Design Credits Project Report

Kethireddy Harshith Reddy (B20AI018)

Project name: Depth-Wise Algorithm and Attacks Implementation on Drone Imagery via CNN and Attention CNN models

Problem Statement

Unmanned Aerial Vehicles (UAVs) or drones have gained significant popularity and application in various fields, including surveillance, agriculture, and delivery services. One crucial aspect of drone operations involves the analysis and interpretation of aerial imagery captured by onboard cameras. However, the security and integrity of these images can be compromised due to potential attacks, leading to significant consequences such as privacy breaches, misinformation, or even physical damage.

The existing methods for analyzing drone imagery primarily rely on traditional image processing techniques, which may not be robust enough to handle complex scenarios and evolving attack strategies. Moreover, these methods often lack the ability to effectively detect and counter adversarial attacks, resulting in inaccurate or misleading interpretations of the imagery.

Therefore, the objective of this project is to develop and implement depth-wise algorithms and attacks on drone imagery using Convolutional Neural Networks (CNN) and Attention CNN models. The depth-wise algorithms aim to enhance the robustness and accuracy of drone image analysis by leveraging the spatial and contextual information present in the data. Additionally, the project will investigate various attack strategies, including adversarial perturbations and camouflage techniques, to assess the vulnerability of existing drone image analysis systems.

Motivation

The motivation behind this project is to address the existing limitations and vulnerabilities in drone image analysis systems, enhance their accuracy and robustness, and contribute to the development of secure frameworks that can withstand potential adversarial attacks. By doing so, we aim to unlock the full potential of drones in various industries while ensuring the integrity and reliability of the captured imagery.

Proposed Methodology

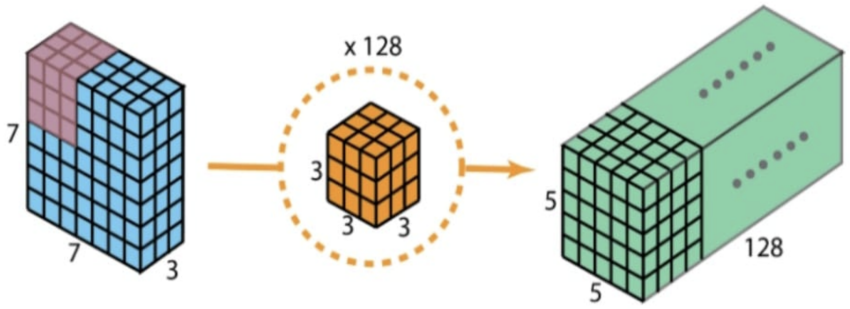
Techniques used for Reconfigurable DNN

The main focus in this part is to reduce the size of the model and this can be achieved by reducing the number of parameters this would cause accuracy to reduce but size of model will also be reduced.

I have implemented 4 approaches which are based on two main concepts.

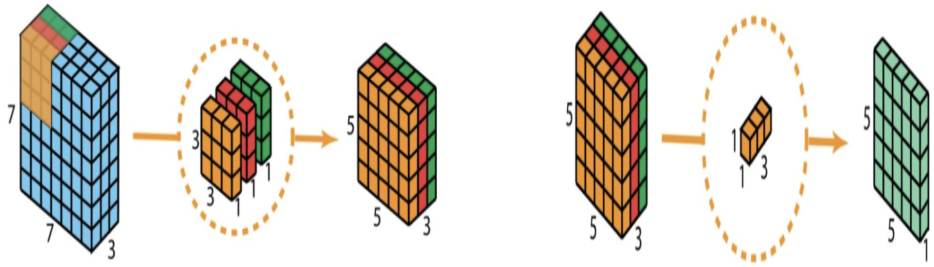
1) Depth-wise separable Convolution
2) Linear Bottlenecks

Computation in CNN:



Computation:
(3*3*3) * (5*5) * 128 == 86,400

Computation in Depth-wise CNN:



