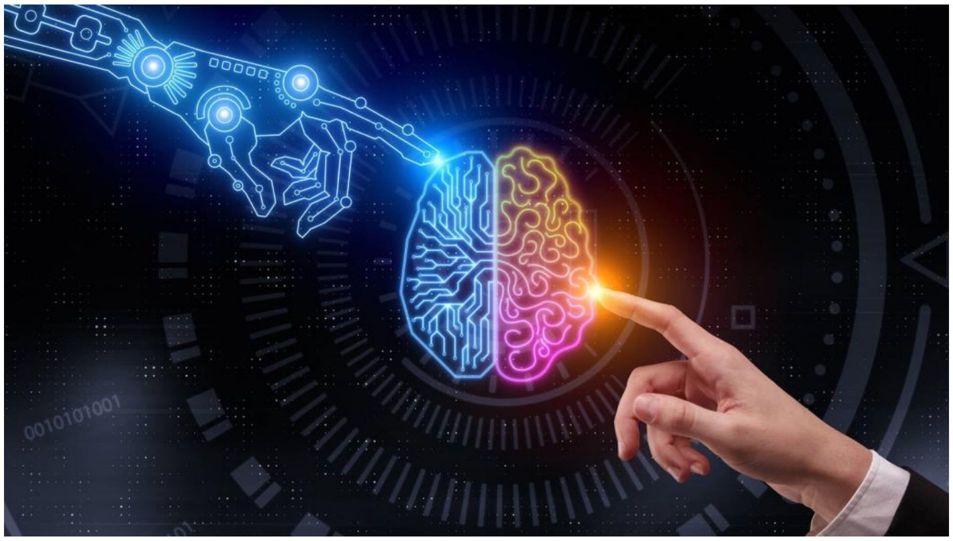
Ahsan Mafaz Hasan Abdul Kader
Supervisor – Ibrahim A. Hameed

Adversarial Attacks on Neural Networks & Defense for it

Report

Ålesund – June 03 – 2020

NTNU
Norwegian University of Science and Technology
Department of Information & Communication Technology (ICT)



NTNU
Norwegian University of Science and Technology

Summary

The Recent Advancement in the field of Machine Learning, enabled the emergence of powerful Neural Networks in domains like Computer Vision, NLP which are capable of achieving Human-level accuracy. As the result of this, incredible applications were born ranging from Face-Based Authorization & Authentication systems, self-driving cars to chatbots, language translators and the number is rising up everyday including in many safety & security sensitive environments. With this huge boon, there comes a major problem alongside "Adversarial Attacks on Neural Networks". Inorder to exploit the advantages offered by Neural Networks and to take over the control of the systems controlled by Neural Networks, Adversaries has developed variety of techniques to attack the Network by feeding in Handcrafted-Inputs known as Adversarial Inputs and make the Neural Network to predict a wrong output rather than the true output, thereby fooling the Network.

The above problem is investigated in multiple-dimensions. The first dimension being the Different flavours of these attacks, their varied nature with respect to factors like accuracy, time-consumption, perturbation-level etc. The different attack strategies are performed on various types of Neural Networks like Convolution Neural Network (CNN), Recurrent Neural Network (RNN) and Long-Short-Term-Memory (LSTM) with standard datasets as input.

The second dimension investigates about popular Defense strategies which can resist/prevent the attacks on the Neural Network, the relative strength and weakness, the resisting capability with respect to different attack types.

In the third dimension, a live-demo of the attack is demonstrated by picking up a real-world dataset and attacking the same to produce the intended output rather than the original one.

Preface

This master thesis is submitted as the final work of the Master of Science degree at the Simulation and Visualization program at the Norwegian University of Science and Technology (NTNU), Department of ICT and Natural Sciences. The research and report are done during the final semester, spring 2020. This thesis aims to explore security vulnerabilities in deep learning & the ways to improve the robustness. It is investigated if it has the potential to improve the current defense strategies by exploiting the loopholes in it or if there is a possibility to come up with a new technique. The main parts of the thesis is to demonstrate & analyze popular attack & defense strategies with the complete analysis of relative strength & weakness, thereby laying the foundation for the bigger map. I've got inspired with this topic by reading across the current trends in the AI world & also strong motivation from my supervisor Ibrahim A Hameed.

Table of Contents

List of Tables

List of Figures

Abbreviations

CNN = Convolutional Neural Network

RNN = Recurrent Neural Network

LSTM = Long Short Term Memory

FGSM = Fast Gradient Sign Method

IFGSM = Iterative Fast Gradient Sign Method

RFGSM = Randomized Fast Gradient Sign Method

PGD = Projected Gradient Descent

RPGD = Randomized Projected Gradient Descent

APGD = Averaged Projected Gradient Descent

ITERLL = Iterative Least Likely Class Method

C&W = Carlini & Wagner Attack

JSMA = Jacobian Saliency Map Based Attack

NLP = Natural Language Processing

iter = iterations

Chapter 1

Introduction

Deep learning (DL) has made significant progress in several dimensions of machine learning (ML) like : image classification, object recognition, object detection ,speech recognition , language translation , voice synthesis . The online Go Master (AlphaGo) beat more than 50 top go players in the world. Recently AlphaGo Zero surpassed its previous version without using human knowledge and a generic version, AlphaZero , achieved a superhuman level within 24 hours of cross domains of chess, Shogi, and Go.

Constantly increasing number of real-world applications and systems have been powered by deep learning. For instance, companies from IT to the auto industry (e.g., Google, Telsa, Mercedes, and Uber) are testing self-driving cars, which require plenty of deep learning techniques such as object recognition, reinforcement learning, and multimodal learning. Face recognition system has been deployed in ATMs as a method of biometric authentication. Apple also provides face authentication to unlock mobile phones.

Despite great successes in numerous applications, many of the deep-learning based empowered applications are life crucial, raising great concerns in the field of safety and security. “With great power comes great responsibility”. Recent studies find that deep learning is vulnerable against well-designed input samples. These samples can easily fool a well-performing deep learning model with little perturbations imperceptible to humans.

Extensive deep learning based applications have been used or planned to be deployed in the physical world, especially in the safety-critical environments. In the meanwhile, recent studies show that adversarial examples can be applied to real world. For instance, an adversary can construct physical adversarial examples and confuse autonomous vehicles by manipulating the stop sign in a traffic sign recognition system or removing the segmentation of pedestrians in an object recognition system. Attackers can generate adversarial commands against Automatic-Speech-Recognition (ASR) models and Voice-Controllable-System(VCS) such as Apple Siri , Amazon Alexa , and Microsoft Cortana .

Deep learning is widely regarded as a “black box” technique as quoted by *Yuan et al*[13] — we all know that it performs well, but with limited knowledge of the reason. Many studies have been proposed to explain and interpret deep neural networks. From inspecting adversarial examples, we may gain insights on semantic inner levels of neural networks and find problematic decision boundaries, which in turn helps to increase robustness and performance of neural networks and improve the interpretability.

1.1 Scope of the Thesis

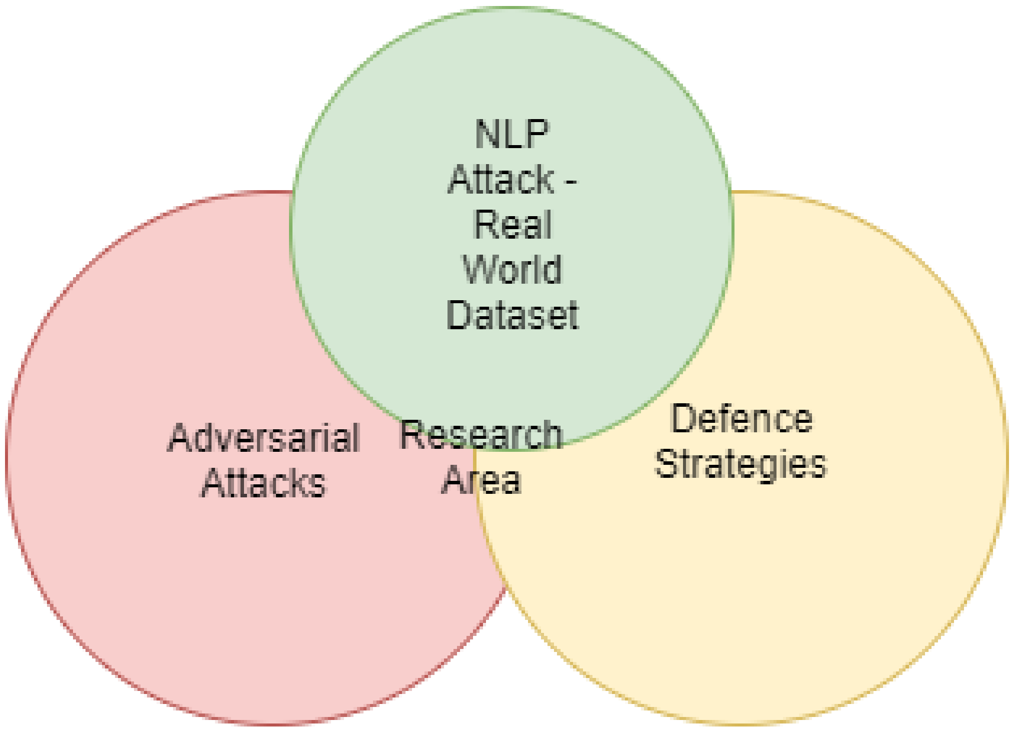


Figure 1.1: Thesis Scope

This Thesis can be dissected into 3 fragments, 1 part deals with Attack strategies along with the analysis while another part in contrary handles the Defense aspect of the same. The third fragment is dedicated exclusively for NLP domain & a real-world dataset is used as opposed to the standard datasets like MNIST, CIFAR10 .. which are used in the other two fragments.

1.2 Brief Introduction To Deep Learning

Deep learning is a type of machine learning method that makes computers to learn from experience and knowledge without explicit programming and extract useful patterns from raw data. For conventional machine learning algorithms, it is difficult to extract well-represented features due to limitations, such as curse of dimensionality, computational bottleneck , and requirement of the domain and expert knowledge.

Deep learning solves the problem of representation by building multiple simple features to represent a sophisticated concept. For example, a deep learning-based image classification system represents an object by describing edges, fabrics, and structures in the hidden layers. With the increasing number of available training data, deep learning becomes more powerful. Deep learning models have solved many complicated problems, with the help of hardware acceleration in computational time.

A neural network layer is composed of a set of perceptrons (artificial neurons). Each perceptron maps a set of inputs to output values with an activation function. The function of a neural network is formed in a chain:

$$f(x) = f^{(k)}(...f^{(2)}(f^{(1)}(x))) \tag{1.1}$$

where f $^{(i)}$ is the function of the ith layer of the network, i = 1,2,···k.

Convolutional neural networks (CNNs) and Recurrent neural networks (RNNs) are the two most widely used neural networks in recent neural network architectures. CNNs deploy convolution operations on hidden layers to share weights and reduce the number of parameters. CNNs can extract local information from grid-like input data. CNNs have shown incredible successes in computer vision tasks, such as image classification , object detection and semantic segmentation. RNNs are neural networks for processing sequential input data with variable length. RNNs produce outputs at each time step. The hidden neuron at each time step is calculated based on current input data and hidden neurons at previous time step. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) with controllable gates are designed to avoid vanishing/exploding gradients of RNNs in long-term dependency. These will be explored in depth in the future sections.

1.3 Standard Deep Neural Network Architectures

Several deep learning architectures are widely used in computer vision tasks: LeNet, VGG, AlexNet, GoogLeNet (Inception V1-V4), and ResNet, from the simplest (oldest) network to the deepest and the most complex (newest) one. AlexNet first showed that deep learning models can largely surpass conventional machine learning algorithms in the ImageNet 2012 challenge and led the future study of deep learning. These architectures made tremendous breakthroughs in the ImageNet challenge and can be seen as milestones in image classification problem. Attackers usually generate adversarial examples against these baseline architectures.

1.4 Standard Deep Learning Datasets

MNIST, CIFAR-10, ImageNet are three widely used datasets in computer vision tasks. The MNIST dataset is for handwritten digits recognition . The CIFAR-10 dataset and the ImageNet dataset are for image recognition task. The CIFAR-10 consists of 60,000 tiny color images (32×32) with ten classes. The ImageNet dataset consists 14,197,122 images with 1,000 classes. Because of the large number of images in the ImageNet dataset, most adversarial approaches are evaluated on only part of the ImageNet dataset. The Fashion MNIST dataset, similar to the MNIST dataset, consists of ten classes of fahion products like T-shirt, shoes, sneakers etc. The Youtube-Dataset is gained from Youtube consisting of about ten million images.

1.5 Types of Neural Networks

This section explores about the different types of Neural Networks, with its applications including the strength & weakness.

1.5.1 FeedForward Neural Network

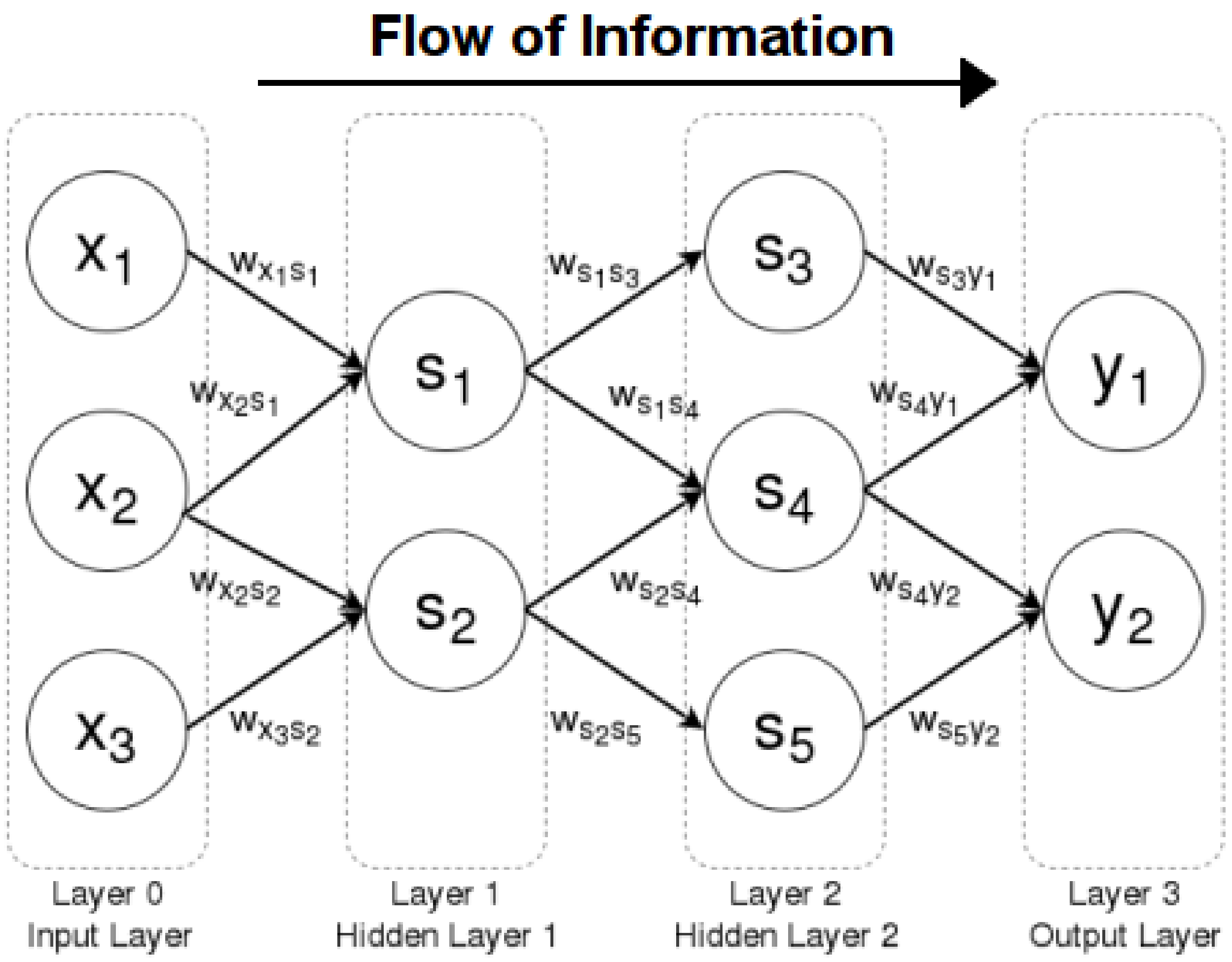


Figure 1.2: FeedForward Neural Network

Also, known as Multi-Layer-Perceptrons (MLP), a FeedForward neural network is the most basic type of vanilla Neural network. As evident from the above figure, there are three types of layers namely Input layer, Hidden layer & Output layer. The Input layer is simply a container which holds the input data which is to be feeded in to the neural network, while the output layer in contrary is a container to hold the output data/response from the neural network. All the remaining layers which are present in-between the input & output layer are called Hidden layers which forms the core of the neural network. From a technical perspective, the Hidden layer represents the input data in fine-discrete & intermediate form in such a way that when one navigates from the left-most hidden layer to the right-most one, the representation complexity increases due to the fact that input source to a particular hidden layer is simply the output from the immediate before hidden layer. It should be noted that process flow is from the left-most input layer to the right-most output layer which is a one-way communication and hence the name FeedForward Neural Network.

Lets now take the mathematical perspective to view the neural network. From this standpoint, neural networks are nothing more than a mapping function of the form y = f$(x : \theta)$ where y denotes the output while x represents the input. θ points to the model's parameters namely *weights* & *biases*. The weight values are connections between the various layers in the neural network and forms the heart of the neural network training as these weights actually encodes the relative importance of the various features in the input towards generating a particular output.

The training performance of a neural network can be evaluated or quantified using a *Cost function*, which actually measures the difference between the approximation learnt by the model & the true actual output. The result will always be a single number & the goal of the training process is to minimize this number & bring it as close to zero, which represents the point where the model has completely learnt the mapping of the given input dataset.

In practice, *optimizers* are used to minimize the cost function by updating the network's weights & biases using the network's *gradients* computed from the cost/loss function. *Stochastic Gradient Descent, Adam, Adagrad* are some of the frequently used optimizers.

The one such application of these neural networks can be attributed to any *supervised learning* problem in which we have a knowledge about the output for a given input and in addition to it, *Linear Regression* problems which calculates the output from the given input features are benefited from this as well.

The main drawback associated with the Feedforward neural networks is it works well only for Linearly separable problems i.e the different output classes can be separated by a line. But when it comes to non-linear problems or if the input is image (Computer Vision) or text (Natural Language Processing), Feed-Forward network is not the best candidate to opt for, which paves the way for other types of neural networks.

1.5.2 Convolutional Neural Network (CNN) -

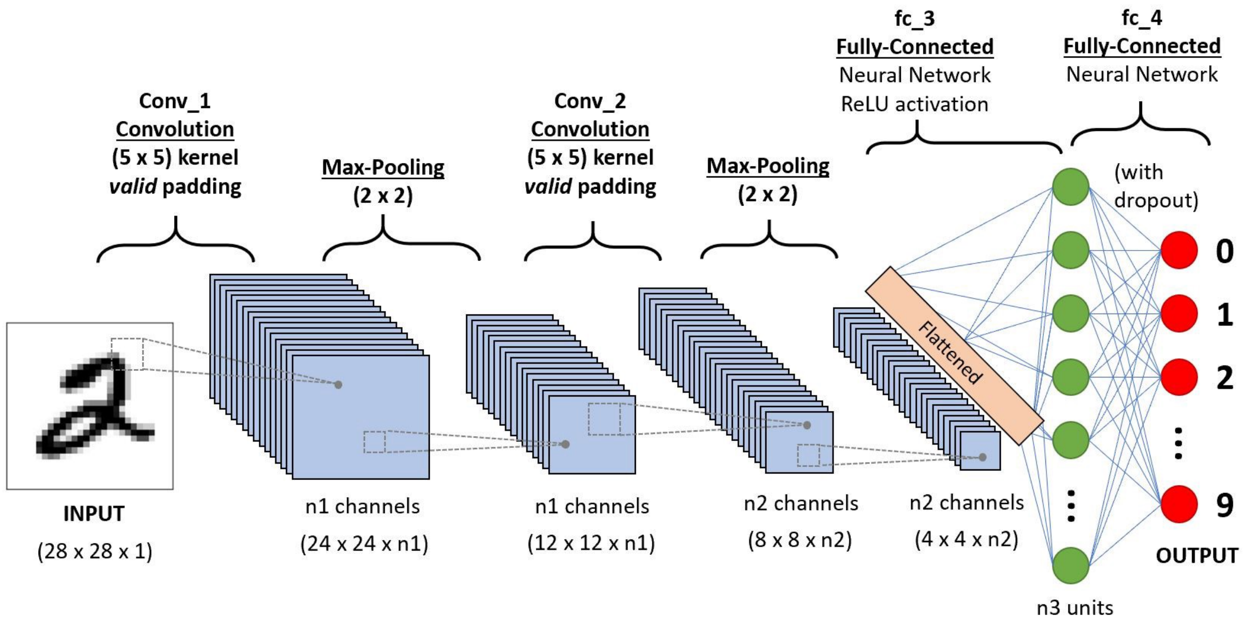


Figure 1.3: Convolutional Neural Network

When it comes to Image processing, CNN would be the best choice to go for it. The basic objective behind the existence of CNN can be attributed to Image classification i.e if a picture of a cat or dog is shown to these neural network, it should clearly identify & distinguish between the two categories, in a similar way a human would do. It would be interesting to see the biological connection between a CNN & Human's visual cortex. The visual cortex in humans is found to consist of small region of cells which are sensitive to specific regions of the visual field. Biological experiments found that specific set of neurons fires/activates when an input with particular edge,curve & orientation is shown. Also, all these neurons were arranged in a columnar architecture. This nature of specialized components within a particular system which is dedicated for a particular input formed basis for CNN.

Lets now investigate about different types of layers present in CNN & their respective roles in it. To begin with an image is feeded as input to the network (CNN also works under Text, NLP domain). One must notice that neural network looks at an image as a matrix of numbers & it would be an array of the form *width * height * depth* like 32*32*3. The depth is given as 3 as it points to RGB color channels of the image. The first layer following the input layer would be the **convolution layer** in CNN always, which forms the heart of this neural network. The convolution layer has a set of matrix of numbers called **Filters**, with width equal to height & depth must match the input image's depth. The main objective behind these filters is to extract a predefined edge/shape/boundary in the given image through a process called **convolving**. By convolving, element-wise multiplication is performed between the pixels in the input image & filter's pixels, post-which a summation is done with the result which finally produces a single number. The notion is if the image contains the expected edge/shape in this region/section of the image, then this number would be very high (similar to neurons firing in visual cortex) else zero otherwise. As a next step, filter is shifted/*strided* some units to the right. These units shifting can be 1 or any other number but higher the number, lesser would be the dimensions for the resulting matrix. When the filter covers the entire region of the image, the resulting matrix is called **activation maps**, the dimensions of which would be less than the original image. In cases in which the dimensions should be maintained, image borders can be padded with zeros which is called **zero-padding**.

The next layer in the sequence which immediately follows the convolution layer would be **ReLU** layer aka Rectilinear activations. The purpose of this layer is to set all the negative activations in the activation maps to zero & has a mathematical representation of **f(x) = max(0,x)**. It was also found that ReLU makes the traning of the network much faster than compared to other activations like *tanh, sigmoid*. This is the part in which a non-linearity is introduced in the network.

Following the ReLU layer comes the **pooling layer**, which down-samples the given input. It comes under different flavours like *MaxPooling, L2, Average pooling*, with **MaxPooling** being the most-frequently used one. The theme behind MaxPooling is to apply a window of specific size like 5X5 to the given input & pick the maximum value within the window, post-which striding happens. The result being the reduced size of the feeded input. It is based on the fact that only the relative positioning of different features in the image matters & not the absolute position.

Until this point, all the layers which are explained above forms the set of layers which are unique to convolutional neural network. This layer set is duplicated for a certain number of times in-order for the filters to extract much complex representations present in the image than the simple curves/shapes like in the initial layers. Next layers in the network would be the normal hidden layers as in the FeedForward Neural Network & the neurons in this layer will be learning the right set of weights to produce a particular output when a specific shape exists in the image. The final output will be set of probabilities which denotes the given input belongs to a particular class.

CNN has huge list of applications under it, most of which involves image-processing in various forms like object detection, image segmentation etc. Commercial applications of CNN includes *Facebook* which uses automatic photo-tagging, *Google* for photo-search, *Instagram* for search infrastructure.

While the strength of the CNN can be attributed to its success in computer vision tasks, its drawback would be it's inefficieny to process sequential input data.

1.5.3 Recurrent Neural Network (RNN) -

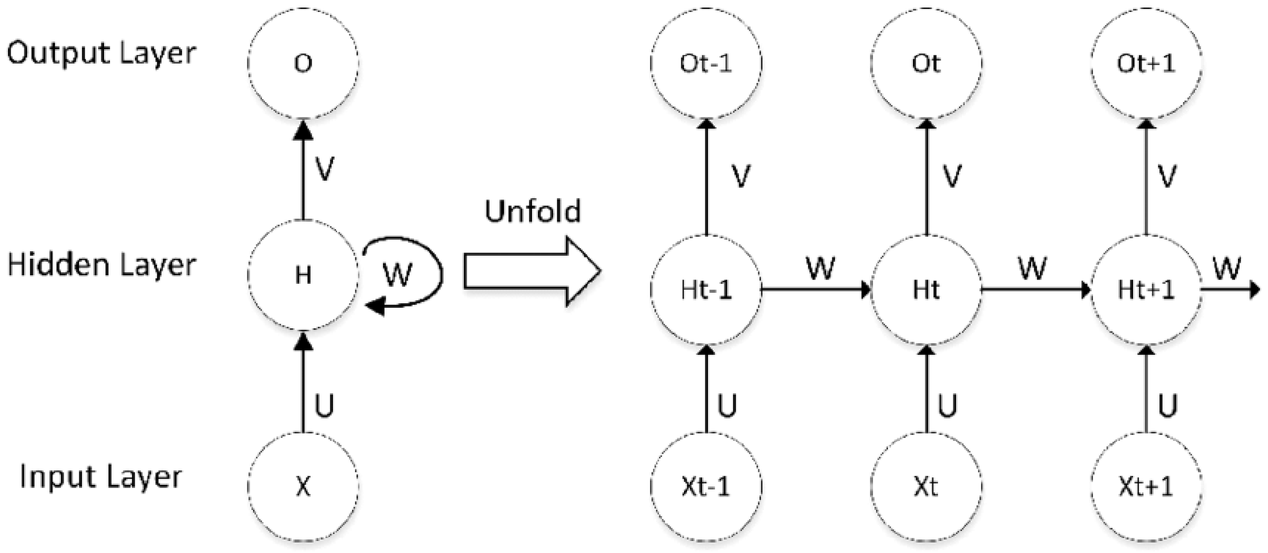


Figure 1.4: Recurrent Neural Network, unrolled over time

RNN solves the major drawback faced by FeedForward network & CNN namely sequential input-data processing. In some sense, RNN is just a flavour of FeedForward networks as it is equivalent to FeedForward Network + Loop as shown in the above figure. In case of FeedForward networks, the generation of a particular output depends only on the input on that specific time-step & is completely independent of all the input/outputs present in the previous layer. This is the notion where RNN stands apart from other neural networks & is capable of remembering all the previous outputs in the sequence using **sequential memory or Hidden state** which is considered before producing an output. It must be noted that the same weight is shared across all the hidden state in the RNN.

A major struggle faced by RNN is the problem of *vanishing/exploding gradient*. The reason behind this problem is the nature of Backpropogation algorithm, which makes a particular weight update to depend upon all the weight updates in the layers succeeding it, in which case if the weight update is a small value in a particular layer, then the layers preceding it will perform even more small weight update with the initial layer doing a weight update close to zero. This is known an **van-**

ishing gradient problem, which implies that RNN can't remember long sequences & is capable of only short-term memory. The vice-versa case of vanishing gradient problem is known as Exploding gradient problem in which the weight update will keep increasing as we navigate from output layer to the first hidden layer.

Coming to the applications of RNN, it is used in NLP tasks like Chatbots, Language Translation, speech recognition, stock prediction, Image captioning etc.

1.5.4 Long Short Term Memory (LSTM) -

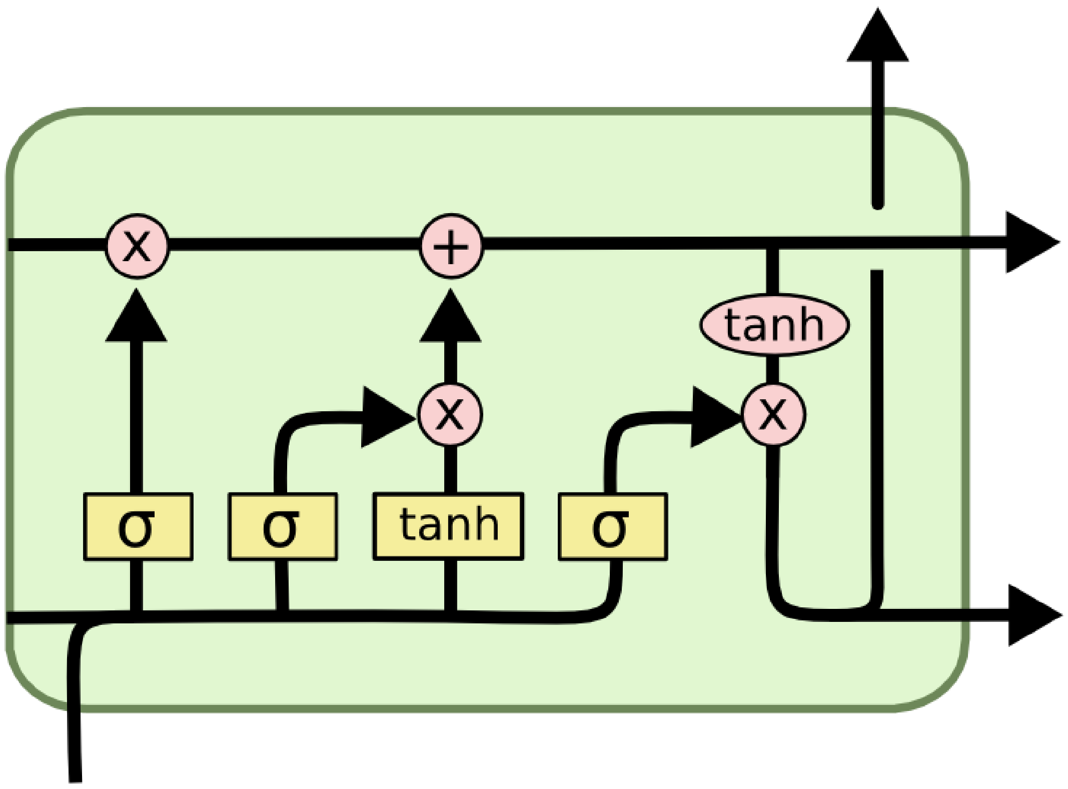


Figure 1.5: LSTM cell

LSTM is simply a RNN but with the cell state (hidden state) controlled in a bit more sophisticated way through *gates & operations*. As shown in the above figure, the first horizontal line from the top represents the cell state, which runs across the entire cell like a conveyor belt & its contents can be dynamically added or removed in a much controlled way by the use of gates.

