INTERNSHIP PROJECT REPORT

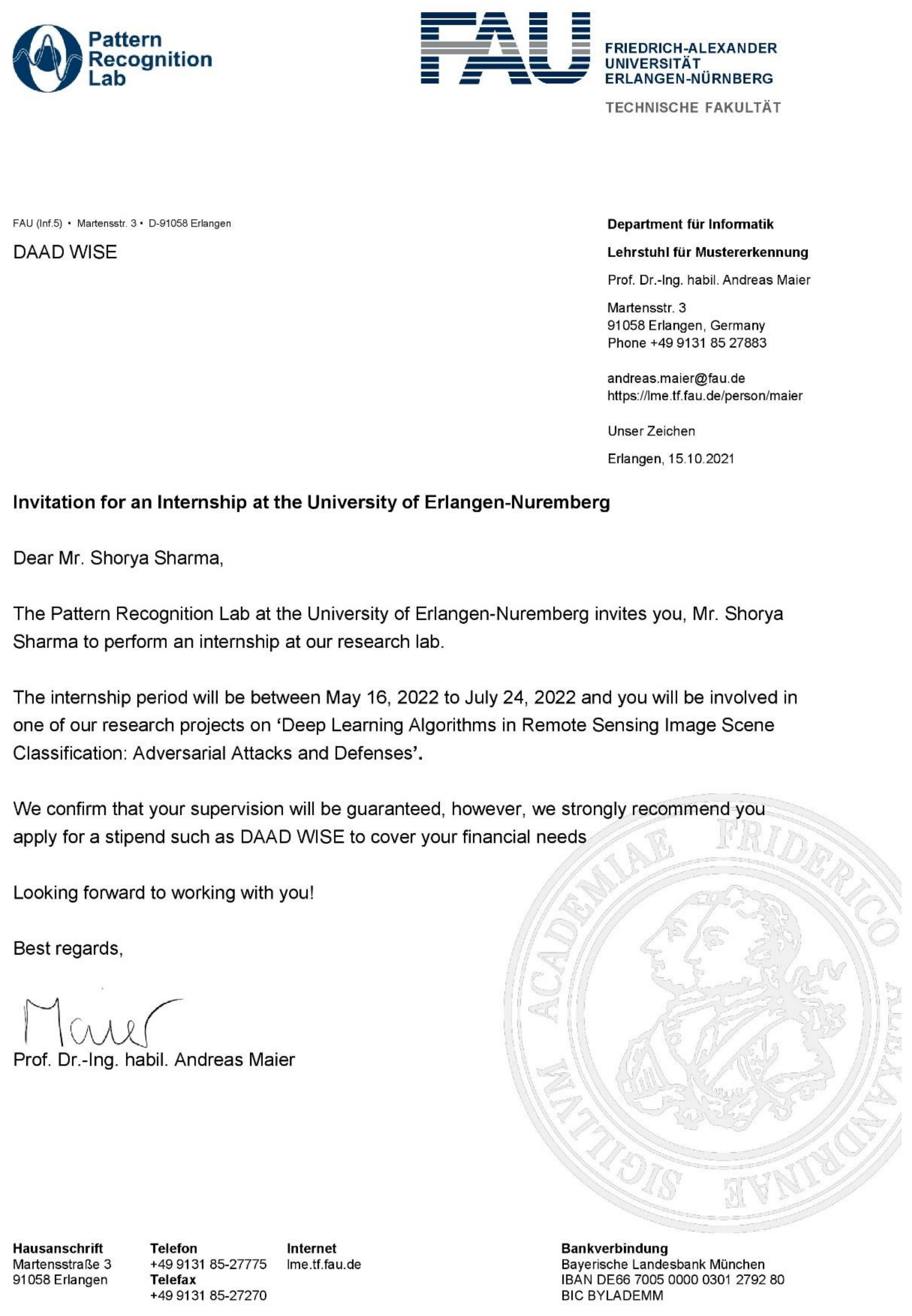
Deep Learning Algorithms in Remote Sensing Image Scene Classification: Adversarial Attacks and Defenses

SHORYA SHARMA
School of Electrical Sciences, Indian Institute of Technology Bhubaneswar
ss118@iitbbs.ac.in

Supervisor: **PROF. Dr. ANDREAS MAIER**
Department of Computer Science, University of Erlangen-Nuremberg

July 2022

LETTER OF INVITATION



Pattern Recognition Lab

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

FAU (Inf.5) • Martensstr. 3 • D-91058 Erlangen

DAAD WISE

Department für Informatik

Lehrstuhl für Mustererkennung

Prof. Dr.-Ing. habil. Andreas Maier

Martensstr. 3
91058 Erlangen, Germany
Phone +49 9131 85 27883

andreas.maier@fau.de
https://lme.tf.fau.de/person/maier

Unser Zeichen

Erlangen, 15.10.2021

Invitation for an Internship at the University of Erlangen-Nuremberg

Dear Mr. Shorya Sharma,

The Pattern Recognition Lab at the University of Erlangen-Nuremberg invites you, Mr. Shorya Sharma to perform an internship at our research lab.

The internship period will be between May 16, 2022 to July 24, 2022 and you will be involved in one of our research projects on **‘**Deep Learning Algorithms in Remote Sensing Image Scene Classification: Adversarial Attacks and Defenses**’.**

We confirm that your supervision will be guaranteed, however, we strongly recommend you apply for a stipend such as DAAD WISE to cover your financial needs

Looking forward to working with you!

Best regards,

Maier

Prof. Dr.-Ing. habil. Andreas Maier

INTERNSHIP COMPLETION CERTIFICATE

Department für Informatik

Lehrstuhl für Mustererkennung

Prof. Dr.-Ing. habil. Andreas Maier

Martensstr. 3
91058 Erlangen, Germany
Phone +49 9131 85 27883

andreas.maier@fau.de
https://lme.tf.fau.de/person/maier

Unser Zeichen

Erlangen, 26.07.2022

To Whom It May Concern

I am writing this letter to certify that **Mr. Shorya Sharma** (Roll No.-19EE01017), a third year B. Tech student of Electrical Engineering, Indian Institute of Technology Bhubaneswar, did his research internship at our **Pattern Recognition Lab** at **University of Erlangen-Nuremberg** from **May 16, 2022** to **July 24, 2022** under my supervision.

During his internship, Mr. Sharma worked for our Computer Vision research group on the ongoing project '**Deep Learning Algorithms in Remote Sensing Image Scene Classification: Adversarial Attacks and Defenses**'. This project explores the properties of adversarial examples of RSI scene classification and develop novel defense approaches to make targeted classifier models more robust to adversarial attacks.

Mr. Sharma has successfully supported this research and in doing so, he learned and applied the scientific procedures and working techniques within this field of research. I observed that he is quite well versed in adapting to multiple programming languages and machine learning libraries along with being dedicated to his work with an interest in providing novel approaches to solve the limitations faced in this new field of research in adversarial learning. His work included the following tasks:

- Literature review on current state-of-the-art adversarial attacks and defense mechanisms in the area of RSI scene classification
- Design and Implementation of a range of white-box adversarial attacks on aerial imagery datasets with the help of Machine Learning libraries such as PyTorch, FoolBox, OpenCV, Scikit-Learn in Python language
- Implementation of algorithms to defend against these attacks using techniques like adversarial training, stochastic activation pruning, Generative Adversarial Networks
- Analysis, evaluation and presentation of his work and findings

The tasks entrusted to Mr. Sharma have always been carried out by him very reliably and to our full satisfaction. During the internship period, he was disciplined and consistent with his work and constantly kept me updated about his ongoing works. His behavior towards supervisors and employees was impeccable at all times. Overall, I appreciate Shorya's work in my lab and wish him all the best for his future career.

Best Regards,

Maier

Prof. Dr.-Ing. habil. Andreas Maier

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
Lehrstuhl für Mustererkennung
Department Informatik
Prof. Dr.-Ing. A. Maier
Martensstr. 3, D-91058 Erlangen

ACKNOWLEDGEMENT

I express my sincere gratitude towards my supervisor, Dr. Andreas Maier for the constant support, motivation and guidance provided during the project work. Without his guidance and support I wouldn’t have progressed such progressively during my project work. I would also like to thank my team leader, Mr. Adarsh Bhandary Panambur for all his support, necessary tips and guidelines during the activation period, and the entire team for being helpful and supportive in every little help I needed and for creating the opportunity for me to bring out my best performance. I am extremely great full to my department staff members and friends who helped me in successful completion of this internship.

Shorya Sharma
19EE01017

DECLARATION

I, Shorya Sharma (Roll No.- 19EE01017), a fourth year B. Tech student in Electrical Engineering, Indian Institute of Technology Bhubaneswar, hereby declare that the entire work presented in this internship report submitted to the Indian Institute of Technology Bhubaneswar during the academic year 2021-22, is a record of an original work done by me under the guidance of Dr. Andreas Maier, Pattern Recognition Lab, University of Erlangen-Nuremberg. This internship work is submitted in partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Electrical Engineering. It is my original work performed in the lab and it has not been submitted elsewhere for the award for any other degree.

Shorya Sharma
19EE01017

TABLE OF CONTENTS

Letter of Invitation **2**

Internship Completion Certificate **3**

Acknowledgement and Declaration **4**

Abstract **6**

1. Adversarial Attacks in RSI **7**

1.1 Introduction 7

1.2 Related Works 8

1.3 Datasets 9

1.4 Method 9

1.5 Experiments 12

1.6 Experimental Observations 17

1.7 Experimental Discussion 17

1.8 Conclusion and Future Works 17

2. Adversarial Defenses in RSI **18**

2.1 Introduction 18

2.2 Related Works 18

2.3 Our Contribution 20

2.4 Experiments 24

2.5 Conclusion 26

2.6 Future Works 26

3. References **27**

Abstract

Remote sensing image (RSI) scene classification is the foundation and important technology of ground object detection, land use management and geographic analysis. Deep learning models have gained immense popularity for machine learning tasks such as image classification and natural language processing due to their high expressibility. During recent years, convolutional neural networks (CNNs) have achieved significant success and are widely applied in RSI scene classification. However, crafted images that serve as adversarial examples can potentially fool CNNs with high confidence and are hard for human eyes to interpret. For the increasing security and robust requirements of RSI scene classification, the adversarial example problem poses a serious problem for the classification results derived from systems using CNN models, which has not been fully recognized by previous research. Hence, some adversarial defense techniques are developed to improve the security and robustness of the models and avoid them being attacked. Gradually, a game-like competition between attackers and defenders formed, in which both players would attempt to play their best strategies against each other while maximizing their own payoffs. To solve the game, each player would choose an optimal strategy against the opponent based on the prediction of the opponent's strategy choice.

In this study, to explore the properties of adversarial examples of RSI scene classification, we create different scenarios by testing 29 major attack algorithms trained on different RSI benchmark datasets (NWPU-RESIC45 and UC Merced Land Use Dataset) to fool CNNs (i.e., AlexNet, ResNet50, ResNet101, MobileNetV_2 and DenseNet121). Attack selectivity reveals potential classes of adversarial examples and provides insights into the design of defensive algorithms in future research.

On the defense side, we followed a game theoretic approach to implement a novel defense strategy, that combines multiple Stochastic Activation Pruning with adversarial training. Our defense accuracy outperforms that of PGD adversarial training, which is known to be the one of the best defenses against several $L\infty$ attacks, by about 6-7%. We are hopeful that our defense strategy can withstand strong attacks leading to more robust deep neural network models. We also trained a generative model in the context of a generative adversarial network, and use this to generate perturbations that can be added to the input image to counter the perturbation added by the attacker. We also explore defense by generating images that are as close as possible to the input adversarial image. We implement our own attacks and train our own baselines to ensure uniform comparison.

Adversarial Attacks in RSI

1. Introduction

With the advancement of remote sensing technology,the automatic interpretation of remote sensing images (RSIs)has greatly improved [1]–[4]. RSIs with higher resolution have led to the wide use of RSI scene classification systems in crop classification [5]–[7], forest resource surveys [8]–[10], land cover classification [11], [12], building detection [13], [14] and other fields [15], [16]. Automatic interpretations of RSIs with robust and high accuracy can deliver high economic efficiency [17]–[19].

