report details a "baseline" of possibilities regarding the possibilities of wielding machine learning for this application.

METHODS, ASSUMPTIONS, AND PROCEDURES

Methodology

All data preprocessing and model building was done using Python 3.6.8. Specific packages include numpy 1.18.1, pandas 1.0.3, TensorFlow 2.1.0, scikitlearn 0.21.1, and matplotlib 3.1.3.

Data Preprocessing

The data used for this project consisted of 10 videos which were provided by the Optical Tracking Group. These videos ranged between 0:25 seconds and 4:59 minutes. The videos contain footage of drones in the sky. At times, clouds are visible in the sky. Frames were cut from these videos at every half second. This resulted in 2812 images. Examples of these frames are below. Notice the green bars that are evident on the top/bottom or sides of the images.

Figure 1
Sample frames from the video dataset

Next, these images were then cropped to a pixel dimensions of 256X256. This 256X256 square was randomly taken from within the image, with a buffer to account for the green edging on the images. This is done so that the model is only looking at the sky view. Examples of these frames are in **Figure 2**.

Figure 2
Sample data points used for model development and testing

Then, all images were labelled according to whether they contained a UAS (1) or not (0).

Data Integrity

The data used for this project consisted of 10 videos which were provided by Precision Targeting and Integration group in the Combat Capabilities Development Command – Armaments Center. These videos ranged between 0:25 minutes and 4:59 minutes. The videos contain footage of drones in the sky. At times, clouds are visible in the sky.

Through the process of working with these videos, there are several things to make note of in regards to the data used and the resulting model. To start, the dataset's negative samples do not contain any non-UAS objects such as birds. All negative images are simply the sky with various clouds. It is expected that the end model will over predict any detected object as a UAS. Furthermore, all UAS in the positive sample class were solid black or white colored. Again, this could have implications for the model's predicting power.

It is important to note that all samples were manually labelled. It is expected that there may be some amount of human error involved in this process. In the training set of 1912 images, 1037 belonged to the 0 class and 875 belonged to the 1 class. That is, roughly 46% of images in the training set contained UAS. This distribution most likely does not align with what the camera actually faces in real life. Because of how the images are cropped, the UAS may be located in any region of the sample image. Furthermore, the size of the UAS within each image does vary, but may not be consistent with what the IR sensor may come across in real life. These examples above are all examples of biases within the data that can have an impact on the resulting model. It is important to keep these caveats in mind when evaluating model performance.

Initial Model Building

Four models were built using different hyperparameters to compare performance. After this initial step, hyperparameters of the models can be tuned for further adjustment. The base model is a convolutional neural network (CNN). Convolutional neural networks pass filters of weights over the data which pick up on spatially located features within the data. This allows features to be noticed in different spatial regions of the image. For example, a cat ear can be noticed no matter what corner/area of the image it is in. While very useful for computer vision on image data, CNN can be applied to other types of data. CNN are often more stable than dense feed forward networks because CNNs keep the total learnable parameter count low without sacrificing model complexity.

Thus, they generally perform better than dense feed forward networks on high resolution images. An example of this architecture is displayed in **Figure 3.** Convolutional layers are often followed by a pooling layer. For these networks, we used a max-pooling layers. Example of how a max pooling layer works is in **Figure 4.** Max-pooling results in an abstract representation of the inputs. It reduces data dimensions, speeds up the learning process, mitigates model overfitting, and allows for assumptions to be made from the down-sampled features.