Lets investigate more on the roles of each of these gates in the LSTM cell. one must note that the input at the present time-step is feeded across all the gates over here. Now, Starting from the left-most sigmoid activation function which is called **forget gate** as the sigmoid function will always outputs a value from 0-1 & when this output is multiplied by the cell's hidden state in the main line, it can actually remove/erase the cell's past memory to the extent of the multiplied number & hence the name. This is identical to the valve that controls the water-flow in a pipe.

The next process in the pipeline will be the cell state update which is done with two sets of gate, one with *sigmoid* function called **input layer gate** & the other with *tanh* function. The tanh layer creates a new candidate vectors while the sigmoid layer again acts as a valve in the pipe that controls the information flow, the result of which is added into the cell's main memory through a summation operation.

The final step in the process is the output generation. The final hidden state is basically the cell's main memory but with some contents filtered. This work can be attributed to the sigmoid & tanh layers present at the right-most side of the cell. To begin with, contents of the cell's main memory is feeded into the tanh layer which outputs the value in the range -1 to +1, which is again made to pass through a valve provided by sigmoid layer, as explained before. This performs the filteration & this filtered value forms the new hidden state which forms the input for the next LSTM cell.

Chapter 2

Literature Review

Until the year 2012, Machine learning models were evaluated only on clean inputs & Adversarial examples has no involvement in impacting the robustness of neural networks until Szegedy et al [1] investigated the existence of adversarial examples in machine learning models through experiments. He proved the fact that it was the High-dimensionality in the input data under a Linear model setting, visits unnaturally occuring points in space i.e adversarial examples and most of the Neural Networks spends the majority of their time in the linear-saturating region, thereby well-aligning with the linear model explanation. In short, this work proved that linearity is the cause here as opposed to the previous hypothesis which was in favour of non-linearity. It was also discovered that the generated adversarial examples can be transferred from one machine-learning model to a completely different one. The reason behind that can be attributed to the contiguous occurence in broad region of 1-D spaces when perturbation points to the right direction with a significant magnitude. It is surprising to know that Szegedy at al [1] also discovered a way to resist/defend adversarial examples through a method called "Adversarial Training", which provides a unique regularization property, which can't be found in traditional regularization techniques like Dropout etc. This way of resistance was strongly backed up by Universal approximator theorem which guarantees that any machine learning model can learn the resistance embedded objective functions if it has atleast one hidden layer and no upper-bound on the number of neurons in those layers.

An analysis about the input representation in the neural network was done by Ian J.Goodfellow et al [2], who came up with two important results from their work. The first one being the information about the semantic structure of the input is stored in activation space and the individual units aka neurons has no involvement in the storage. While the second result revolves around the fact that the hidden layers present in the neural networks is responsible for representing non-local generalization information. In simple terms, the output layer assigns some non-significant probabilities to those regions of the input which doesn't has any training examples in their vicinity. It is this region which represents the input from different perspective. Also, this work has discovered a way to traverse the manifold represented by the network to reach the adversarial examples which occur in High-dimensional space that are quite unnatural.

A much interesting work was done by Sameer Singh et al [3] with respect to Natural Adversarial examples generation. In contrary to the previous works which mostly concerned about adversarial examples generation under white-box setting, this work focuses on developing a framework which produces semantically similar adversarial examples through a special type of Neural Network known as Generative-Adversarial Network (GAN) and feed this natural adversarial samples to any Black-box classifier. It is important to note that the proposed framework is capable of producing adversarial samples for both Image as well as Text domain, which makes this to stand apart from other existing approaches. The main theme behind this approach is instead of modifying the given input sample be it an image or a text, it produces a dense-vector representation of this input through a pre-trained Inverter, which forms one part of the GAN. Once the Dense vector is obtained, then the modifications are applied on these original dense-vector to get perturbed dense-vector which are then fed into pre-trained generator which finally produces set of natural adversarial samples. A notable observation over here is, it is quite easy to generate adversarial samples for images as they are represented in continuous value range which isn’t the same case when it comes to text as they have discrete values. All the previous works, exploited strong NLP-oriented strategies and manual intervention to develop adversarial samples and that too it wasn’t semantically similar. But in this approach, a special technique known as Textual Entailment was used which uses a pair of sentences called ’preamble’ and ’hypothesis’ and modify the ’hypothesis’ sentence to produce natural adversarial text. The adversaries generated were able to attack ’Google Translate’ and sounded perfectly well in both semantic and logical aspects.

Szegedy et al [1] came up with the very first method to generate adversarial example by exploiting the gradients of the underlying model which was known as FGSM. Although it was the first method, the attack was quite rapid & powerful which eventually paved the way for other types of attacks. Goodfellow et al [7] proposed an Iterative version of FGSM, IFGSM in which all the operations of FGSM was done in an iterative way with the intermittent values clipped within some range & similarly, Papernot et al’s [11] work adds one more step of randomizing the initial starting points before starting the iteration in FGSM, inorder to escape getting struck in the local minimum. Madry et al [8] did bit different by clipping the noise level produced by the iterative version within a particular range i.e always maintain the noise within a circle of particular radius & adds this controlled noise to the original input without the input values exceeding the permissible range. This method was known as PGD, which emerged as one of the most powerful attack with Randomized-PGD & Averaged-PGD as its flavours to escape the local optimum & attack the Bayesian Neural Networks. Carlini et al [9] came up with the most powerful attack ever devised so far which is known as C&W attack that considers both the intensity of perturbation level & misclassification rate by encoding both of them in the objective function to be optimized. C&W attack rose to the power in such a way that it is used as the Benchmark to evaluate the robustness of neural network or the resisting capability of a particular defense strategy.

The first practical demonstration of the Black-box attacks in the real-world setting was performed by Papernot et al [4] in a unique approach which doesn't require any knowledge about the target Neural Network's weights and hyperparameters as well as avoids the access to large training dataset. Theme was that the target Neural Network was hosted at a third-party API service and adversary has zero knowledge about the model type, architecture and its hyperparameters while the only access the adversary has is to query the model by feeding an input and get the predicted output label (not the probability) by the model. And there is a restriction to the number of queries made to the target model to view the output because if it is permissible to query infinite times, then adversary can completely learn about the target model. The proposed solution under this setting was to construct a substitute model compatible with the problem domain say Convolution Neural Network for computer vision domain, with random architecture and hyperparameter and train the model in a normal manner but with a *synthetic dataset*. The Synthetic dataset is formed by taking very few samples from the test-set of the given problem say images representing 0-9 in the MNIST dataset but with the labels omitted. In order to get the labels, the raw input samples are fed into the target model & the predicted output is assigned as the label for that particular sample. Then, the model is trained with this dataset and post-which *synthetic generation* happens in which Jacobian derivatives of each samples in the present dataset are calculated with respect to the labels and added to the corresponding samples after the Jacobian is multiplied with a hyperparameter λ and the new sample is appended to the original dataset. In this way the substitute model is able to approximate the target model's decision boundaries. Then, the adversarial examples are generated using traditional approaches say FGSM with the substitute model and fed into the target model. Due to the property of Transferability, the target model misclassifies the sample as well. By tuning the hyperparameter λ, the rate of Transferability was increased.

There was a speculation within the machine-learning community that adversarial attacks are applicable only in theory but it is highly uncertain when it comes to practical real-world attacks. This was broken with the work done by

Stephan et al [6] by demonstrating an attack on Real Facial-Recognition system using the notion of adversarial patches. Unlike the previous works which strongly demanded the need for the creation of complex-shaped objects and specific lighting conditions, this strategy can be universally applied across any model without any strong restrictions. The demonstration was done on a Facial-Recognition system called ArcFace which uses state-of-the-art Face ID model called LResNet100E-IR with the novel strategy called AdvHat. The main objective behind the AdvHat technique is to create a rectangular image using a novel algorithm and finally place this generated image on a hat which is to be weared by a person who wants to dodge the system and hence the name AdvHat. The proposed pipeline behind AdvHat is to create a rectangular image using a novel off-plane transformation as the image has to go through a series of transformations and project this generated image on a high-quality image of a face with some small perturbations in the projection parameters to make the patch more robust. Following this, the obtained patch is transformed to fit into the ArcFace format which can be feeded in as input. As a final transformation of the pipeline, a sum is computed between two parameters namely TV loss & cosine similarity between embeddings and the patch is tweaked to reduce this sum. It was also observed that maximum intensity of the attack was witnessed when the patch was pasted at the bottom of the hat and this attack falls into Dodge category and not impersonation.

Another similar work which proved the practicality of adversarial examples was done by Alexey et al[7] who proved the practical possibility of these attacks. They have done series of experiments in which inputs to the neural networks are provided by some physical mediums like camera, sensor etc. and explored the adversarial attack behaviours through it. Surprisingly, it worked for both White-box & Black-Box attacks in the physical world. one such notable work was using Imagenet Inception classifier but simply feed the adversarial-inputs via cellphone-camera as opposed to previous style of images. Most of the adversarial samples got misclassified & the accuracy is nearly same as that of before. By this, lower-bound classification accuracy was established for white/box attacks. Regarding the Black-box attack, a pair of real & adversarial images were printed adjacently in a piece of paper pointed to a object recognition software through cell-phone camera. Again, the adversarial samples got misclassified. It was also observed that adversarial samples generated via Fast Gradient Sign Method is more robust to series of transformations like cropping, lighting conditions etc. to name a few.

With the rise in popularity of these attacks, the exploration of defense strategies got widened & *papernot et al*[4] came up with the quite successful defense method known as *Adversarial Training*, a unique regularization technique which possess some capabilities that can't be seen in conventional L_0 & L_2 regularization. The idea behind this approach is to inject the adversarial examples during the routine training of the neural networks, so that model gets aware of the false samples & becomes cautious accordingly. It should be noted that the model's very own parameters are used to generate adversarial samples. *Alexey et al*[11] modified this approach a bit & proposed that rather than using the model's own parameters to generate adversarial samples, use a substitute model to generate these samples which are then used in the training process. In this way, model gets more robust towards Black-Box attacks.

The state-of-the-art defense was developed by Xi Wu et al[5] using a variant of Distillation. The *Distillation* technique actually refers to transferring the knowledge learned by big neural networks in the form of probability vectors to another neural network which is relatively of small size without any loss in accuracy, so that the small-sized neural network can be implemented on resource-constrained devices like smartphones. The variant proposed in this work was to transfer the output probability vectors learnt to another network which is an exact replica of the original one and train the second network with the feeded probability vectors as the *soft-labels* rather than the hard-labels used in the original network. A special hyperparameter called 'Distillation Temperature' was used at the neurons present in the final-softmax layer of the network, which proved to be a core component behind this technique.

Chapter 3

Basic Theory

3.1 Adversarial Examples - What is it?

Szegedy et al. [1] first noticed the existence of adversarial examples in the image classification domain, when he observed that on applying an imperceptible non-random perturbation to a test image, it was possible to change the network's prediction arbitrarily. These perturbations were found by optimizing the input to maximize the prediction error and were termed as Adversarial Examples which can be defined as "Instances with small, intentional feature perturbations to fool the Machine-Learning model". To put it in simple words,

Adversarial Input = Original Input **+ carefully calculated noise**

Ideally the Noise must be imperceptible to Human observers/listeners, meaning if a Human looks at an adversarial image or listens to adversarially modified audio file, he must still believe that it is normal input & there is no tampering in it. Also, this attacks can be applied in any domains like *computer vision, NLP* where there is a room for Deep Neural Networks.

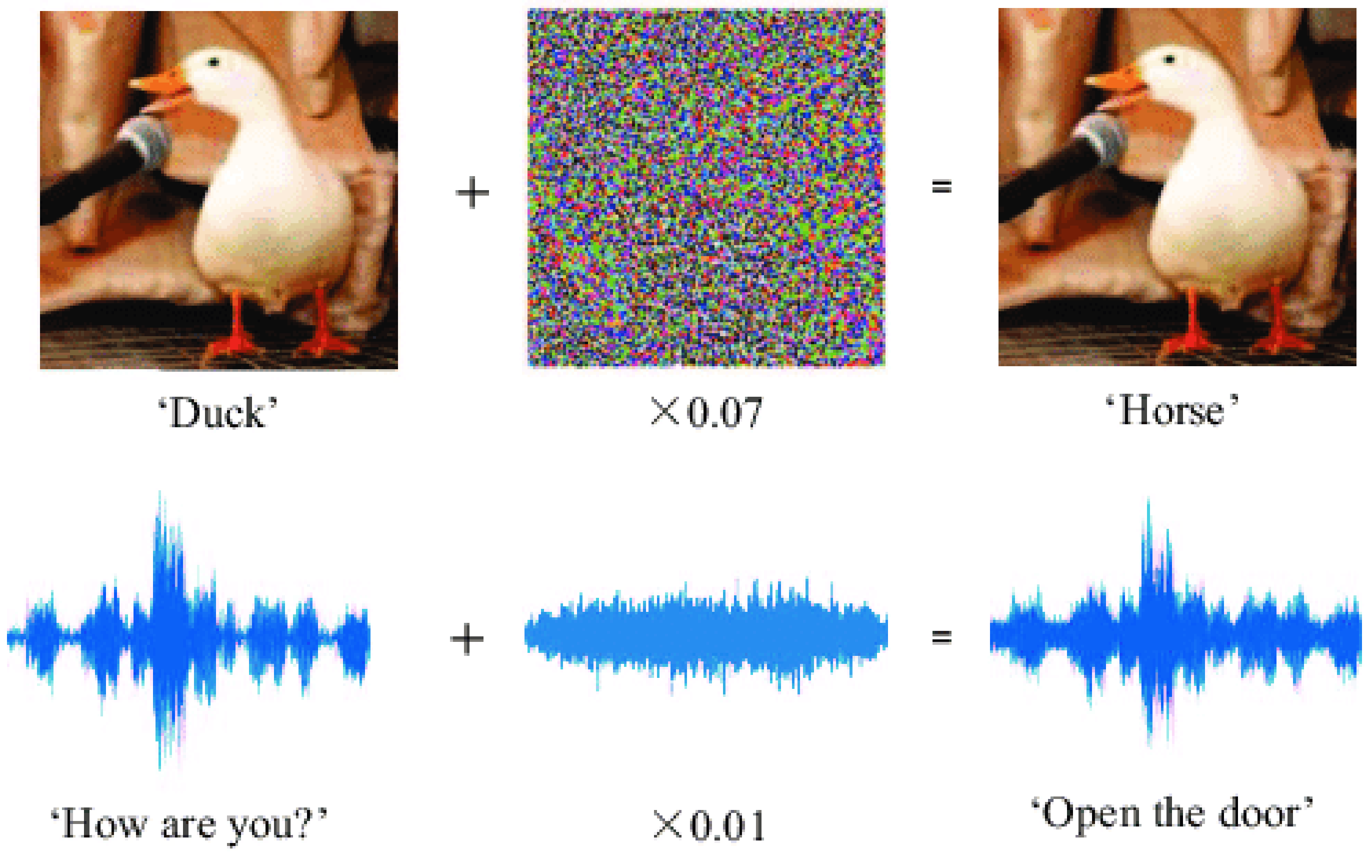


Figure 3.1: Adversarial Examples in **computer-vision & speech-recognition** domains

South Africa's historic Soweto township marks its 100th birthday on Tuesday in a mood of optimism. 57% **World**
South Africa's historic Soweto township marks its 100th birthday on Tuesday in a moo**P** of optimism. 95% **Sci/Tech**
Chancellor Gordon Brown has sought to quell speculation over who should run the Labour Party and turned the attack on the opposition Conservatives. 75% **World**
Chancellor Gordon Brown has sought to quell speculation over who should run the Labour Party and turned the attack on the o**B**position Conservatives. 94% **Business**

Figure 3.2: Adversarial Examples in **NLP** domain

3.2 Why to worry about Adversarial Examples?

Recent breakthroughs in Computer Vision have led to near human level performance in various tasks like Image Classification and Object Detection. There is also breakthrough in speech-recognition & NLP areas like *Google Translate*. These advances also call for widespread adoption of such methods for various problems by individuals and industry because of their accurate results. But this also means that such systems would naturally be applied in security-critical scenarios. It turns out that these "highly accurate classifiers" show an inherent weakness known as adversarial examples.

Let’s take an example to illustrate the importance of defenses required against adversarial attacks. State-of-the-art neural networks are extensively used in autonomous vehicles for Computer Vision tasks. Generally some decision regarding the path, speed, etc is taken after doing some computation based on the images taken by the camera. Say our car encounters a signal which says to stop. On an unperturbed input, our car would stop but if our input is perturbed (imperceptible but non-random perturbation), our network’s prediction would be of some arbitrary class, say ’Go Right’. As you can imagine all autonomous cars on the road would behave in a similar manner, and it would lead to chaos.

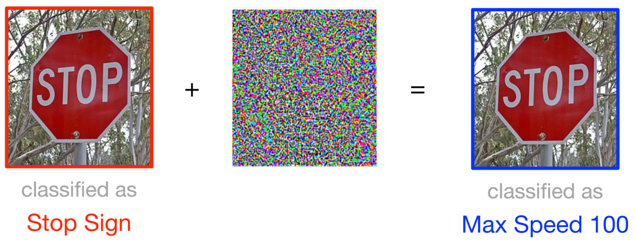


Figure 3.3: Fooling of Autonomous cars - Scenario 1

In the autonomous cars, another scenario could be lets say some pedestrains are crossing the road. The trained neural network model could correctly recognizes the pedestrains in the image but with the adversarial attack, the neural network can be fooled to get an impression like no one is on the road & the car can move forward freely which in reality would cause a disaster.

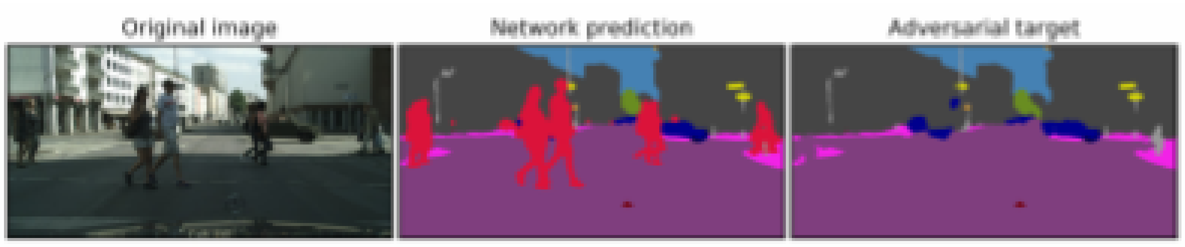


Figure 3.4: Fooling of Autonomous cars - Scenario 2

Another case of these attacks can be seen in **Face-Recognition** domain. In Today's world, Face-based authorization system has carved out a unique place & it's application ranges from simple work-attendance to secured Bank vaults, military. But hackers successfully attacked the neural networks behind these systems, forcing the network to predict their own identity as some specific person who has authorized access. Here, two things are possible, either *prevent* the network from making the correct prediction which is called **dodging** or *impersonate* a particular person.

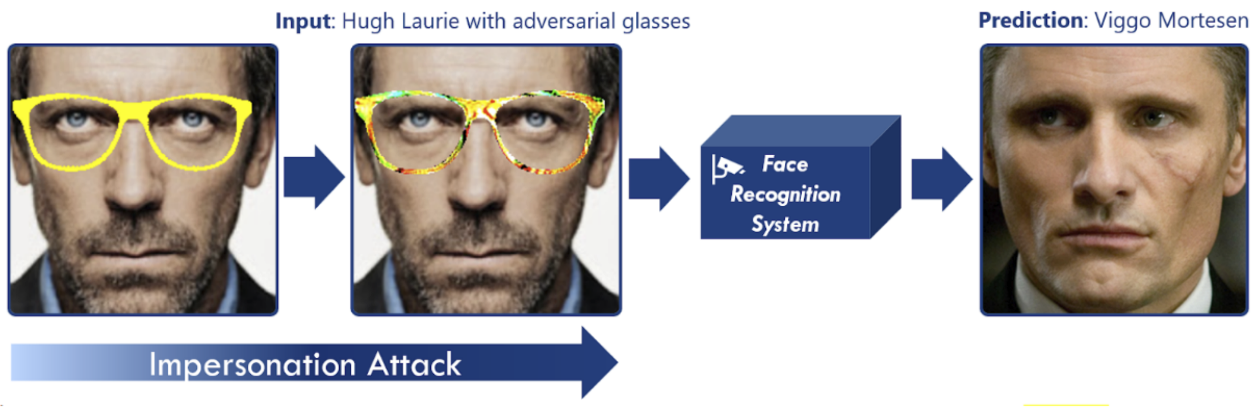

Input: Hugh Laurie with adversarial glasses
Prediction: Viggo Mortesen
Face Recognition System
Impersonation Attack

Figure 3.5: Facial Impersonation Attack

Another example would be of a malicious file or malware. By changing some bits of the file which are benign, our systems might predict that it is a benign file (though it did not alter the malicious part of the file, so it still should be classified as malware) and would not block it, therefore posing a security risk to the computer.

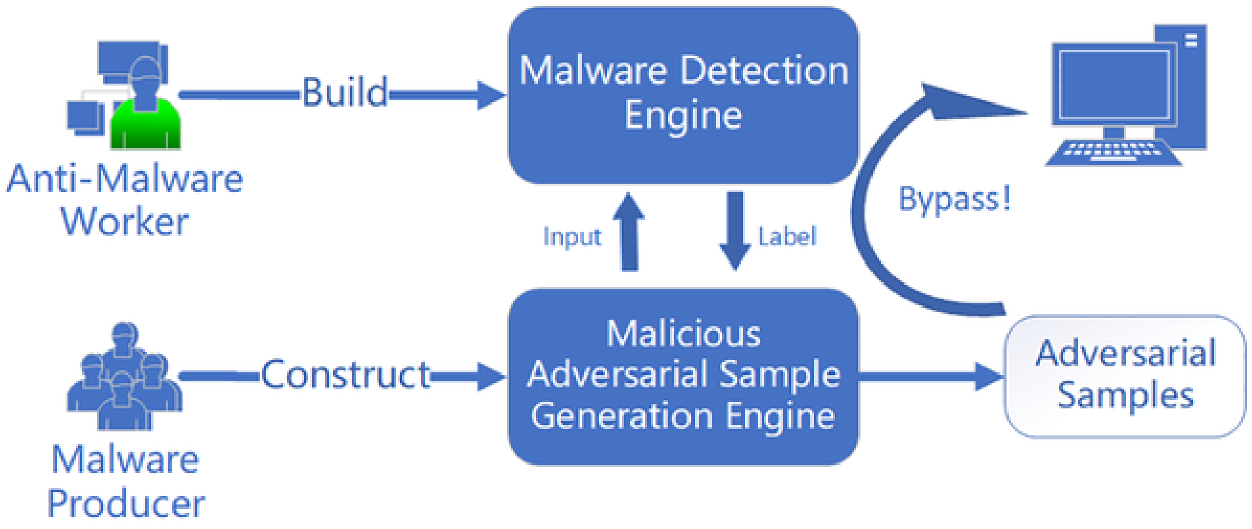


Figure 3.6: Malware Bypassing Attack

Hence, it became the fact that all the machine models including the state-of-the art neural networks are vulnerable to adversarial attacks. The misclassification happens even when the same adversarial example is fed into different neural networks with completely different architectures and trained with different subsets of data. This raised a serious concern among the machine learning scientists and community as its applications involves safety-critical environment and this technically restricts the application of machine-learning in many areas despite of its huge success.

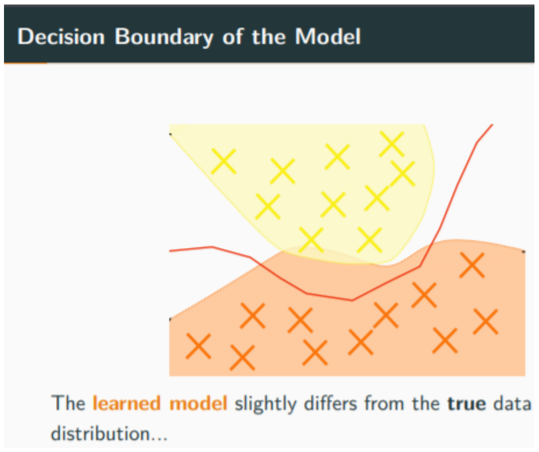
3.3 Where do Adversarial examples come from?

Machine learning algorithms are usually designed under the assumption that models are trained on samples drawn from a distribution that is representative of test samples for which we would later make predictions. However, this is not true in the case of adversarial examples. Let us define a Machine Learning Classifier and we will then try to understand the statistical properties of adversarial examples through the eyes of the classifier.

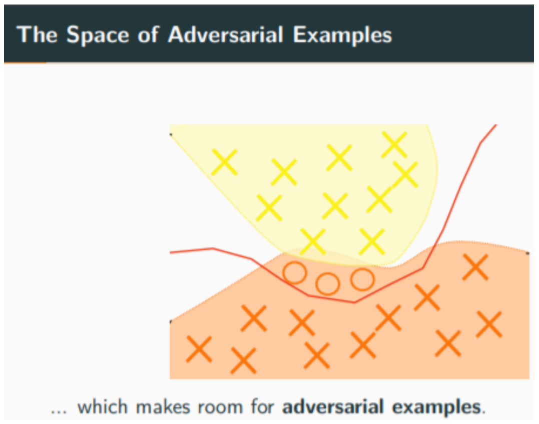
An input $x \varepsilon X$ with n features and $y \varepsilon Y$ be the label for that input. A classifier then tries to learn a mapping $f: x \rightarrow y$ such that the function tries to minimize the empirical risk. The classifier outputs probabilities and then the label corresponding to the largest probability is chosen as the prediction of the model. There is an unknown distribution $D^{C_i}{}_{real}$ for each class C_i and the training data X is obtained from this distribution, and the classifier tries to approximate this distribution during training, thereby learning $D^{C_i}{}_{train}$.

A notable result in ML is that any stable learning algorithms will learn the real distribution $D^{C_i}{}_{real}$ up to any multiplicative factor given a sufficient number of training examples drawn from $D^{C_i}{}_{real}$. But complete generalization is not possible due to finite number of training samples.

The existence of adversarial examples is a manifestation of the difference between the real distribution $D^{C_i}{}_{real}$ and the learned training distribution $D^{C_i}{}_{train}$, i.e the adversary aims to find a sample from $D^{C_i}{}_{real}$ whose behavior is not captured by the learned distribution $D^{C_i}{}_{train}$. But the adversary does not know the real distribution (if we would, then no need to learn anything) so it takes a sample from $D^{C_i}{}_{train}$ and tries to perturb it to craft adversarial examples. Each such example made would belong to $D^{C_i}{}_{adv}$ for the class. Since the perturbations applied are tiny in nature, we know that $D^{C_i}{}_{adv}$ is consistent with $D^{C_i}{}_{real}$ (since a sample in $D^{C_i}{}_{adv}$ would also be in $D^{C_i}{}_{real}$) , but $D^{C_i}{}_{adv}$ differs from $D^{C_i}{}_{train}$.


Decision Boundary of the Model
The learned model slightly differs from the true data distribution...

(a) Non-aligning Decision Boundary


The Space of Adversarial Examples
... which makes room for adversarial examples.

(b) Room for adversarial example

Figure 3.7: It is the imperfect fit of the learned Decision boundary by the machine-learning model which creates a room for adversarial examples & by the adversarial attack, we are trying to shift the correct samples towards this region

3.4 Linear Explanation of Adversarial Examples

Adversarial examples not only exist in big neural networks but also for shallow linear models. We explain the existence of such examples in linear models in this section.

Digital images are capable of storing 8 bits of information per pixel (i.e from 0-255) and changes below the order of 2^{-8} are not recognized by a computer let alone human vision. So maximum change per pixel should be $||\eta||_\infty < \varepsilon$ where ε is small enough to be discarded by the sensor. For linear models, we calculate the inner product between the weights of the network w and the adversarial example $\overline{x}$, as :

$$w^T\overline{x} = w^Tx + w^T\eta \tag{3.1}$$

where x is the unperturbed image

This equation tells us that the activation increases by w^T . Our goal is to increase this activation as much as possible, while having capped the maximum change per pixel. This can be achieved by assigning η = sign(w). To analyze this behaviour, we assume w is of n dimensions and average value of elements of w is m. This implies that our activation grows by mn on average, which tells us that activation can grow linearly with the amount of perturbation added unlike $||\eta||_\infty$. So for large images(or high dimensional data), we could make very small changes to pixel which would add up to something substantial to increase our activation values.

This explanation shows that a simple linear model can have adversarial examples if its input has sufficient dimensionality.

3.5 Why do Adversarial examples Generalize? - (Transferability)

An Intriguing aspect of adversarial examples is that an example generated for one model is often misclassified by other models, even when they have different architectures or were trained on disjoint training sets. Moreover, when these different models misclassify an adversarial example, they often agree with each other on its class.

Under the linear view, adversarial examples occur in broad subspaces. one must note that adversarial example generation methods depends only on the sign of the gradient & not on its magnitude. This means that as long as signs are matching with noise level bit significant, one can observe the adversarial examples in contiguous regions of the 1-D subspace as defined by the fast gradient sign method, & not in fine pockets. This explains why adversarial examples are abundant and why an example misclassified by one classifier has a fair high probability of being misclassified by another classifier.

To explain why multiple classifiers assign the same class to adversarial examples, lets claim a hypothesize that neural networks trained with current methodologies learned on the same training set. This reference classifier is able to learn approximately the same classification weights when trained on different subsets of the training set, simply because machine learning algorithms are able to generalize. The stability of the underlying classification weights in turn results in the stability of adversarial examples.

3.6 Taxonomy of Adversarial Examples

To systematically analyze approaches for generating adversarial examples, we analyze and categorize them along three dimensions: threat model, perturbation, and benchmark. Note the *Yuan et al*[13] is used as a reference for this section.

3.6.1 Threat Model

- The adversaries can attack only at the testing/deploying stage. They can tamper the input data only in the testing stage after the victim deep learning model got trained. Neither the trained model or the training dataset can be modified. The adversaries may have knowledge of trained models (architectures and parameters) but are **not allowed to modify models**, which is a common assumption for many online machine learning services. Attacking at the training stage (e.g., training data poisoning) is not performed here.

- We focus on attacks against models built with deep neural networks, due to the great performance achieved. Adversarial examples against deep neural networks proved effective than the conventional machine learning models aka (linear-models).
- Adversaries only aim at compromising integrity. Integrity is presented by performance metrics (e.g., accuracy, F1 score, AUC), which is essential to a deep learning model. Although other security issues pertaining to confidentiality and privacy have been drawn attention in deep learning, we focus on the attacks that degrade the performance of deep learning models, which cause an increase of false positives and false negatives.