A good image classification algorithm can effectivelyextract target features and quickly locate new targets, which is the key to RSI interpretation. Therefore, the variabilityin features can be massive among different classification tasks. most traditional RSI interpretation algorithms design the hand-craft features for different applications, such as road detection [20], [21], vegetation measurement [22], drought monitoring [23], etc. [24]–[26]. However, these well-designed features cannot solve the RSI interpretation problem well since traditional algorithms need to be carefully designed with solid domain knowledge in order to be effective in different applications [27]–[29]. And most traditional classification algorithms have a low classification accuracy and calculation inefficiency, which makes it difficult to meet thesystem requirements.

In recent years, deep learning has achieved remarkable progress in the field of computer vision [30]. Many excellent algorithms, especially convolutional neural networks (CNNs), have been introduced into the RSI scene classification systems [5], [11], [15]. Chen *et al.* [31] used a single-layer autoencoder (AE) and multi-layer stacked autoencoder (SAE) to learn the features of RSIs and proposed a classification method using the extracted spatial features. This method performs better than do support vector machines (SVM)and K-nearest neighbor (KNN) classification models. Chen *et al.* [32] proposed a deep belief network (DBN)-based image classification framework that combined the spectral and spatial features of RSIs and used a CNN model toextract robust features from images, thereby greatly reducing the number of hyperparameters. To alleviate the overfitting problem of CNNs, Cheng *et al.* [33] added the metric learning regularization term to the CNN model by optimizing the discriminant objective function. This approach efficiently reduced classification error. Zhang *et al.* [34] proposed a gradient boosting random convolution network that reused the weight of each CNN model and reduced the parameters, thusmaking feature extraction more efficient. Chaib *et al.* [35] used the CNN model in feature extraction and discriminant correlation analysis (DCA) for data fusion. These RSI sceneclassification models perform much better in terms of accuracy than traditional RSIs and in the efficiency of RSI sceneclassification achieved using high-performance computing. Meanwhile, previous research has also demonstrated the excellent feature extraction abilities of CNNs.

However, CNNs have many limitations and still haveroom for improvements, especially concerning security problems [36]–[39]. Szegedy *et al.* [40] find that a small adversarial perturbation can be achieved on a trained model with agradient algorithm. This happens when we deliberately add noises to the input images to fool the CNN classifier and prompt it to make wrong predictions with high confidence. The modified images are called adversarial examples.

The problem of adversarial examples exists in various application areas of deep learning [41]. The classification problem of RSIs presents a great security risk, especially forapplications in the military and automatic driving fields [42].As shown in Figure 1, in military applications, an attacker can generate an adversarial perturbation for a target such as an airplane to camouflage the object using a physical object.The adversarial example of the target can easily escape the detection by the military target detection systems. In this case, the originally robust target would be wrongly classified as the wrong class with high confidence. State-of-the-art RSI sceneclassification systems depend heavily on CNNs to extract features from the target. Unlike RGB images, RSIs have unique properties, such as spectra, bands, etc. In addition, RSIs are usually obtained from the nadir perspective, without the foreground and back views of natural images. Therefore,it is necessary to study the characteristics of adversarial examples in RSI scene classification systems.

In the experiment, we train several CNN models withthe widest application in RSI scene classification systems.In these high-accuracy CNNs, we use a variety of attack algorithms to generate different adversarial examples. The result shows the vulnerability of several mainstream CNNs and demonstrates the importance of adversarial examples in RSI scene classification. Furthermore, we find that the fundamental issues related to adversarial examples of RSIs are the model vulnerability and attack selectivity. This means that different RSI scene classification CNNs have differ- ent security characteristics, so the cost to obtain adversarial examples varies. We also find that attack selectivity is related to the model structure and the training data, producing a clustered pattern of errors in generating adversarial examples. The misclassified adversarial examples are highly similar to the correct original scenes. These properties also reflect the characteristics of adversarial examples in RSI sceneclassification.

The main contributions of this article are listed as follows:

- We implemented white-box adversarial attacks using 5 different neural network architectures on different aerial scene classification datasets under two different configurations: eps=0.05 and eps=1.0.
- We tabulated our results and inferred the best-attack on each dataset using a particular network architecture. Confusion Matrix were also plotted for evaluation purposes.
- We implemented the Transferable Sparse Adversarial Attack (TSAA) and deployed it on our dataset to infer the transferability of this black-box attack on our datasets using different neural network architectures.
- We implemented state-of-the-art black box attacks on our datasets using Adversarial Attack libraries.

2. Related work

In recent years, CNNs have been applied in many tasks in the field of remote sensing, such as landmark classification [43], land use classification [44] and change detection [45]. CNNs have produced outstanding results in these areas. However, most of the research focuses on how to improve the model accuracy and design good CNN structures [46]–[48], and insufficient attention has been placed onthe adversarial example problem of RSI scene classificationsystems. Since the adversarial example was proposed by Szegedy *et al.* [40], the security of deep learning has been the subject of widespread discussion.
The adversarial example refers to an input image formed by adding a special adversarial perturbation, which results in the model producing a false output with high confidence. Generally, a small adversarial perturbation is not perceived bythe human eye. Goodfellow *et al.* [37] stated that adversarialexamples in CNNs are produced due to the linear operationsof the model in a high-dimensional space. For feature vectoroperations in a high-dimensional space, small perturbations can cause substantial errors by reducing the robustness of themodel. This problem not only is encountered in computer vision tasks but also exists in many fields such as natural language processing and sentiment analysis [49]. It is well known that RSIs consist of high-dimensional data, so the problem of adversarial examples in RSI scene classificationsystems is also unavoidable.

The attack algorithms of adversarial examples are also diverse. According to the number of iterations, attack algorithms can be divided into the one-step attack and iterative methods. These attack algorithms are gradient-based optimization methods that try to maximize the loss of the objective function. The fast gradient notation (FGSM) method is a one-step attack proposed by Goodfellow *et al.* [37], which first computes the gradient direction of the model loss function and subsequently increases the value of the loss function by adding a small adversarial perturbation in that direction. Thus, the prediction result of the model deviates from the real class. This attack algorithm is simple, but the fooling rate of deceived models is low. Therefore, an iterative method named basic iterative method (BIM) is proposed [38]. The algorithm is an extension of the FGSM, and with multiple iterations, it attempts to add noise in each step to increase the value of the loss. In each iteration, the changed image pixels are controlled within a certain neighborhood of the original picture. Different from the FGSM, the BIM algorithm can effectively improve the fooling rate of attacks. FGSM and BIM are the two most widely applied attack algorithms.

Attack algorithms can also be divided into other types. Attack algorithms can be divided into white-box attacks andblack-box attacks [36] according to the model information acquired by the attacker. In a white-box attack, the attacker knows the type of CNN, the number of layers, and the optimization method used for training and can obtain the trainingdata and even discern the hyperparameters of the model. Attackers can launch additional targeted attacks based on thedetails of the model. In contrast to white-box attacks, attackers in black-box attacks are unaware of the model details. They can use only the model application background and outputs to analyze the vulnerability of the model. According to the characteristics of the attack algorithm, Prem-latha *et al.* [50] and Chakraborty *et al.* [51] classified attack scenarios into three categories: evasive attacks, poisoned attacks and exploratory attacks. Attackers in an evasive attack attempt to avoid detection by the system by constructing malicious input. In this scenario, the attacker cannot influence the training data of the model. The attacker in a poisoned attack scenario attempts to inject constructed malicious data to poi-son the model during the training process. In the exploratoryattack scenario, the attacker cannot affect the training datasetor the details of the model and acquires knowledge of the learning algorithms and training data by testing the responseof the model to the input data, thereby constructing effectiveadversarial examples. According to whether the attack class is clear, attack algorithms are also divided into targeted attacksand untargeted attacks [36]. Targeted attacks attempt to makethe model classify the adversarial example as a specific class,whereas untargeted attacks do not care about the prediction labels of adversarial examples. Untargeted attacks require only the model to misclassify data that were originally correctly classified.

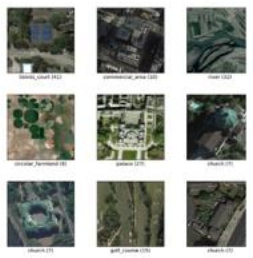
Above all, adversarial example attacks might occur during training and testing of a model. The type of attack is related to the model details and training data. The type of attack algorithm depends on the setting of the attack scenario. For example, FGSM and BIM can be both white-box and black-box attack algorithms. It depends on whether the information (e.g., model structure, hyperparameters) of the attacked model is known before the attack. Similarly, FGSMand BIM can also be targeted and untargeted attacks, depending on whether the image is misclassified into a specific class. Under different attack types, the object function of the attack algorithm changes. For example, the FGSM targeted attack algorithm only needs to change the loss function into the optimization of a specific class. In our research, we use all white-box, untargeted attack algorithms.

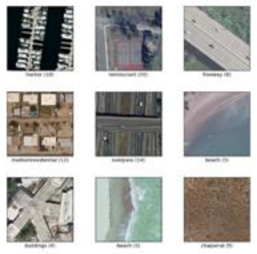
Currently, RSI scene classification involves many CNN-based applications that might also have adversarial example problems. These problems could have highly serious consequences for the application of RSI scene classification. Moreover, since RSIs have spatial and spectral properties, they are different from the RGB images. Therefore, the adversarial example problem is of great importance in RSI scene classification.

3. Data

Two Datasets were used:

- RESISC45 dataset is a publicly available benchmark for Remote Sensing Image Scene Classification (RESISC), created by Northwestern Polytechnical University (NWPU). This dataset contains 31,500 images, covering 45 scene classes with 700 images in each class.
- UC Merced is a 21-class land use remote sensing image dataset, with 100 images per class. The dataset contains 2100 images which were manually extracted from large images from the USGS National Map Urban Area Imagery collection for various urban areas around the country. The pixel resolution of this public domain imagery is 0.3 m


NWPU-RESIC 45


UC Merced Dataset

4. Method

This section gives a brief introduction of CNNs and the most widely used robust CNN model structure in RSI scene classification. Subsequently, it explains how to use attack algorithms to generate an adversarial example of an RSI.

A. CONVOLUTIONAL NEURAL NETWORKS

Current RSI scene classification systems have better classification capabilities, and they are mostly based on CNNs. CNNs usually consist of convolutional layers, pooling layers, and fully connected layers [52]. Each layer has a different function. Through these structures, RSIs are continuously reduced in dimension, and semantic features are gradually abstracted. Eventually, the model implements the classification of the RSIs.

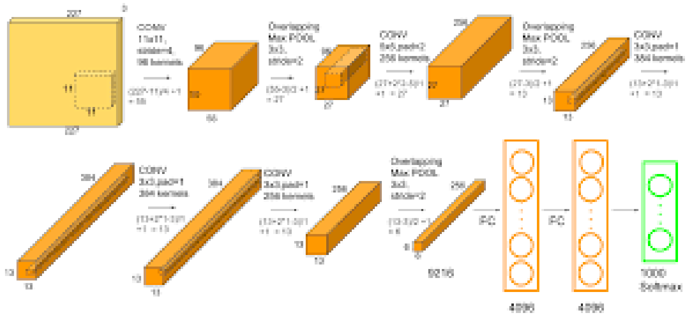
Pooling operations can reduce the dimensions of the RSI to reduce the computational complexity and improve the generalization ability of the model. Finally, A^l denotes the feature maps produced after the convolution and pooling operations. When the feature maps are fed into the fully connected layer, all of the feature maps need to be flattened and trans-formed into vectors. Finally, A classifier can predict a class y based on the final outputs. When the prediction is notcorrect, we use the cross-entropy loss function to compute thedistance between the predicted results and the real labels. Themodel can be updated by the backpropagation algorithm [52],which aids the model in reducing the loss value. The modelgradually converges in the process of iteration

1) AlexNet

In 2012, *Alex Krizhevesky* and others proposed a deeper and wider CNN model compared to LeNet and won the most difficult ImageNet challenge for visual object recognition called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 [7]. AlexNet achieved state-of-the-art recognition accuracy against all the traditional machine learning and computer vision approaches. It was a significant breakthrough in the field of machine learning and computer vision for visual recognition and classification tasks and is the point in history where interest in deep learning increased rapidly.