Computations:
(3*3*1) * 3 * (5*5) == 675

Computations:
(1*1*3) * (5*5) * 128 == 9600

Total = 9600 + 675 == 10,275

General Approach 1:

Let us assume we want to do convolution of
Input - Hi x Wi x n
kernel - $k * k * n$ with m filters

Output - $H * W * m$

With normal convolution computation are

$$H * W * n * k^2 * m$$

If we break this convolution into two parts

1) **Depth wise** - Here we will first do convolution on
   Input - Hi x Wi x n
   kernel - $k * k * 1$ with n filters
   Output - $H * W * n$
   Computation = $n * k^2 * H * W$
2) **Point Wise** - Here we do convolution as follows
   Input - H x W x n
   kernel - 1 x 1 x n with m filters
   Output - H x W x m
   Computation = $n * m * H * W$

   Hence total computation are

   = $n * k^2 * H * W + n * m * H * W$

   = $n * H * W * (k^2 + m)$

We can see we have reduced the computation for this convolution.

General Approach 2 :

We know feature maps in low-dimensional subspaces can be encoded and for non linearity which do boost representational complexity, non-linear activations cause information loss.

Keeping this in mind we will take input of size $h * w$ with k channels and will make it pass through three steps which are described below.

- ➢ Given an input of size $h * w * k$
- ➢ We first expand it by factor t by applying 1x1 convolution with non linear activation output is $h * w * (tk)$
- ➢ After this we apply 3x3 depthwise convolution with stride of s with non linear
- ➢ Activation output is $\frac{h}{s} * \frac{w}{s} * (tk)$
- ➢ Finally we apply 1x1 convolution with linear optimization and the output is
- ➢ $\frac{h}{s} * \frac{w}{s} * k'$

We also add a residual connection so that when backpropagating it will help in gradient flow and result in better performance.

All operations summarized into a table.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Here the total computation will be

$$H * W * n * t * (k^2 + m + n)$$

Where m and n are output and input channels, which are less compared to what original would have been as we are applying depthwise convolution.

Techniques used for Reconfigurable DNN

The main focus is to reduce the size of the model and this can be achieved by reducing the number of parameters this would cause accuracy to reduce but size of model will also be reduced.

Approach 1

Keeping the baseline model we look upon the convolution layer with maximum number of parameters and switch it with depthwise convolution layer and here we are replacing only 1 convolution layer from the block of the model.

Approach 2

We will now change multiple convolution layers from each block with the maximum number of parameters in that block of model.

Approach 3

We will now integrate reverse linear bottlenecks along with Depth wise Convolution.

Approach 4

This will be the hybrid approach of all convolutional layers as depthwise convolutional layers with bottleneck blocks.

Attacks

3 attacks have been implemented. Here's an explanation of each attack:

FGSM (Fast Gradient Sign Method) Attack:

The FGSM attack is a simple and fast adversarial attack method. It perturbs the input data by adding a small perturbation in the direction of the gradient of the loss function with respect to the input. The attack is performed using a single step of perturbation, where the magnitude of the perturbation is determined by a hyperparameter called epsilon. The FGSM attack aims to maximize the loss to induce misclassification or misbehavior of the target model. However, FGSM might not be effective against models with strong defenses.

PGD (Projected Gradient Descent) Attack:

The PGD attack is an iterative version of the FGSM attack. It performs multiple iterations of perturbation, taking small steps in the direction that maximizes the loss. After each perturbation step, the perturbed data is projected back onto an epsilon-constraint region to ensure it stays within a permissible range. The PGD attack aims to find the maximum perturbation that can fool the model while satisfying the

constraints. It is a stronger attack than FGSM and can overcome some defense mechanisms.