Based on different scenarios, assumptions, and quality requirements, adversaries decide the attributes they need in adversarial examples and then deploy specific attack approaches. We further decompose the threat model into four aspects: adversarial falsification, adversary's knowledge, adversarial specificity, and attack frequency. For example, if an adversarial example is required to be generated in real-time, adversaries should choose a onetime attack instead of an iterative attack, in order to complete the task.

- Adversarial Falsification
  - **False positive** attacks generate a negative sample which is misclassified as a positive one (Type I Error). In a malware detection task, a benign software being classified as malware is a false positive. In an image classification task, a false positive can be an adversarial image unrecognizable to human, while deep neural networks predict it to a class with a high confidence score.
  - **False negative** attacks generate a positive sample which is misclassified as a negative one (Type II Error). In a malware detection task, a false negative can be the condition that a malware (usually considered as positive) cannot be identified by the trained model. False negative attack is also called machine learning evasion. This error is shown in most adversarial images,

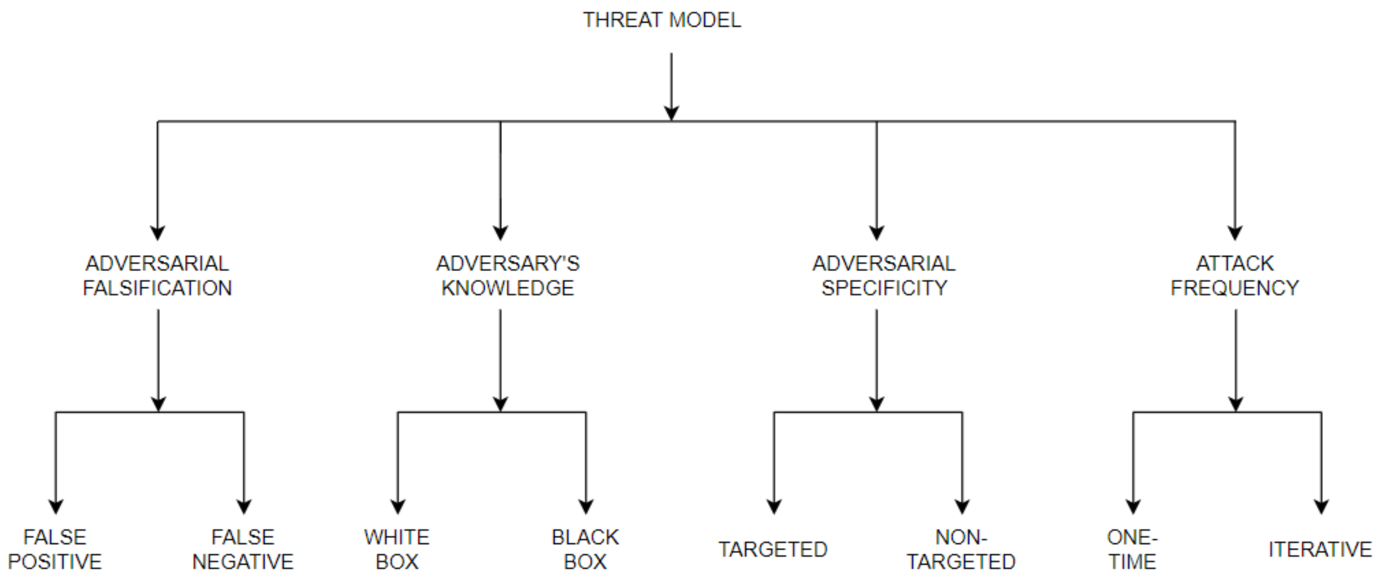


Figure 3.8: Threat Model Decomposition

where human can recognize the image, but the neural networks cannot identify it.

- Adversary's Knowledge
  - **White-box** attacks assume that the adversary knows everything related to trained neural network models, including training data, model architectures, hyper-parameters, numbers of layers, activation functions, model weights. Many adversarial examples are generated by calculating model gradients.Since deep neural networks tend to require only raw input data without handcrafted features and to deploy end-to-end structure, feature selection is not necessary compared to adversarial examples in machine learning.
  - **Black-box** attacks assume the adversary has no access to the trained neural network model. The adversary, acting as a standard user, only knows the output of the model (label or confidence score). This assumption is common for attacking online Machine Learning services (e.g., Machine Learning on AWS,Google Cloud AI, Microsoft Azure, Face++). Most adversarial example attacks are white-box attacks. However, they can be transferred to attack black-box services due to the **transferability** of adversarial examples.

- Adversarial Specificity
  - **Targeted attacks** misguide deep neural networks to a specific class. Targeted attacks usually occur in the multiclass classification problem. For example, an adversary fools an image classifier to predict all adversarial examples as one class. In a face recognition/biometric system, an adversary tries to disguise a face as an authorized user (Impersonation). Targeted attacks usually maximize the probability of targeted adversarial class.
  - **Non-targeted attacks** do not assign a specific class to the neural network output. The adversarial class of output can be arbitrary except the original one. For example, an adversary makes his/her face misidentified as an arbitrary face in face recognition system to evade detection (dodging). Non-targeted attacks are easier to implement compared to targeted attacks since it has more options and space to redirect the output. Non-targeted adversarial examples are usually generated in two ways: 1) running several targeted attacks and taking the one with the smallest perturbation from the results; 2) minimizing the probability of the correct class. Some generation approaches (e.g., extended BIM, ZOO) can be applied to both targeted and non-targeted attacks. For binary classification, targeted attacks are equivalent to non-targeted attacks.
- Attack Frequency
  - **One-time attacks** take only one time to optimize the adversarial examples.
  - **Iterative attacks** take multiple times to update the adversarial examples. Compared with one-time attacks, iterative attacks usually generate better adversarial examples, but require more interactions with victim classifier (more queries) and cost more computational time to generate them. For some computational-intensive tasks (e.g., reinforcement learning), one-time attacking may be the only feasible choice.

3.6.2 Perturbation

Small perturbation is a fundamental premise for adversarial examples. Adversarial examples are designed to be close to the original samples and imperceptible to a human, which causes the performance degradation of deep learning models compared to that of a human. We analyze three aspects of perturbation: perturbation scope, perturbation limitation, and perturbation measurement.

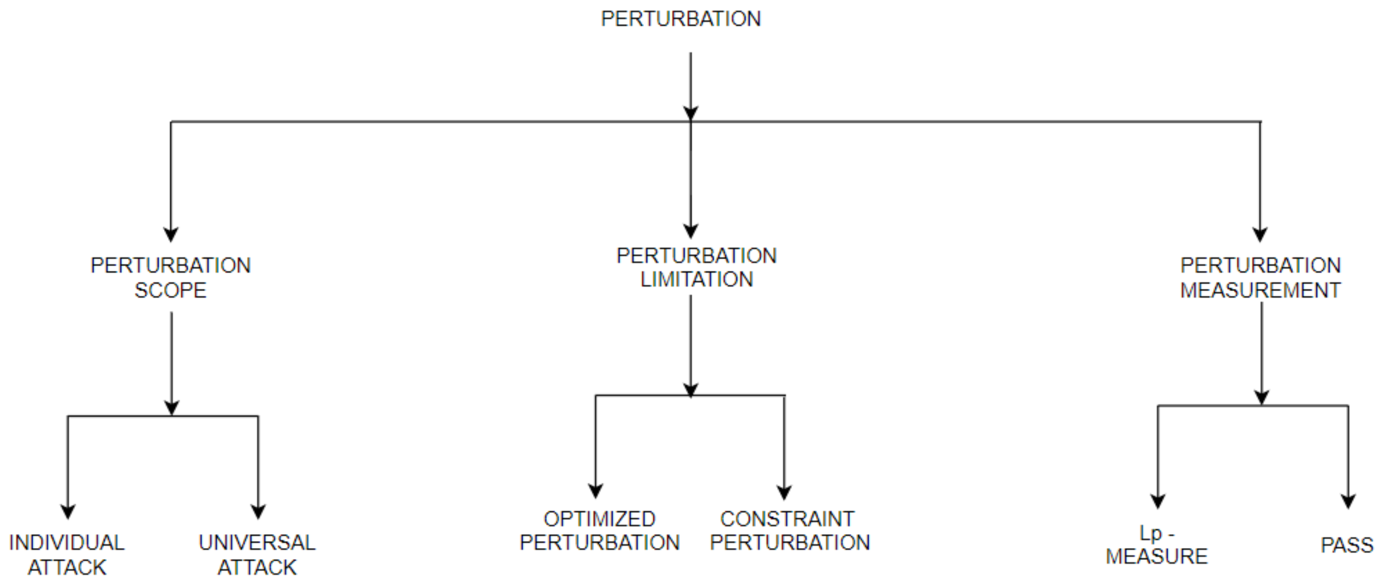


Figure 3.9: Perturbation Classification

- Perturbation Scope
  - **Individual attacks** generate different perturbations for each clean input.
  - **Universal attacks** creates a universal perturbation for the whole dataset. This perturbation can be applied to all clean input data. Most of the current attacks generate adversarial examples individually. However, universal perturbations make it easier to deploy adversary examples in the real world. Adversaries do not require to change the perturbation when the input sample changes.
- Perturbation Limitation

  - **Optimized Perturbation** sets perturbation as the goal of the optimization problem. These methods aim to minimize the perturbation so that humans cannot recognize the perturbation.
  - **Constraint Perturbation** sets perturbation as the constraint of the optimization problem. These methods only require the perturbation to be small enough.

- Perturbation Measurement

  - l_p (L_0, L_2, L_∞) measures the magnitude of perturbation by p-norm distance:

$$||x||_p = (\sum_{i=1}^{n} ||x_i||^p)^{1/p} \tag{3.2}$$

  - **L_0 perturbation** - Restricts the number of pixels to be perturbed. This ultimately leads to Larger-Amplitude variations in individual pixels.
  - **L_2 perturbation** - Root Mean Square (RMS) between the original input the perturbed image. This takes into account both the *number of pixels* to be modified as well as the *maximum change* per pixel.
  - **L_∞ perturbation** - Puts a cap on the maximum limit a given pixel can be perturbed.
  - Psychometric perceptual adversarial similarity score (PASS) is a new metric introduced which is consistent with human perception. But this is seldom followed in any type of attack.

3.6.3 Benchmark

Adversaries show the performance of their adversarial attacks based on different datasets and victim models. This inconsistency brings obstacles to evaluate the adversarial attacks and measure the robustness of deep learning models. Large and high-quality datasets, complex and high-performance deep learning models usually make adversaries/defenders hard to attack/defend. The diversity of datasets and victim models also makes researchers hard to tell whether the existence of adversarial examples is due to datasets or models.

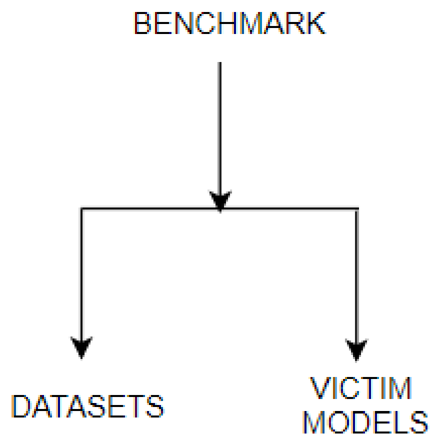


Figure 3.10: Benchmark

- **Datasets** - MNIST, CIFAR-10, and ImageNet are three most widely used image classification datasets to evaluate adversarial attacks. Because MNIST and CIFAR-10 are proved easy to attack and defend due to its simplicity and small size, ImageNet is the best dataset to evaluate adversarial attacks so far. A well-designed dataset is required to evaluate adversarial attacks.

- **Victim Models** - Adversaries usually attack several well-known deep learning models, such as LeNet, InceptionV3, VGG, AlexNet, GoogLeNet, CaffeNet, and ResNet.

3.7 Adversarial Attacks on Neural Networks

This section explores about different types of popular adversarial attacks on Neural Networks, both white-box & black-box attacks, targeted & non-targeted, with the key-properties, strength & weakness alongside.

3.7.1 Fast Gradient Sign Method - FGSM

Goodfellow et al. [1] proposed a fast method called Fast Gradient Sign Method to generate adversarial examples. They only performed one step gradient update along the direction of the sign of gradient at each pixel. Let θ be the parameters of the model, x is the input image, y is the true label for x, and J(θ,x,y) is the loss function for the network. We now linearize the loss function at the current value of θ, then their perturbation (Noise) can be expressed as:

$$\eta = \epsilon sign(\nabla_x J_\theta(x,l)) \tag{3.3}$$

where ε is the magnitude of the perturbation. The generated adversarial example x^l is calculated as: $x^l = x + \eta$. This perturbation can be computed by using back-propagation. One thing to notice is that we assume that parameters of the model is fixed and compute the gradient with respect to the input, thereby getting a matrix of the same size as that of the input. This cheap method is able to get high levels of mis-classifications on datasets like MNIST, CIFAR-10. Figure 3.11 shows an adversarial example on ImageNet.

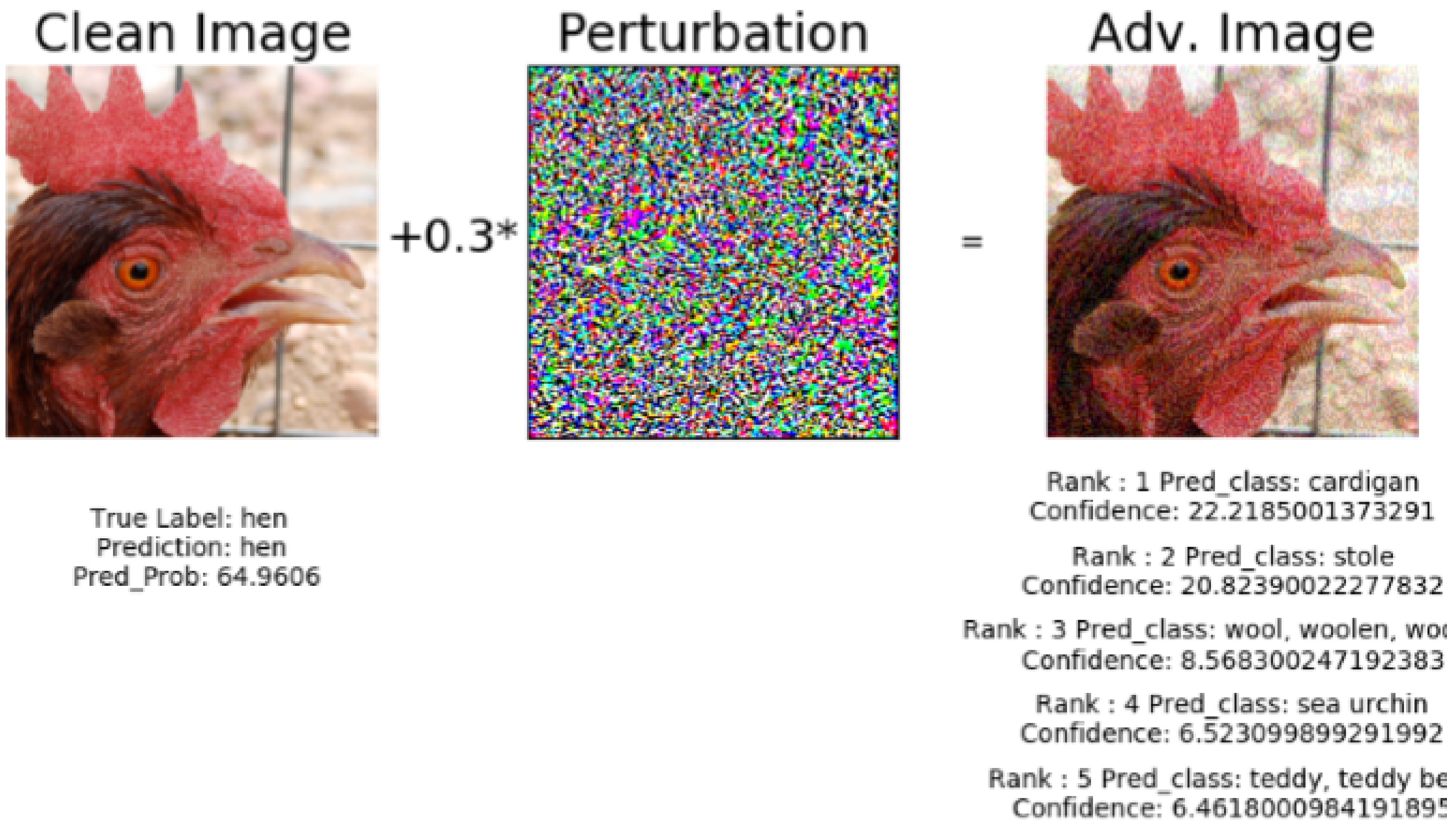

Clean Image
+0.3*
Perturbation
=
Adv. Image
True Label: hen
Prediction: hen
Pred_Prob: 64.9606
Rank : 1 Pred_class: cardigan
Confidence: 22.2185001373291
Rank : 2 Pred_class: stole
Confidence: 20.82390022277832
Rank : 3 Pred_class: wool, woolen, wo
Confidence: 8.568300247192383
Rank : 4 Pred_class: sea urchin
Confidence: 6.523099899291992
Rank : 5 Pred_class: teddy, teddy be
Confidence: 6.461800098419189

Figure 3.11: FGSM performed on an image of hen as predicted by the unperturbed model. After adding some perturbation, we can see that the image is still a hen but our model does not predict it as hen even in its top five predictions. Here our ε of 0.3 corresponds to the magnitude of the perturbation.

Key-Properties :

- Non-Targeted
- one-step attack
- L_∞ perturbation

Strength : Faster generation of adversarial images.

Weakness : Although it can fool some models, It is not powerful & fails to fool many models.

3.7.2 BIM - Basic Iterative Method aka IFGSM

The Basic Iterative Method is simply a flavour of Fast Gradient Sign Method. Goodfellow et al [7] observed that by taking multiple smaller steps of size α instead of one step of ε as in FGSM, one can obtain finely grinded pixels in the output. It is basically an extension of Fast Gradient Sign method by running a finer optimization (smaller change) for multiple iterations. In each iteration, pixel values are clipped to avoid large change on each pixel:

$$X_0^{adv} = X, \quad X_{N+1}^{adv} = Clip_{X,\epsilon}(X_N^{adv} + \alpha sign(\nabla_X J(X_N^{adv}, y_{true}))) \tag{3.4}$$

The reason behind the clipping at every iteration is to ensure that they are in ε neighbourhood of the original image. The number of iterations are taken to be
min(ε + 4, 1.25 ε). This was chosen heuristically ; it is sufficient for the adversarial example to reach the edge of the max-norm ball but restricted enough to keep the computational cost of the experiments manageable.

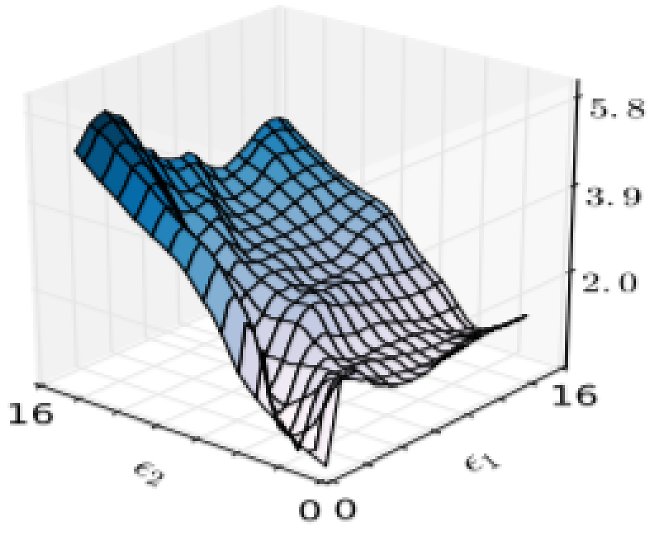
Key-Properties :

- Non-Targeted
- Iterative attack
- L_∞ perturbation

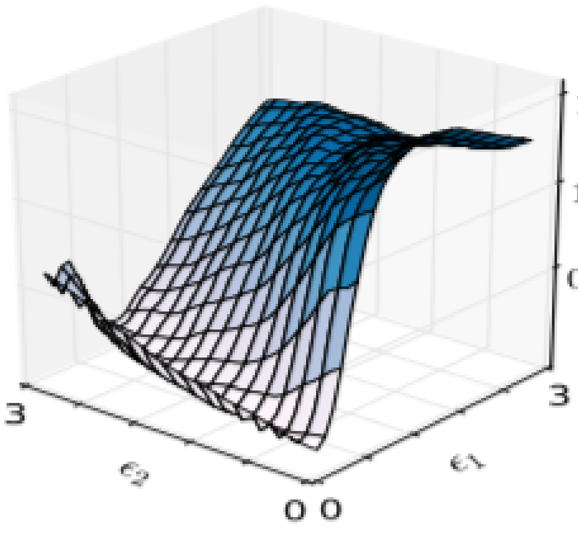
3.7.3 RFGSM - Randomized Fast Gradient Sign Method

RFGSM is another flavour of FGSM founded by Papernot et al[8] in which a randomness is added to the original input before feeding in to the machine-learning model. The reason with randomness is to escape the non-smoothness vicinity in the loss-function curve which is obtained by feeding in the original input & it was also observed that gradients are able to represent the loss function in a poor manner.

$$x^{adv} = x^{'} + (\epsilon - \alpha).sign(\nabla_{X^{'}} J(X^{'}, y_{true}))) \tag{3.5}$$



(a) Loss of model $v3_{adv}$.



(b) Zoom in for small ϵ_1, ϵ_2.

Figure 3.12: (a) denotes the curvature in the model's loss function while (b) denotes the same but in much zoomed-in level

Key-Properties :

- Non-Targeted
- Iterative attack
- L_∞ perturbation

Advantages :

- Model gets exposed to much diversified perturbations
- Naive FGSM transformed to powerful single-step attack
- Low computational time

Drawback : Still not strong enough to fool many powerful models.

3.7.4 Projected Gradient Descent - PGD

PGD, a technique proposed by Madry et al [8] is basically a Projected method which is generally used when dealing with *box constrained optimization* problems like -5<=x<=5, where the constraint is imposed on feasible set of parameters, **NOISE** in this case. As we optimize the function, we might take a step that takes us outside of the feasible set and we need to find a way to correct for that. A way to achieve it is to project the point back to the closest boundary of this feasible set. It is just gradient descent, where for every step taken, we project the resulting point to the closest feasible point. From another perspective, it keeps the *noise* level within a circle of particular radius.

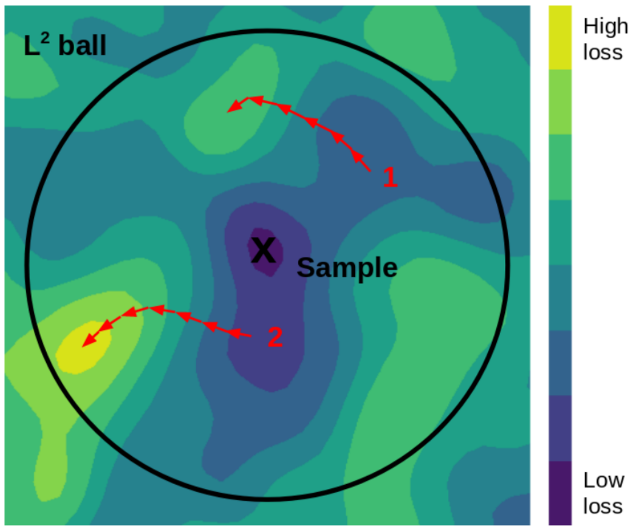


Figure 3.13: PGD maintaining the noise within a circle of particular radius, L_2 norm in this case

Algorithm for PGD:

- pick an initial point $x_0 \varepsilon Q$
- Loop until stopping condition is met
  - Descent direction: pick the descent direction as $-\nabla f(x_k)$
  - Step size: pick a step size t_k
  - Update $y_{k+1} = x_k - t_k \nabla f(x_k)$

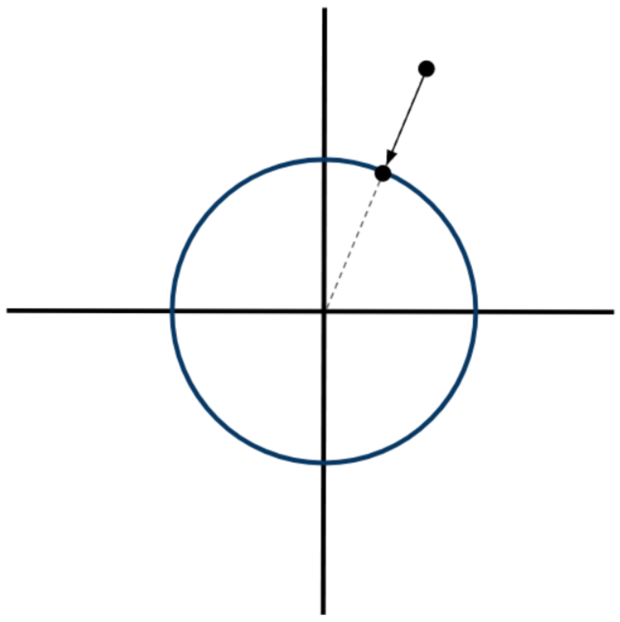


Figure 3.14: Projection of the noise which lies outside the feasible set back into the permissible set

– Projection : $x_{k+1} = \text{argmin}_{x \varepsilon Q} 1/2 \| x - y_{k+1} \|^2_2$

Key-Properties :

- Non-Targeted
- Iterative attack
- L_∞ & L_2 perturbation

Advantages :

- Much powerful than the FGSM & all its flavours
- Good candidate to generate strong adversaries
- Used as a part of Adversarial Training in many cases inorder to harden the neural-networks.

Drawbacks : The major drawback of this approach would be if there is any error during the updation at any point of the iteration, then it implies a wrong input is being fed into the next step of the iteration, which on overall gives a wrong output at the end.

3.7.5 Randomized Projected Gradient Descent - RPGD

Similar to RFGSM, this is just another flavour of vanilla PGD, a randomized version of PGD as proposed by Madry et al [8]. Here, before the input is being fed into the model, it is added with a random value of identical shape. The reason behind the initial random restarts is to escape being struck in the local minimum if it happens in any iteration & to reach the global optimum. Another supporting reason could be that a single gradient descent alone cannot solve the problem.

Key-Properties :

- Non-Targeted
- Iterative attack
- L_∞ & L_2 perturbation

Advantages :

- Encodes a way to escape the local minimum & reach the global optimum
- Solves the problem of incapability of single gradient descent alone to generate adversaries
- By Randomization, aggressive search of input space is obtained

Drawbacks : At times, if randomization is bit higher, then the algorithm may end up searching up in unwanted space which in reality wastes significant amount of time.

3.7.6 Averaged Projected Gradient Descent - APGD

APGD is a special case of PGD which came into existence due to the inability of vanilla PGD to attack & fool *Bayesian Neural network* (BNN). It was observed that inorder to attack BNN, there is a need to incorporate stochastic nature into the existing PGD & to cater that Madry et al [8] came up with this approach in which for each sample, gradients are calculated multiple times & the average of these values is found out & used as the final gradient value & hence the name Averaged -PGD.

Key-Properties :

- Non-Targeted
- Iterative attack
- L_∞ & L_2 perturbation

Advantages :

- Capable enough to attack Bayesian Neural network
- Due to its success, it's also used in Adversarial training of BNN, inorder to harden it.

Drawbacks : Not wise to use this on general type of neural network as this is way too much & it wastes out computational resources.

3.7.7 Iterative Least Likely Class Method - ILL

Both of the above methods, we discussed are untargeted in nature, i.e they increase the cost of the correct class without choosing the target class for misclassification. Now we will discuss a method which targets a specific class for misclassifications proposed by Papernot et al [7]. This would in some sense will fall under **targeted attack**. This attack is of interest on datasets with large number of classes, since on datasets like MNIST we have lower number of classes with higher distinction between classes. If we use untargeted attacks on ImageNet we can get uninteresting misclassifications like a change in breed of an animal as a misclassification. On the other hand it would indeed be interesting to see a misclassification of a spider as a towel.

This method tries to misclassify the original image to the class with the least probability according to prediction of original image. This class is found by :

$$y_{LL} = argmin_y(P(y|X)) \tag{3.6}$$

To achieve this misclassification we try to maximize log p($\mathrm{y_{LL}}$|X) by making iterative steps in the *opposite* direction of sign(∇_{X} log p($\mathrm{y_{LL}}$|X)). This expression equals sign(∇_{X}J(X,$\mathrm{y_{LL}}$)) for networks with cross entropy loss. Therefore we have the following method :

$$X_0^{adv} = X, \quad X_{N+1}^{adv} = Clip_{X,\epsilon}(X_N^{adv} - \alpha sign(\nabla_X J(X_N^{adv}, y_{LL}))) \tag{3.7}$$

We follow the same number of iterations and value of alpha as that in Basic Iterative Method.It is not guaranteed that we would always have the corresponding adversarial example since the above methods are not exact.

Key-Properties :

- Targeted
- Iterative
- L_∞ perturbation

Strength : Brings in a new feature to select the target class

Weakness :

- Not guaranteed every time that output will be misclassified as specified targeted class

- Other than this feature, it is identical to IFGSM method. Nothing much new here.