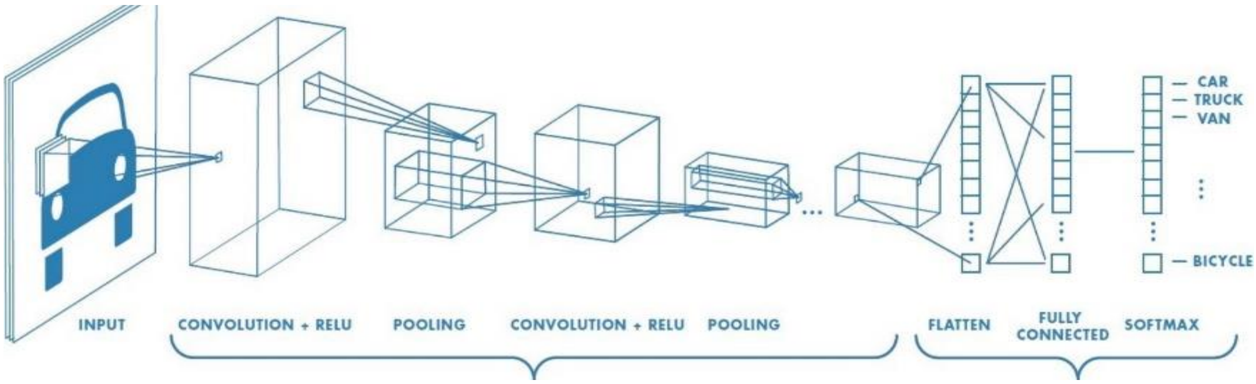


Figure 3
Convolutional Neural Network Example

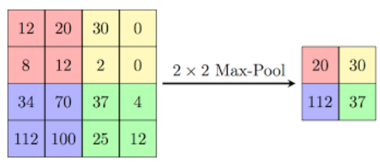


Figure 4
Max Pooling Layers

The model was built using 100X100 images and trained on up to 50 epochs. The models are convolutional neural networks using the Adam optimizer. The Adam optimizer is a combination of the Momentum optimizer and the RMS Prop optimizer. The loss function use is binary cross entropy, and the metric captured during training is accuracy. The function for binary cross entropy is **Equation 1.** Note in this equation that y_i is the label and $p(y_i)$ is the predicted probability for sample i. The models examine here used three or four convolutional layers, with max pooling in between. Two of these models employ an early stopping condition.

The training dataset contains 1912 samples, the validation set has 477 samples, and the test data set contains 423 images. The training set was split into 25 batches for training, with each batch containing 75 samples.

$$Loss = -\frac{1}{N}\sum_{i=1} y_i * \log\big(p(y_i)\big) + (1 - y_i) * \log\big(1 - p(y_i)\big)$$

Equation 1
Binary Cross Entropy Function

The confusion matrix for the validation set is used to compare model performance. Metrics calculated from the confusion matrix are precision, recall, the F1 score, and accuracy. Precision is the percentage of predicted positives that are correctly classified. It is represented by $Precision = \frac{TP}{TP+FP}$. Recall is the percentage of actual positives that are correctly classified. It is represented by $Recall = \frac{TP}{TP+FN}$. The F1 score represents the harmonic mean of precision and recall and is represented by $F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$. The last statistic is accuracy, which is a measure of the proportion of correct classifications, which is $Accuracy = \frac{TP+TF}{TP+FP+TN+TN}$. It is expected that a "good" model performs well according to these (and other) performance metrics. While this is not always true, and these metrics can be misleading at times, it is still worth it to examine this data.

For our specific problem, false negatives are "worse" than false positives. It is much better to take second look at a sample where no UAS is present than to pass over a sample where a UAS is indeed present, in regards to protecting the warfighter. We want to minimize false negatives in our models.

The performance of all the models on the test dataset will be compared at the end of this section. Performance statistics comparing all models can be found in **Table 9.**

RESULTS AND DISCUSSION

Model A

The first model used three convolutional layers, each followed by a max pooling layer. A visual diagram of the network is depicted in **Figure 5.** This model did employ an early stopping condition. The early stopping condition was based on the validation accuracy, and had a condition to stop training after 5 epochs of no 'improvement' which is defined by a change of 0.001. The model has a maximum number of epochs to train set at 50, but stopped training after 12 epochs.