BIM (Basic Iterative Method) Attack:
The BIM attack is similar to the PGD attack but differs in the way perturbations are applied. It also performs multiple iterations of perturbation, but instead of taking a single step, it applies small perturbations along the gradient direction for each iteration. The step size for perturbation is determined by a hyperparameter called alpha. The BIM attack gradually accumulates the effect of perturbations over multiple iterations, allowing it to find more potent adversarial examples. It is also more computationally expensive than FGSM but can be more effective against defenses.

Datasets and their characteristics

I have used SHVN, CiFAR100, MNIST and FashionMNIST datasets as well alongwith VisDrone19, the reason which is given in the results section of the implementation of the attacks.

CiFAR10 - CIFAR-10 dataset consists of 60,000 color images in 10 classes. Each image in CIFAR-10 is a low-resolution (32x32 pixels) RGB image representing one of the following classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is commonly used for image classification tasks and benchmarking deep learning models.

SVHN - The SVHN dataset is a real-world image dataset of house numbers captured from Google Street View. It contains over 600,000 digit images for training and 26,032 digit images for testing.

MNIST - MNIST is a widely known dataset that has been a standard benchmark for image classification tasks. It contains a collection of 60,000 handwritten digit images for training and 10,000 images for testing. The images in MNIST are grayscale and have a size of 28x28 pixels.

FashionMNIST - FashionMNIST is a dataset designed as a drop-in replacement for the traditional MNIST dataset. It consists of 70,000 grayscale images of 10 different fashion categories, including T-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots. Each image in FashionMNIST has a size of 28x28 pixels.

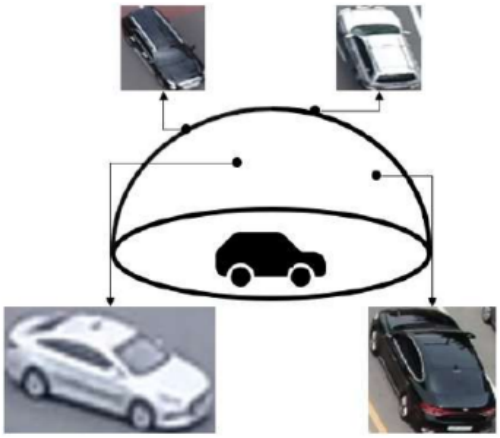
VisDrone19 - The VisDrone2019 dataset, which was put together by the AISKYEYE team at the Lab of Machine Learning and Data Mining, was used. This dataset includes:

- 6471 training samples
- 548 validation samples
- 1610 test samples

Some sample images from the dataset:

There are many things about the drone dataset. In the datasets we already have, objects are taken by people or by CCTV. A circle around a car is a path that can be taken based on the car. So, you can mostly see the front, back, and sides of the object, and you can also see a little bit of the top. Because the picture was taken up close, it is big and clear.

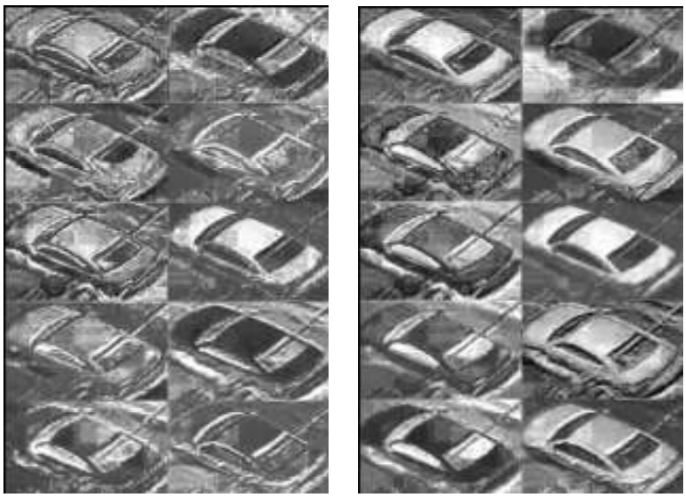
On the other hand, drones can take pictures in a variety of car-shaped, semi-spherical shapes. It would have more information than traditional datasets, especially from a bird's-eye view and a plane. So, the object looks the same from the front, back, side, and top. But it looks different when taken at a 90-degree angle. Because they can only see the top, people look like dots, and street lamps look like straight lines. In this case, it's hard to put things into groups, which makes the classification task difficult.