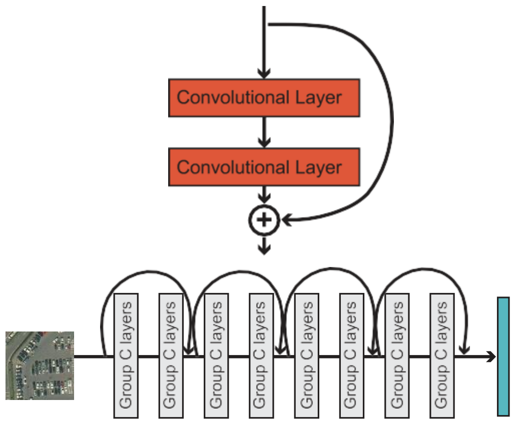
The architecture of AlexNet is shown in Fig. 14. The first convolutional layer performs convolution and max pooling with Local Response Normalization (LRN) where 96 different receptive filters are used that are 11×11 in size. The max pooling operations are performed with 3×3 filters with a stride size of 2. The same operations are performed in the second layer with 5×5 filters. 3×3 filters are used in the third, fourth, and fifth convolutional layers with 384, 384, and 296 feature maps respectively. Two fully connected (FC) layers are used with dropout followed by a Softmax layer at the end. Two networks with similar structure and the same number of feature maps are trained in parallel for this model. Two new concepts, Local Response Normalization (LRN) and dropout, are introduced in this network. LRN can be applied in two different ways: first applying on single channel or feature maps, where an N×N patch is selected from same feature map and normalized based one the neighborhood values. Second, LRN can be applied across the channels or feature maps (neighborhood along the third dimension but a single pixel or location).

AlexNet has 3 convolution layers and 2 fully connected layers. When processing the ImageNet dataset, the total number of parameters for AlexNet can be calculated as follows for the first layer: input samples are 224×224×3, filters (kernels or masks) or a receptive field that has a size 11, the stride is 4, and the output of the first convolution layer is 55×55×96. According to the equations in section 3.1.4, we can calculate that this first layer has 290400 (55×55×96) neurons and 364 (11 ×11×3 = 363 + 1 bias) weights. The parameters for the first convolution layer are 290400×364 = 105,705,600. Table II shows the number of parameters for each layer in millions. The total number of weights and MACs for the whole network are 61M and 724M respectively



2) Resnet

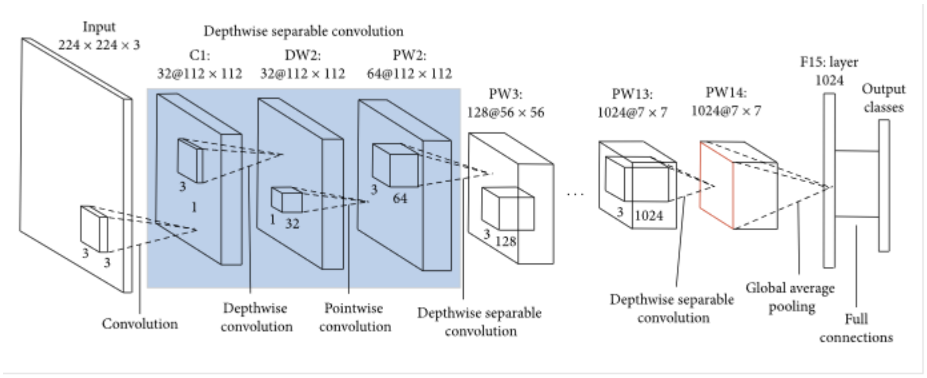
Another model structure, ResNet [57], won the 2015 ILSVRcompetition, as shown in Figure 3(b). The model takes the depth to extreme values, and for the first time, the model achieved a depth of 152 layers. ResNet50 is the 50-layer version of ResNet. To train deeper structures, ResNet introduced a residual module, as shown in Figure 3(a), which implements the delivery of information between layers through a shortcut connection. Without adding parameters, this model can directly place shallow features into the upperlayer, thus greatly increasing the training speed of the model.ResNet achieved a milestone in the ImageNet competition [56] and was the first model to surpass human classification in the ImageNet classification task. Moreover, as the main backbone of the detection model, ResNet has achieved good results for the VOC Pascal [58] and COCO [59] datasets and in other detection tasks [60]. Many RSI detection applications are based on ResNet [61].



3) MobileNet

MobileNet model is designed to be used in mobile applications, and it is TensorFlow's first mobile computer vision model. MobileNet uses depthwise separable convolutions**.** It significantly reduces the number of parameters when compared to the network with regular convolutions with the same depth in the nets. This results in lightweight deep neural networks.

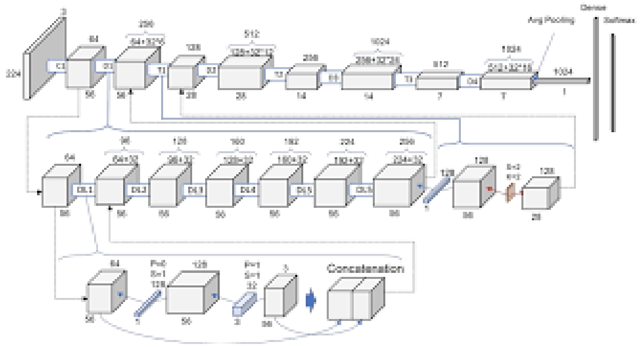
A depthwise separable convolution is made from two operations: Depthwise Convolution and Pointwise Convolution. MobileNet is a class of CNN that was open-sourced by Google, and therefore, this gives us an excellent starting point for training our classifiers that are insanely small and insanely fast.



4) DenseNet

In a normal feedforward network, an output from a layer is passed on to only the next occurring layer and as the network goes deep it would have access to only the higher-level features but not those from lover levels closer to input. In Densenet every subsequent layer receives input from all the previous layers in the chain as can be observed from above figure. All these values are concatenated so that information is not lost. The concatenated feature map is passed through a composite function consisting of Batch Normalization, Relu and 3x3 Convolution. The output is then passed on to every subsequent layer and the process is repeated.

But simple concatenation can lead to very huge number of parameters in a deep network. So the architecture is divided into multiple dense blocks each separated by a block called transition block. In dense block the information flows in the way described in above paragraph. Transition block is the place where downsampling occurs to prevent blowing up. The transition layer consists of batch normalization layers followed by 1x1 convolution layer which is followed by 2x2 average pooling layer.



B. ATTACK ALGORITHM

These excellent models still cannot escape the problem of adversarial examples. The goal of the adversarial example is primarily to make the model misclassify the minimally changed image when the original image can be classified correctly. The attack algorithms of adversarial examples arealso applicable to RSIs.

1) White-Box Adversarial Attacks

- Gradient-Based Attacks: Gradient Attack, Gradient Sign Attack (FGSM), Iterative Gradient Attack, Iterative Gradient Sign Attack, DeepFool L2 Attack, DeepFool L∞ Attack, L-BFGS Attack, SLSQP Attack, Jacobian-Based Saliency Map Attack
- Score-Based Attacks: Single Pixel Attack, Local Search Attack, Approximate L-BFGS Attack
- Decision-Based Attacks: Boundary Attack, Pointwise Attack, Additive Uniform Noise Attack, Additive Gaussian Noise Attack, Salt and Pepper Noise Attack, Contrast Reduction Attack, Gaussian Blur Attack, Precomputed Images Attack

2) Fooling Rate

To evaluate the effects of attack algorithms, we introduce thedefinition of the fooling rate [36], which reflects the model'sratio of misclassified images to attack images and can be represented as follows,

$$\text{fooling rate} = \frac{N_{misclassification}}{N_{attack}} \times 100\%,$$

where $N_{misclassification}$ represents the number of misclassified images after the attack. Similarly, N_{attack} represents the number of images attacked. The fooling rate represents how many adversarial examples can successfully fool the model. It canbe used to evaluate both models and attack algorithms. For the same model, a strong attack algorithm can obtain more adversarial examples that can fool the model with a higher fooling rate. For the same attack algorithm, a high fooling rate indicates that the model has poor robustness.

5. Experiments

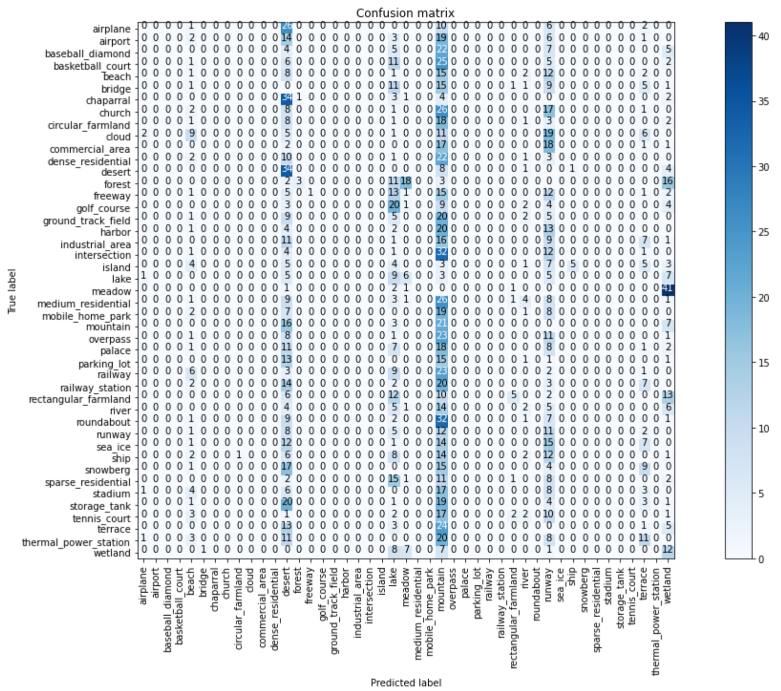
In this section, we first use the white-box attack algorithms to attack multiple converged RSI scene classification modelsunder different datasets. Second, we analyze the transferability of TSAA. The attack object of the adversarial example is a model with high accuracy that has been converged. In the experiment, we use three commonly used RSI datasets, including UC Merced Land Use [62], NWPU-RESISC45 [63]. For all datasets, we choose 80% as a training set and the remaining 20% as a test set during the training progress. For deep learning models, we choose AlexNet, ResNet50, Resnet101, MobileNetV$_2$ and DenseNet121 which are widely used in RSI scene classification systems. The experimental platform is based on Ubuntu 16.04, with a 12 × 3.20 GHz Core i5 CPU, Tesla M60 GPU and PyTorch Framework. Cross-Entropy Function was used as the loss function with Adam Optimizer being selected for optimizing the weights. Confusion Matrix was used as the evaluation Metrics along with accuracy.