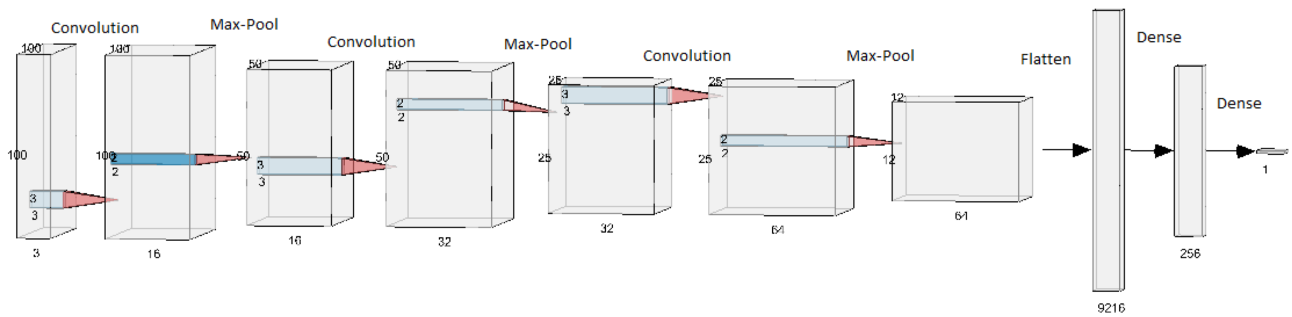


Figure 5
Model A Architecture

A confusion matrix of the validation set is in **Table 1.** The validation set contains 477 samples. This table includes precision, recall, F1 score, and accuracy against the validation set. This model has a precision of 0.929, a recall score of 0.925 an F1 score is 0.927, and the accuracy is 0.931. Something to note from the confusion matrix in **Table 1** is that false positives occur just as much as false negatives. In the final model, the rate of false negatives should be minimized. Out of the 477 samples, 33 were incorrectly classified in total.

Table 1
Confusion Matrix of Validation Set for Model A

	Predicted 0	**Predicted 1**
Actual 0	234	16
Actual 1	17	210

The loss and accuracy over all epochs is displayed in **Figure 6.** The model approaches a 98% accuracy rate on the training set, while the validation accuracy hits a maximum of 0.9578 after the 7th epoch. This indicates that the model may be beginning to overfit on the training data, which can present an issue if this model is deployed on new data. Possible solutions are discussed at the end of this paper.

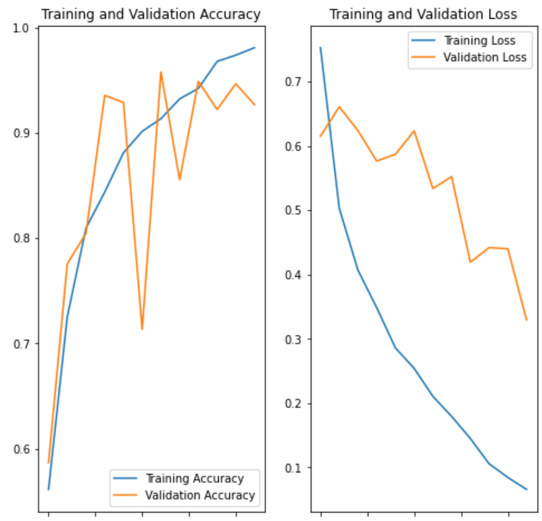


Figure 6
Learning Curves for Model A

Model B

This model used three convolutional layers, each followed by a max pooling layer. A visual diagram of the network is depicted in **Figure 7.** This model uses the same model architecture as Model A, but does not employ an early stopping condition. This model trains for 50 epochs.

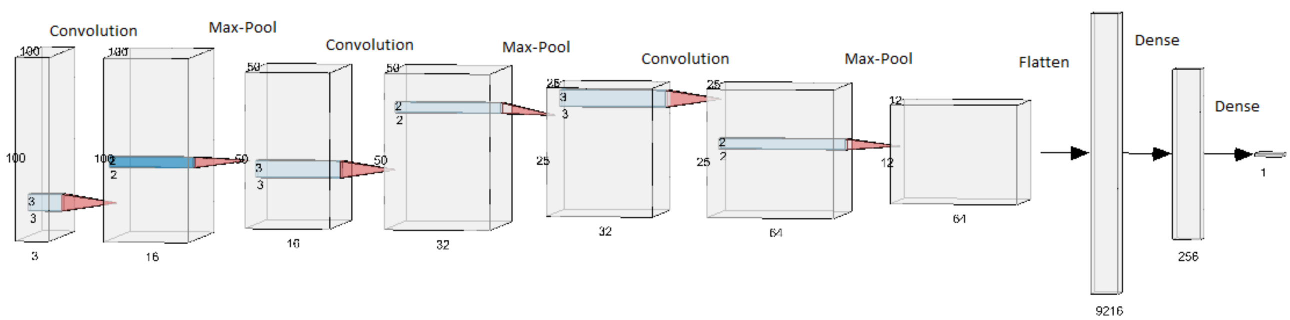


Figure 7
Model B Architecture

A confusion matrix of the validation set is in **Table 2.** This model has a precision of 0.948, a recall of 0.922, the F1 score is 0.935, and the accuracy is 0.941. This model incorrectly classified 28 samples from the validation set. This model has 6 more false negatives than false positives. For our problem of identifying UAS, false negatives are less desirable than false positives.