In the picture above, we can see that the shape of the cars changes depending on how the picture was taken. Because there are images from different angles and at different resolutions, it is hard to put them into groups.

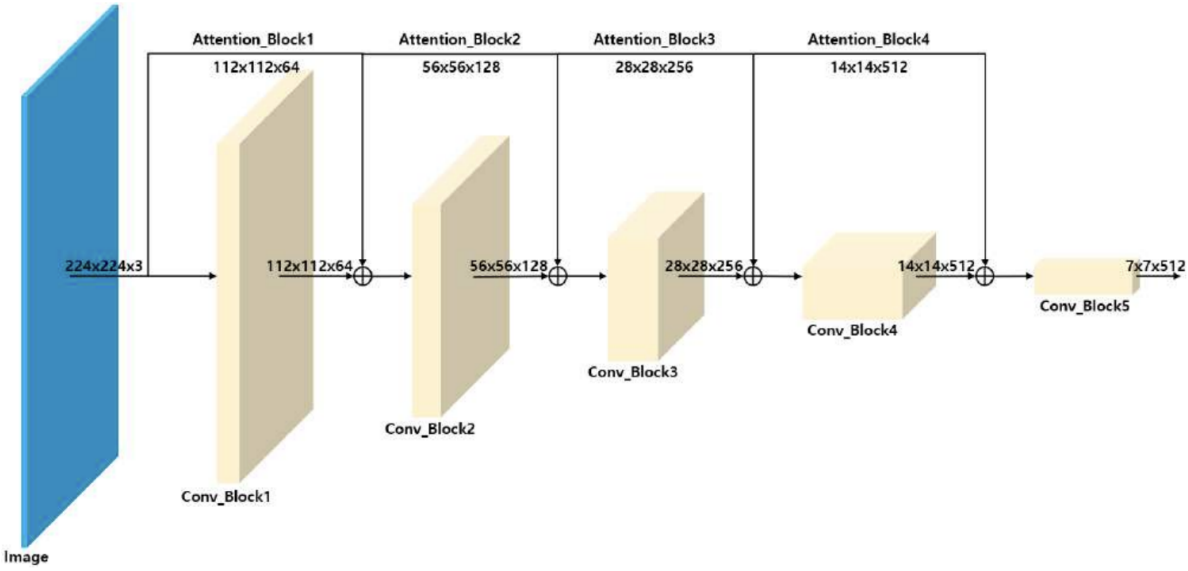
Attention Block

Attention block is a proposed technique that complements lost features and extracta more diverse features through convolution layers. A feature map called attention block is added to the output of the convolution layer before entering the next convolution layer's input.



In the image above, the class on the image is a car, and the initial width and height are about 100 pixels each. The picture on the left shows what happened when it was shrunk to 224x224 and put into the convolution layer. When you add the attention block to the picture on the left, you get the image on the right. In the picture on the right, you can clearly see what happens when the attention block is used. So, as the network gets

deeper, continuing to feed it features from the original picture makes the learning process better.



In this project, convolutional and attentional blocks are used to make networks. The internal structure of the convolution block is simple because it is made up of only four convolution layers and one max-pooling layer. In the convolution layer part, the bottleneck method is used on two layers with kernel sizes of 33 and two layers with kernel sizes of 11. This method is used because the bottleneck technique keeps the same speed while reducing the number of parameters by about 28%. The output of the convolution block will be a quarter the size of the picture that went into it, and the number of channels will double. The depth-wise algorithm is applied to the convolution network.