A. *White-Box Attacks*

1) NWPU-RESIC45

- Epsilon: 0.05

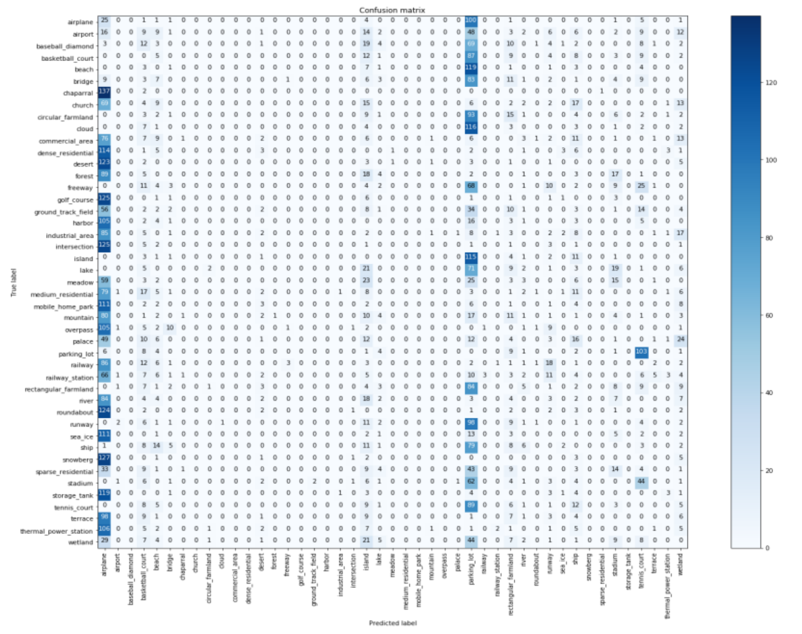
	Original Accuracy	0.747058824	0.885373609	0.892686804	0.894117647	0.914149444
	Best Attack Accuracy	0.185	0.479	0.381	0.573	0.449
	Percentage of Performance Drop	**75.23622047**	45.89854552	57.31985752	35.91447368	50.88330435
Attack No.	**Attack Name**	**AlexNet**	**Resnet50**	**Resnet101**	**MobileNet_v2**	**DensetNet**
1	**L2 Contrast Reduction Attack**	0.206999958	0.582499981	0.55249998	0.604999989	0.653999984
2	**Virtual Adversarial Attack**	**0.185499966**	0.566999972	0.567499965	0.605999976	0.655499995
3	**DDNA Attack**	0.204999983	0.565999985	0.566499978	0.597499967	0.660999984
4	**L2 Projected Gradient Descent Attack**	0.20449996	0.549499989	0.56249997	0.602999985	0.648499995
5	**Linf Projected Gradient Descent Attack**	0.20449996	0.552999973	0.561499983	0.592499971	0.661999971
6	**L2 Basic Iterative Attack**	0.194999933	0.57249999	0.563999981	0.583499968	0.67049998
7	**Linf Basic Iterative Attack**	0.20449996	0.555999964	0.554499984	**0.573499978**	0.631499976
8	**L2 Fast Gradient Attack**	0.21299994	0.555999964	0.557999969	0.627999991	0.66049999
9	**Linf Fast Gradient Attack**	0.201499939	0.563999981	0.551999986	0.602499992	0.660999984
10	**L2 Repeated Additive Gaussian Noise**	0.21299994	0.552999973	0.552999973	0.593499988	0.664499998
11	**L2 Repeated Additive Uniform Noise**	0.206499934	0.584999979	0.539499968	0.617499977	0.662999988
12	**L2 Clipping Aware Repeated Additive Gaussian Noise Attack**	0.20599997	0.564499974	0.559999973	0.604999989	0.661499977
13	**L2 Clipping Aware Repeated Additive Uniform Noise Attack**	0.193499982	0.588999987	0.573999971	0.603499979	0.666999996
14	**Attack**	0.218999982	0.564499974	0.576499969	0.59799999	0.673999995
15	**Newton Fool Attack**	0.201499939	**0.479499996**	**0.380499959**	0.615999997	**0.449499965**
16	**Linf Deep Fool Attack**	0.213499963	0.539999992	0.552999973	0.594999969	0.661499977
17	**Salt And Pepper Noise Attack**	0.19599998	0.546999991	0.564999968	0.593499988	0.659999996
18	**L2 Deep Fool Attack**	0.20599997	0.548499972	0.57249999	0.586499989	0.66049999
19	**L2 Additive Gaussian Noise Attack**	0.237999976	0.536999971	0.548999965	0.613999993	0.676999986
20	**L2 Additive Gaussian Noise Attack**	0.23149997	0.532999992	0.563499987	0.602499992	0.662999988
21	**L2 Clipping Aware Additive Gaussian**	0.219499946	0.532999992	0.569499969	0.57949999	0.675499976
22	**Attack**	0.222499967	0.539499968	0.561499983	0.572999984	0.672499985
23	**Linf Additive Uniform Noise Attack**	0.224499941	0.523499966	0.570499986	0.59799999	0.659499973
24	**L2 Carlini Wagner Attack**	0.229499936	0.532499969	0.57249999	0.597499967	0.656499982
25	**FGM**	0.229999959	0.544999987	0.566999972	0.599499971	0.672999978
26	**FGSM**	0.220999956	0.54549998	0.533499986	0.584499985	0.663999975
27	**L2 PGD**	0.227999985	0.522999972	0.571999967	0.573999971	0.657999992
28	**Linf PGD**	0.229499936	0.548999965	0.530499965	0.596999973	0.667499989
29	**PGD**	0.209999979	0.565999985	0.564499974	0.600499988	0.664999992



Confusion Matrix for Virtual Adversarial Attack on AlexNet

- Epsilon: 1.0

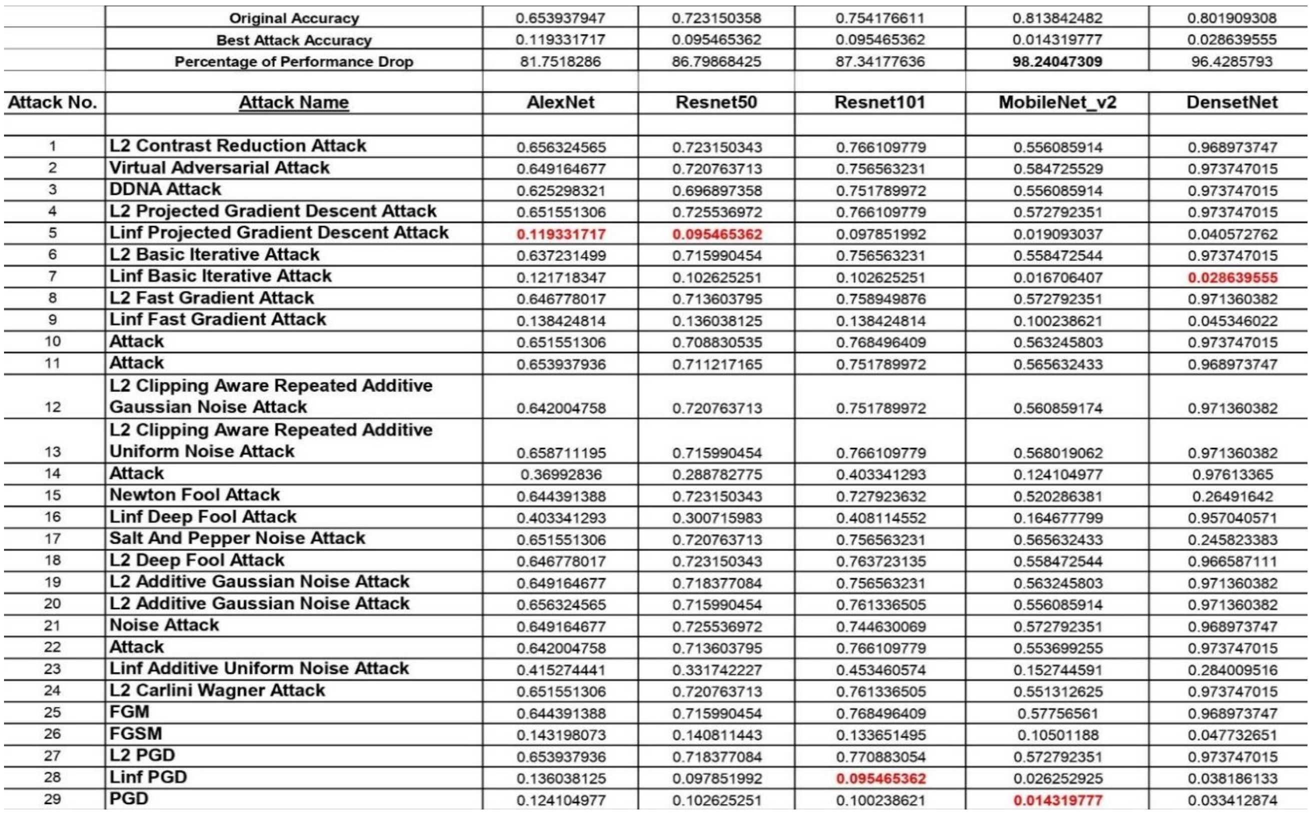
	Original Accuracy	0.747058824	0.885373609	0.892686804	0.894117647	0.914149444
	Best Attack Accuracy	0.718600959	0.848807633	0.841494441	0.885055646	0.505723357
	Percentage of Performance Drop	3.809320504	4.130005188	5.734638752	1.013513243	**44.67826232**
Attack No.	**Attack Name**	**AlexNet**	**Resnet50**	**Resnet101**	**MobileNet_v2**	**DensetNet**
1	**L2 Contrast Reduction Attack**	0.744038165	0.887599364	0.896820351	0.89125596	0.91462639
2	**Virtual Adversarial Attack**	0.746263921	0.887758344	0.894117646	0.891732909	0.915103339
3	**DDNA Attack**	0.750238478	0.886645466	0.895389505	0.892368838	0.915262319
4	**L2 Projected Gradient Descent Attack**	0.748012722	0.886327505	0.892686807	0.891096979	0.915739268
5	**Linf Projected Gradient Descent Attack**	0.746740848	0.883942768	0.892209858	0.889984101	0.916375197
6	**L2 Basic Iterative Attack**	0.747694761	0.886804454	0.895707473	0.892368838	0.917329095
7	**Linf Basic Iterative Attack**	0.74244833	0.8836248	0.894753575	0.892209858	0.915580288
8	**L2 Fast Gradient Attack**	0.743720204	0.884737678	0.895707473	0.892686807	0.914467409
9	**Linf Fast Gradient Attack**	0.740381569	0.883465819	0.892050877	0.887122415	0.912718602
10	**Attack**	0.743402213	0.886804454	0.896184422	0.893640697	0.914308429
11	**Attack**	0.746581882	0.884101748	0.895548493	0.893004768	0.91478537
12	**L2 Clipping Aware Repeated Additive Gaussian Noise Attack**	0.745469004	0.884737678	0.894594595	0.893958665	0.913036563
13	**L2 Clipping Aware Repeated Additive Uniform Noise Attack**	0.745310009	0.885532595	0.893958665	0.891891889	0.91399046
14	**Attack**	0.746422887	0.888553262	0.89793323	0.889348172	0.91399046
15	**Newton Fool Attack**	**0.718600959**	**0.848807633**	**0.841494441**	**0.89062003**	**0.505723357**
16	**Linf Deep Fool Attack**	0.741176486	0.884737678	0.889507152	0.888871223	0.915103339
17	**Salt And Pepper Noise Attack**	0.747853726	0.887758344	0.884737492	0.888871223	0.915262319
18	**L2 Deep Fool Attack**	0.747853726	0.883783787	0.887421242	0.890937999	0.916693166
19	**L2 Additive Gaussian Noise Attack**	0.743402213	0.884578697	0.894117646	0.892209858	0.912400633
20	**L2 Additive Gaussian Noise Attack**	0.750397459	0.888394274	0.893163756	0.89062003	0.916057236
21	**L2 Clipping Aware Additive Gaussian Noise Attack**	0.74562797	0.888394274	0.896025434	0.891096979	0.917170115
22	**Attack**	0.74594596	0.884101748	0.897774242	0.892368838	0.916057236
23	**Linf Additive Uniform Noise Attack**	0.744992048	0.885532595	0.892368838	0.890779011	0.915421307
24	**L2 Carlini Wagner Attack**	0.747694761	0.884101748	0.895230524	0.894117646	0.915898249
25	**FGM**	0.746581882	0.887122415	0.894594595	0.89062003	0.913354531
26	**FGSM**	0.740540534	0.880286172	0.88966614	0.885055646	0.913513511
27	**L2 PGD**	0.747535765	0.887281403	0.892209858	0.891732909	0.914308429
28	**Linf PGD**	0.747853726	0.884737678	0.893958665	0.891096979	0.914467409