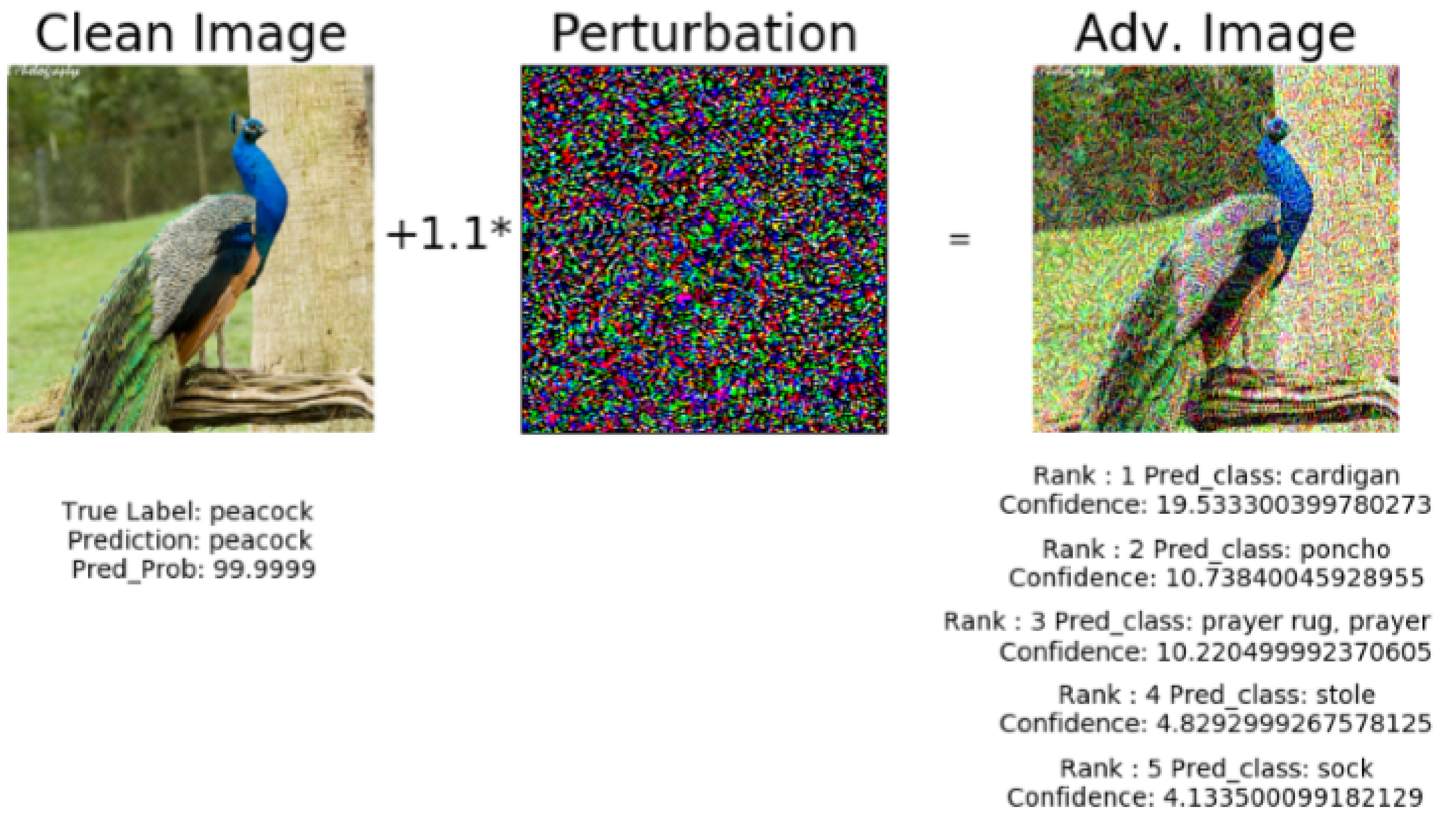

Clean Image
Perturbation
Adv. Image
+1.1*
=
True Label: peacock
Prediction: peacock
Pred_Prob: 99.9999
Rank : 1 Pred_class: cardigan
Confidence: 19.533300399780273
Rank : 2 Pred_class: poncho
Confidence: 10.73840045928955
Rank : 3 Pred_class: prayer rug, prayer
Confidence: 10.220499992370605
Rank : 4 Pred_class: stole
Confidence: 4.8292999267578125
Rank : 5 Pred_class: sock
Confidence: 4.133500099182129

Figure 3.15: We kept the target class as ”hen” (instead of choosing the least likely class, we selected a random class) for this image but this method could not yield that prediction in this case. But we can observe that it has misclassified the image.

3.7.8 Carlini & Wagner Attack - C&W Attack

This is one of the **most powerful attack** ever devised so far which was proposed by *Carlini et al* [9]. The reason behind this statement is as follows:

- All the remaining attacks whatever we explored above, was easily defeated by Defensive Distillation, a powerful technique on the Defence side but when it comes to C&W attack, Defensive Distillation was unable to defend it and in addition to it, C&W attack was successfully able to generate adversarial images for all the 1000 images provided as input and visually indistinguishable from the original image.
- Another reason could be that it was this specific attack which takes into consideration of both *the misclassification rate & the perturbation level* into the objective function, which was seldom noticed in the previous attack strategies.

Lets explore more about the C&W attack in this section.

There are 3 distance metrics that have been defined as part of this attack which are described below:

- L_0 distance measures the number of coordinates i such that $x_i \neq x_i^{|}$. Thus, the L_0 distance corresponds to the number of pixels that have been altered in an image.
- L_2 distance measures the standard Euclidean (rootmean-square) distance between x and x_0. The L_2 distance can remain small when there are many small changes to many pixels.
- L_∞ distance measures the maximum change to any of the coordinates:

$$||x - x^{|}||_\infty = max(|x_1 - x_1^{|}|, ...|x_n - x_1^{n}|) \tag{3.8}$$

For images, we can imagine there is a maximum budget, and each pixel is allowed to be changed by up to this limit, with no limit on the number of pixels that are modified.

It would be interesting to see the history behind the existence of the C&W Attack. The main reason this attack came into the picture is as a solution to **Box-Constrained** optimization problem, very similar to *projected gradient descent.*

While modifying the input image, to ensure the modification yields a valid image, we have a constraint on δ: we must have $0 \leq x_i + \delta_i \leq 1$ for all i. In the optimization literature, this is known as a "box constraint." There were three approaches to address this problem:

- **Projected gradient descent** performs one step of standard gradient descent, and then clips all the coordinates to be within the box.
- **Clipped gradient descent** does not clip x_i on each iteration; rather, it incorporates the clipping into the objective function to be minimized. In other words, we replace f(x + δ) with f(min(max(x + δ, 0), 1)), with the min and max taken component-wise.
- **Change of variables** introduces a new variable w and instead of optimizing over the variable δ defined above, we apply a change-of-variables and optimize over w, setting
$$\delta_i = 1/2(tanh(w_i) + 1) - x_i \tag{3.9}$$

The problem formulation for C&W attack can be defined as:

$$\begin{aligned}&\text{minimize}(x,x+\delta)_D\\&\text{such that } c(x+\delta) = t\\&x+\delta \; \varepsilon \; [0,1]^n\end{aligned}$$

where x is fixed and we need to find δ to minimize D(x,x+δ). D is our distance metric and can be any of L0, L2, L∞. There are many different approaches to solve this optimization problem, and as a first step the problem is reformulated in a bit different way as the constraint is highly non-linear.

Re-Formulation: We define an objective function f such that C(x+δ) = t if and only if f(x+δ) $\leq$ 0. Now we have an alternate formulation of the problem :

$$\text{minimize}(x,x+\delta)_D + c.f(x+\delta)$$

$$\text{such that } x+\delta \ \varepsilon \ [0,1]^n$$

where $c \geq 0$ is a constant. After assigning an L_p norm to D the problem finally becomes:

$$\text{minimize } ||\delta||_p + c.f(x+\delta)$$

$$\text{such that } x+\delta \ \varepsilon[0,1]^n$$

1-D optimization techniques like binary search is used to find optimum c. Also we need to take care of the range of $x_i + \delta_i$, i.e the pixels should remain between 0 and 1. To solve this *Box-constrained* optimization problem the change of variables technique is exploited out, which introduces a new variable w, due to which the problem becomes as optimizing over w instead of δ.

$$\delta_i = 0.5(\tanh(w_i+1)-x_i)$$

The advantage of this equation is that it automatically follows box constraints, i.e $-1 \leq \tanh(w_i) \leq 1 \Rightarrow 0 \leq x_i + \delta_i \leq 1$.

There were around 9 candidate functions for the proposed "f" function in the original work but majority of them failed in 1 edge case or the other but there was one which met all the specified condition, which is:

$$f(x^{'}) = max(max(Z(x^{'}_i) - Z(x^{'}_t))), -\kappa) \tag{3.10}$$

where Z(x) denotes the *logits* i.e the input values provided to the final softmax layer & $\varkappa$ is a hyperparameter which provides a way to control the prediction's *confidence level* as well.

Key-Properties :

- Targeted & Non-Targeted
- Iterative
- L_0, L_2, L_∞ perturbations

Strength :

- At present, this is the Benchmark to evaluate the Robustness of Neural network. In other words, if someone comes up with a new defense technique, then it is tested against this attack technique to evaluate how powerful the proposed defense method is.
- Can be Tailored to all three types of perturbations
- Gives the user, the control to even set the model's final prediction confidence level too
- Broke the hypothesis that *Defensive Distillation* was the most resistable & unbreakable of all the defense technique & exposed the truth behind defensive distillation that even it is not learning the true underlying concepts.

Weakness :

- Takes long time to generate adversarial examples
- In some cases, it even fails to generate one as the loss function used fails to converge which leads to early-stopping of the program.

3.7.9 Jacobian Saliency Map based Attack - JSMA

The purest form of ***targeted attack*** can be probably attributed to JSMA, although in some sense *IterLL, C&W* attacks would fall under this category as well. It was proposed by *Anqi Xu et al*[10] with some variants of it alongside. Before diving into the details, lets explore a bit about the Saliency maps & the purpose of it.

Saliency Maps are basically a visualization tool which can provide the user an insight into the model's prediction decision.

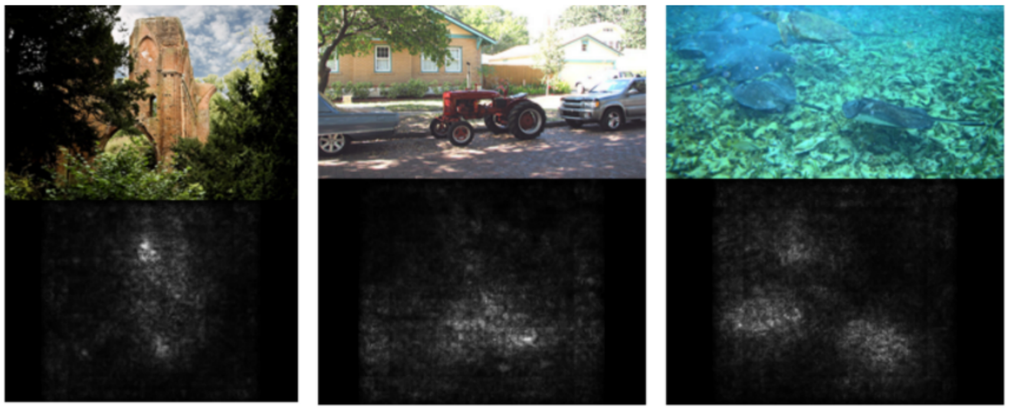
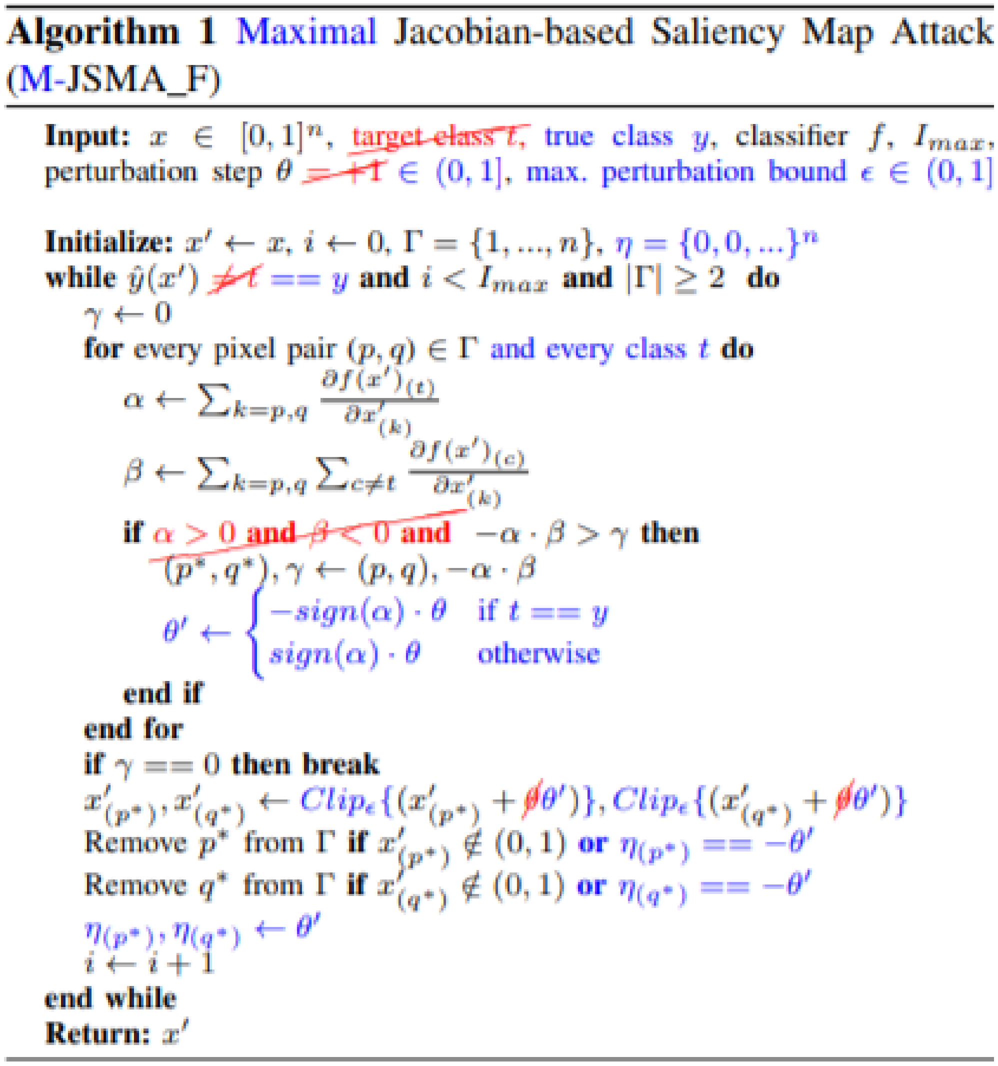


Figure 3.16: Saliency Maps with the corresponding input image

It is usually rendered as heatmaps as shown in the image above where the *hotness* corresponds to the regions of big impact in the model's decision process. From an image perspective, it identifies the most influential pixels towards a particular class prediction. In this aspect, this can be used to perform a post-mortem analysis to find the reason behind the wrong prediction by the model.

With this background, lets now analyze about the JSMA technique. It basically exploits the saliency maps to generate adversarial images by minimizing the most influential pixels towards the *original or true* class (assigns 0)& maximizing the influential pixels towards the *target* class (assigns 1).

$$S^{+}(x_{(i)}, c) = \begin{cases} 0 & if \ \frac{\partial f(x)_c}{\partial x_i} < 0 \ \ or \ \ \sum \frac{\partial f(x)_{c'}}{\partial x_i} > 0 \\ -\frac{\partial f(x)_c}{\partial x_i} . \sum \frac{\partial f(x)_{c'}}{\partial x_i} & \text{otherwise} \end{cases} \tag{3.11}$$



```
Algorithm 1 Maximal Jacobian-based Saliency Map Attack (M-JSMA_F)
  Input: $x \in [0,1]^n$, ~~target class $t$,~~ true class $y$, classifier $f$, $I_{max}$,
  perturbation step $\theta$ ~~$= +1$~~ $\in (0,1]$, max. perturbation bound $\epsilon \in (0,1]$

  Initialize: $x' \leftarrow x$, $i \leftarrow 0$, $\Gamma = \{1, ..., n\}$, $\eta = \{0, 0, ...\}^n$
  while $\hat{y}(x')$ ~~$\neq t$~~ $== y$ and $i < I_{max}$ and $|\Gamma| \geq 2$ do
    $\gamma \leftarrow 0$
    for every pixel pair $(p, q) \in \Gamma$ and every class $t$ do
      $\alpha \leftarrow \sum_{k=p,q} \frac{\partial f(x')_{(t)}}{\partial x'_{(k)}}$
      $\beta \leftarrow \sum_{k=p,q} \sum_{c \neq t} \frac{\partial f(x')_{(c)}}{\partial x'_{(k)}}$
      if ~~$\alpha > 0$ and $\beta < 0$ and~~ $-\alpha \cdot \beta > \gamma$ then
        $(p^*, q^*), \gamma \leftarrow (p, q), -\alpha \cdot \beta$
        $\theta' \leftarrow \begin{cases} -sign(\alpha) \cdot \theta & \text{if } t == y \\ sign(\alpha) \cdot \theta & \text{otherwise} \end{cases}$
      end if
    end for
    if $\gamma == 0$ then break
    $x'_{(p^*)}, x'_{(q^*)} \leftarrow Clip_\epsilon\{(x'_{(p^*)} + \cancel{\theta}\theta')\}, Clip_\epsilon\{(x'_{(q^*)} + \cancel{\theta}\theta')\}$
    Remove $p^*$ from $\Gamma$ if $x'_{(p^*)} \notin (0,1)$ or $\eta_{(p^*)} == -\theta'$
    Remove $q^*$ from $\Gamma$ if $x'_{(q^*)} \notin (0,1)$ or $\eta_{(q^*)} == -\theta'$
    $\eta_{(p^*)}, \eta_{(q^*)} \leftarrow \theta'$
    $i \leftarrow i + 1$
  end while
  Return: $x'$
```

Figure 3.17: JSMA Algorithm as defined in the original paper *Anqi Xu et al*[10]

It is in this way the input image is bit perturbed inorder to yield the specified target as the model's prediction.

Key-Properties :

- Targeted
- Iterative
- L_0 perturbation

Strength :

- Allows the user to specify the target & tries to make the model predict the specified target as the output.
- Also, Targeted attack is the most difficult attack to perform than the non-targeted one.

Drawback :

- At first case, this is not at all a guaranteed attack. In fact, the results are noticed in rare cases.
- The property of ***structural similarity*** is expected between the original input & the specified target which makes the success rate higher. For Example, incase of MNIST dataset '7' & '1' would meet this condition.
- In some cases, there is way too much distortion to the input image which gets clearly visible to the outsider.

3.7.10 Black-Box Attack:

Lets navigate from *White-Box* attack mode to a completely different form of attack namely *Black-Box* attack. As opposed to white-box attack, in which the attacker has complete knowledge about the victim model right from its architecture to its trainable parameters values, here in contrary to that, attacker has zero knowledge about the victim model with no gradient information available (which is exploited in all the previous methods). This leaves us with a question 'How to attack such a model?'. The Answer is :

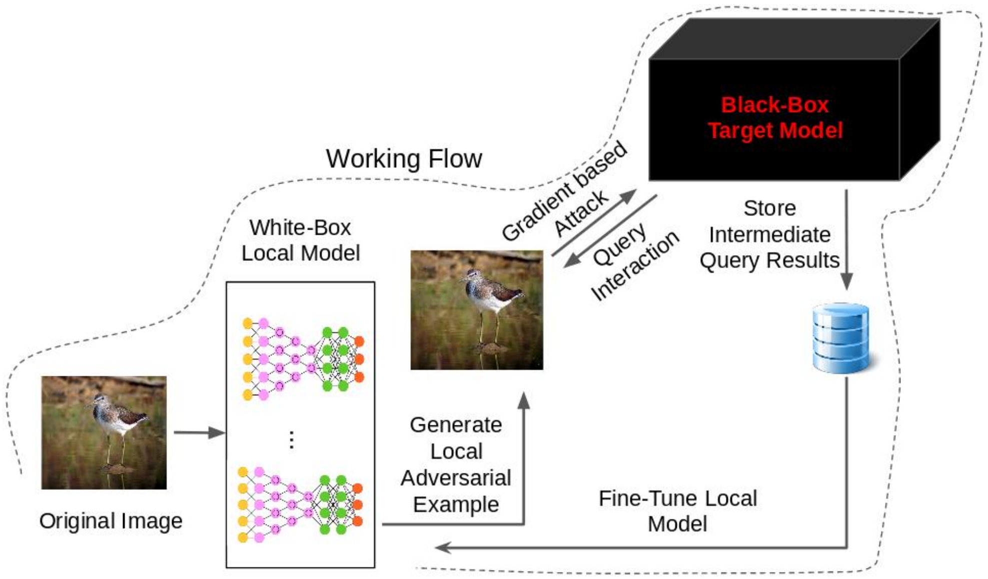


Figure 3.18: Process Flow in a Black-Box attack mode

As the attacker is unaware of the victim model, he develops another model locally called *substitute model* for which he has full-control over it i.e complete knowledge about the model's architecture & its parameters. with this model in hand, the situation becomes similar to white-box attack mode & the attacker uses this substitute model to generate adversarial examples, which are then feeded into the Black-box victim model & the output is noted. Based on the output, the parameters of the substitute model is tuned which will eventually match or gets closer to the victim model's parameters & thereby generating strong adversaries & successfully fooling the black-box model. It must be noted that this attack gets harder based on the restrictions imposed by the black-box model like no.of permissible queries, shape of the output response i.e if the model spits out *class-probabilities*, it is quite easy to guess the gradient information.

3.8 DEFENSE towards Adversarial Attacks

In general, the defense strategies can be classified into two types namely, *reactive* where the model has the capability to detect adversarial examples while the other being *proactive* in which the model itself is robust against adversarial examples. The defenders perspective will be mostly towards training a model at high cost by increasing model's complexity or using High-dimensional input.

3.8.1 Adversarial Training

It has been observed that by training the neural network on a mixture of adversarial and clean examples, a neural network could be regularized to an extent. Training on adversarial examples basically involves special kind of data augmentation which uses input that are unlikely to occur naturally. The working nature of this technique is very similar to Human's Immunity system which when provided with a sample of particular virus (anti-vaccine), it becomes aware of this & the next time when it encounters the same virus, it offers complete resistance. The same analogy can be applied to this technique as well.

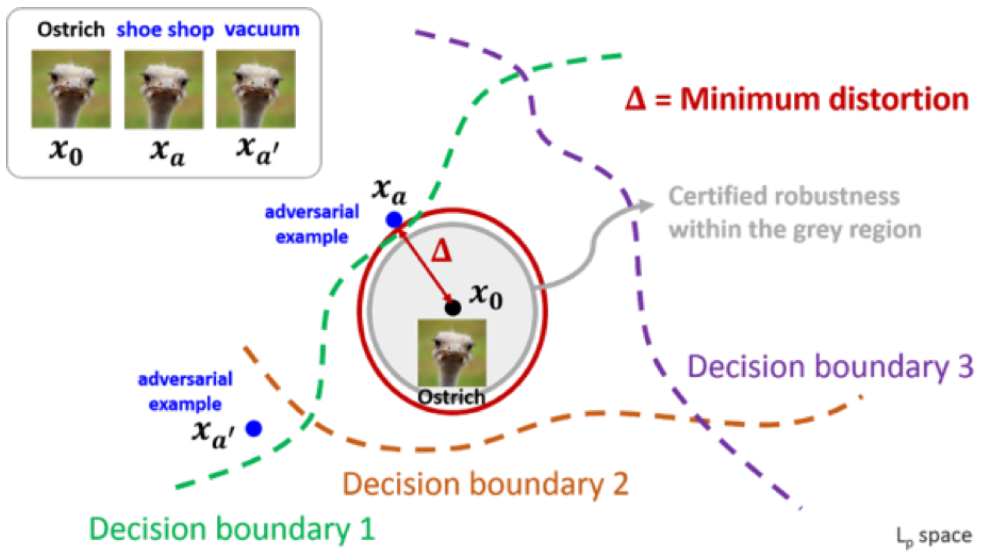


Figure 3.19: Representation of Decision boundary of the model

From the model's perspective, it is learning the true decision boundary which is responsible for accommodating the adversarial sample, which the sample exploits as well.

Strength:

Able to successfully harden the model & thereby resist/defend against adversarial attacks.

Weakness:

- Is completely irresistant towards ***Black-Box Attacks***
- Even in case of white-box attack mode, it cannot defend against some powerful attacks like C&W attack method.

3.8.2 Ensemble Adversarial Training

This is probably another variant of vanilla Adversarial Training which was proposed by *Florian et al*[11]. The main objective behind this method is to provide a way to defend against Black-box attacks & it does so by using some random pre-trained model's parameters to generate adversaries samples & transfers this samples to adversarially train the needed model. In this way, it incorporates the black-box nature into the naive adversarial training.

Strength:

- Defends against Black-box attacks
- Decouples adversarial sample generation from model's parameters
- Brings in the Transfer nature of Black-Box attacks
- Model gets exposed to Diversification of perturbations.

Drawback:

- Still vulnerable to some powerful attacks
- Doesn't provides a guarantee.

3.8.3 Defensive Distillation

The most powerful defense technique ever found so far can be attributed to *Defensive Distillation* as it can practically defend against all types of attacks except for C&W Attack. To be precise, *Distillation* is completely a different idea, the goal of which is to run giant neural networks on low-powered devices like smartphones which otherwise doesn't has the capability to do so. The following section briefs out in a detailed way.

Neural Network Distillation -

The ultimate goal behind Neural Network Distillation [5] is to provide low-powered devices like smartphones to utilise the knowledge learnt by Highly-complex neural networks by transferring the knowledge in the form of class-probability vectors to a relatively small-sized network. The main foundation behind this approach is that Deep Neural Networks's knowledge is not only embedded in the weights between the various layers but also in the final class-probability vectors produced in the softmax layer.

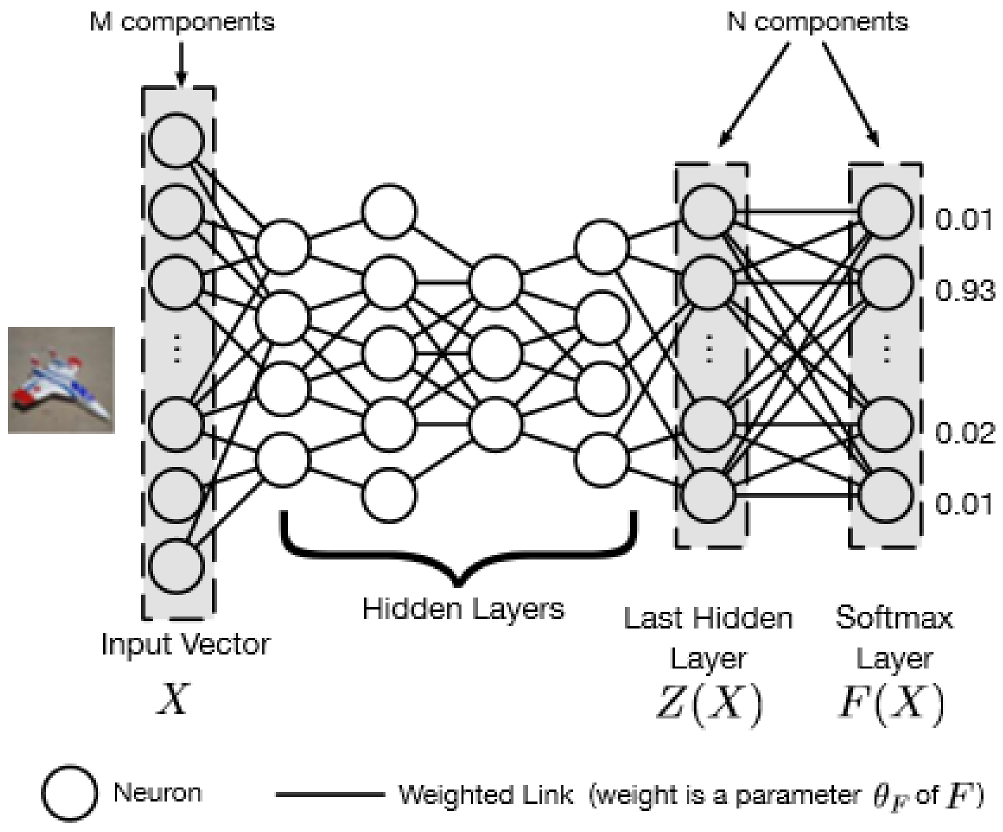


Figure 3.20: Deep Neural Network Architecture

The above figure can be considered as a reference under Distillation context. There are some conditions which has to be met to perform Network Distillation which are described as follows:

- At present, Distillation is done only under 'Supervised learning' environment

- The final most layer in the Neural Network has to be the softmax layer as evident from the above figure.
- In the softmax layer, there is an additional parameter called 'Distillation Temperature' which forms the core of this technique and explained below :

$$F(X) = [e^{z_i(X)/T} / \sum_{l=0}^{N-1} e^{z_l(X)/T}]_{i \in 0 \ldots N-1} \tag{3.12}$$

- During the Training process, the Distillation Temperature T > 1 and the ideal value is T = 20 but while testing T = 1 which restores back to the normal softmax layer.

In the Distillation terminology, the first big neural network is called as 'Teacher' network while the reduced second network is called as 'Child' network. Also, the term 'logits' refers to the output produced by the last hidden layer which is also the input to the softmax layer.

Lets shift the focus towards the Training process. As explained previously, the final softmax layer is tweaked a bit to include Distillation Temperature in it and the value is set to 20 post-which neural network is trained in a normal way with the final class-probability vector is recorded to the corresponding inputs. These class-probability vectors which are now used to label the inputs are called as 'soft-labels' as opposed to previously marked 'hard-labels'. Now with the new Dataset which is marked with soft-labels, the second small-sized network, 'child' network is trained using this dataset. In this way, the child network completely utilises the knowledge learnt by the teacher network.

Defensive Distillation -

Armed with Network Distillation in the background, lets dive into the Defensive Distillation aspect. The majority of the things here works in the same way as that of Network Distillation with the only change being the teacher and the child network are same. The objective here is not to reduce the size of the network but to make the given network more robust towards adversarial examples by training it with Defensive Distillation and hence this approach is adapted. It has been found that by this training the amplitude of the network's gradients was reduced to 10^{30} times which is the main factor behind reducing the model's sensitivity towards minor perturbations.

3.9 Real-World Dataset Attack

This section explores about the Real-world Dataset Attack which forms the third part of the Thesis. As all the previous sections dealt with computer vision problems, this section is dedicated completely for NLP domain which is explained as follows:

3.9.1 DeepWordBug

DeepWordBug is a novel algorithm exclusively dedicated for NLP attacks in ***Black-box*** mode, which was invented by *Ji Gao et al [12].* Unlike the attack algorithms which works only for computer vision domain, DeepWordBug possess some unique capabilities as it needs to handle some additional challenges involved with NLP attacks. The one such major challenge is ***discrete*** nature of the NLP in contrary to ***continuous*** nature of computer vision problems i.e while performing attack on images, it perfectly makes sense to modify the pixel to any value within the permissible set but this is not the case with NLP problems, as the new *transformed* character must be a valid one like *alphabets* or *alpha-numeric* & the new word produced after the transformed character should completely make sense i.e *ma**n*** & *ma**p*** is completely valid here. There are 2 novel things which makes this a unique approach:

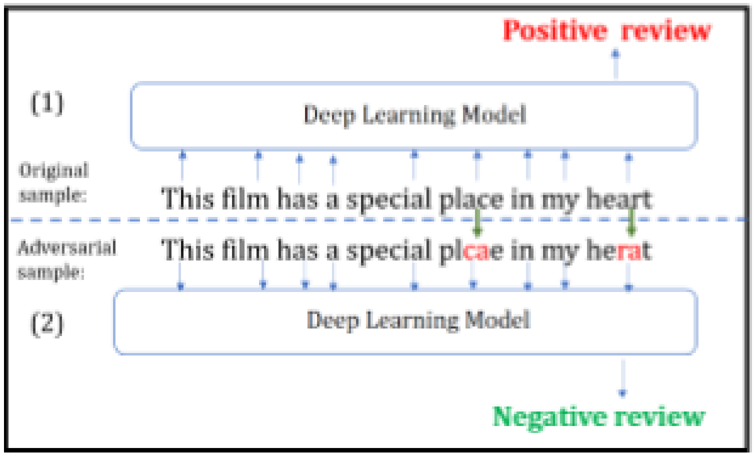


Figure 3.21: Outcome of DeepWordBug

Scoring Function - The Scoring function forms the main foundation & the objective is to identify the most ***sensitive/influential*** characters in the given sentence & assign score values to it. It is interesting to note that this scoring function is independent of model's parameters & not attributed to any specific model which means it can work well across any type of model. The main idea behind this working is it scans the given sentence from left to right & for each character, it records the output classification label while that character is present & records the same by removing that character. If the output changes, then it implies that this character is influential towards this particular class. The score calculation happens in 3 parts namely:

Temporal Score - Let the given input sentence be X = x_1,x_2,x_3,...x_n,, where x_i represents the i^{th} character in the sentence.

$$TS(x_i) = F(x_1, x_2, x_3, ...x_{i-1}, x_i) - F(x_1, x_2, x_3, ...x_{i-1}) \tag{3.13}$$

Temporal Tail Score - One problem with *Temporal Score* is it gives importance only to the characters preceding the given character & it completely ignores the characters succeeding it, which was the reason TTS came into existence:

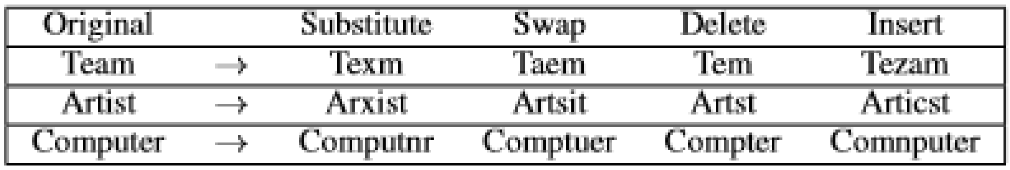
$$TTS(x_i) = F(x_i, x_{i+1}, x_{i+2}....x_n) - F(x_{i+1}, x_{i+2}....x_n) \tag{3.14}$$

Combined Score - Like in the name, CS considers both the above approaches to calculate the final score, which is:

$$CS(x_i) = TS(x_i) + \lambda.TTS(x_i) \tag{3.15}$$

where λ is hyperparameter.