The attention block has a simple structure on the inside as well. At first, it had just 11 convolution layers and one max-pooling layer. Each layer of the attention block goes up by one layer as the convolution block goes deeper. It only helps to make a new feature map by making the number of channels bigger and the size of the picture smaller. The reason for setting up as stated is to keep as many of the properties of the input picture as possible and to reduce the amount of work that needs to be done. After these outputs are added, they are sent to the next convolution layer and continue to be learned through batch normalization and activation functions.

Layer	Output
Input Image	224 x 224 x 3

Conv_Block1	112 x 112 x 64
Attention_Block1	112 x 112 x 64
Conv_Block1 + Attention_Block1	112 x 112 x 64
Conv_Block2	56 x 56 x 128
Attention_Block2	56 x 56 x 128
Conv_Block2 + Attention_Block2	56 x 56 x 128
Conv_Block3	28 x 28 x 256
Attention_Block3	28 x 28 x 256
Conv_Block3 + Attention_Block3	28 x 28 x 256
Conv_Block4	14 x 14 x 512
Attention_Block4	14 x 14 x 512
Conv_Block4 + Attention_Block4	14 x 14 x 512
Conv_Block5	7 x 7 x 512

The structure of the image showing the architecture of the model is described in the table above.

Comparison Of Parameters And Model Size For Different Approaches

Models:

CNN - Baseline CNN Model
Attention CNN - CNN with Attention mechanism
CNN_v1 - Approach 1 implemented on Baseline CNN model
CNN_v2 - Approach 2 implemented on Baseline CNN model
CNN_v3 - Approach 3 implemented on Baseline CNN model
CNN_v4 - Approach 4 implemented on Baseline CNN model

Model	Total Parameters	Size (MB)
CNN	11,856,834	93.90
CNN_v1	9,768,898	86.00

CNN_v2	8,403,296	81.76
CNN_v3	9,936,578	95.64
CNN_v4	8,570,976	91.40
Attention CNN	6,404,704	26.64

Dataset - VisDrone19

The discrepancy in sizes can be attributed to several factors, including architecture design choices, parameter sizes, and compression techniques. The following are the reasons for the discrepancy in size:

Architecture design: The basic CNN architecture typically consists of convolutional layers, pooling layers, and fully connected layers. On the other hand, an attention CNN incorporates attention mechanisms, which focus on relevant regions of the input, allowing the model to attend to specific features.

Parameter sizes: The CNN has more layers and filters will generally have more parameters, resulting in a larger file size. Conversely, an attention CNN has a smaller number of parameters due to the attention mechanisms focusing on specific regions of the input.

Table of test accuracies with and without attacks on the various datasets:

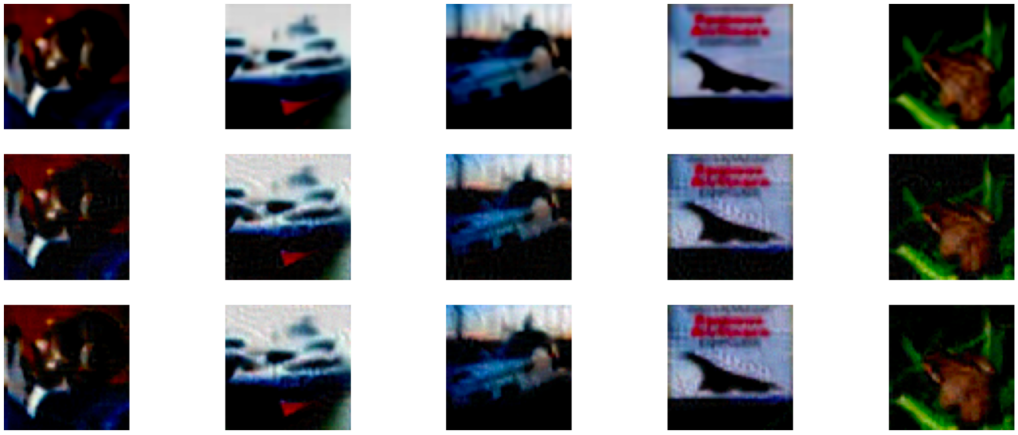
Dataset	Type of Model	Without Attack	FGSM Attack	BIM Attack	PGD Attack
CiFAR10	CNN	82.68%	14.88%	0.00%	0.06%
MNIST	CNN	99.04%	82.3%	6.76%	23.93%
Fashion MNIST	CNN	92.39%	15.03%	0.59%	0.62%
SVHN	CNN	94.06%	32.18%	0.21%	7.55%
VisDrone19	CNN	99.44%	99.44%	99.44%	99.44%
VisDrone19	Attention CNN	99.44%	99.44%	99.44%	99.44%