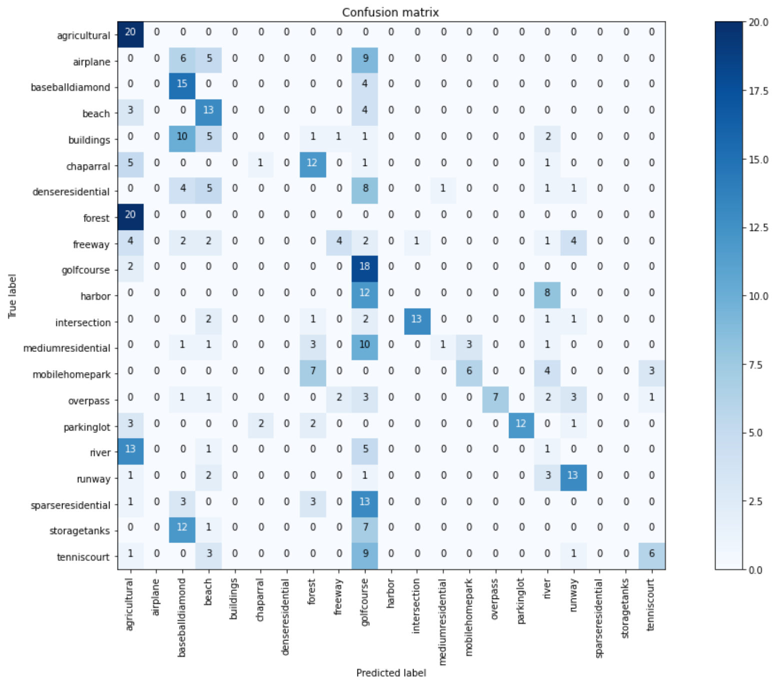
Confusion Matrix for Newton Fool Attack on DenseNet121

2) UC Merced Land Use Dataset

- Epsilon: 0.05



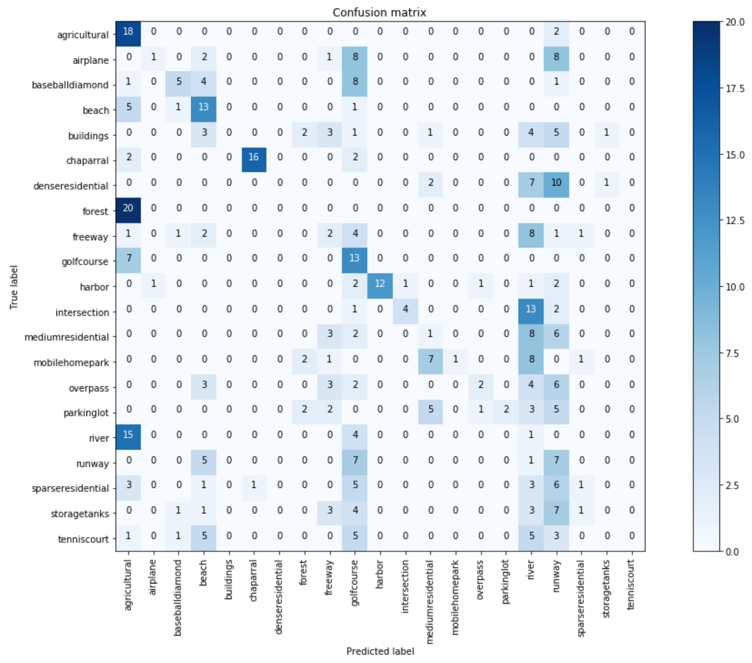
Attack No.	Attack Name	AlexNet	Resnet50	Resnet101	MobileNet_v2	DensetNet
	Original Accuracy	0.653937947	0.723150358	0.754176611	0.813842482	0.801909308
	Best Attack Accuracy	0.119331717	0.095465362	0.095465362	0.014319777	0.028639555
	Percentage of Performance Drop	81.7518286	86.79868425	87.34177636	**98.24047309**	96.4285793
1	**L2 Contrast Reduction Attack**	0.656324565	0.723150343	0.766109779	0.556085914	0.968973747
2	**Virtual Adversarial Attack**	0.649164677	0.720763713	0.756563231	0.584725529	0.973747015
3	**DDNA Attack**	0.625298321	0.696897358	0.751789972	0.556085914	0.973747015
4	**L2 Projected Gradient Descent Attack**	0.651551306	0.725536972	0.766109779	0.572792351	0.973747015
5	**Linf Projected Gradient Descent Attack**	**0.119331717**	**0.095465362**	0.097851992	0.019093037	0.040572762
6	**L2 Basic Iterative Attack**	0.637231499	0.715990454	0.756563231	0.558472544	0.973747015
7	**Linf Basic Iterative Attack**	0.121718347	0.102625251	0.102625251	0.016706407	**0.028639555**
8	**L2 Fast Gradient Attack**	0.646778017	0.713603795	0.758949876	0.572792351	0.971360382
9	**Linf Fast Gradient Attack**	0.138424814	0.136038125	0.138424814	0.100238621	0.045346022
10	**Attack**	0.651551306	0.708830535	0.768496409	0.563245803	0.973747015
11	**Attack**	0.653937936	0.711217165	0.751789972	0.565632433	0.968973747
12	**L2 Clipping Aware Repeated Additive Gaussian Noise Attack**	0.642004758	0.720763713	0.751789972	0.560859174	0.971360382
13	**L2 Clipping Aware Repeated Additive Uniform Noise Attack**	0.658711195	0.715990454	0.766109779	0.568019062	0.971360382
14	**Attack**	0.36992836	0.288782775	0.403341293	0.124104977	0.97613365
15	**Newton Fool Attack**	0.644391388	0.723150343	0.727923632	0.520286381	0.26491642
16	**Linf Deep Fool Attack**	0.403341293	0.300715983	0.408114552	0.164677799	0.957040571
17	**Salt And Pepper Noise Attack**	0.651551306	0.720763713	0.756563231	0.565632433	0.245823383
18	**L2 Deep Fool Attack**	0.646778017	0.723150343	0.763723135	0.558472544	0.966587111
19	**L2 Additive Gaussian Noise Attack**	0.649164677	0.718377084	0.756563231	0.563245803	0.971360382
20	**L2 Additive Gaussian Noise Attack**	0.656324565	0.715990454	0.761336505	0.556085914	0.971360382
21	**Noise Attack**	0.649164677	0.725536972	0.744630069	0.572792351	0.968973747
22	**Attack**	0.642004758	0.713603795	0.766109779	0.553699255	0.973747015
23	**Linf Additive Uniform Noise Attack**	0.415274441	0.331742227	0.453460574	0.152744591	0.284009516
24	**L2 Carlini Wagner Attack**	0.651551306	0.720763713	0.761336505	0.551312625	0.973747015
25	**FGM**	0.644391388	0.715990454	0.768496409	0.57756561	0.968973747
26	**FGSM**	0.143198073	0.140811443	0.133651495	0.10501188	0.047732651
27	**L2 PGD**	0.653937936	0.718377084	0.770883054	0.572792351	0.973747015
28	**Linf PGD**	0.136038125	0.097851992	**0.095465362**	0.026252925	0.038186133
29	**PGD**	0.124104977	0.102625251	0.100238621	**0.014319777**	0.033412874



Confusion Matrix for PGD Attack on MobileNetV$_2$

- Epsilon: 1.0

	Original Accuracy	0.653937947	0.723150358	0.754176611	0.813842482	0.801909308
	Best Attack Accuracy	0.627684951	0.706443906	0.732696891	0.799522668	0.785202861
	Percentage of Performance Drop	**4.01460059**	2.310232169	2.84810214	1.759531398	2.083333731
Attack No.	**Attack Name**	**AlexNet**	**Resnet50**	**Resnet101**	**MobileNet_v2**	**DensetNet**
1	**L2 Contrast Reduction Attack**	0.642004758	0.708830535	0.766109779	0.809069201	0.811455846
2	**Virtual Adversarial Attack**	0.670644373	0.708830535	0.751789972	0.818615749	0.794749394
3	**DDNA Attack**	0.634844869	0.715990454	0.747016698	0.809069201	0.799522668
4	**L2 Projected Gradient Descent Attack**	**0.627684951**	0.720763713	0.756563231	0.825775653	0.801909298
5	**Linf Projected Gradient Descent Attack**	0.649164677	0.715990454	0.763723135	0.818615749	0.806682572
6	**L2 Basic Iterative Attack**	0.644391388	0.725536972	0.761336505	0.825775653	0.801909298
7	**Linf Basic Iterative Attack**	0.646778017	0.708830535	0.773269683	0.813842475	0.806682572
8	**L2 Fast Gradient Attack**	0.651551306	0.713603795	0.756563231	0.830548912	0.801909298
9	**Linf Fast Gradient Attack**	0.656324565	0.708830535	0.766109779	0.811455846	0.797136024
10	**Attack**	0.651551306	0.723150343	0.758949876	**0.799522668**	0.799522668
11	**L2 Repeated Additive Uniform Noise Attack**	0.649164677	0.706443906	0.761336505	0.806682572	0.799522668
12	**L2 Clipping Aware Repeated Additive Gaussian Noise Attack**	0.651551306	0.725536972	0.754176602	0.823389009	0.809069201
13	**L2 Clipping Aware Repeated Additive Uniform Noise Attack**	0.644391388	0.718377084	0.758949876	0.804295942	0.801909298
14	**Linf Repeated Additive Uniform Noise Attack**	0.646778017	0.723150343	0.756563231	0.818615749	0.797136024
15	**Newton Fool Attack**	0.649164677	0.711217165	0.732696891	0.816229105	0.794749394
16	**Linf Deep Fool Attack**	0.663484484	0.730310261	0.766109779	0.809069201	0.801909298
17	**Salt And Pepper Noise Attack**	0.646778017	0.715990454	0.763723135	0.818615749	0.797136024
18	**L2 Deep Fool Attack**	0.649164677	0.723150343	0.768496409	0.816229105	0.801909298
19	**L2 Additive Gaussian Noise Attack**	0.639618129	0.727923632	0.780429587	0.801909298	0.804295942
20	**L2 Additive Gaussian Noise Attack**	0.653937936	0.708830535	0.761336505	0.809069201	0.794749394
21	**L2 Clipping Aware Additive Gaussian**	0.634844869	0.727923632	0.758949876	0.806682572	0.804295942
22	**L2 Clipping Aware Additive Uniform Noise**	0.639618129	0.720763713	0.763723135	0.804295942	0.78758949
23	**Linf Additive Uniform Noise Attack**	0.658711195	**0.706443906**	0.751789972	0.811455846	0.801909298
24	**L2 Carlini Wagner Attack**	0.651551306	0.715990454	0.766109779	0.821002379	0.801909298
25	**FGM**	0.658711195	0.718377084	0.758949876	0.813842475	0.804295942
26	**FGSM**	0.63245821	0.713603795	0.773269683	0.813842475	0.797136024
27	**L2 PGD**	0.63245821	0.730310261	0.768496409	0.816229105	0.801909298
28	**Linf PGD**	0.637231499	0.708830535	0.758949876	0.809069201	**0.785202861**
29	**PGD**	0.642004758	0.725536972	**0.732696891**	0.816229105	0.806682572



Confusion Matrix for L_2-Projected Gradient Descent Attack on AlexNet

6. Experiment Observations

Following Observations were made on the basis of the experiments conducted in the above study:

- On NWPU-RESIC45, the highest drop in accuracy was observed when Virtual Adversarial Attack **(75.23%)** and Newton Fool Attack **(44.67%)** were deployed under eps= 0.05 and eps=1.0 respectively. to create adversarial examples on the AlexNet and DenseNet respectively.
- On UC Merced Dataset, the highest drop in accuracy was observed when L_2 Projected Gradient Descent Attack **(4.01%)** and PGD Attack **(98.24%)** were deployed under eps= 0.05 and eps=1.0 respectively. to create adversarial examples on the AlexNet and MobileNetV_2 respectively.

7. Experiment Discussion

In this experiment, we verify that adversarial examples also exist in RSI scene classification systems. We find that the use of training datasets with different scales and different model structures has an impact on model vulnerability; eventhe same model exhibits different model vulnerabilities for different attack algorithms.

Adversarial examples of RSIs also have attack selectivity.From a geometric point of view, the RSI dataset is represented as a data point in the high-dimensional space in the CNN model, and the adversarial example is deviated from the boundary of the original cluster by adding small perturbations to the data points, which causes the models to misclassifyit. Therefore, data points at a cluster boundary are more likely to be successfully attacked than data points near the center of the cluster. In addition, we find that most of the adversarial classes are beaches. Beaches might have special characteristics. We believe that this research can supplyideas for future research on security and defense methods concerning adversarial examples in RSI scene classification systems.