Token Transformation - Armed with the scores of each character in the given sentence in our hand, next step is to transform these characters inorder to fool the model. The first step involves sorting the scores of the characters in the descending order i.e sort by the most-influential characters. Then one of the following transformation techniques can be applied:



Original		Substitute	Swap	Delete	Insert
Team	$\rightarrow$	Texm	Taem	Tem	Tezam
Artist	$\rightarrow$	Arxist	Artsit	Artst	Articst
Computer	$\rightarrow$	Computnr	Comptuer	Compter	Comnputer

Figure 3.22: Token Transformation

Note that in the modern way, when it comes to *substitute,insert,swap* operations, ***homoglyph*** characters are used so that it makes nearly impossible for a human observer to distinguish it.

This in overall makes the Adversarial attack possible & can be applied to any type of model like *CNN,RNN,LSTM*.

Chapter 4

Experiment

A. **White-Box Non-Targeted-Attack**

Attack Type : The following types of non-targeted white-box attacks are experimented out in this Thesis:

- FGSM - Fast Gradient Sign Method
- IFGSM - Iterative Fast Gradient Sign Method
- RFGSM - Randomized Fast Gradient Sign Method
- PGD - Projected Gradient Descent
- RPGD - Randomized Projected Gradient Descent
- APGD - Averaged Projected Gradient Descent
- ITERLL - Iterative Least-Likely Class Method
- C&W - Carlini & Wagner Attack

Dataset : All the attack variants are experimented with three types of datasets namely MNIST Dataset, CIFAR10 Dataset and ImageNet Dataset.

The first dataset MNIST consist of 60,000 training samples and 10,000 test samples of handwritten-digits from 0-9. Each image is a gray-scale image with 28 X 28 pixels, size-normalized and centered. The 60,000 image samples present in the training set were composed by 250 writers & 10,000 test image samples were produced by completely different set of 250 writers.

Following the MNIST dataset, we have CIFAR 10 dataset. This dataset consist of 60,000 images in total & the images are colored images with size 32X32. There are in total 10 output classes which corresponds to some real-world entities like cat,deer,ship,truck etc. with each class having 1000 images under it & the classes are mutually exclusive. The entire dataset is divided into training set with 50,000 images & test set with 10,000 images.

ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet, majority of them are nouns (80,000+). In ImageNet, there are on average 1000 images, to illustrate each synset. Images of each concept are quality-controlled and human-annotated. This Dataset is constantly updated from time to time and continues evolving. In its completion, ImageNet will offer tens of millions of cleanly sorted images for most of the concepts in the WordNet hierarchy.

Dataset	**Training**	**Test**	**Total**
MNIST	60000	10000	70000
CIFAR10	50000	10000	60000

Table 4.1: Datasets Summary

Model -

All the above datasets are experimented with 3 different models of Convolution Neural Network (CNN) with each model type dedicated for particular dataset, the architectures of which are described below:

Layer Type	I/P Feature	O/P Feature	Filters/Pooling
convolutional	1	10	10 filters, 5X5
Relu Pooling	10	10	Max Pool, 2X2
convolutional	10	20	20 filters, 5X5
Relu Pooling	20	20	Max Pool, 2X2
Dropout	-	-	-
Fully connected	320	50	-
Fully connected	50	10	-
Softmax	10	10	-

Table 4.2: MNIST Architecture

Layer Type	I/P Feature	O/P Feature	Filters/Pooling
Relu convolutional	3	6	6 filters, 5X5
Pooling	6	6	Max Pool, 2X2
Relu convolutional	6	16	16 filters, 5X5
Pooling	16	16	Max Pool, 2X2
Fully connected	450	120	-
Fully connected	120	84	-
Fully connected	84	10	-

Table 4.3: CIFAR10 Architecture

Layer Type	**patch size**	**stride**	**I/P size**
convolutional	3X3	2	299X299X3
convolutional	3X3	1	149X149X32
conv padded	3X3	1	147X147X32
pool	3X3	2	147X147X64
convolutional	3X3	1	73X73X64
convolutional	3X3	2	71X71X80
convolutional	3X3	1	35X35X192
3 X Inception	-	-	35X35X288
5 X Inception	-	-	17X17X768
2 X Inception	-	-	8X8X1280
Pool	8X8	-	8X8X2048
Linear	logits	-	1X1X2048
Softmax	classifier	-	1X1X1000

Table 4.4: InceptionV3 Architecture(ImageNet)

Justification for Model Architectures & Datasets-

- In-order to analyze the strength of different attack strategies, models of varying sizes are chosen & this is complete random.
- In accordance with the ***Benchmark*** aspect of the *Threat model,* standard datasets of MNIST,CIFAR10, ImageNet are chosen with the state-of-the-art **Inception v3** being dedicated for ImageNet example.
- Due to the huge size of ImageNet dataset, the whole dataset can't be considered which makes the Thesis scope infeasible. Hence just 1 single image of *giantPanda* is used (randomly) as a representation for this dataset but all the input items in the dataset along with the labels is feeded into the model in the form of python dictionaries which safely permits to use any of these class labels for the attack methods & hence this single image is referred as 'ImageNet' Dataset across the Thesis.

Attack -

The following table summarizes about the **set of epsilon** values(perturbation level) experimented with for the different types of attack techniques:

Attack	Experiment(1)	Experiment(2)	Experiment(3)
FGSM	0.7	0.07	0.2
IFGSM	40/255	4/255	15/255
RFGSM	160/255	16/255	70/255
PGD	0.3	0.03	0.1
RPGD	0.3	0.03	0.1
APGD	0.3	0.03	0.1
ITERLL	40/255	4/255	15/255

Table 4.5: Epsilon values set

The way these epsilon values chosen is, first a standard recommended value from the original paper is chosen & based on this, other two values are chosen in a complete random fashion with 1 value above the recommended value while the other one below it. Readers might wonder why same set of epsilon values isn't used across all the attacks which may yield a clear analysis,and the reason is majority of these attacks are mutually exclusive & for the proper functioning of these attacks, the epsilons should fall under some specific scale range. In cases, where the attacks shares similarity, same value set is used as evident from the above table.

In addition to the above mentioned epsilon values which is the major factor which controls the intensity of the attacks, there are also **other hyperparameters**, which are described in the following table:

Attack	**Alpha**	**Iteration**
IFGSM	1/255	0
RFGSM	8/255	1
PGD	2/255	40
RPGD	2/255	40
APGD	2/255	40
ITERLL	1/255	0

Table 4.6: Additional default Hyperparameters

In some cases, iterations are set to zero or 1 as it will get inferred automatically from the formula & in addition to it, it will expose how much difference is created from the naive methods.**APGD** has additional hyperparameter called *sampling* which determines the fine sampling of the gradients & is described below:

APGD	**Experiment(1)**	**Experiment(2)**	**Experiment(3)**
Sampling	10	5	15

Table 4.7: Sampling values set for APGD

Note that C&W attack is quite unique from other attack types & doesn't have overlapping hyperparameters, which is explained below:

parameter	**value**
C	10
kappa	0
Iteration	1000
learning rate	0.01

Table 4.8: C&W Attack Hyperparameter

B. **White-Box Targeted Attack**

When it comes to attacks, the most difficult attack is the *Targeted attack* in which the model is forced to predict a particular target instead of a random one & this is done by using **JSMA** (Jacobian based Saliency Map) Attack.

Input -

MNIST dataset is used as input here, as JSMA is a targeted attack, it works best when both the given input & the specified target has high *structural similarity*. For this reason, two specific cases are chosen:

Case 1: Input Image - 7 & Target - 1

Case 2: Input Image - 1 & Target - 7

This setting gives JSMA an easy-environment, post-which experiment is done.

Model Architecture -

Layer Type	I/P Feature	O/P Feature	Filters/Pooling
convolutional	1	10	10 filters, 5X5
Relu Pooling	10	10	Max Pool, 2X2
convolutional	10	20	20 filters, 5X5
Relu Pooling	20	20	Max Pool, 2X2
Dropout	-	-	-
Fully connected	320	50	-
Fully connected	50	10	-
Softmax	10	10	-

Table 4.9: MNIST Architecture

Attack -

There isn't much hyperparameter to experiment with in this case except for **Max-Iteration** which is set to *10000* as the whole algorithm keeps running until the model makes the given target as the prediction or there are no more valid pixels left out to perturb it.

C. **Black-Box Attack**

As the Black-box attack should be performed on an unknown model, two types of models are used here namely **substitute model** & **target model**. Note that size of the target model is pretty big as it will clearly expose the power of **Transferability** in Black-Box attacks. The following section explains in a much detailed way:

Dataset -

Dataset	Training	Test	Total
CIFAR10	50000	10000	60000

Table 4.10: CIFAR10 Dataset

Model -

Layer Type	**I/P Feature**	**O/P Feature**	**Filters/Pooling**
convolutional	3	32	32 filters, 5X5
Relu	32	32	-
BatchNormalization	32	32	-
MaxPooling	32	32	Max Pool, 2X2
Convolutional	32	64	64 filters, 5X5
BatchNormalization	64	64	-
Relu	64	64	-
MaxPooling	64	64	2X2

Table 4.11: CIFAR10 Substitute Model Architecture

Layer Type	**I/P Feature**	**O/P Feature**	**Filters/Pooling**
convolutional	3	96	96 filters, 3X3
GroupNormalization	32	96	-
ELU	-	-	-
Dropout2D	-	-	0.5
convolutional	96	96	96 filters, 3X3
GroupNormalization	32	96	-
ELU	-	-	-
convolutional	96	96	96 filters, 3X3
GroupNormalization	32	96	-
ELU	-	-	-
Dropout2D	-	-	0.5
convolutional	96	192	192 filters, 3X3
GroupNormalization	32	192	-
ELU	-	-	-
convolutional	192	192	192 filters, 3X3
GroupNormalization	32	192	-
ELU	-	-	-
Dropout2D	-	-	0.5
convolutional	192	256	256 filters, 3X3
GroupNormalization	32	256	-
ELU	-	-	-
convolutional	256	256	256 filters, 1X1
GroupNormalization	32	256	-
ELU	-	-	-
convolutional	256	10	10 filters, 1X1
Avg Pooling	10	10	20X20

Table 4.12: CIFAR10 Target Model Architecture

Attack -

Attack	Epsilon	Alpha	Iteration
PGD	0.3	2/255	10

Table 4.13: Attack Hyperparameter

The above experiments summarizes different attack strategies that can fool the neural network. The next section explores about some popular defense strategies which can defend from above attacks.

D. **Adversarial Training -**

Adversarial Training uses a custom MNIST Architecture with MNIST dataset similar to that of previous experiments & PGD is used to adversarially train the network with configurations same as that of *JSMAs PGD attack* while FGSM is used in the Testing phase to analyze the cross-attack resistivity.

Layer Type	I/P Feature	O/P Feature	Filters/Pooling
convolutional	1	16	16 filters, 5X5
Relu	16	16	-
convolutional	16	32	32 filters, 5X5
Relu	32	32	-
Pooling	32	32	Max Pool, 2X2
convolutional	32	64	64 filters, 5X5
Relu	64	64	-
Pooling	64	64	Max Pool, 2X2
Fully connected	576	100	-
Fully connected	100	10	-

Table 4.14: Custom MNIST Architecture

E. **Defensive Distillation -**

Defensive Distillation involves training the model *twice*, with the first model undergoes normal training & the *soft-labels* (class-probabilities) are extracted out of it. The second time training is done by using these soft-labels as their target instead of hard-labels as in the first training.

Dataset-

Dataset	Training	Test	Total
MNIST	60000	10000	70000

Table 4.15: MNIST Dataset

Model Architecture -

Layer Type	I/P Feature	O/P Feature	Filters/Pooling
convolutional	1	10	10 filters, 5X5
Relu Pooling	10	10	Max Pool, 2X2
convolutional	10	20	20 filters, 5X5
Relu Pooling	20	20	Max Pool, 2X2
Dropout	-	-	-
Fully connected	320	50	-
Fully connected	50	10	-
Softmax	10	10	-

Table 4.16: MNIST Architecture

Defense -

Process	Temperature	Label	Loss
Train 1	20	Hard	CrossEntropy
Train 2	20	Soft	KLDivLoss
Test 2	1	Hard	-

Table 4.17: Defense Hyperparameters

F. **Real World Dataset, NLP Attack -**

To demonstrate this attack, character-level CNN is used as the model & 2 types of datasets are used namely *Ag_News* & *NoRec*. 'Ag_news' dataset is the English news dataset which classifies the news/input sentence into 4 categories while 'NoREC' is the norwegian movie review dataset that sentiments the movie reviews into 6 classes.

Dataset -

Dataset	Training	Test	Total	Class
AG_NEWS	120000	7600	127600	4
NOREC	28155	7038	35194	6

Table 4.18: NLP Datasets

Model Architecture -

The major hyperparameter associated with this approach is *power* which denotes the number of characters in the input sentence to be modified. It is set as **10** for the experiments. If the value is high, then the attack might get visible to the human observers but if its low then it might impact the strength of the attack. So proper balance should be maintained. Another hyperparameter is maximum length of the input sentence which is set to **1014** in this case.

Layer Type	I/P Feature	O/P Feature	Filters/Pooling
convolutional-1D	69	256	256 filters, 7X7
Relu 1D-Pooling	256	256	Max Pool, 3X3
convolutional-1D	256	256	256 filters, 7X7
Relu 1D-Pooling	256	256	Max Pool, 3X3
convolutional-1D	256	256	256 filters, 3X3
Relu	256	256	-
convolutional-1D	256	256	256 filters, 3X3
Relu	256	256	-
convolutional-1D	256	256	256 filters, 3X3
Relu	256	256	-
convolutional-1D	256	256	256 filters, 7X7
Relu 1D-Pooling	256	256	Max Pool, 3X3
Fully connected	8704	1024	-
Relu	1024	1024	-
Dropout	-	-	0.5 dropout rate
Fully connected	1024	1024	-
Relu	1024	1024	-
Dropout	-	-	0.5 dropout rate
Fully connected	1024	4/6	-
Softmax	4/6	4/6	-

Table 4.19: NLP Model Architecture

Chapter 5

Analysis

A. **White-Box Non-Targeted-Attack :**

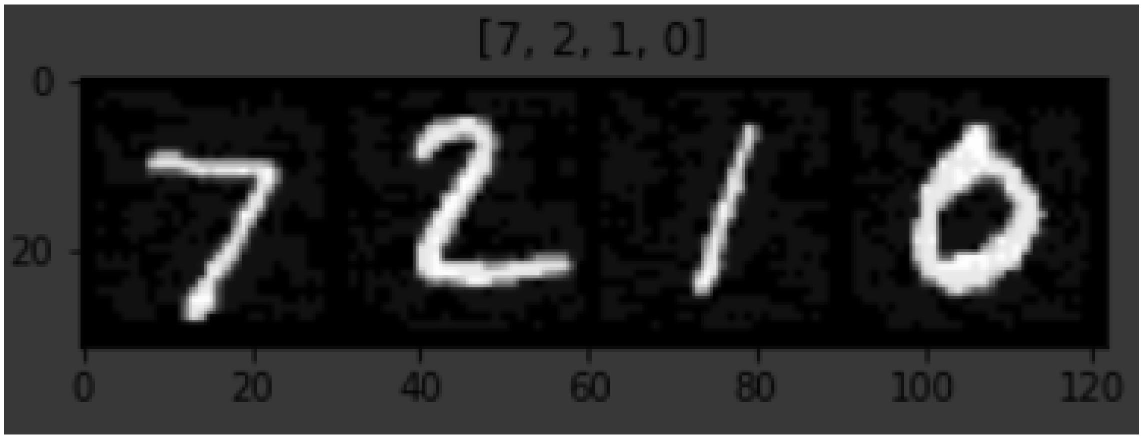
The following section does an analysis about the Test-accuracies i.e the total number of correctly classified samples, after the attack is made on previously described models which is followed by a sample perturbed image which shows the impact of the attack.

Attack Type - FGSM

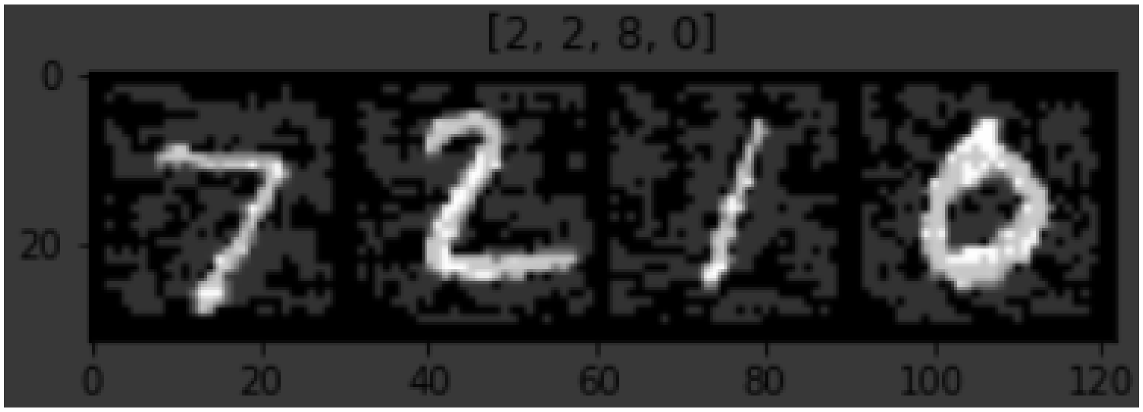
Model \ Epsilon	0.07	0.2	0.7
MNIST	91.22	43.01	0.77
CIFAR10	8.69	1.75	0.36
ImageNet	100	100	0.0

Table 5.1: FGSM - Test Accuracy Percentage

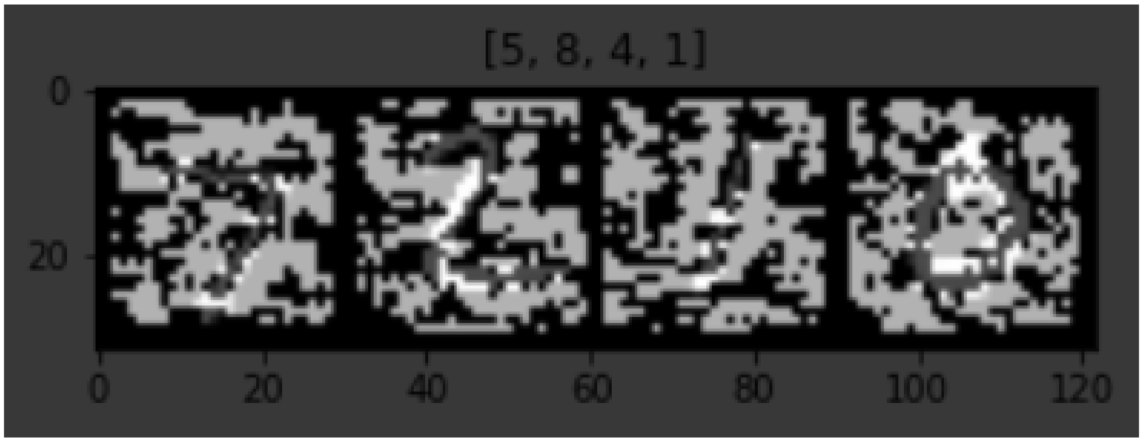
Perturbed Image -



(a) Epsilon - 0.07

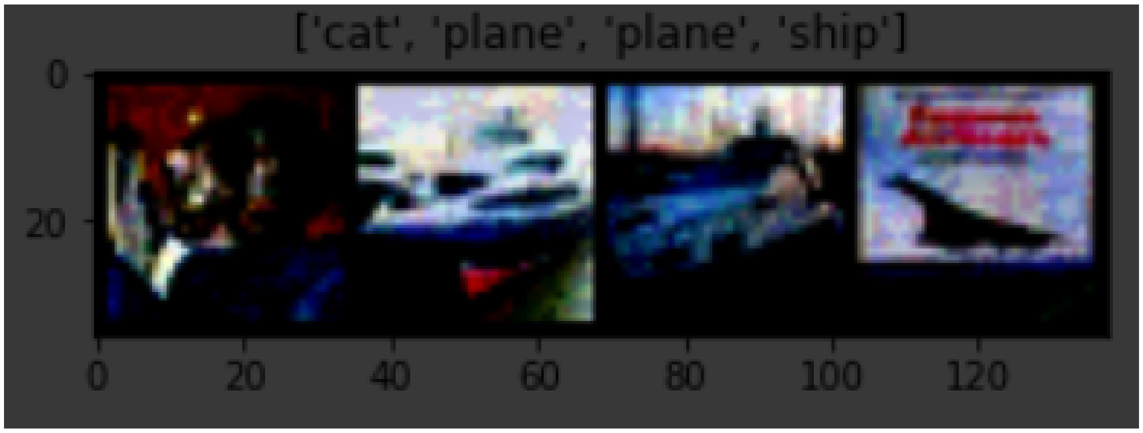


(b) Epsilon - 0.2

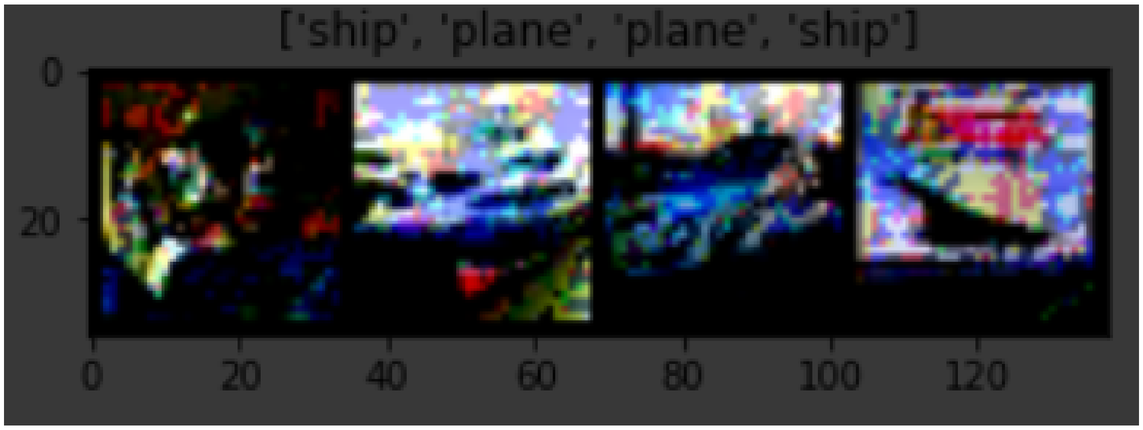


(c) Epsilon - 0.7

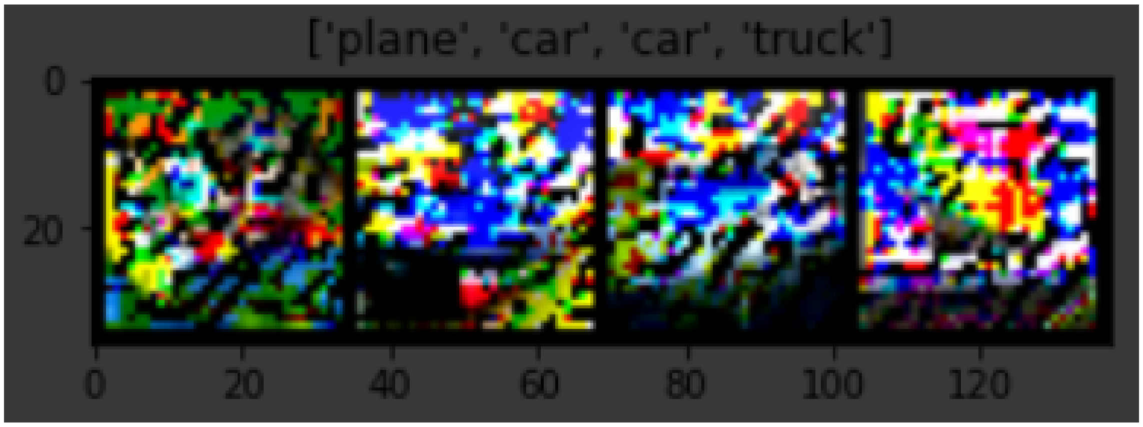
Figure 5.1: FGSM's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) Epsilon - 0.07

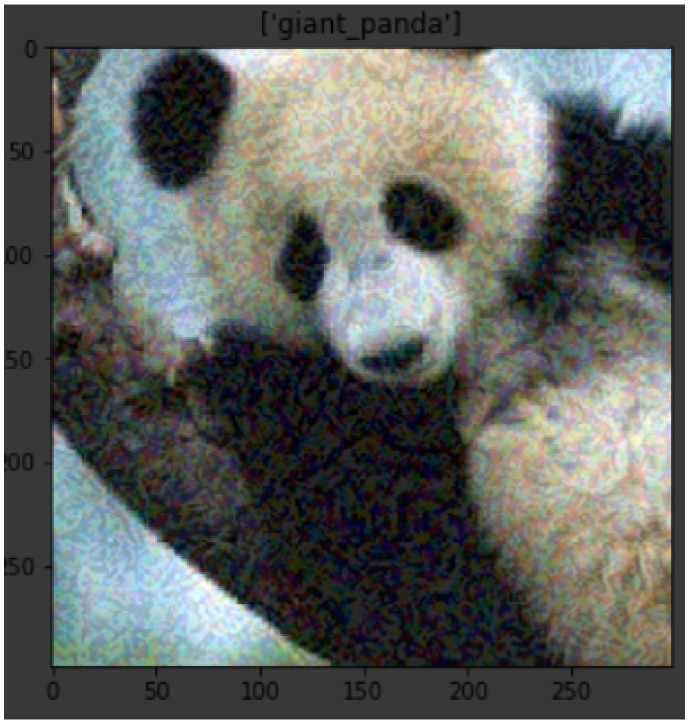


(b) Epsilon - 0.2

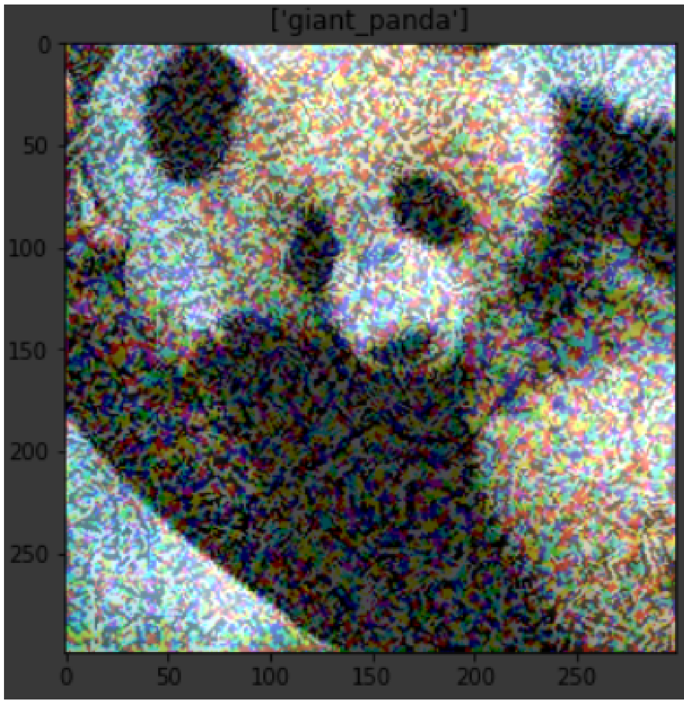


(c) Epsilon - 0.7

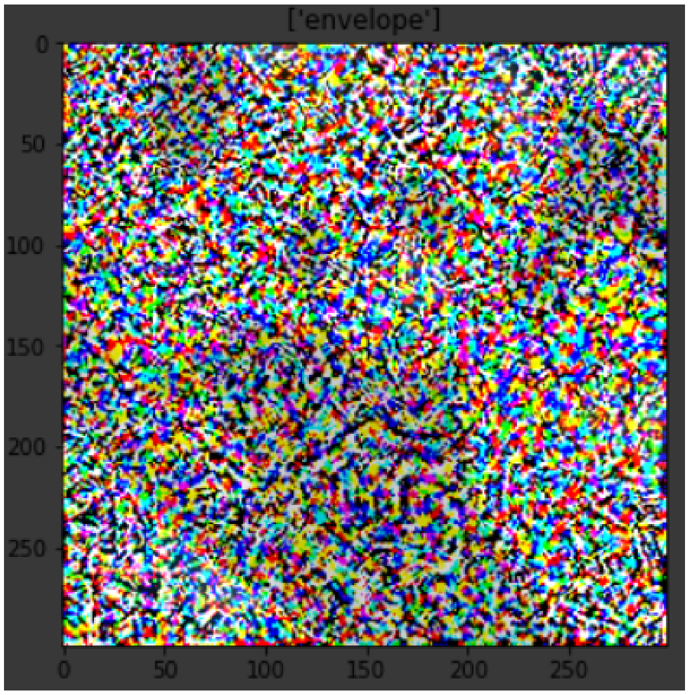
Figure 5.2: FGSM's impact on sample CIFAR10 Input batch, with predicted outputs



(a) Epsilon - 0.07



(b) Epsilon - 0.2



(c) Epsilon - 0.7

Figure 5.3: FGSM's impact on ImageNet's Input Image of giantPanda, with predicted outputs

Analysis -

From the first sight it can be inferred that FGSM attack is able to quickly generate adversarial image using the gradient information & the attack strength or misclassification accuracy is completely dependent upon the hyperparameter, *epsilon*. For all the three models, highest misclassification accuracy or least test accuracy of about 0.5, was reached when *epsilon=0.7*. When *epsilon=0.07 & eplsilon=0.2*, ImageNet has a constant accuracy of 100% while CIFAR10 maintains an accuracy of less than 10% in both the cases. But interestingly, MNIST accuracies decreases in some proportion with the increase of epsilon values. Thus, with high values of epsilon, maximum misclassification is possible.