Table 2
Confusion Matrix of Validation Set for Model B

	Predicted 0	**Predicted 1**
Actual 0	247	11
Actual 1	17	202

The loss and accuracy over all epochs is displayed in **Figure 8.** The model severely over trains on the training set, settling at 100% accurate, while the validation accuracy hovers around 0.95. This overfitting could be mitigated by regularization techniques such as dropout or data augmentation, which are covered at the end of this paper. It can also be seen that the validation loss began rising in the last 30 or so epochs in the training process. This should be investigated in future iterations of this project.

Figure 8
Learning Curves for Model B

Model C

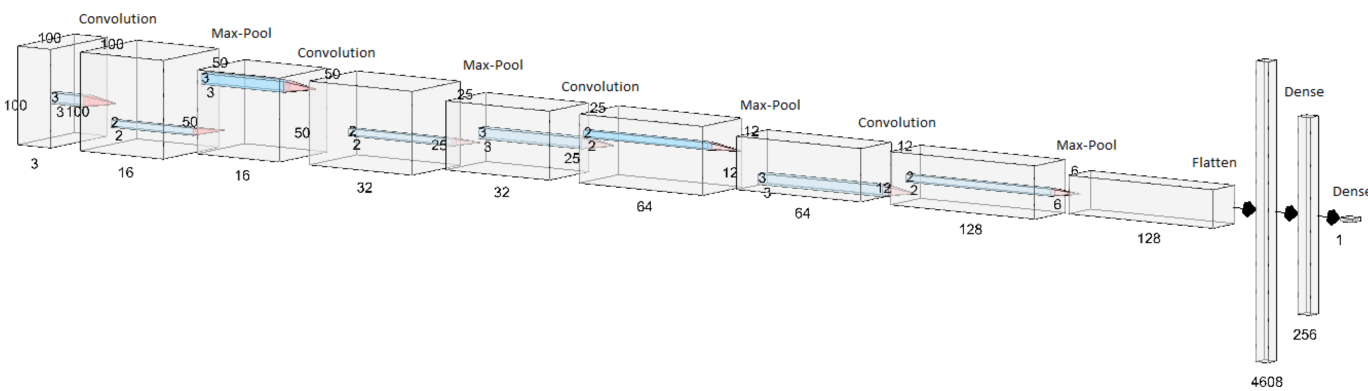


Figure 9
Model C Architecture

This model used four convolutional layers, each followed by a max pooling layer. Notice this is one more convolutional layer than both Model A and Model B. A visual diagram of the network is depicted in **Figure 9**. The model did not employ an early stopping condition. This model trained for 50 epochs.

A confusion matrix of the validation set is in **Table 3**. This model has a precision of 0.971 and a recall of 0.923. For this model, the F1 score is 0.946 and the accuracy is 0.952. This model has more than three the number of false positives as false negatives, which is concerning. One the other hand, out of 477 samples, only 23 were incorrectly labelled.

Table 3
Confusion Matrix of Validation Set for Model C

	Predicted 0	**Predicted 1**
Actual 0	251	6
Actual 1	17	203

The loss and accuracy over all epochs is displayed in **Figure 10.** The model over trains on the training set, settling at 100% accurate, while the validation accuracy hovers around 95%. This overfitting could be mitigated by regularization techniques such as dropout or data augmentation. The methods are mentioned at the end of this paper for future use. It can also be seen that the validation loss began rising in the last 30 or so epochs in the training process. This should be investigated in future iterations of this project. It also appears that the model stopped learning/improving its accuracy at about 30 epochs in. The model may have gotten stuck in a local minimum or saddle point. This should also be investigated in future work.