The research on adversarial examples of RSIs is stillrare. Compared to natural images, adversarial examples of RSIs also have the same properties in some aspects. However, attack selectivity of adversarial examples is a property not explored in natural images. This property might be more helpful for understanding the correlation between ground objects. This study also has somelimitations: (1) in the real scenario, adversarial examples of RSIs are usually placed with some physical objects, which would change the impact of the adversarial example problem [69]. (2) RSI has spectral and spatial properties. Different spatial resolutions would produce different adversarial examples, and these adversarial examples would have different properties [70]. In further research, we will explore these issues. Importantly, we will design defensive algorithms to solve the adversarial example problem.

8. Conclusion and Future Work

In this article, we discuss the adversarial example problem in RSI scene classification for the first time. The experimentshows that adversarial examples are difficult for the human eye to recognize, and they allow the RSI scene classification system to achieve false results. This situation can causeserious problems for applications based on RSI scene classification systems.

Adversarial examples in the field of RSIs are a problem that should be addressed. In the field of RSI applications, many detection and segmentation tasks remain, such as building detection and land cover classification. The processing of these tasks is different from that of image classification. Theadversarial example problems of these tasks require further study, and this topic is of great significance for the design ofa secure and robust CNN-based remote sensing application system.

Adversarial Defenses in RSI

1 Introduction

In an adversarial setting, “attacker” and “defender” can be thought of as two parties playing a multi- move game, taking alternative turns to play their move. A successful “attack” will be a gain for the attacker and a loss for the defender, and likewise, a successful “defense” will be a gain for the defender and a loss for the attacker. Thus, this approach manifests itself into a zero-sum two-person game. We hypothesize that such a zero-sum game approach will enable us to develop a strong adversarial defense.

We designed our mixed strategy defense that builds on the Stochastic Activation Pruning (SAP) algorithm. Stochastic Activation Pruning is a method which stochastically drops out nodes in each layer during forward propagation. The SAP function converts the activation map of a layer to a multinomial distribution and samples the nodes with replacement with a probability proportional totheir activation magnitudes. Unlike the original SAP algorithm [15], we leverage the SAP method tocreate a novel technique utilizing multiple SAP (Multi-SAP) networks. Instead of creating a single model, this multi-SAP method iterates over the SAP function multiple times creating differently pruned networks attributed to the variation in sampling in every iteration. We implemented two defense strategies using multi-SAP on top of adversarial training which improved PGD adversarialdefense accuracy by roughly 6-7%.

We also explore two kinds of defenses against such adversarial attacks using generators in the context of generative adversarial networks. The first is training a generator that can generate defensive perturbations, which when added to the input adversarial image cause it to be correctly classified. The second is to use a generator trained on the original image distribution to generate an image thatis as close as possible to the adversarial input.

These methods are aimed at reducing the amount of information that flows back from the classification to the input, which is what adversarial attacks depend on. Our first approach, dubbed Counter GAN, attempts to use both random noise vectors and noise vectors which are a non-differentiable function of the input images. The second method, Defense GAN [8] aims to project the adversarial image back to the natural image space such that it gets classified correctly. To this end, we generate our own adversarial dataset, train our own classifiers and implement a simple baseline detector to detectadversarial images coming from black-box attacks.

2 Related Work

2.1 Our attacks and defenses

Gradient based attacks leverage the backpropagation process to develop a perturbation vector for the input image by making a slight change to the input gradients. These methods consider model parameters to be constant and the input to be a variable which in turn is used to obtain the perturbation vector. This vector satisfies the condition of being very similar to the input while fooling the neuralnetwork model at the same time such that it’s imperceptible to the human eye. FGSM and PGD aretwo such gradient based attacks.

The two main approaches to perform gradient-based attacks are one-shot and iterative attacks. In one-shot attacks, the attacker takes a single step along the direction of the gradient, whereas in iterative attacks, the attacker takes several steps, instead of a single one. The FGSM method is a one-shot attack, whereas PGD is an iterative attack.

2.1.1 FGSM Attack

The Fast Gradient Sign Method (FGSM) is a form of untargeted attack, proposed by Goodfellow et al, that generates adversarial samples in the L_∞ neighborhood of benign samples. It generates adversarial images by adding pixel-wide perturbations along the direction of the gradient. An FGSM-generatedadversarial sample can be defined as,

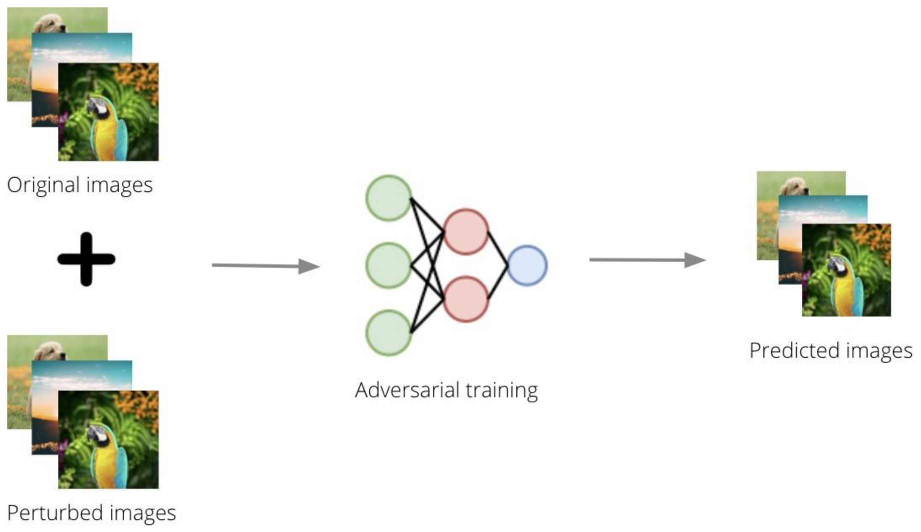
$$x' = x + \epsilon.sign[\nabla_x J(\theta, x, y)] \quad (1)$$

In the above equation, x is the benign sample that has been perturbed by the addition of the gradient to obtain an adversarial image, x’. The gradient can be computed as shown in equation (2), where $\delta$is 0 in the first time-step. δ can be adjusted according to a step size α (given by equation (3)), suchthat δ lies in the range [-ϵ, ϵ], thereby satisfying the inequality in equation (4).

$$g = \nabla_{\delta} l(h_{\theta}(x+\delta), y) \tag{2}$$

$$\delta = \delta + \alpha g \tag{3}$$

$$l_{\infty} norm, ||\delta||_{\infty} \le \epsilon \tag{4}$$



Adversarial training defense illustration

2.1.2 PGD Attack

Projected Gradient Descent (PGD) is an iterative attack where it takes the same step as an FGSM multiple times in the direction of maximising loss on a particular input while keeping the size of perturbation smaller than epsilon. In PGD, one starts from a random perturbation in the L_∞ space around an input and takes a gradient step in the direction of greatest loss in an iterative manner untilconvergence. To satisfy the constraint, the PGD projects the adversarial samples learned from eachiteration in the epsilon constrained L_∞ space of the benign input sample.

$$x'_{t+1} = Projected\{x'_t + \alpha.sign[\nabla_x J(\theta, x'_t, y)]\} \tag{5}$$

2.1.3 Adversarial training defense

Adversarial training is the most common defense technique is adversarial defense where adversarial samples generated from adversarial attacks are fed into DNNs during training to increase their robustness against adversaries. In simple terms, it is like an added data augmentation step where perturbed images are fed into the model as a preprocessing step where these perturbed imagesare created to best fool the deep neural-network model and train to reduce overfitting. Figure 1 demonstrates a visualization of adversarial training defense.

2.1.4 Stochastic Activation Pruning (SAP)

Stochastic Activation Pruning is a method which stochastically drops out nodes in each layer during forward propagation. The probability of retaining a node is directly proportional to the magnitude ofits activation. So, the SAP function converts the activation map of a layer to a multinomial distribution and samples the nodes with replacement with a probability proportional to their activation magnitudes. Equation 6 shows the probability of sampling the j'th activation value, $(h^i)_j$. After pruning, the remaining surviving nodes are scaled up by the inverse of the probability of sampling the nodes overall the draws, following equation 7.

$$p_j{}^i = \frac{|(h^i)_j|}{\sum_{k=1}^{a^i} |(h^i)_k|} \tag{6}$$

$$(m_p{}^i)_j = \frac{I((h^i)_j)}{1-(1-p_j{}^i)^{r_p{}^i}} \tag{7}$$

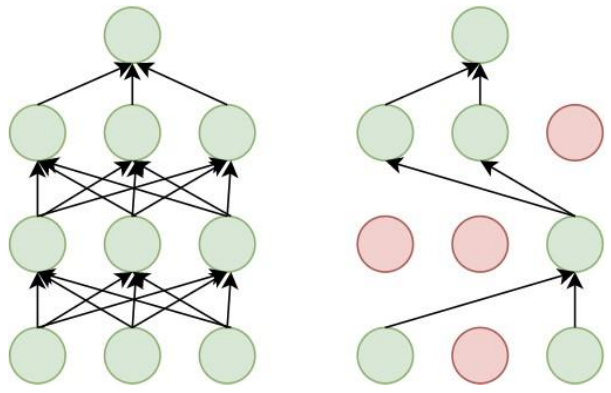


Illustration of SAP

2.2 DefenseGAN

DefenseGAN is a generative model that is trained on unperturbed images and learns their distribution [8]. During inference it aims to iteratively change the noise vector to generate an image that is as close as possible to the input image. The idea here is that the generated image would be really close to the input image while still lying in the distribution of normal images, thus being classifies correctly.In practice, DefenseGAN is able to resist a range of black-box and white-box attacks using the NWPU-RESIC45 dataset.

2.3 AdvGAN

AdvGAN is a generative model that can generate an adversarial attack given an input image instance both in the white-box and black box settings [2]. The generator is trained in a GAN, wherein it generates a noise vector. This vector is added to the input image to generate the perturbed image which is subsequently fed to the discriminator and the image classifier. A later improvement on this model, AdvGAN++ extracts a feature embedding from the input image and adds noise to the embedding.

2.4 Generative Adversarial Networks for Adversarial Attack Generation

AdvGAN is proposed in [2] which generates adversarial examples with generative adversarial networks (GANs), that can learn and approximate the distribution of original instances. We will be adopting a similar architecture to learn the mapping from an original image to a perturbed output such that the perturbed image cannot be distinguished from real images in the original class. In [3], the latent features of images are used instead of images for adversary generation. AdvGAN++, a version of AdvGAN is proposed that achieves higher attack rates than AdvGAN.

3 Our Contribution

3.1 Intuition

A preliminary strategy that we used in our baseline was to have multiple L attacks on our defense model (ResNet18) and measure the impact of each on the accuracy. Another reason to have multipleattacks was to train our defense model on more than one to make it more robust. Ideally, we would want to have our defense model to be adversarially trained against all possible attacks, but this can be a computational nightmare. Since we can’t adversarially train our defense model on all possible attacks, we hypothesize that having a defense network consisting of multiple adversarially trained models that are different from each other, and taking a combination or intelligent selection of the classification results from these diverse models is more robust against a variety of attacks. This method should make it difficult for the attacker to emulate the defense network since implementingan attack against a large number of models is computationally expensive.

We can also tie this strategy back to the zero-sum two player game in game theory, in which the maxmin value is the highest value that the attacker (Player 1) can be sure to get without knowing the defender’s (Player 2’s) moves, and equivalently, it is also the lowest value the defender can force the attacker to receive when they know their moves. Thus, we can replicate this game theory based mixed strategy by using a set of diverse models to reduce the reward that an attacker gets by making an intelligent classification decision that is based on the input, that is, the attacker’s move.

3.2 Diverse Network Creation

To test our hypothesis, our first step was to create a set of multiple models. We used three different methods to create different models from the ResNet18 base network. The first method was to use different subsets of the training dataset to train our model while keeping the accuracy levels close to the baseline model. The second method was to use different weight initializations and train the model on the entire training dataset. The final method involved the addition of skip connections every fourth and sixth layers in the ResNet architecture.