We can clearly see that the Attacks have no effect on the VisDrone19 dataset. The VisDrone19 dataset may have been specifically designed to evaluate the robustness of models against adversarial attacks. It's possible that the models trained on this dataset have incorporated robust defense mechanisms that make them more resilient to attacks like FGSM.

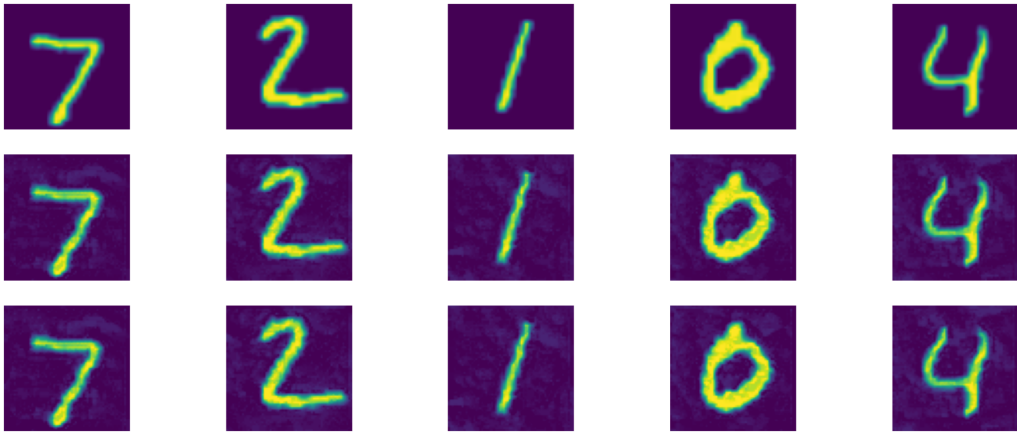
I have used SHVN, CiFAR100, MNIST and FashionMNIST datasets as well alongwith VisDrone19 to confirm the same, i.e, the VisDrone19 dataset is robust against the traditional attacks

Image plots of each dataset's images before and after the attacks: (1st row - No Attack, 2nd row - BIM Attack, 3rd row - PGD Attack):

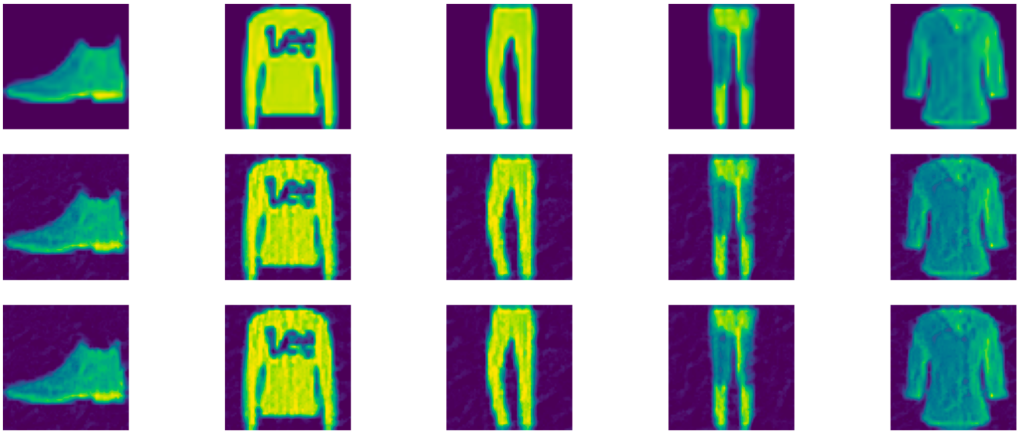
1. CiFAR10 Dataset:



2. MNIST Dataset:

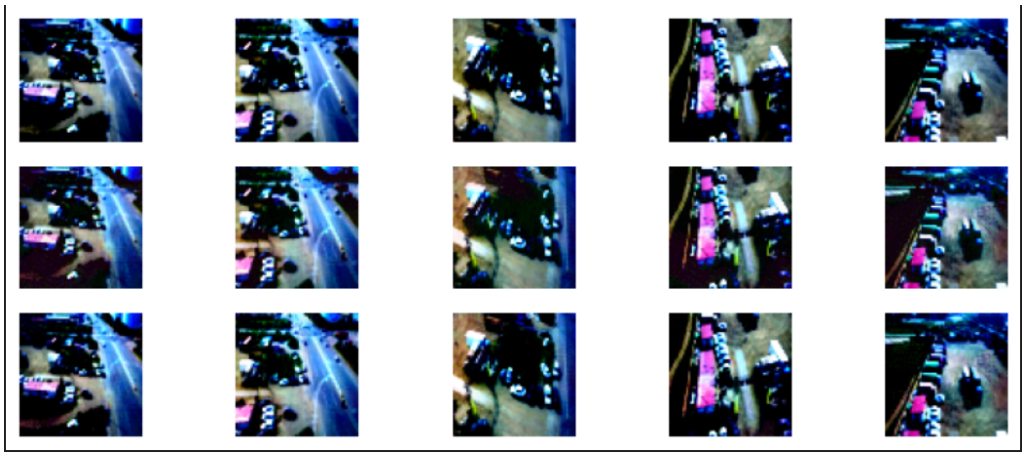


3. FashionMNIST Dataset:



4. SVHN Dataset:

5. VisDrone19 Dataset:



Conclusion

- The proposed depth-wise algorithms tailored for drone imagery analysis have demonstrated improved model size and robustness compared to traditional image processing techniques.
- The evaluation of the models using appropriate metrics has quantitatively and qualitatively validated their performance.
- The implementation of various attack strategies has revealed vulnerabilities in drone image analysis systems and highlighted the need for enhanced security measures.
- Our proposed attention mechanism has the potential to be applied to other computer vision tasks for drone imagery and to facilitate the development of more advanced and intelligent drone systems.
- VisDrone19 dataset is robust against traditional attacks like FSGM, BIM and PGD.

Code

- If you do not have the VisDrone19 dataset, then the code is present on the server at the path ***/home/user/Desktop/Vitis-AI/HLS4ML/DC_Project***.
- You can execute the code in any text editor (like VS Code) by installing the extension for .ipynb as defined in the README file. The dataset is present in the server in the same folder as the code file.
- Otherwise, you can just follow the README file to execute the code locally on your PC.
- If you already have the VisDrone19 dataset, you can access the code for the implementation of Depth-wise algorithm and the attacks on the VisDrone19 dataset by clicking the following link: Code
- You can also access the code for the implementation of the attacks on the SVHN, MNIST, FashionMNIST and CiFAR10 datasets by clicking on the following link: Code

References

https://ieeexplore.ieee.org/document/9589099

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9589099

https://www.geeksforgeeks.org/depth-wise-separable-convolutional-neural-networks/

https://www.javatpoint.com/depth-wise-separable-convolutional-neural-networks