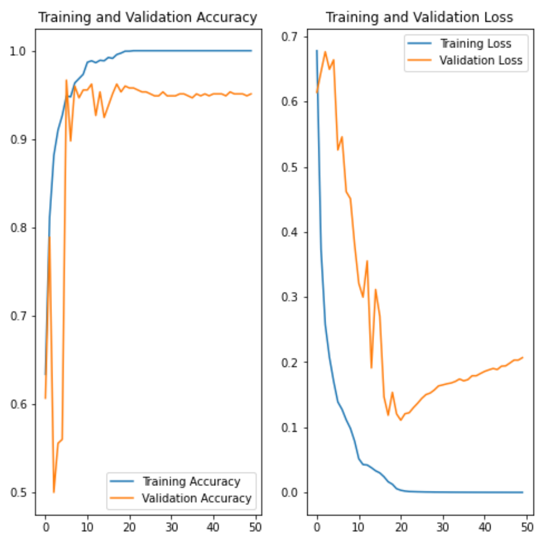


Figure 10
Learning Curves for Model C

Model D

This model used four convolutional layers, each followed by a max pooling layer. A visual diagram of the network is depicted in **Figure 11**. The model did employ an early stopping condition. This model did employ an early stopping condition. The early stopping condition was based on the validation accuracy, and had a condition to stop training after 5 epochs of no ‘improvement’ which is defined by a change of

0.001. The model has a maximum number of epochs to train set at 50, but stopped training after 15 epochs.

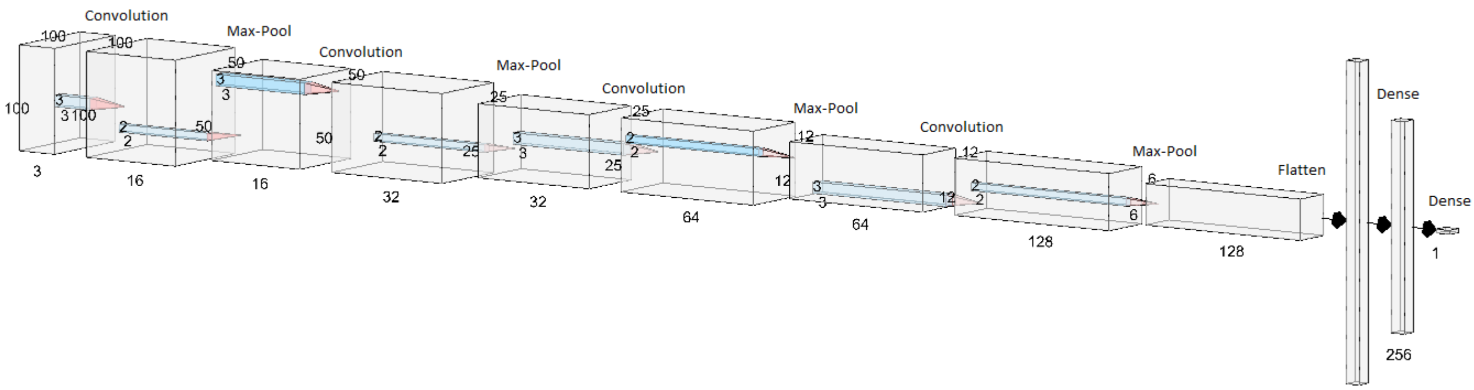


Figure 11
Model D Architecture

A confusion matrix of the validation set is in **Table 4**. This model has a precision of 0.963, a recall of 0.986, the F1 score is 0.975, and the accuracy is 0.977. This model has a low number of false negatives in relation to false positives. For this particular problem, this is preferable. Out of the 477 samples in the validation set, 11 samples were mislabeled, and 3 of these were UAS which were labelled as “not UAS”.

Table 4
Confusion Matrix for Validation Set of Model D

	Predicted 0	**Predicted 1**
Actual 0	256	8
Actual 1	3	210

The loss and accuracy over all epochs is displayed in **Figure 12.** The model reaches a maximum training set accuracy of 0.9848, while the maximum value the validation accuracy hits is 0.9733. The early stopping condition stopped the model training before it had the opportunity to severely over train on the training set. If this model were to be deployed on new data, this overfitting would most likely not present as large of an issue as the previous models.