The second step was to adversarially train these multiple models because we wanted to leverage both diversity and adversarial training to defend against an attack. Diversity among the models would ensure that not all the models fail against a particular attack and that the majority of them will still succeed against the attack. This way we can take the majority vote from these models and predict thecorrect class. Thus if not all, some of the robust networks will be able to defend against the attack. We created a pool of models containing both vanilla and adversarially trained models. However, having multiple models was not enough and it was important to ensure that these models were diverse.

For checking the diversity of the models in the pool that we created, we tried to find the correlation between the models. A lower correlation between two models would indicate that they are more diverse, whereas a higher correlation would indicate that they are less diverse. We computed the correlation between two models by calculating the cosine similarity between the gradients of the loss of the models with respect to the test datum . The pairwise correlation values that we obtained using this method for our set of models were quite low and seemed to suggest that our models were diverse. However, when we tested these models against FGSM attack, they performed as poorly as our baseline model, and in some cases even worse. Given our hypothesis that having a diverse set of uncorrelated models implies that not all models fail against a particular attack, we also tried to randomly pick a model from our pool to classify each input, hoping that the inherent diversity would yield more robustness. However, we found that this technique did not work either, and performed poorly against the attack.

We thus realized that the low values of correlation as a test of diversity was misleading. As an alternative method, we turned to a naive method of calculating the number of agreements and disagreements in the predictions of the models to gauge how diverse they are from one another. If a set of models are robust to an adversarial image, we would expect them to give the correct prediction, and be in agreement with each other. However, if the attack is successful, then the models would give incorrect predictions. In this scenario, if we look at the incorrect predictions of the set of models and find that they are the same incorrect predictions, then we can say that the models are not diverse from one another. Our calculated agreement and disagreement numbers between the models indicated that they were not very diverse to one another, but it was still difficult to ascertain anything about the diversity using these numbers. We summarized all of these experiments and the results in Section 6.4.

3.3 Stochastic Activation Pruning

After conducting the diversity tests as explained in the above section, we realised that our approaches for creating models were not producing a very diverse pool of models. Thus, we tried to find a new method to create a pool of diverse networks. We decided to try Stochastic Activation Pruning (SAP)[15] which is activated after receiving the input (attacker's last move). This was a better approach because instead of finding new methods to create new models and training them, we could just use our existing models and prune them without any need for further training, saving us tons of time.

Stochastic Activation Pruning is a method which stochastically drops out nodes in each layer during forward propagation. The probability of retaining a node is directly proportional to the magnitude of its activation. So, the SAP function converts the activation map of a layer to a multinomial distribution and samples the nodes with replacement with a probability proportional to their activation magnitudes. After pruning, the remaining surviving nodes are scaled up by the inverse of the probability of sampling the nodes over all the draws. In our case, we are performing SAP after every ReLU layer and sampling 100% of the nodes every layer. Unlike the SAP paper, we wanted to leverage the novel SAP method to create multiple SAP (Multi-SAP) networks instead of creating just one by iterating over the SAP function multiple times which creates differently pruned networks attributed to the variation in sampling in every iteration.

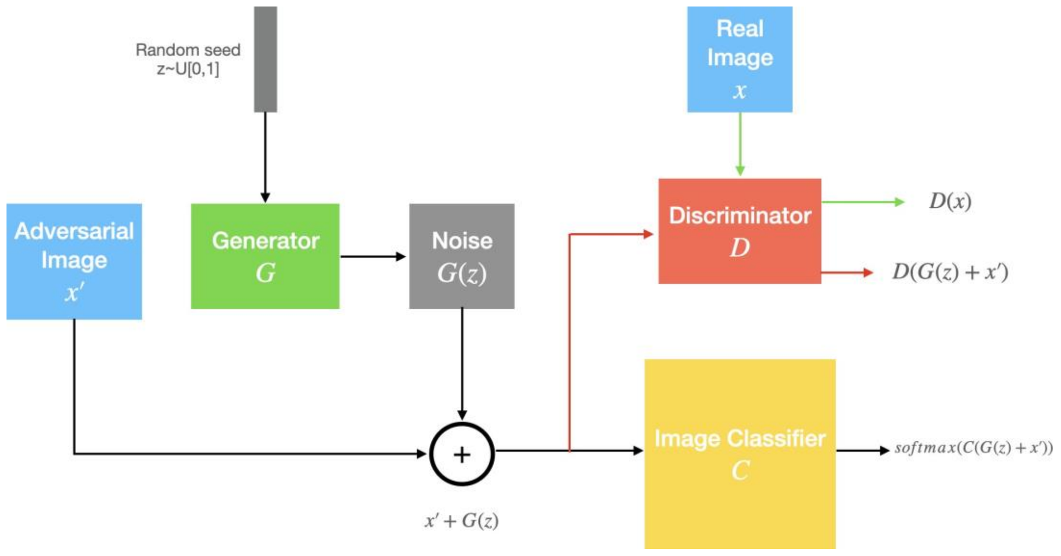
3.3.1 Novel Multi-SAP Defense Approach

Our updated defense strategy was to apply Multi-SAP on adversarially trained models and then take a majority vote of classification results from these models. We experimented with two defense strategies to evaluate the efficacy of adding Multi-SAP on top of our adversarially trained models. In the first approach we built a set of 50 networks by applying SAP with 50 iterations to one adversarially trained base network. The second approach was to take a pool (for example 5) of adversarially trained networks and apply SAP on all of them for 10 iterations, each giving us 50 networks. This combined with the initial five adversarially trained models gave us a total of 55 models. Both the strategies were followed by taking a majority vote of classification results obtained from the diverse set of networks to obtain a final prediction. A visualization of this can be observed in Figure 5.

3.4 CounterGAN

We designed a generative adversarial network to counter the effects of the adversarial perturbationadded by the attacker. We have adversarially trained a generative model to learn the distribution ofthe perturbation added by the attacker. Thus, it generates a noise that can drown out the adversarialeffect of the attack.

The generator consists of fully connected layers (Table 2), which takes a seed vector(random or image based, as discussed later) and generates a tensor of dimension 3 32 32. We add this generated noise tensor to our input image before passing it onto the discriminator. The discriminator, which consists of fully connected layers as well, acts as a binary classifier on the input image and classifies whether it belongs to the unperturbed image distribution or the adversarially perturbed image distribution (Figure 3). Our model also includes a pre-trained image classifier that has been adapted from VGG16 [13] to account for images from NWPU-RESIC dataset.



counterGAN architecture. Green arrows stand for the real image inputs and outputs, and red arrows stand for mixed image inputs and their outputs.

For every epoch, the first step is to train the discriminator. To do so, the unperturbed image is mixed with Gaussian noise and passed to the discriminator as the real label. The perturbed image is mixedwith the noise generated by the generator and passed to the discriminator as the fake label. The gradients are accumulated over these two passes and the gradient descent update is performed for the discriminator. This is followed by the training of the generator, wherein we pass the perturbed image mixed with generated noise through the discriminator as the real label and backpropagate thegradients to the generator. This mixed input is also passed to the image classifier and its gradient is backpropagated back to the generator as well. The gradient accumulation and gradient update step ofthe generator ends the epoch of training.

Generator and Discriminator architectures used in our model

Generator	Discriminator
ReLU	FC (1024)
FC (256)	Batchnorm
Batchnorm	Leaky ReLU (0.2)
ReLU	Dropout (0.3)
FC (512)	FC (512)
Batchnorm	Batchnorm
ReLU	Leaky Leaky ReLU (0.2)
FC (1024)	Dropout (0.3)
Batchnorm	FC (128)
ReLU	Batchnorm
FC (3072)	Leaky ReLU (0.2)
Sigmoid	Dropout (0.3)
	FC (1)
	Sigmoid

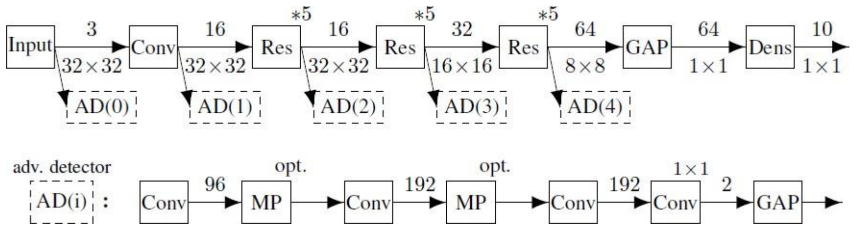
As a seed to the generator, we pass in information about the input image to facilitate training. The idea here is that the random seed should contain some information about the image but the way this information is obtained should be non-differentiable or difficult to approximate. To do this, we flatten the image into a vector and convert it into a binary vector by comparing whether the value is greater than 0.5. This is similar to a threshold function which makes this operation non-differentiable. This flattened vector can then be resized into the size of the generator. The generator can then be trained using this input noise vector.

Using a seed is important to ensure that our model is non-differentiable. This makes it difficult for an attacker to attack our model. Had we not used a seed, the model would've become deterministic, allowing an attacker to learn the distribution of noise being generated by our model. Thus, using a seed ads robustness to our model. We are using binary cross entropy as the loss metric for both the generator and the discriminator. Cross entropy loss is used as the loss metric for the image classifier. We use AdamW as our optimizer with an L_2 regularization of 0.004.

3.5 Implementation of Baselines

3.5.1 Simple Baseline

We have implemented a simple baseline model based on [7] which augments deep neural networks with a small sub-network "detector". The adversarial detector is basically a binary classifier that distinguishes genuine images from the adversarial images.



Baseline architecture including the sub networks for adversarial detection

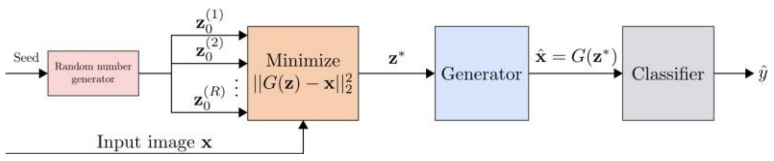
We built a classifier (shown on the upper half of Figure 4) as our baseline Whitebox model. We wrote our custom code for this model to allow for the extraction of the feature maps at the intermediate layers during the forward pass for any input. As mentioned in the paper we extracted data from 5 places in the model pipeline. These feature maps will go into the relevant Adversarial Detector model (AD(0) to AD(4)). Each of the AD models is a binary classifier that predicts if the image is an adversary or not. We used the non adversary images from the NWPU-RESIC45 and the intermediate feature map values for those images as out "Not Adversary" class and used an adversarial image generation approach for the "Adversary" class. We are in the process of training this baseline and calculating the results.

In the inference stage, for each image the 5 models will each look at the different intermediate stages, we will look at the 5 outputs and generate a final score for whether the image is an adversary or not.

3.5.2 DefenseGAN

DefenseGAN [8] is a method that relies on the generator of a pre-trained Generative Adversarial Network. It is based on the fact that adversarial images come from a different data-distribution as compared to natural images. It leverages the fact that the generator can only generate images from the natural distribution to generate an image that is as close to the adversarial image as possible. By virtue of the generator's output distribution being in the natural image space, we know that the reconstruction will not be adversarial and will be correctly classified by the classifier.

The DefenseGAN algorithm is as follows - given an adversarial image and a generator, we use a random noise vector to generate an image from the generator. Now, this is compared to the adversarial image, and a loss value corresponding to the difference between the two images is obtained. We can now use stochastic gradient descent (SGD) on the *noise vector* to make the generated image more similar to the adversarial image. Over a number of such SGD steps, the generated image becomes as close as possible to the adversarial image while still being in the realm of natural images. This reconstructed "natural" image can be fed into the classifier, where it should be correctly classified. The gradient descent is also randomly restarted a number of times, in order to ensure the closest possible reconstruction.