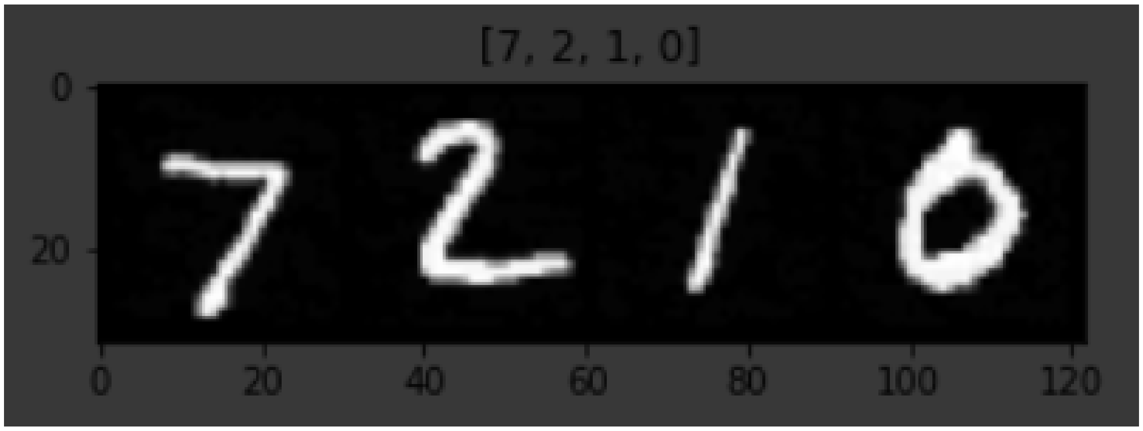
From the perspective of perturbed image, almost zero noise is visible when *epsilon=0.07* & the image is completely blurred or destroyed if *epsilon=0.7* but when *epsilon=0.2*, a mixture of both the true image & the noise can be seen. From these observations, with high values of epsilon although high misclassification rate can be achieved it becomes visually detectable & vice-versa with low epsilon values. Thus, inorder to perform a good attack, one must pick an accuracy that maintains a good balance on both the sides.

Attack Type - IFGSM

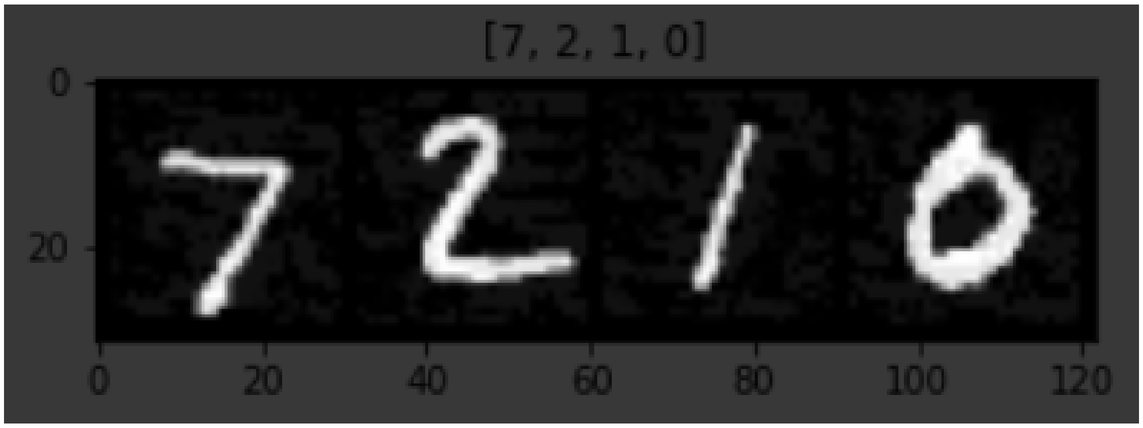
Epsilon / Model	4/255	15/255	40/255
MNIST	96.69	88.83	31.89
CIFAR10	41.59	1.27	0.0
ImageNet	0.0	0.0	0.0

Table 5.2: IFGSM - Test Accuracy Percentage

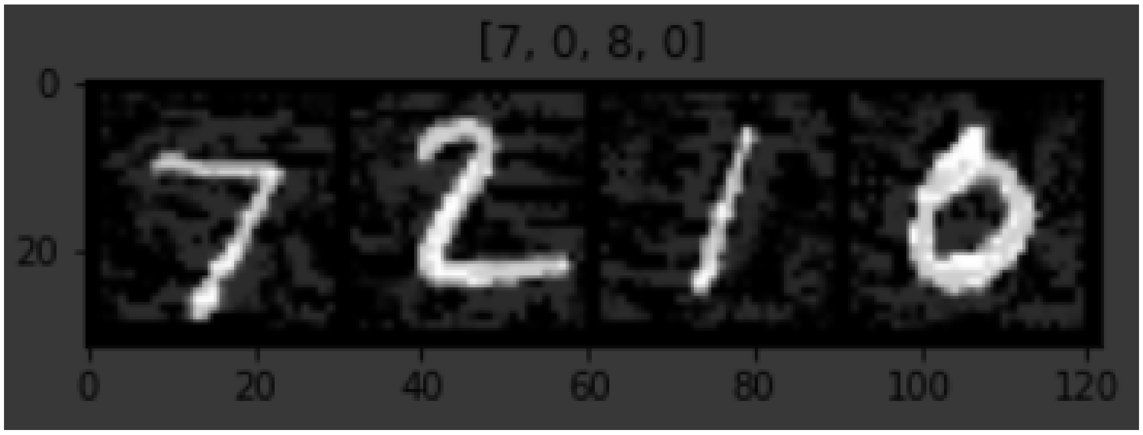
Perturbed Image -



(a) Epsilon - 4/255

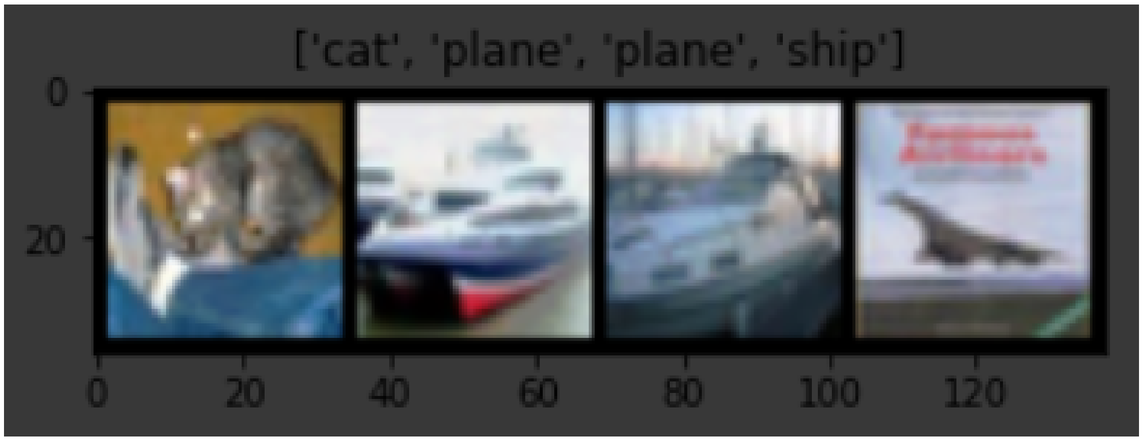


(b) Epsilon - 15/255

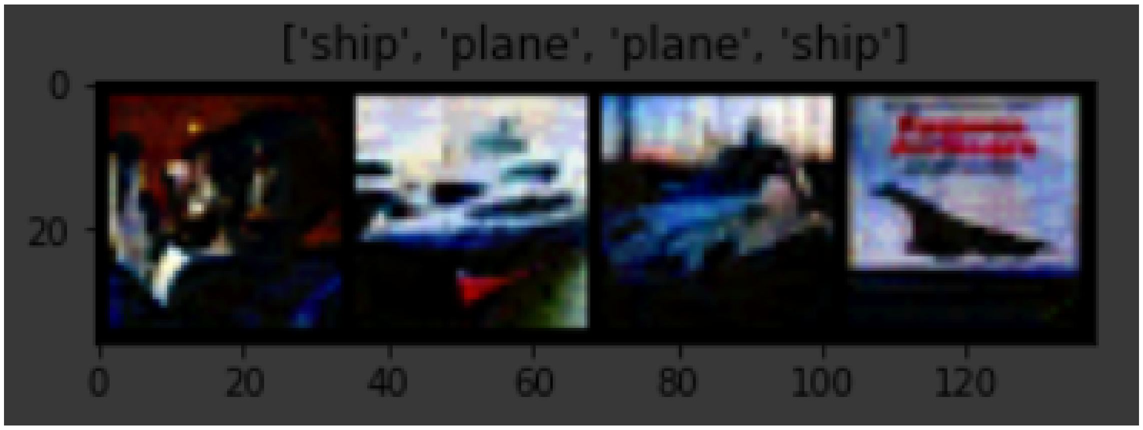


(c) Epsilon - 40/255

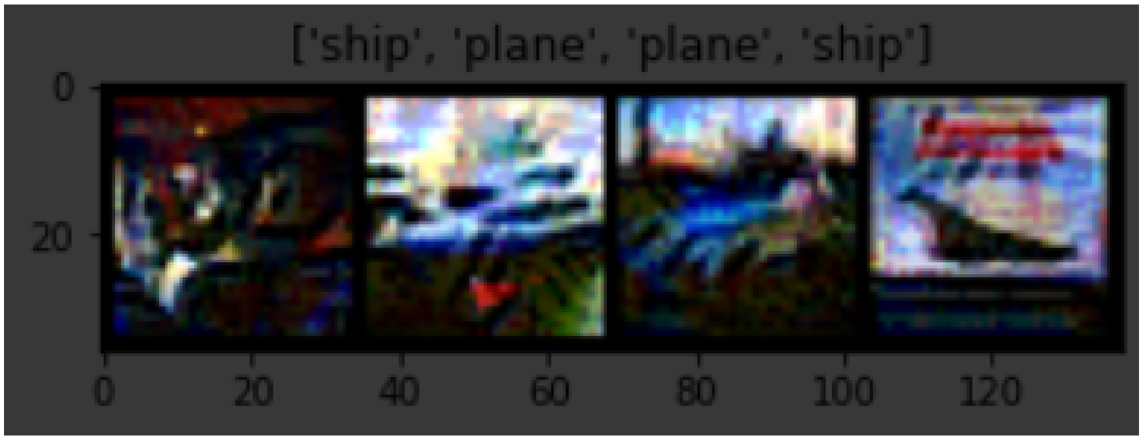
Figure 5.4: IFGSM's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) Epsilon - 4/255

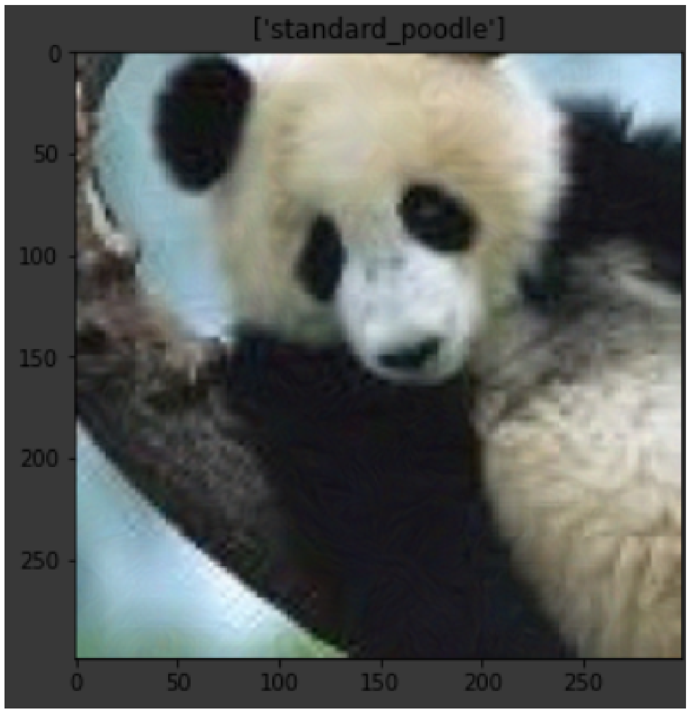


(b) Epsilon - 15/255

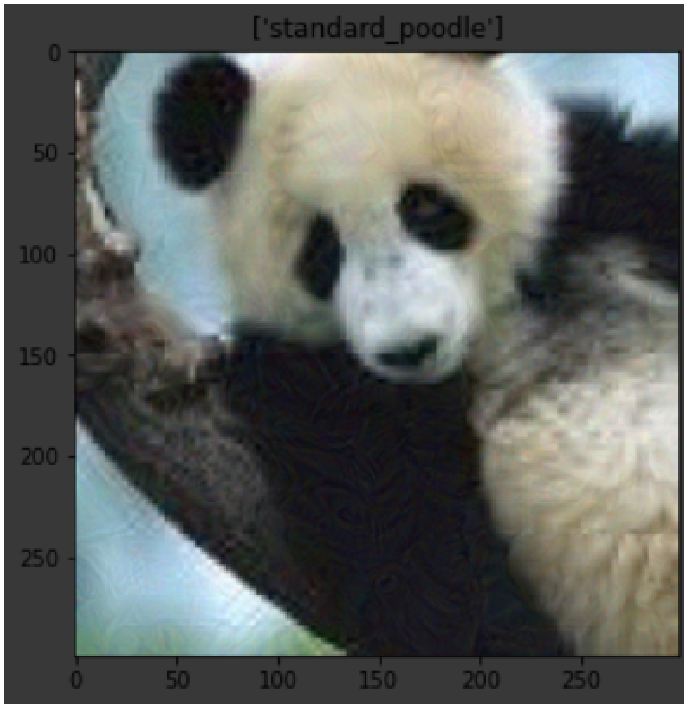


(c) Epsilon - 40/255

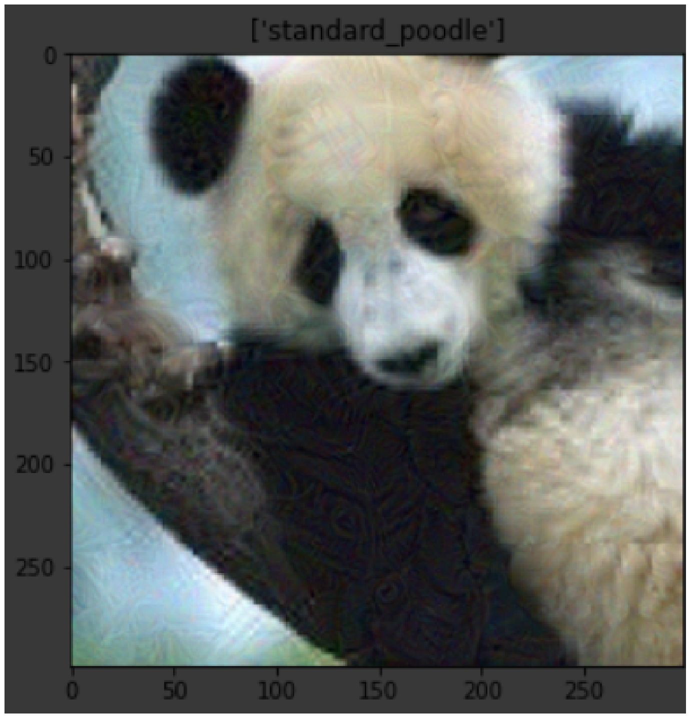
Figure 5.5: IFGSM's impact on sample CIFAR10 input batch of [cat,ship,ship,plane], with predicted outputs



(a) Epsilon - 4/255



(b) Epsilon - 15/255



(c) Epsilon - 40/255

Figure 5.6: IFGSM's impact on ImageNet's Input Image of giantPanda, with predicted outputs

Analysis -

Despite of being the fact that IFGSM is basically an Iterative version of FGSM, we can notice considerable differences in the impact of the attack. To begin with test-accuracy, ImageNet always maintains a constant accuracy of zero while CIFAR10 showing a sharp dip in the accuracy with just a minor shift of *epsilon from 4/255 to 15/255* & finally reaches zero with *epsilon=40/255*. It is the MNIST alone, which shows accuracy variations directly proportional with respect to the epsilon values & it never reached zero in any case.

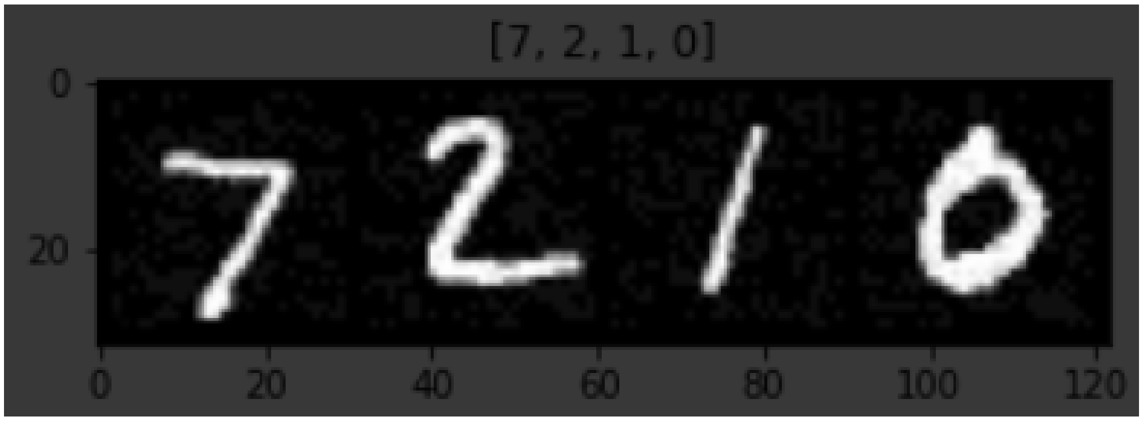
In case of perturbed images, no complete destruction or blurring of the input image is seen in any case with these epsilon value set & when epsilon=40/255, we can see the embedded noise to a small extent. From this, we can tell that IFGSM is bit more powerful than the plain FGSM.

Attack Type - RFGSM

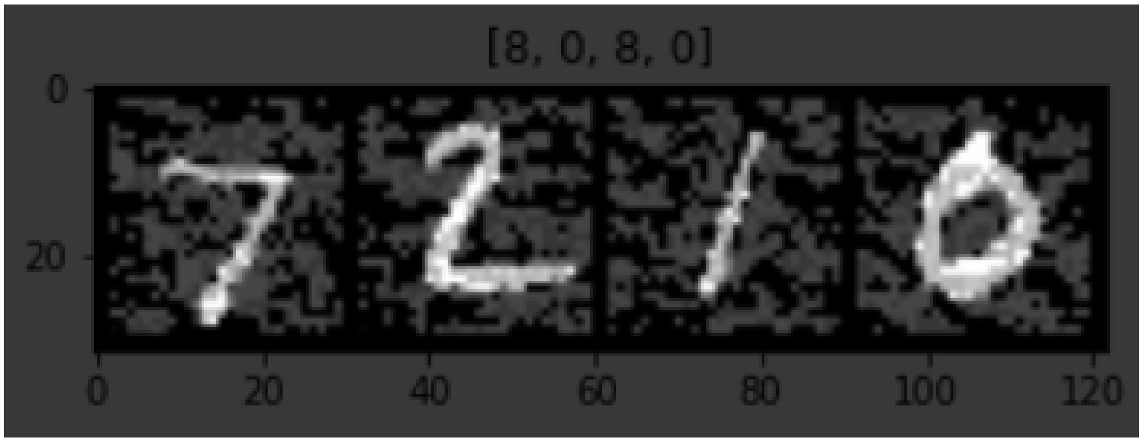
Model \ Epsilon	16/255	70/255	160/255
MNIST	95.91	22.02	0.33
CIFAR10	17.85	1.3	0.46
ImageNet	0.0	100.0	0.0

Table 5.3: RFGSM - Test Accuracy Percentage

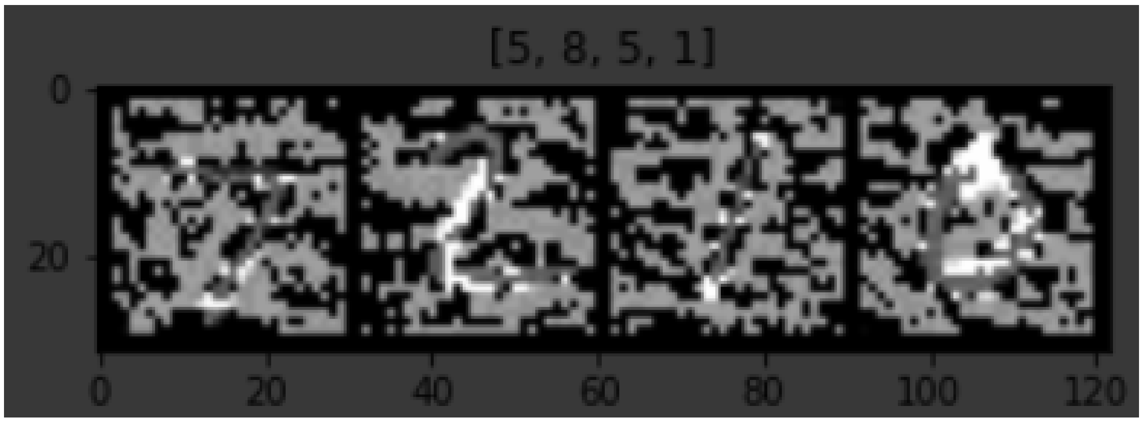
Perturbed Image -



(a) Epsilon - 16/255

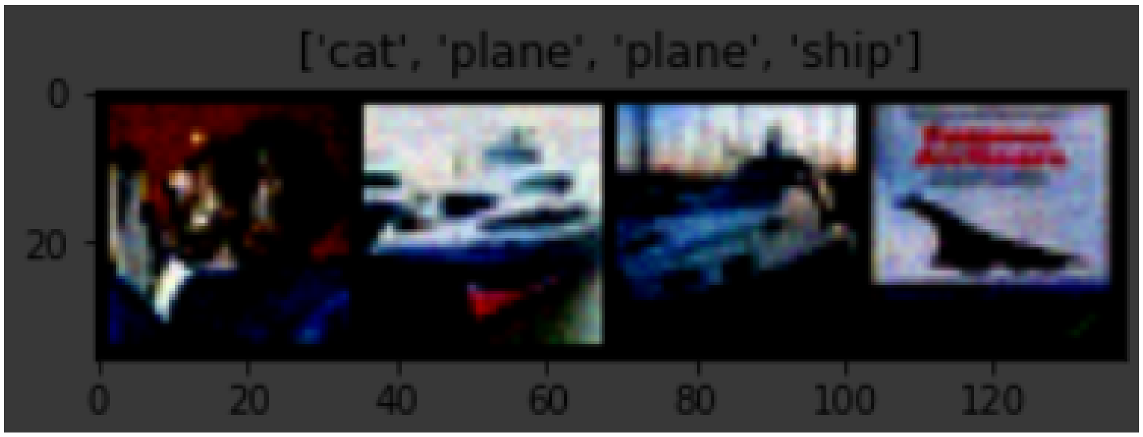


(b) Epsilon - 70/255

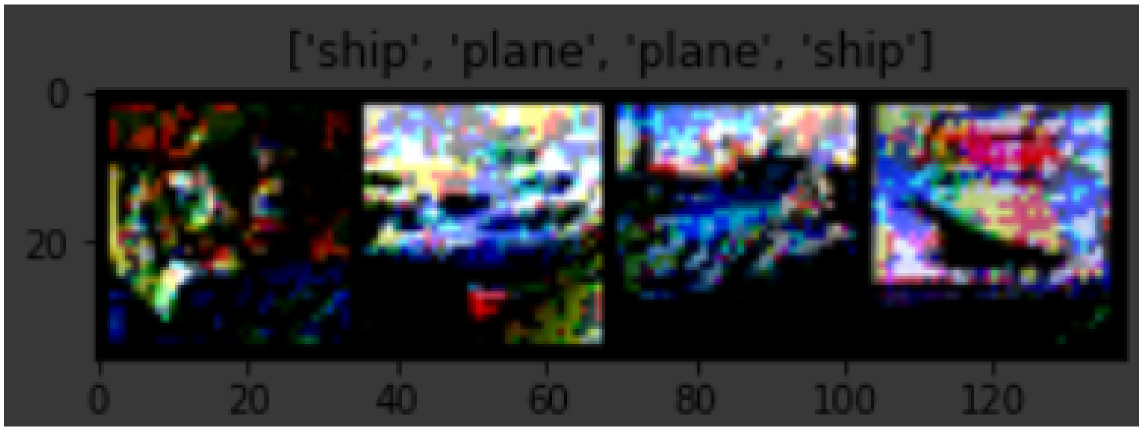


(c) Epsilon - 160/255

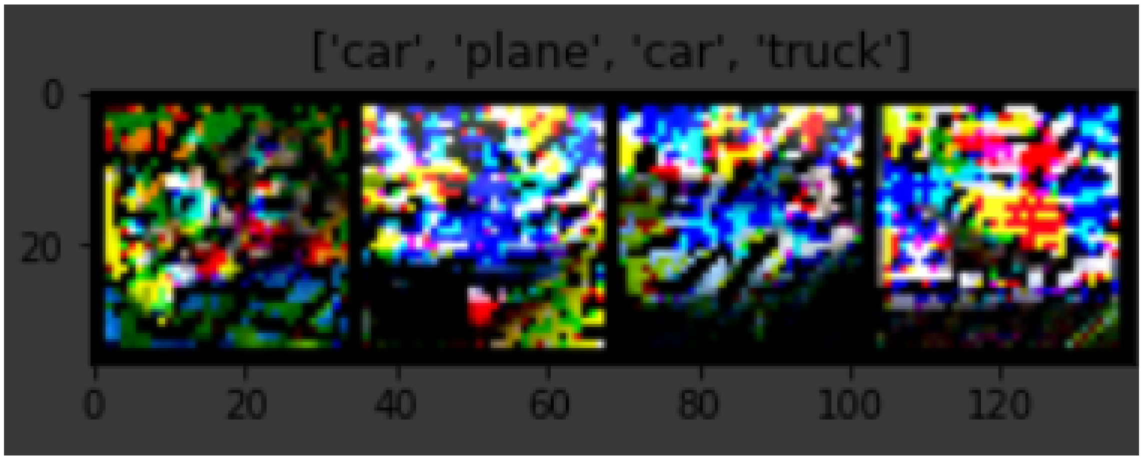
Figure 5.7: RFGSM's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) Epsilon - 16/255

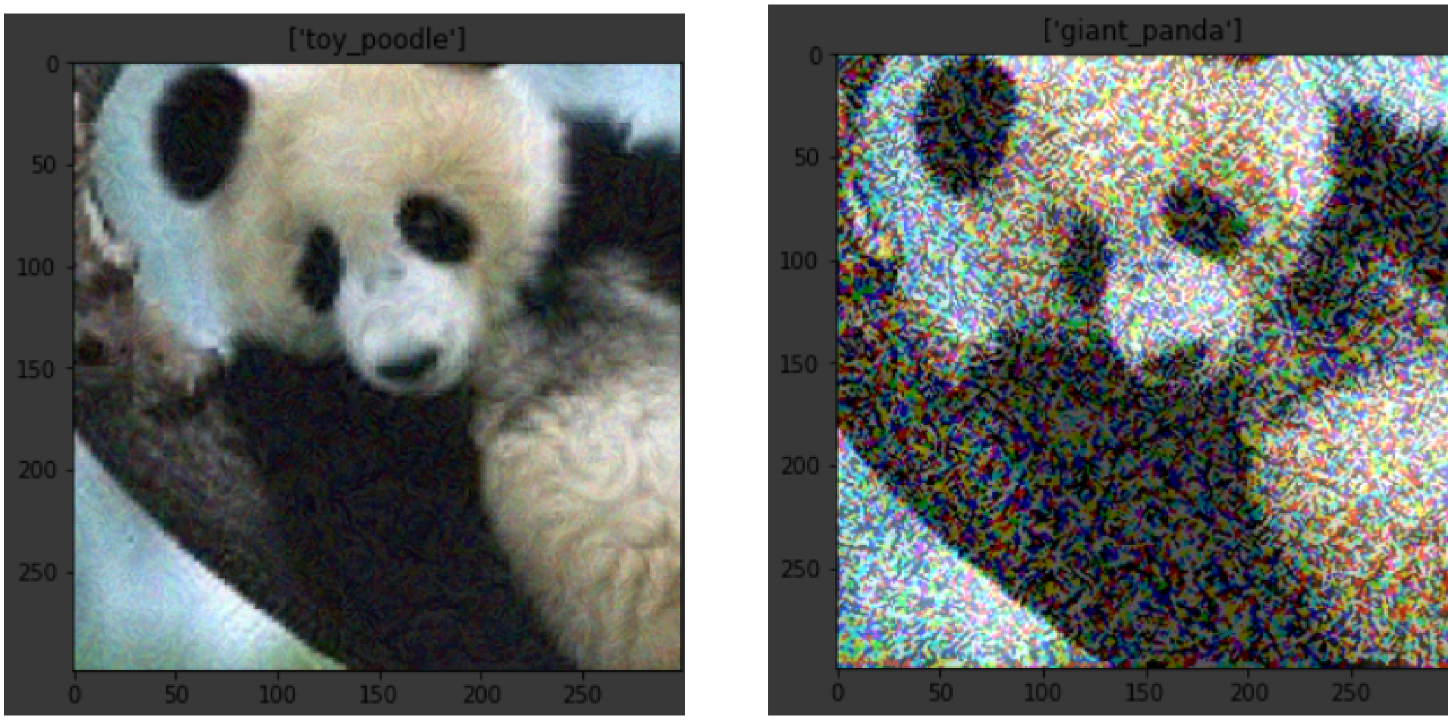


(b) Epsilon - 70/255



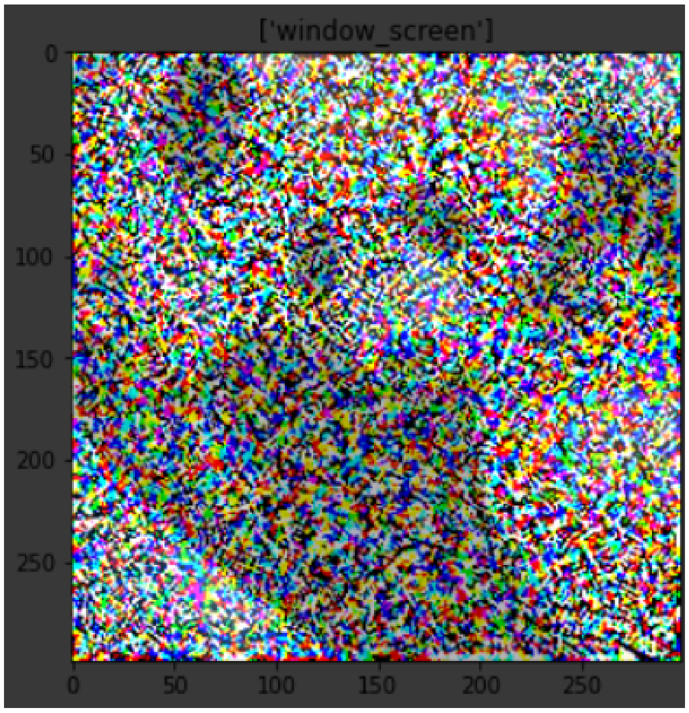
(c) Epsilon - 160/255

Figure 5.8: RFGSM's impact on sample CIFAR10 input batch of [cat,ship,ship,plane], with predicted outputs



(a) Epsilon - 16/255

(b) Epsilon - 70/255



(c) Epsilon - 160/255

Figure 5.9: RFGSM's impact on ImageNet's Input Image of giantPanda, with predicted outputs

Analysis -

With respect to the Test-Accuracy, there are some interesting results here as compared with the previous attacks. To begin with even with the least epsilon of 16/255, CIFAR10 shows an accuracy of 17.85% but for higher epsilon values, it becomes *less than 1*. In accordance with previous cases, MNIST varies in proportion with epsilon values. Surprisingly, ImagNet shows an accuracy of about 100% for *epsilon=70/255* & zero for other cases.