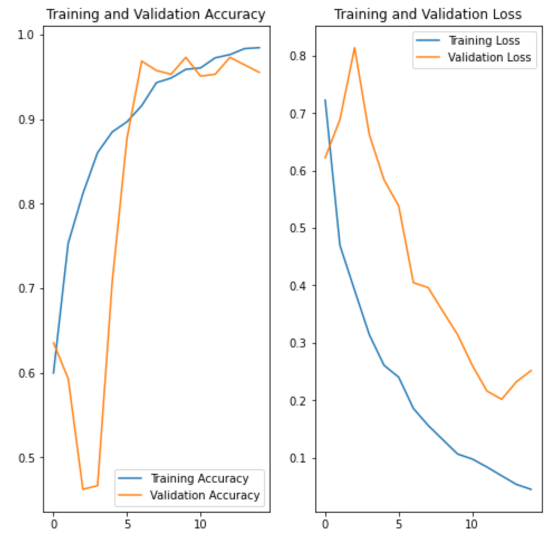


Figure 12
Learning Curves for Model D

Test Data Set Performance

423 images were reserved from the data set to be used in the test data set. Confusion matrices for the performance on the test data set of the four models are located in **Tables 5, 6, 7,** and **8**. The performance statistics comparing all models can be found in **Table 9**. As seen in **Table 5**, model A has 54 false negatives and 1 false positive. The high amount of false negatives contributes to the low recall score for this model, which is in **Table 9**. Out of al 423 images in the test data set, this model incorrectly classified 55 of them.

Table 5
Confusion Matrix for Model A on the Test Data

	Predicted 0	**Predicted 1**
Actual 0	228	1
Actual 1	54	140

The confusion matrix for model B is in **Table 6**. This model has 31 incorrect predictions, with 10 false positives and 21 false negatives. Performance statistics are in **Table 9.** Model B performs better than model A when comparing the recall, F1 score, and the accuracy. However, it still has a high proportion of false negatives.

Table 6
Confusion Matrix for Model B on the Test Data

	Predicted 0	**Predicted 1**
Actual 0	219	10
Actual 1	21	173

The confusion matrix for model C is in **Table 7**. This model has only 20 incorrect predictions, with 5 false positives and 15 false negatives. Performance statistics are in **Table 9.** Model C performs better than model A and B when comparing the precision, recall, F1 score, and the accuracy.

Table 7
Confusion Matrix for Model C on the Test Data

	Predicted 0	Predicted 1
Actual 0	224	5
Actual 1	15	179

The confusion matrix for model D is in **Table 8**. This model has 20 incorrect predictions, with 8 false positives and 12 false negatives. Performance statistics are in **Table 9.** Model D performs better than model A and B when comparing the precision, recall, F1 score, and the accuracy. It also performs better than model C when comparing the recall and F1 score.

Table 8
Confusion Matrix for Model D on the Test Data

	Predicted 0	Predicted 1
Actual 0	221	8
Actual 1	12	182

Performance statistics are in **Table 9.** Model D outperforms models A, B, and C overall. However, model C is very close/ equal for the F1 score and accuracy level. The significance of these results are discussed in the next section.

Table 9
Test Data Confusion Matrix Statistics for Model Comparison

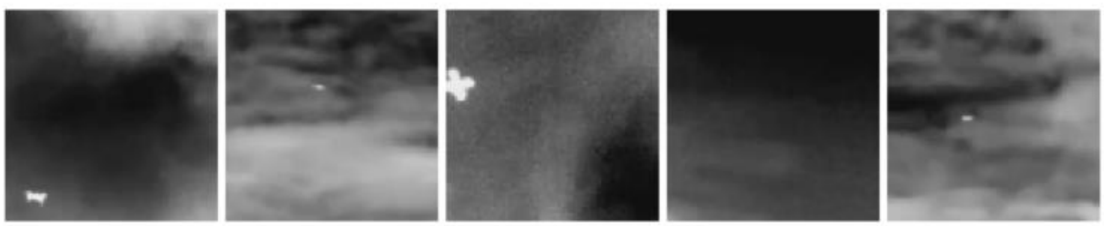
	Model A	Model B	Model C	Model D
Precision	0.993	0.945	**0.973**	0.958
Recall	0.722	0.892	0.923	**0.938**
F1 Score	0.839	0.918	0.947	**0.948**
Accuracy	0.870	0.923	**0.953**	**0.953**

CONCLUSIONS

When comparing the performance of these four models, model D shows the most promising results. It has the highest recall and F1 score. In addition, it only has 20 incorrect prediction in the test set, with 12 of those being false negatives. Model D has the same accuracy on the test set as model C. Model C also has the highest precision score on the test data set. Recall that model A and B has three convolutional layers, while model C and D had four. This added convolutional layer allows the model to better learn the complex features in the dataset. The difference between model C and D is the number of epochs trained for. Model C trained for 50 epochs while model D had an early stopping condition and stopped after 15 epochs. Overall, model D performed slightly better than model C, a sign that model C is likely over trained to the training data and the 50 epochs were not all necessary.