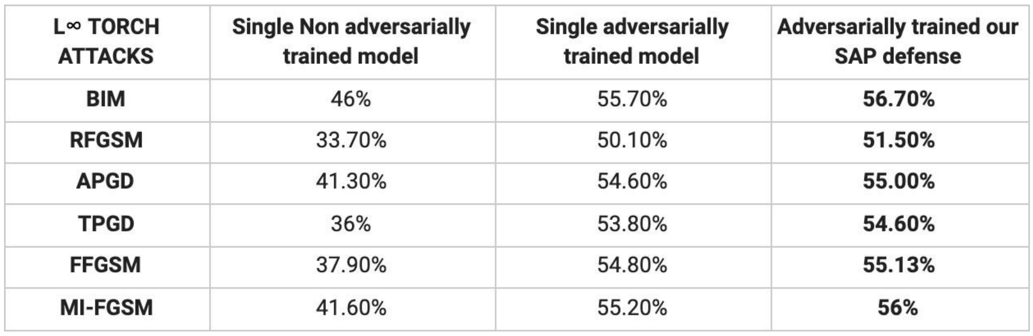
Defense-GAN architecture

4 Experiments

4.1.1 Defense Strategy 1

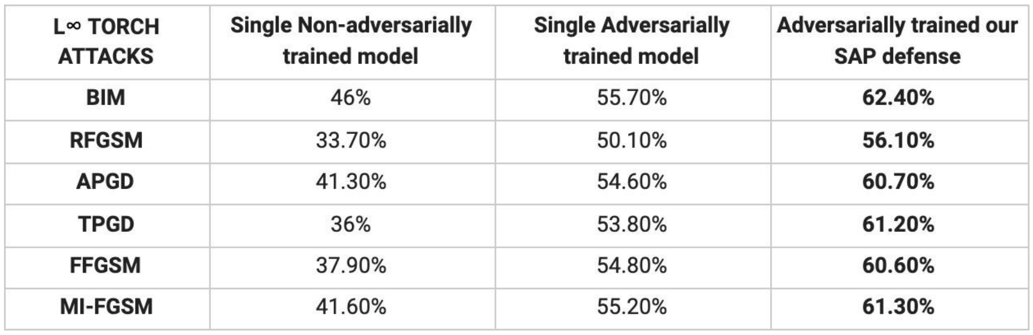
In our first defense strategy, we use one base network, here, the base ResNet18 model and adversarially train it. As mentioned in the above Section, different models are produced every time SAP is applied, owing to the variation in sampling at each iteration. Therefore, we applied 50 random seeds and obtained 50 differently pruned models of the base adversarially trained model. We then obtained a majority vote among the classification results from each of these 50 models. The results are further elucidated in the table in Figure 9.

In the table, we demonstrate our defense technique against six L attacks, imported from the torchattacks library in Pytorch. The first column represents the defense performance when the attacks were performed against a single non-adversarially trained model. This is a vanilla model with no defense applied to it. The second column consists of the results when the base model was adversarially trained and hence the defense performance improved from the previous one. The final column demonstrates the results of Multi-SAP, which are comparable to those of adversarial defense.



L∞ TORCH ATTACKS	Single Non adversarially trained model	Single adversarially trained model	Adversarially trained our SAP defense
BIM	46%	55.70%	**56.70%**
RFGSM	33.70%	50.10%	**51.50%**
APGD	41.30%	54.60%	**55.00%**
TPGD	36%	53.80%	**54.60%**
FFGSM	37.90%	54.80%	**55.13%**
MI-FGSM	41.60%	55.20%	**56%**

Performance of defense strategy 1 against different L attacks



L∞ TORCH ATTACKS	Single Non-adversarially trained model	Single Adversarially trained model	Adversarially trained our SAP defense
BIM	46%	55.70%	**62.40%**
RFGSM	33.70%	50.10%	**56.10%**
APGD	41.30%	54.60%	**60.70%**
TPGD	36%	53.80%	**61.20%**
FFGSM	37.90%	54.80%	**60.60%**
MI-FGSM	41.60%	55.20%	**61.30%**

Performance of defense strategy 2 against different L attacks

4.1.2 Defense Strategy 2

In our second defense strategy, we utilize 5 different base networks that are obtained using different splits of training data and weight initializations. These 5 networks are then adversarially trained. Now, using 10 random seeds, we create 10 diverse networks from each of these five adversarially trained networks, thereby producing 50 diverse networks. These combined with the 5 non-SAP base adversarial networks produced a total of 55 networks. The results are further demonstrated in the table in Figure 10.

In the table, we demonstrate our defense technique against six L attacks, imported from the torchattacks library in Pytorch. The table follows the same format as table in Figure 10 in the previoussection. Here, the final column demonstrates the results of this strategy of multi-SAP, which show an improvement of about 6-7% from the adversarial defense technique. As mentioned in the Literature Review, the PGD adversarially trained defense is one of the strongest defenses against most L attacks [13]. Our defense managed to outperform adversarial training as well, which can prove verypromising.

4.2 DefenseGAN Results

The results for DefenseGAN are shown in the table below. Accuracy refers to the accuracy without the defense, while accuracy with defense refers to the accuracy after applying the DefenseGAN.

DefenseGAN Results

Dataset	Accuracy	Accuracy with Defense
Real Dataset	76%	42%
Adversarial Dataset	6%	37%

The two observations to be made here are that defenseGAN is successful in improving the performance of the model against adversarial attacks. However, it does lower the accuracy of natural images as well. It is important to note here that with an increase in the number of iterations, the accuracy for both the datasets should converge to the accuracy on the natural dataset, but due to a limited number of changes (improvements) to the generated images, they do not always look like the target image. The inputs to the network are shown in Figure 6. The "defended" images generated by the DefenseGAN are shown in Figure 7, where the green boxes show correctly generated images while the red boxes show the incorrectly generated outputs. Also this method takes a long time to run (3 orders of magnitude slower than other inference methods) and each image takes a few minutes to run.In real systems, this is not feasible since it would add a lot of latency and cost to the system.

4.3 counterGAN Results

We tried the experiments outlined in Table 8 to tune our model and come up with the appropriate architecture. For each experiment we measured the accuracy of the classifier with and without the defense on both the adversarial dataset and the real dataset. These results have led us to our final model.

counterGAN experiment results

Experiment	Linear+BCE	Linear+MSE	Conv+MSE	Conv+BCE	Linear + WGAN-GP
Adv Classifier accuracy	1.03%	1.03%	1.03%	1.03%	1.03%
Adv Defense Accuracy	14.88%	11.87%	13.03%	12.47%	10.70%
Real Classifier accuracy	92.21%	92.21%	92.12%	92.14%	92.21%
Real Defense Accuracy	31.01%	28.38%	35.15%	35.27%	30.87%

5 Conclusion

Adversarial attacks against state-of-the-art deep neural networks is indeed a tough battle to combat, and building models robust to these attacks is imperative for the adoption of such systems in the real world. We looked at this problem from a game theoretic perspective and were successful in designing a defense strategy against L_∞ attacks that takes into account the attacker's current move. The results obtained from our novel multi-SAP defense technique demonstrate that the usage of diverse networks on top of adversarial training can prove to be a strong defense. The fact that our defense outperforms PGD adversarial training, one of the most powerful defenses against L_∞ attacks, by about 6-7%, is reflective of our contribution to the research in this area.

We were able to get 15% of the images correctly classified. This was an improvementover the 1% without our solution. We see that this method is promising in some ways, but thereis scope for further experimentation. Especially in areas like the noise generation methods and the generator and discriminator architectures. Besides the core idea, there were various other learnings from the multiple experiments that we performed. From our results we conclude that fully connected layers might work better with images than convolutional networks for generating noise, such asin our use case. Our experiments with various losses suggest that simple loss functions such as mean squared error and binary cross entropy might work just as well as more complicated ones likeWasserstein Loss. Our future work would focus on conducting more experiments with architectures and hyperparameters to better tune our model.

6 Future Work

While it is impressive to come up with a novel defense strategy that outperforms PGD adversarial training, we think that we can improve our results by intelligently selecting classification results from the diverse models instead of using a simple majority vote. Additionally, detecting whether aninput is adversarial or not after the attacker's last move will help us further optimize our defense byrunning Multi SAP only in the case of an attack. Moreover, we can also tune the sampling parameter in the SAP algorithm and evaluate how it affects the accuracy numbers. We are hopeful that these additional steps will strengthen our defense and help us secure our deep learning systems.

References

1. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9146660
2. https://ieeexplore.ieee.org/document/9339955
3. https://www.hindawi.com/journals/scn/2021/6663028/
4. https://ieeexplore.ieee.org/document/9119167
5. https://arxiv.org/pdf/1805.10997
6. https://www.tensorflow.org/datasets/catalog/resisc45
7. https://ieeexplore.ieee.org/document/7891544
8. https://www.tensorflow.org/datasets/catalog/uc_merced https://faculty.ucmerced.edu/snewsam/papers/Zhu_SIGSPATIAL15_LandUseClassification.pdf
9. https://github.com/bethgelab/foolbox*(GitHub)*
10. https://foolbox.jonasrauber.de
11. https://arxiv.org/abs/1707.04131
12. Ren, K., Zheng, T., Qin, Z. and Liu, X., 2020. Adversarial attacks and defenses in deep learning. Engineering.
13. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, et al. Intriguing properties of neural networks. 2013. arXiv:1312.6199.
14. Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. 2014. arXiv:1412.6572.
15. Kurakin A, Goodfellow I, Bengio S. Adversarial examples in the physical world. 2016. arXiv:1607.02533.
16. Dong Y, Liao F, Pang T, Su H, Zhu J, Hu X, et al. Boosting adversarial attacks with momentum. In: Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition; 2018 Jun 18–23; Salt Lake City, UT, USA; 2018. p. 9185–193.
17. Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: Proceedings of the 2017 IEEE Symposium on Security and Privacy; 2017 May 22–26; San Jose, CA, USA; 2017. p. 39–57.\
18. Zheng T, Chen C, Ren K. Distributionally adversarial attack. 2018. arXiv:1808.05537.
19. Moosavi-Dezfooli SM, Fawzi A, Fawzi O, Frossard P. Universal adversarial perturbations. In: Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition; 2017 Jul 21–26; Honolulu, HI, USA; 2017. p. 1765–73.
20. Xiao C, Li B, Zhu JY, He W, Liu M, Song D. Generating adversarial examples with adversarial networks. 2018. arXiv:1801.02610.
21. Madry, Aleksander, et al. "Towards deep learning models resistant to adversarial attacks." arXiv preprint arXiv:1706.06083 (2017).
22. Jan Hendrik Metzen, Tim Genewein, et al. "On Detecting Adversarial Perturbations. " arXiv preprint arXiv:1702.04267 [stat.ML] (2017).
23. Samangouei, Pouya, Maya Kabkab, and Rama Chellappa. "Defense-gan: Protecting classifiers against adversarial attacks using generative models." arXiv preprint arXiv:1805.06605 (2018).
24. Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.
25. Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572 (2014
26. Alec Radford, Luke Metz, Soumith Chintala, "Unsupervised Representation Learning with Deep Convolu- tional Generative Adversarial Networks". ICLR 2016
27. Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).
28. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
29. https://github.com/chengyangfu/pytorch-vgg-cifar10Mao, Xudong, et al. "Least squares generative adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017.
30. Martin Arjovsky, Soumith Chintala, and Leon Bottou. "Wasserstein GAN". In:ArXiV abs/1701.07875 (2017) https://arxiv.org/abs/1701.07875
31. Nicholas Carlini, David Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. 2017.
32. Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, Dawn Song.(Revised version: 14 Feb 2019) Generating Adversarial Examples with Adversarial Networks arXiv:1801.02610v5 [cs.CR]
33. Puneet Mangla, Surgan Jandial, Sakshi Varshney, Vineeth N Balasubramanian.(Revised version: 23 Dec 2019) AdvGAN++ : Harnessing latent layers for adversary generation. arXiv:1908.00706v2 [cs.CV]
34. Yann LeCun, Corinna Cortes, and Christopher JC Burges. 1998. The MNIST database of handwritten digits. (1998)
35. Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).
36. Madry, Aleksander, et al. "Towards deep learning models resistant to adversarial attacks." arXiv preprint arXiv:1706.06083 (2017).
37. an Hendrik Metzen, Tim Genewein, et al. "On Detecting Adversarial Perturbations. " arXiv preprint arXiv:1702.04267 [stat.ML] (2017).
38. Samangouei, Pouya, Maya Kabkab, and Rama Chellappa. "Defense-gan: Protecting classifiers against adversarial attacks using generative models." arXiv preprint arXiv:1805.06605 (2018).
39. Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.
40. Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572 (2014
41. Alec Radford, Luke Metz, Soumith Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". ICLR 2016