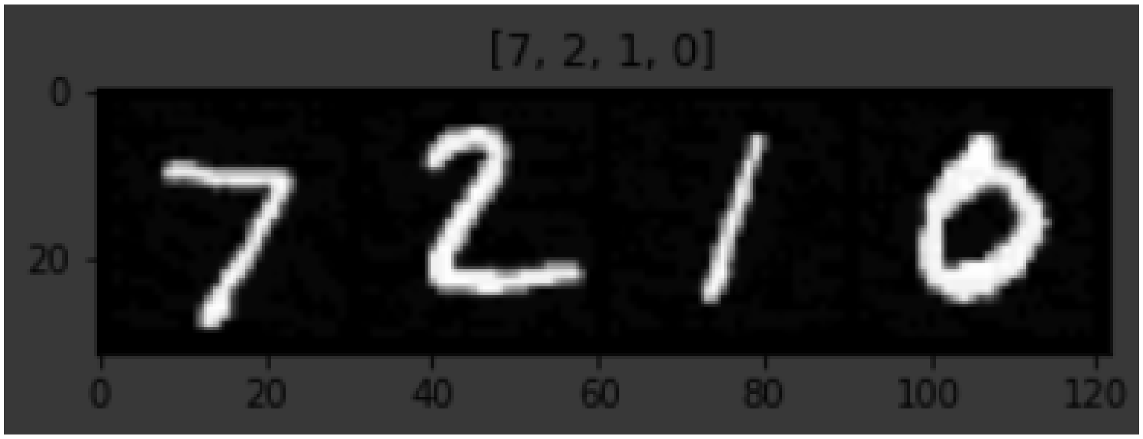
In case of perturbed images, the result is very similar to plain FGSM attack, in which the noise is clearly visible with high epsilon values but just to note that when epsilon=160/255, the image is bit more visible as opposed to the complete destruction in case of FGSM.

Attack Type - PGD

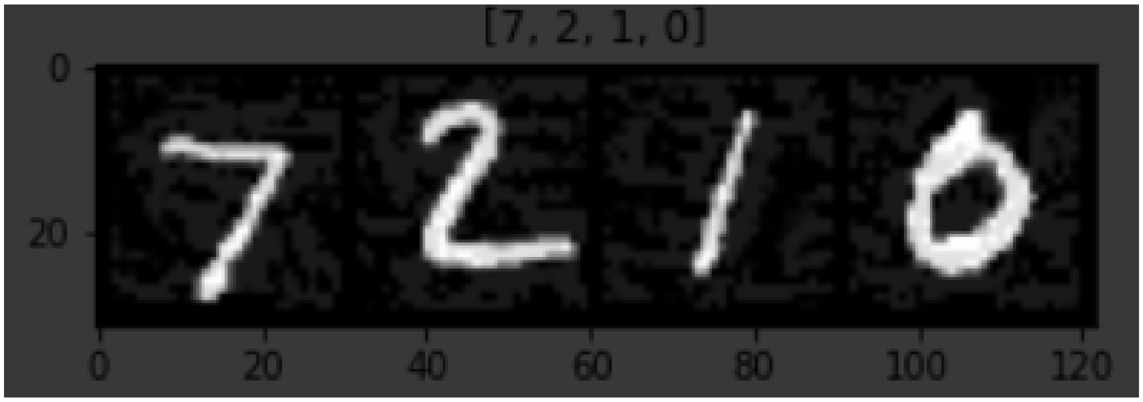
Epsilon / Model	0.03	0.1	0.3
MNIST	95.85	74.58	0.12
CIFAR10	10.00	0.36	0.0
ImageNet	0.0	0.0	0.0

Table 5.4: PGD - Test Accuracy Percentage

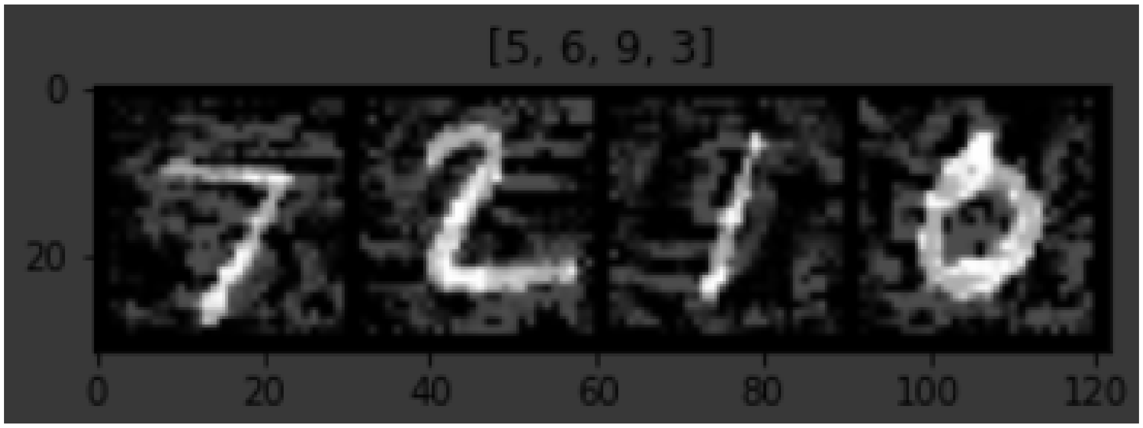
Perturbed Image -



(a) Epsilon - 0.03

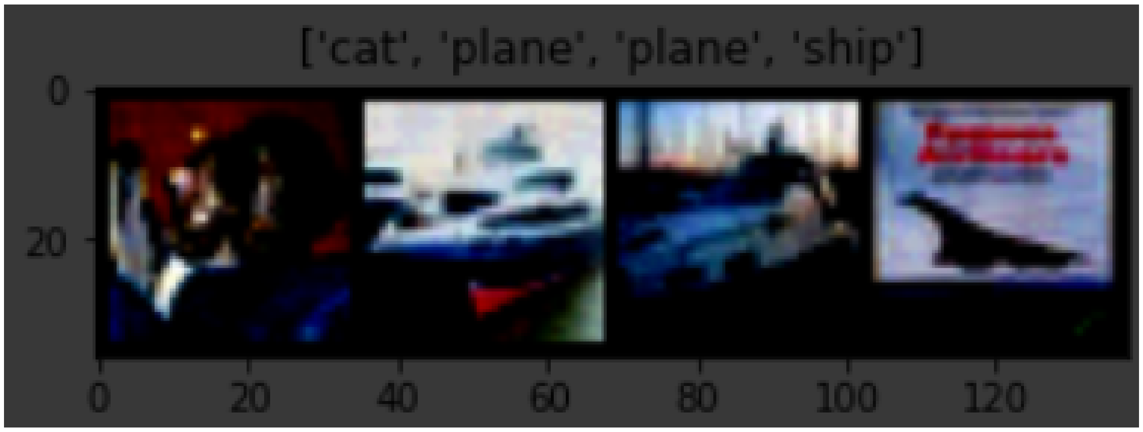


(b) Epsilon - 0.1

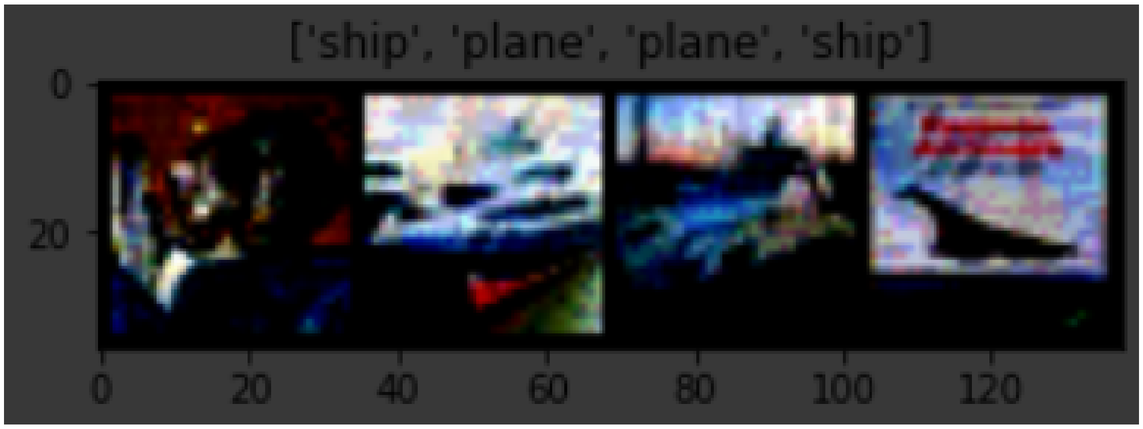


(c) Epsilon - 0.3

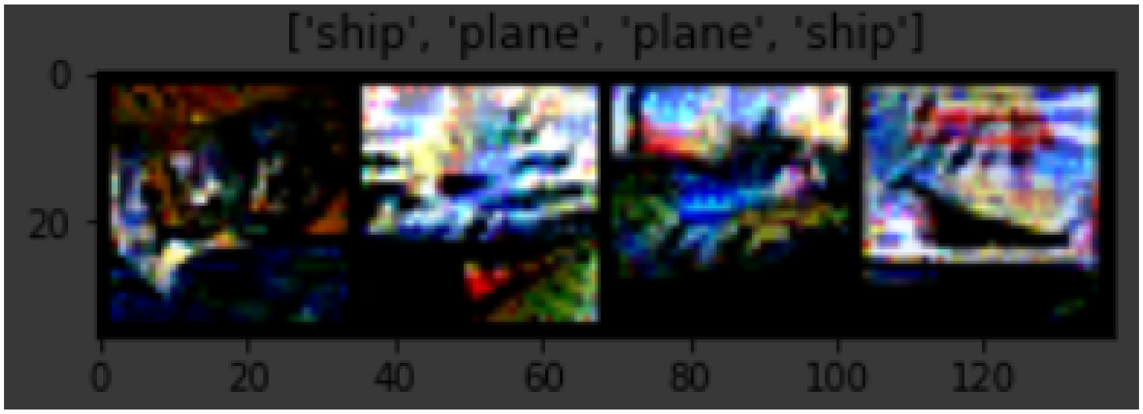
Figure 5.10: PGD's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) Epsilon - 0.03

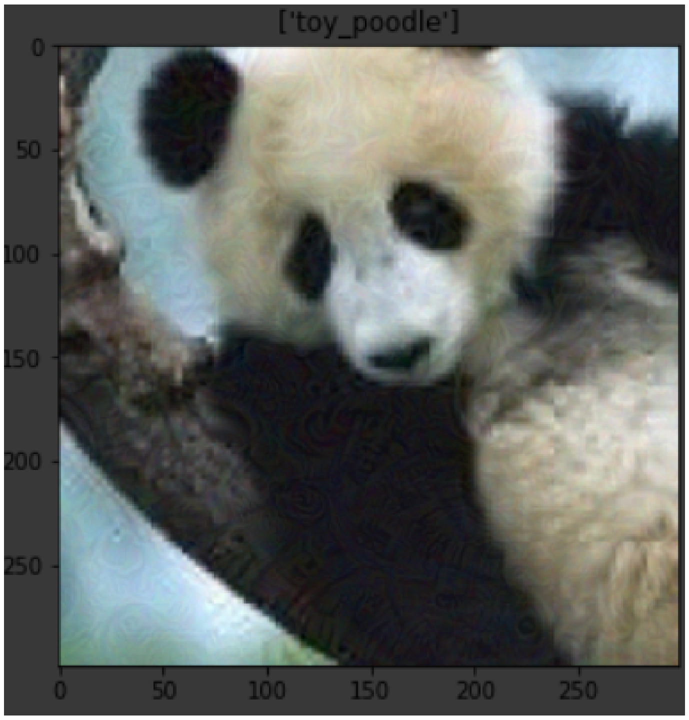


(b) Epsilon - 0.1

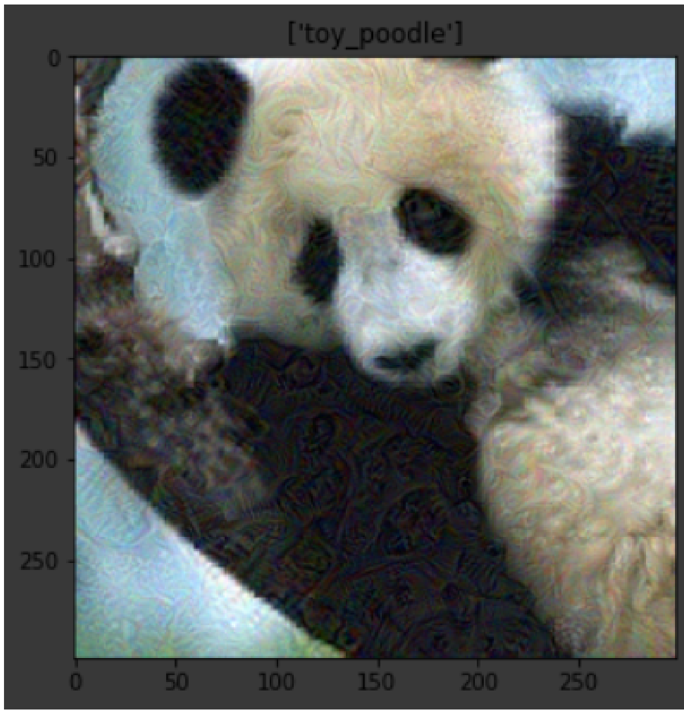


(c) Epsilon - 0.3

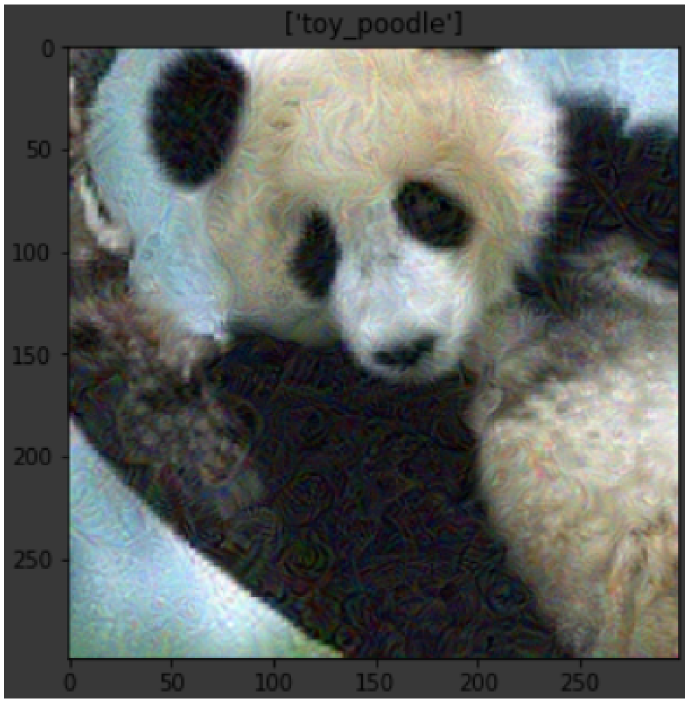
Figure 5.11: PGD's impact on sample CIFAR10 input batch of [cat,ship,ship,plane], with predicted outputs



(a) Epsilon - 0.03



(b) Epsilon - 0.1



(c) Epsilon - 0.3

Figure 5.12: PGD's impact on sample ImageNet image of giantPanda, with predicted outputs

Analysis -

Here ImagNet shows a constant accuracy of 0 in all the cases whereas CIFAR10 shows an accuracy of 10% when epsilon=0.03 but in the remaining cases it is close to zero. Even in case of MNIST, it doesn't vary in the normal pattern as compared with previous cases & note that just with a shift of 0.07 in the epsilon, accuracy shows a dip of about 20%.

In case of Image perturbation, there isn't much noise visible except for the case of epsilon=0.3 with MNIST in which a small amount of noise is still visible.

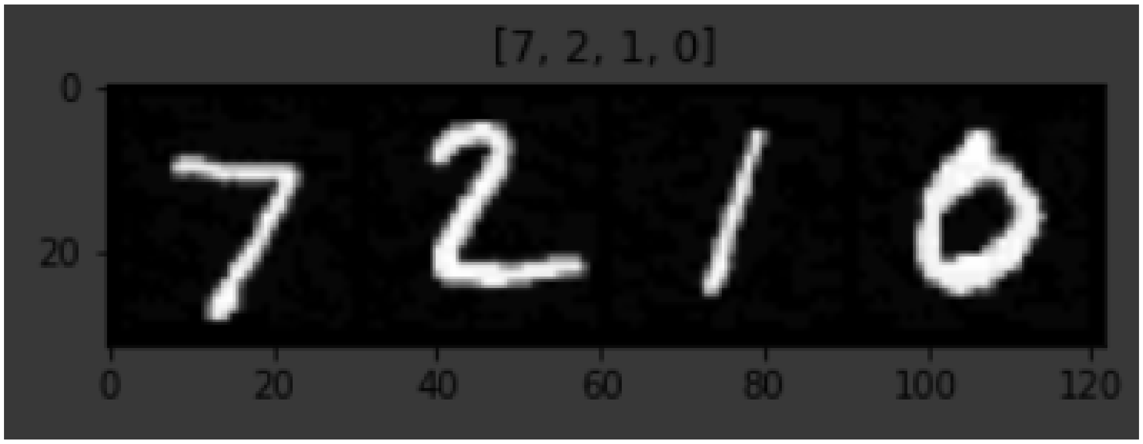
From this, it is not wrong to say that PGD is quite strong & powerful attack than compared with all the variants of FGSM as it can maintain a good balance between the two aspects of the attack.

Attack Type - RPGD

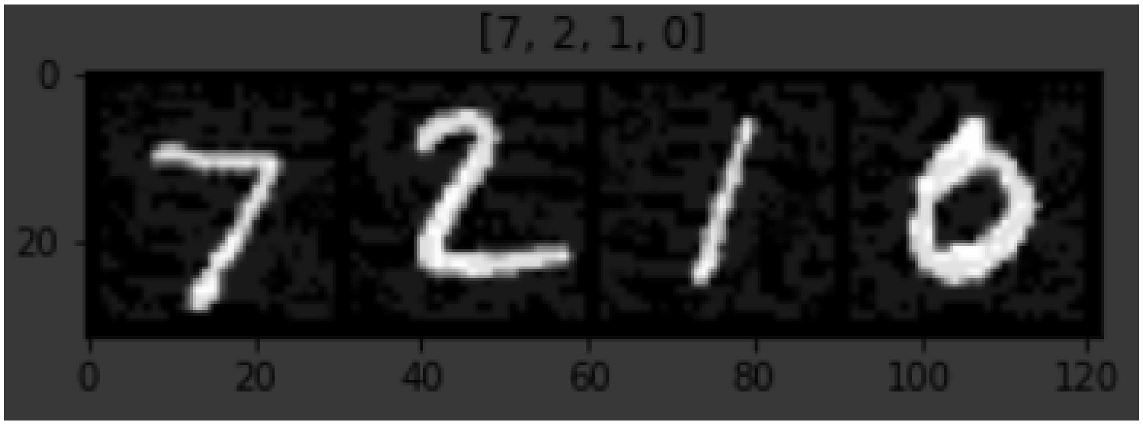
Epsilon / Model	0.03	0.1	0.3
MNIST	95.88	75.27	0.22
CIFAR10	9.96	0.37	0.0
ImageNet	0.0	0.0	0.0

Table 5.5: RPGD - Test Accuracy Percentage

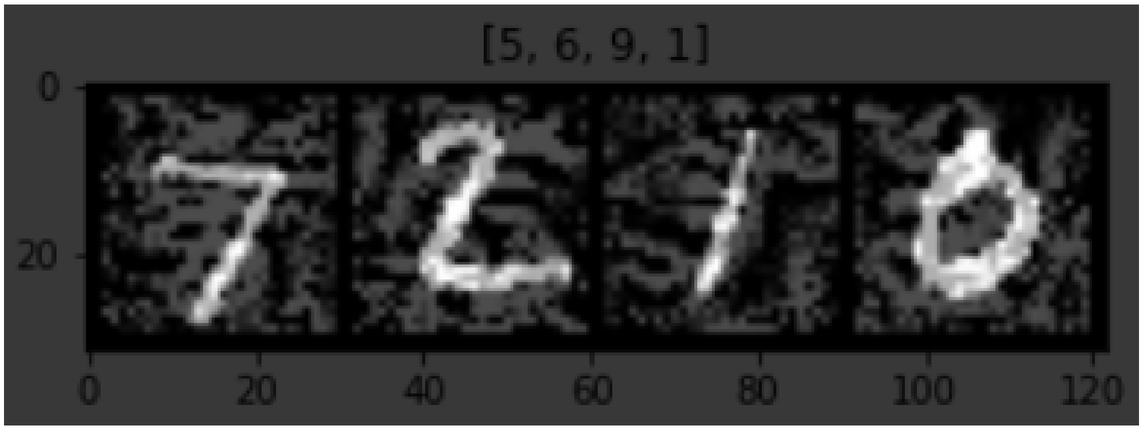
Perturbed Image -



(a) Epsilon - 0.03

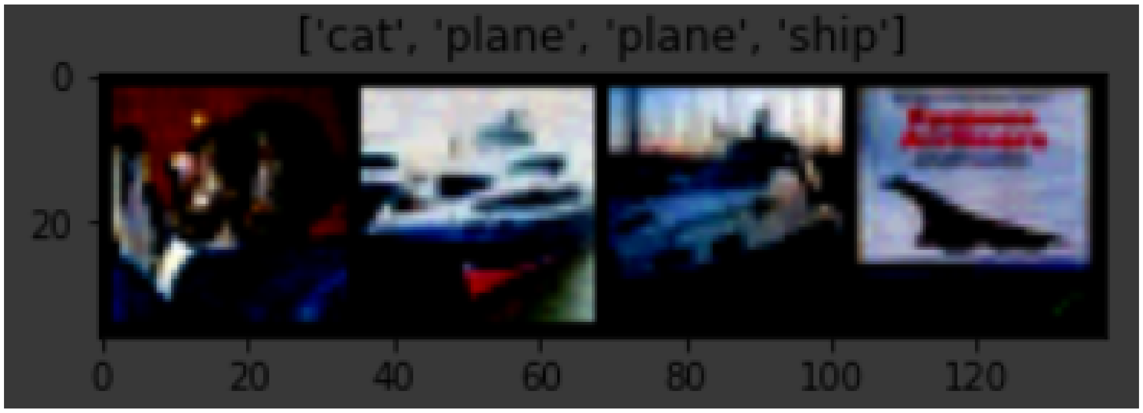


(b) Epsilon - 0.1

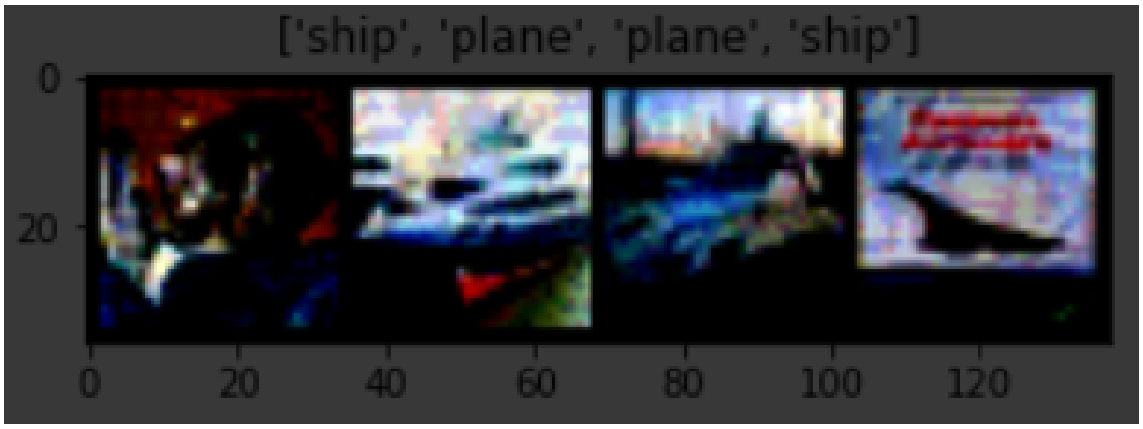


(c) Epsilon - 0.3

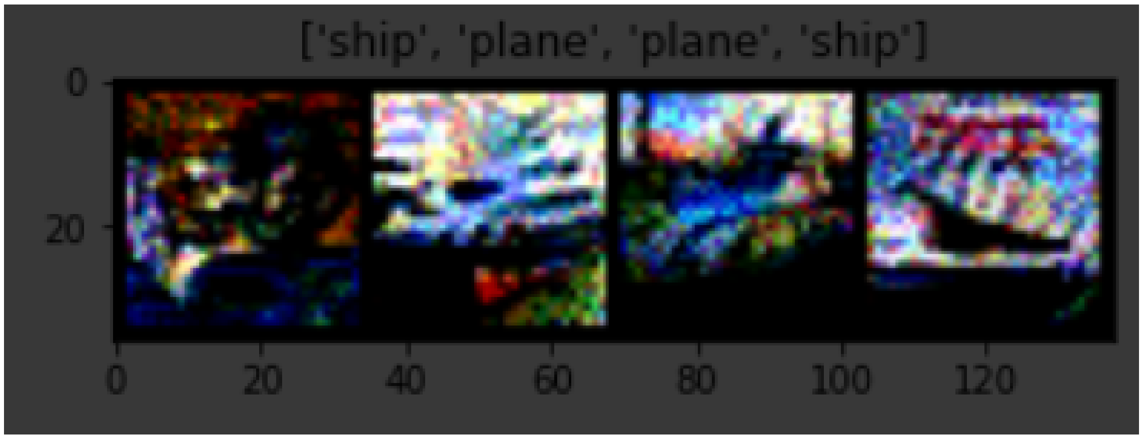
Figure 5.13: RPGD's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) Epsilon - 0.03

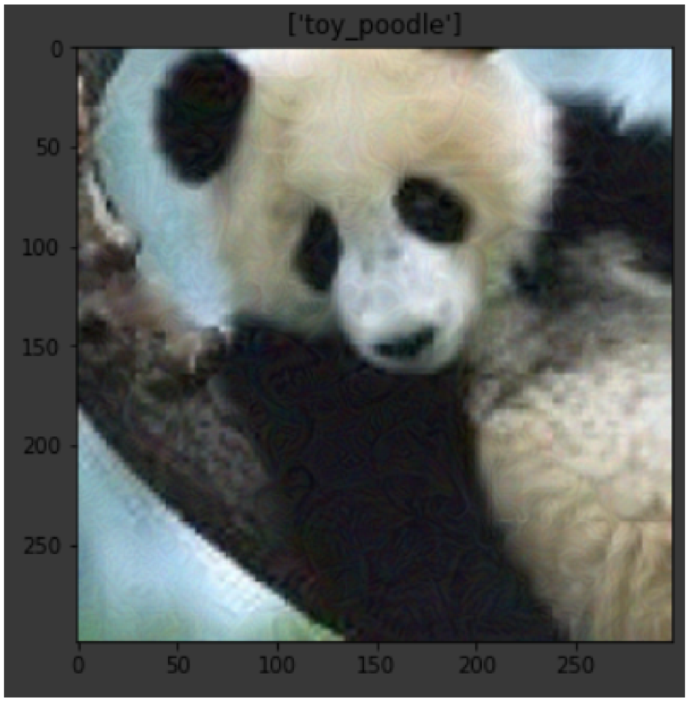


(b) Epsilon - 0.1

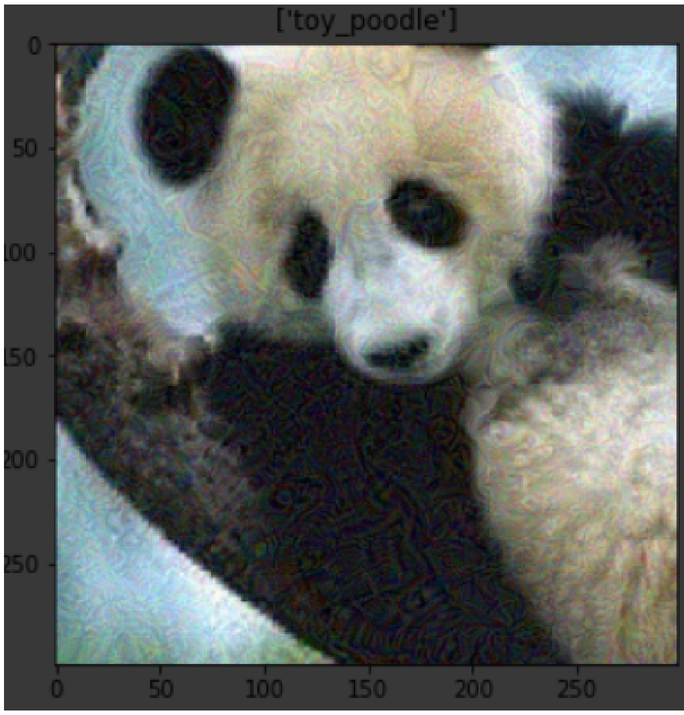


(c) Epsilon - 0.3

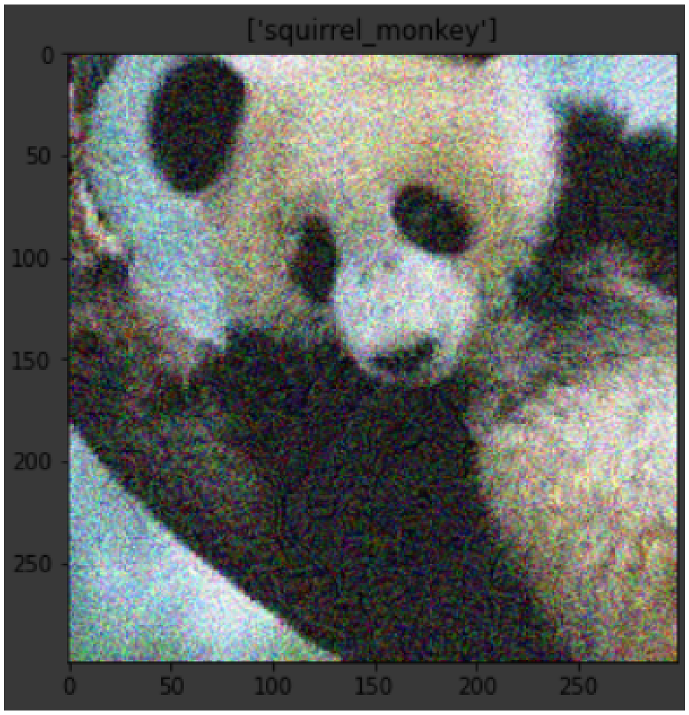
Figure 5.14: RPGD's impact on sample CIFAR10 input batch of [cat,ship,ship,plane], with predicted outputs



(a) Epsilon - 0.03



(b) Epsilon - 0.1



(c) Epsilon - 0.3

Figure 5.15: RPGD's impact on sample ImageNet image of giantPanda, with predicted outputs

Analysis -

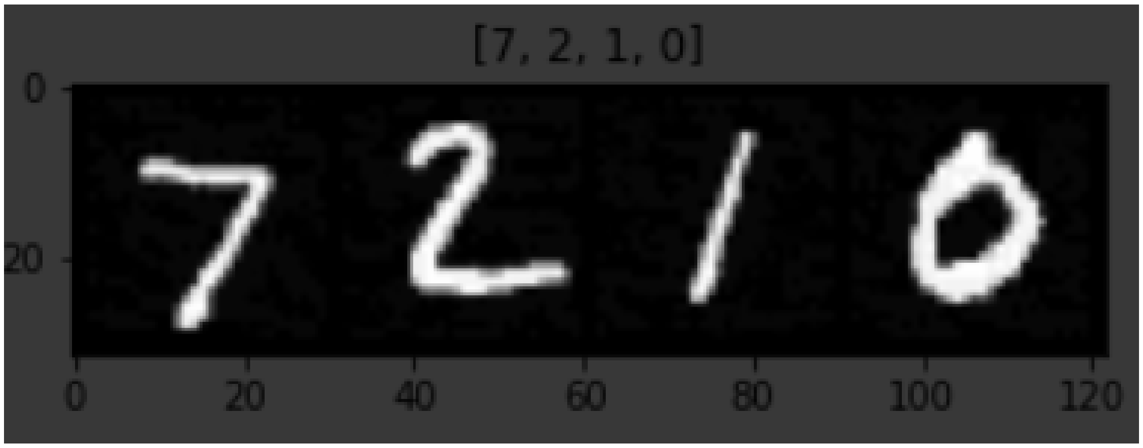
It is surprising to note that the behaviour of RPGD is almost identical to that of naive PGD both in terms of Test-Accuracies & the level of perturbation in the resulting images. This clearly proves the point that other than escaping the local optimum RPGD has no other additional benefit & if the naive PGD can reach the global optimum, then both becomes same.