Below is an example of what model output looks like when predicting if an image has a UAS or not. The tensor containing 0’s and 1’s are the actual values for the subsequent five images. The five values after the images are the predictions. Positive values indicate that the model identified a UAS in the image, while negative values mean it did not.

```
[1. 1. 1. 0. 1.]
```



```
tf.Tensor(
[[ 17.]
 [  5.]
 [  6.]
 [-10.]
 [ 14.]], shape=(5, 1), dtype=float32)
10 15
[1. 0. 1. 0. 1.]
```

```
tf.Tensor(
[[ 10.]
 [-10.]
 [ 11.]
 [ -8.]
 [  6.]], shape=(5, 1), dtype=float32)
20 25
```

Figure 13
Sample Model Output

Based on the performance of all of these models on this data, this application seems to be feasible, and seems to have fruitful outputs that can be useful. In addition, there is still work to be done with this data and these models.

RECOMMENDATIONS

Increase Training Dataset

This model could be made more robust through increasing the size of the training data set. Only 2250 images were used to train this model. The data set could be increased through a few different ways. One method is through data augmentation. Data augmentation allows us to increase diversity in the data we have available to us without collecting more images. Some data augmentation techniques are: rotating images, flipping images, adjusting brightness levels, cropping images, etc. This added diversity to the training set can also help mitigate overfitting.

Another way to increase the size of the training dataset could be to capture more frames from the original videos. Of the ten videos we have access to, nine videos have a frame rate of 29.97 frames/second. One has a frame rate of 25 frames/second. To create the dataset used for these models, roughly two frames were captured per second. It is recommend to capturing capture more frames if more computational power is available.

Hyperparameter Tuning

For this initial exploration, four models were tested, with small differences between them. It is recommended to consider hyperparameter tuning for future investigations. Hyperparameters that should be looked into for tuning are: learning rate, batch size, size and number of convolutional layers, the architecture of dense layers, the size and stride of max-pooling filters, beta 1 and beta 2 for the Adam optimizer. Depending on time and computational resources, computational power, it may not be feasible to efficiently tune all of the hyperparameters previously listed. In this case, it may be helpful to have the resources to run different models in parallel to save time.

Hyperparameter tuning is an active area of research and there are many ways to optimization techniques to consider for future work. One such way is the grid search method. This imposes a grid on the hyperparameter space and tests values in each cell of the grid to evaluate the model. Depending on the possible space of hyperparameters, this grid of possible values may include some combinations of hyperparameters that do not make sense given the problem scope. Another method, which is recommended for this problem, is the random sampling method. Random sampling allows for a more efficient search of the hyperparameter space and allows for a high probability of finding a reasonable set of well performing parameters quickly. In this manner, it is possible to focus in on the “area” of well performing parameters.

Mitigating Overfitting

In all models, we saw a tendency for the model to overfit to the training dataset. Overfitting is dangerous because it can prevent the model from generalizing well to data (images) it has not seen

before. There are many methods used to prevent overfitting, some of which were attempted or previously mentioned.
One method is implementing an early stopping condition. The idea behind this is that it identifies when a model has reach a point of diminishing returns in regards to a specified metric, and stops the model from continuing to train, ideally stopping it before it can overfit to the biases within the training dataset. Another method is dropout. Dropout is one of the most popular regularization techniques for neural networks. Dropout eliminates a certain percentage of connections between one layer of the neural network and the next, according to a dropout probability (a hyperparameter set by the user). With dropout in place, individual nodes cannot rely on the output of others as they previously did, so every node must output useful features on their own.

Lasso and ridge regularization are also regularization options that can help with overfitting. These weight regularizations add a weighted parameter to the cost function (binary cross entropy) which encourage the network to learn a “simpler” model. Lasso regularization adds a value to the cost function that is proportional to the L1 norm. The L1 norm is the sum of the absolute value of all weight coefficients. Ridge regression adds a value to the cost function that is proportional to the L2 norm. The L2 norm is the squared sum of all weight coefficients. While effective, these methods offer yet another hyperparameter to the model which must be tuned.

There are other methods that mitigate overfitting that can be explored but which are not mentioned in this initial exploratory paper.

Future Considerations

This paper aims to serve as a baseline in terms of this future of the project possibility. This model may be able to at least identify the “obvious” clutter in the sky. Looking forwards, this project has the potential to be picked up by the Optical Tracking group for further work on this application.
Another future for this application lies in utilizing the “You only look once” (YOLO) algorithm. The YOLO algorithm is a network architecture used for real time object detection. Object detection is a more complex application than classification. The models above predicted whether or not a UAS was in an image. Object detection recognizes what objects are in an image as well as where in the image the object is located.

The structure of YOLO is based on CNN. The algorithm applies a neural network to the entire image, then splits up the image by region. In each region, the model predicts bounding boxes and probabilities. The YOLO algorithm runs very quickly because it only requires one forward propagation pass through the network in order to make predictions. It also looks at the whole image at test time so its predictions are informed by global context in the image, which speeds up the training process. The YOLO algorithm has also performed well when compared to other network architectures.

Another performance metric to worth looking into is the AUROC curve, which is the Area Under the Receiver Operating Characteristics curve. The ROC curve depicts the tradeoff between the true positive rate (which is equal to recall) and the false positive rate for varying decision rules. AUROC is an informative metric to calculate “discrimination” for binary classification problems when distinguishing between two classes is important. AUROC is a measure of the model’s ability to discriminate between the positive and negative class.

APPENDIX A : PYTHON PACKAGES

```
# Name                    Version                   Build  Channel
keras                     2.3.1                         0
keras-applications        1.0.8                      py_0
keras-base                2.3.1                    py36_0
keras-preprocessing       1.1.0                      py_1
kiwisolver                1.1.0            py36ha925a31_0
matplotlib                3.1.3                    py36_0
matplotlib-base           3.1.3            py36h64f37c6_0
numpy                     1.18.1           py36h93ca92e_0
numpy-base                1.18.1           py36hc3f5095_1
pandas                    1.0.3            py36h47e9c7a_0
pillow                    7.0.0            py36hcc1f983_0
pip                       20.0.2                   py36_1
python                    3.6.8                h9f7ef89_7
python-dateutil           2.8.1                      py_0
python-libarchive-c       2.9                        py_0
python_abi                3.6                     1_cp36m    conda-forge
scikit-image              0.16.2           py36he350917_0    conda-forge
scikit-learn              0.21.1           py36h6288b17_0
scipy                     1.4.1            py36h9439919_0
seaborn                   0.9.0                    pypi_0    pypi
statsmodels               0.11.0           py36he774522_0
tensorboard               2.1.0                     py3_0
tensorflow                2.1.0           eigen_py36hdbbabfe_0
tensorflow-base           2.1.0           eigen_py36h49b2757_0
tensorflow-datasets       1.2.0                    py36_0    anaconda
tensorflow-estimator      2.1.0              pyhd54b08b_0
tensorflow-metadata       0.14.0             pyhe6710b0_1    anaconda
```

SYMBOLS, ABBREVIATIONS, AND ACRONYMS

Acronym	**Meaning**
AUROC	Area Under the Receiver Operating Characteristics curve
CCDC-AC	Combat Capability Development Command's Armament Center
CNN	Convolutional Neural Network
EOIR	Electro-Optical Infrared
UAS	Unmanned Aircraft System
YOLO	You Only Look Once

BIBLIOGRAPHY

Diederik P. Kingma, et al. "Adam: A Method for Stochastic Optimization." (2014).

Krizhevsky, Alex et al. "Imagenet classification with deep convolutional neural networks." *Advances in Neural Information Processing Systems*.

Narkhede, Sarang. “Understanding AUC - ROC Curve.” *Medium*, Medium, 26 June. 2018, https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

Redmon, Joseph, and Ali Farhadi. *YOLO: Real-Time Object Detection*, 2018, pjreddie.com/darknet/yolo/.

Science, ODSC - Open Data. “Overview of the YOLO Object Detection Algorithm.” *Medium*, Medium, 25 Sept. 2018, medium.com/@ODSC/overview-of-the-yolo-object-detection algorithm-7b52a745d3e0.

DISTRIBUTION LIST

U.S. Army CCDC AC
ATTN: FCDD-ACE-SAP Victoria Gerardi
FCDD-ACE-SAP Charles Douros
Marian Popescu
Picatinny Arsenal, NJ 07806-5000