Attack Type - APGD

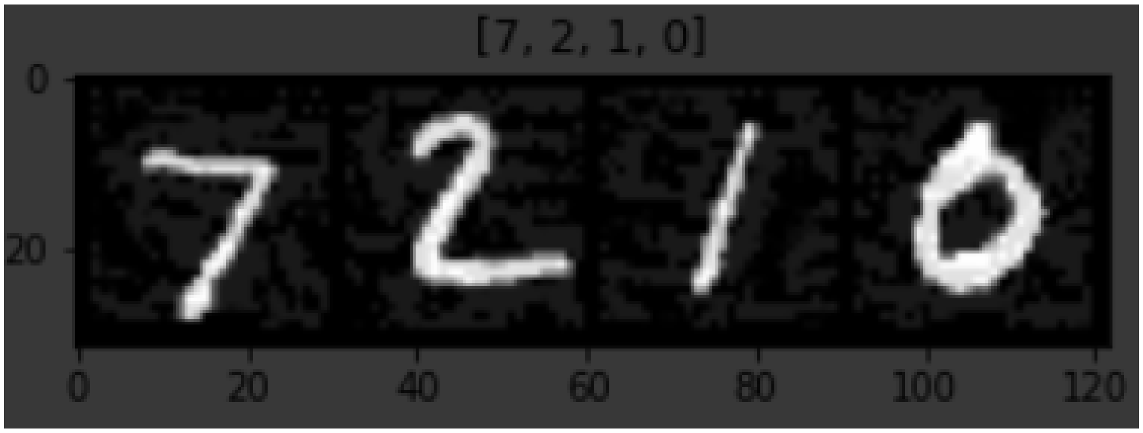
Epsilon / Model	0.03	0.1	0.3
MNIST	95.85	74.58	0.12
CIFAR10	10.00	0.36	0.0
ImageNet	0.0	0.0	0.0

Table 5.6: APGD - Test Accuracy Percentage

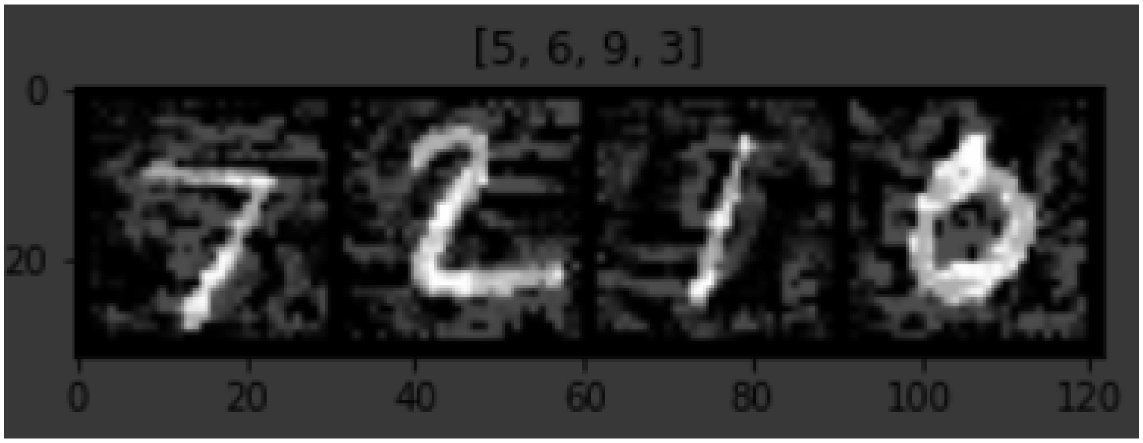
Perturbed Image -



(a) Epsilon - 0.03

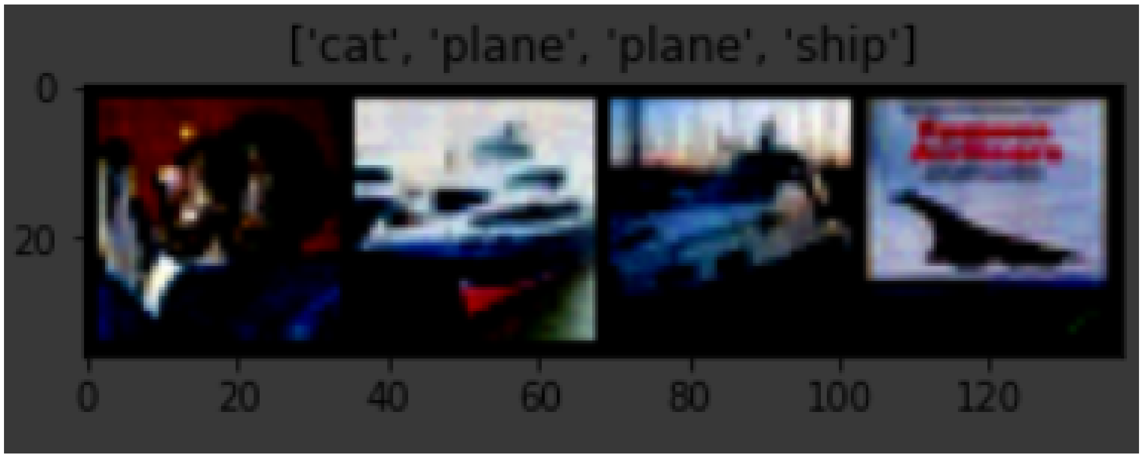


(b) Epsilon - 0.1

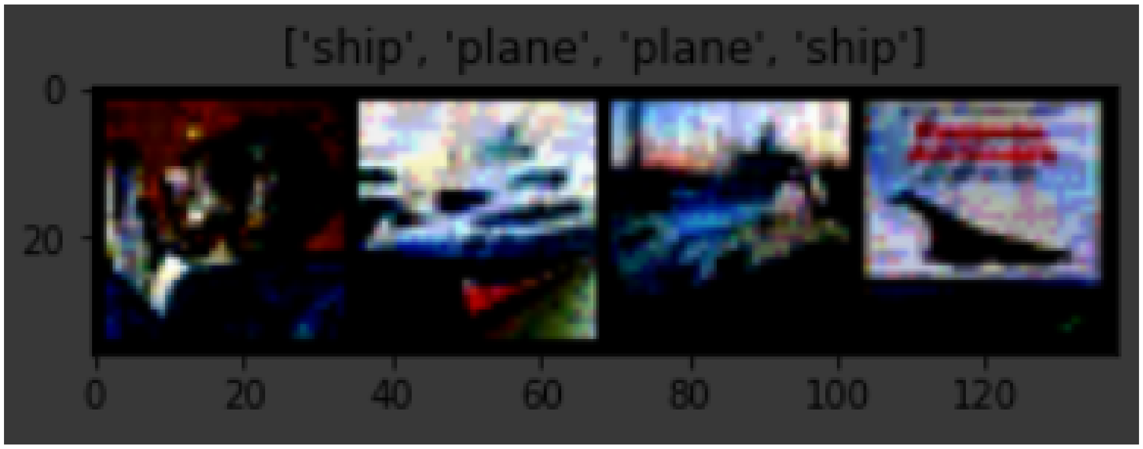


(c) Epsilon - 0.3

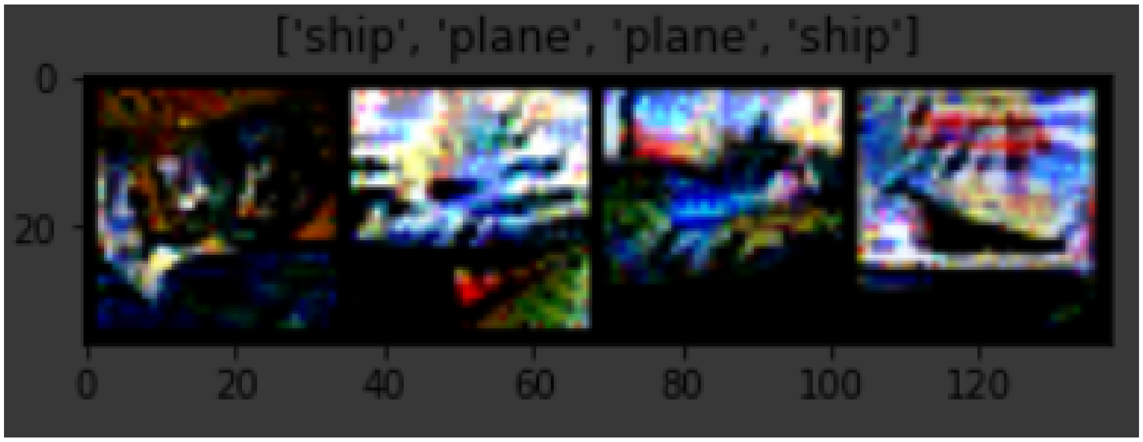
Figure 5.16: APGD's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) Epsilon - 0.03

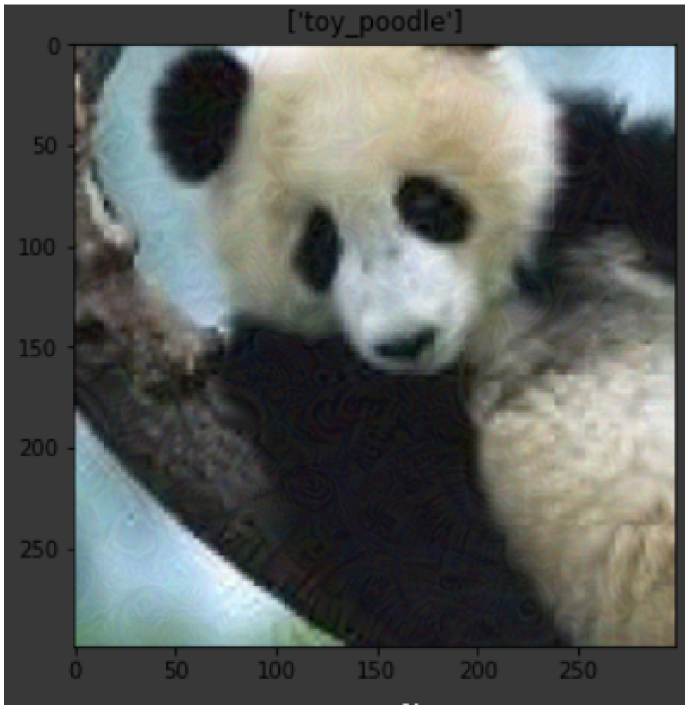


(b) Epsilon - 0.1

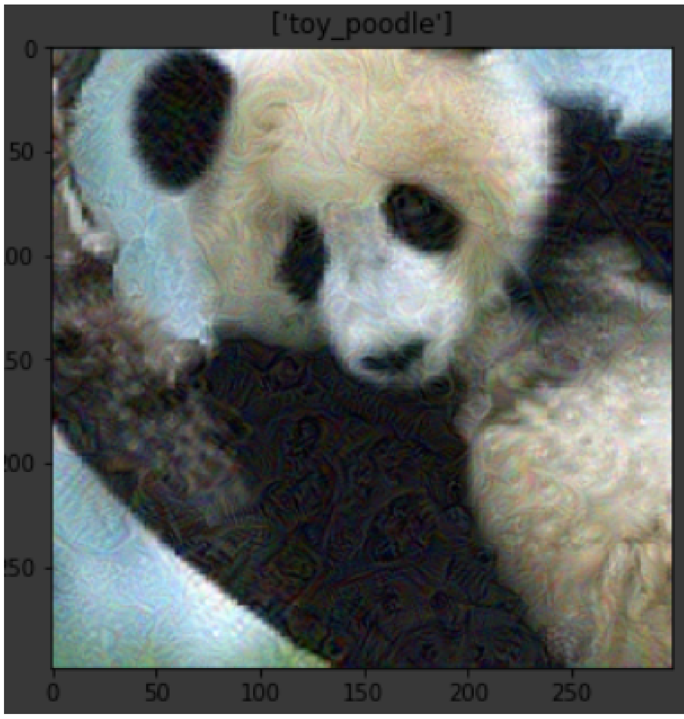


(c) Epsilon - 0.3

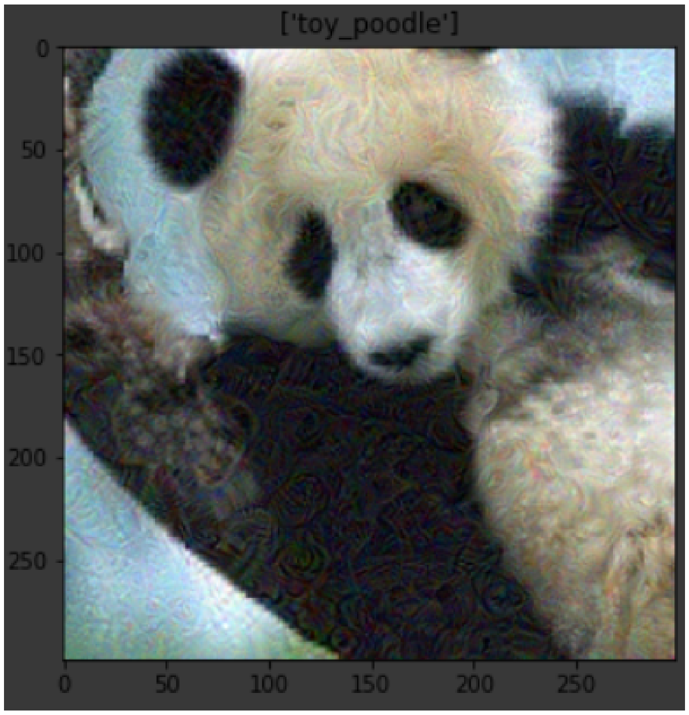
Figure 5.17: APGD's impact on sample CIFAR10 input batch of [cat,ship,ship,plane], with predicted outputs



(a) Epsilon - 0.03



(b) Epsilon - 0.1



(c) Epsilon - 0.3

Figure 5.18: APGD's impact on sample ImageNet image of giantPanda, with predicted outputs

Analysis -

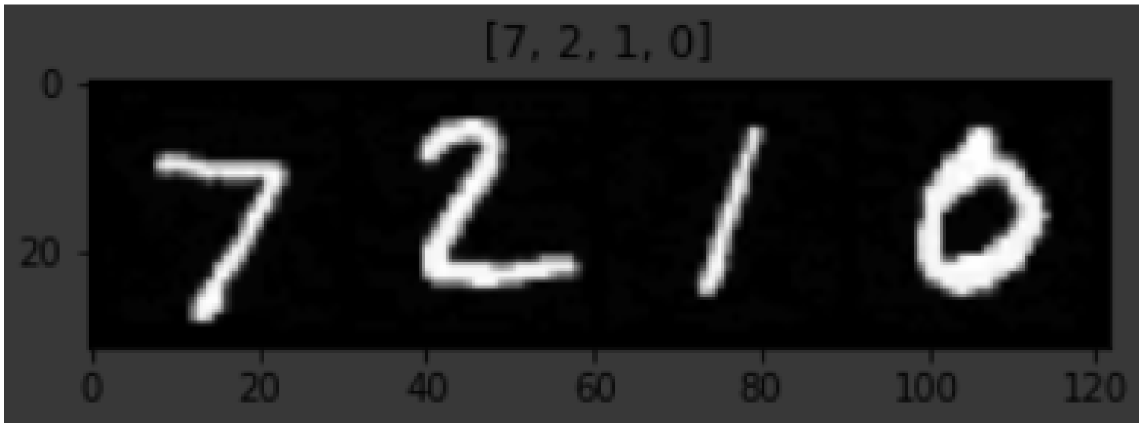
Similar to RPGD, here as well results are interesting that the behaviour is identical to that of PGD & RPGD. From this, it can be inferred that all the flavours of a particular algorithm exist only to solve some specific edge cases which the naive algorithm may fail to handle at times but in the cases in which the naive algorithm handles that perfectly then the outcome/behaviour becomes identical.

Attack Type - ITERLL

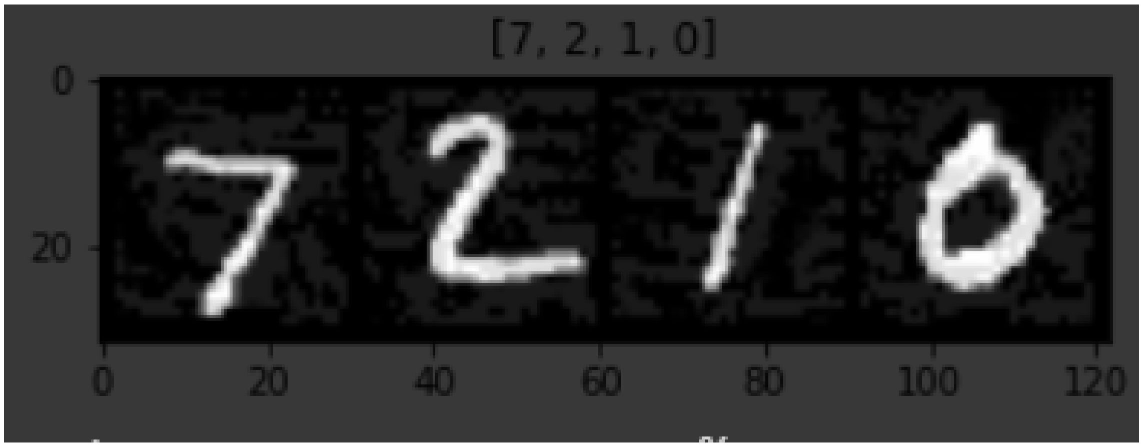
Epsilon / Model	4/255	15/255	40/255
MNIST	97.82	95.63	66.69
CIFAR10	54.94	13.09	1.34
ImageNet	0.0	0.0	0.0

Table 5.7: ITERLL - Test Accuracy Percentage

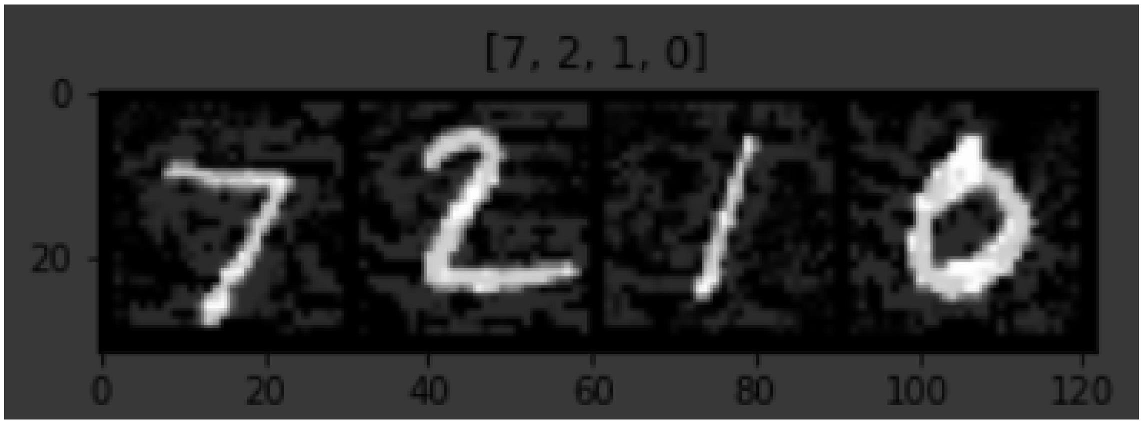
Perturbed Image -



(a) Epsilon - 4/255

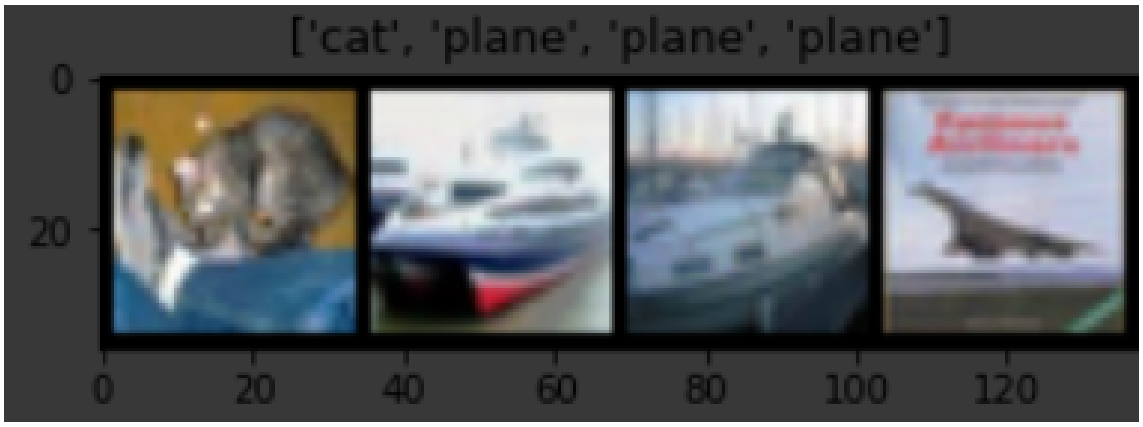


(b) Epsilon - 15/255

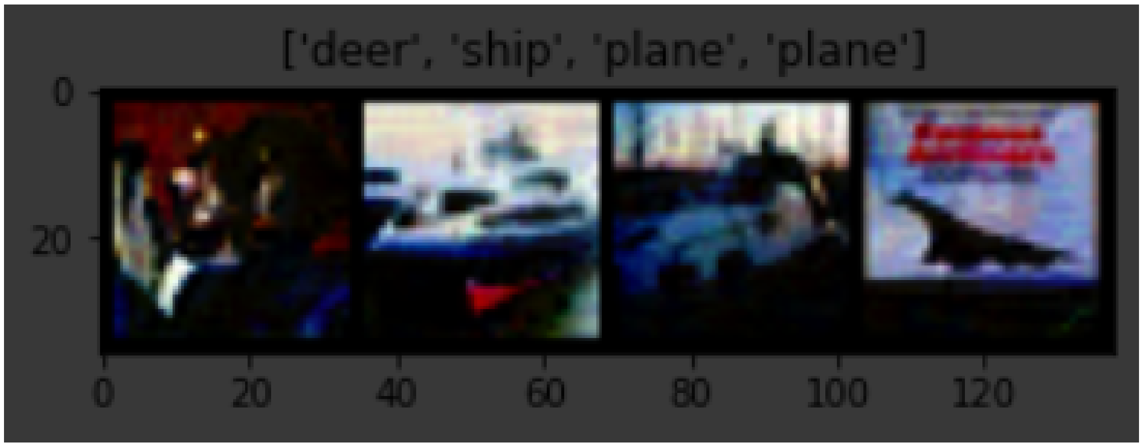


(c) Epsilon - 40/255

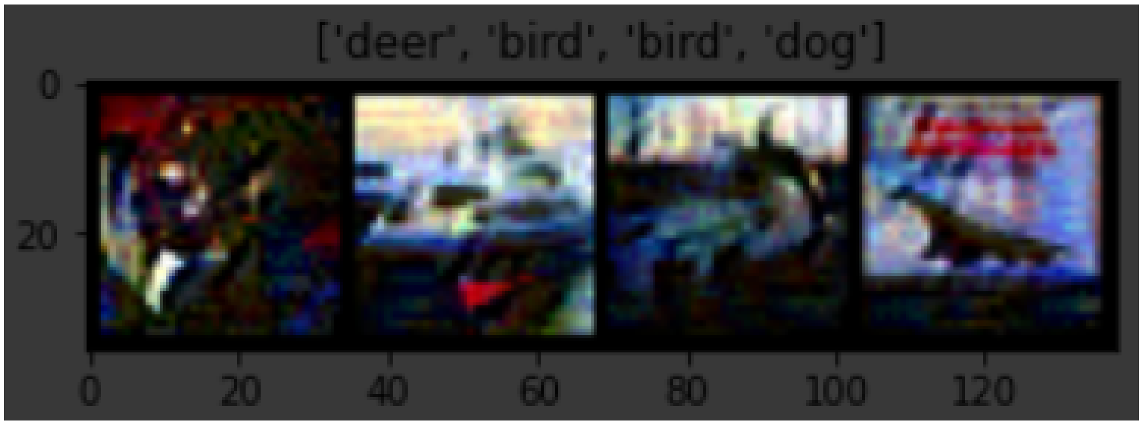
Figure 5.19: ITERLL's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) Epsilon - 4/255

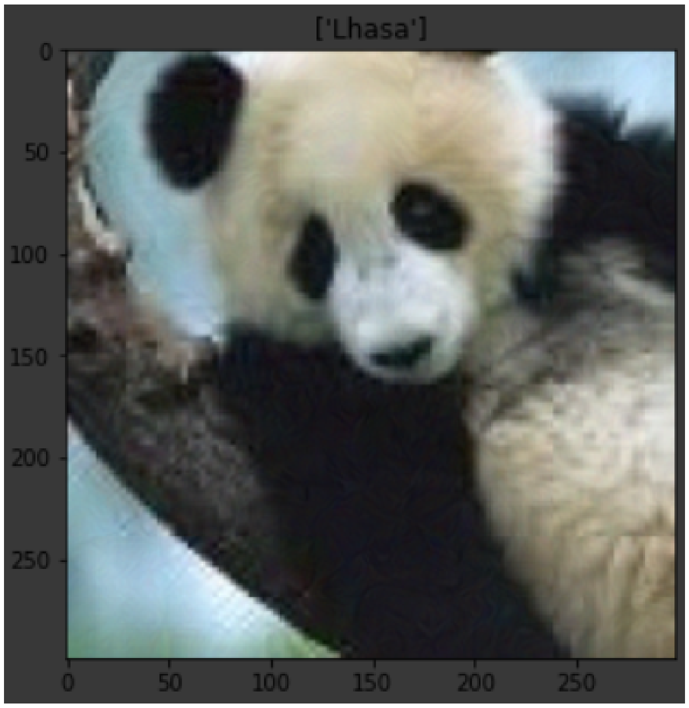


(b) Epsilon - 15/255

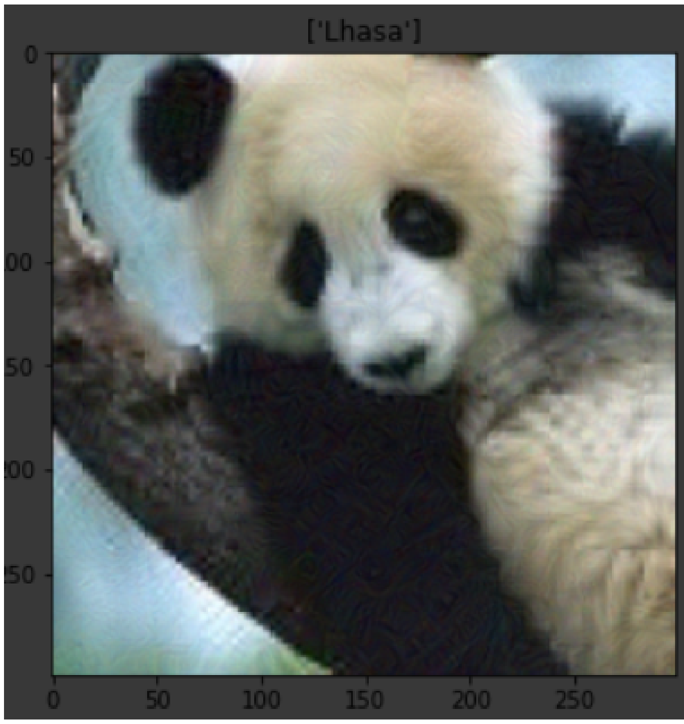


(c) Epsilon - 40/255

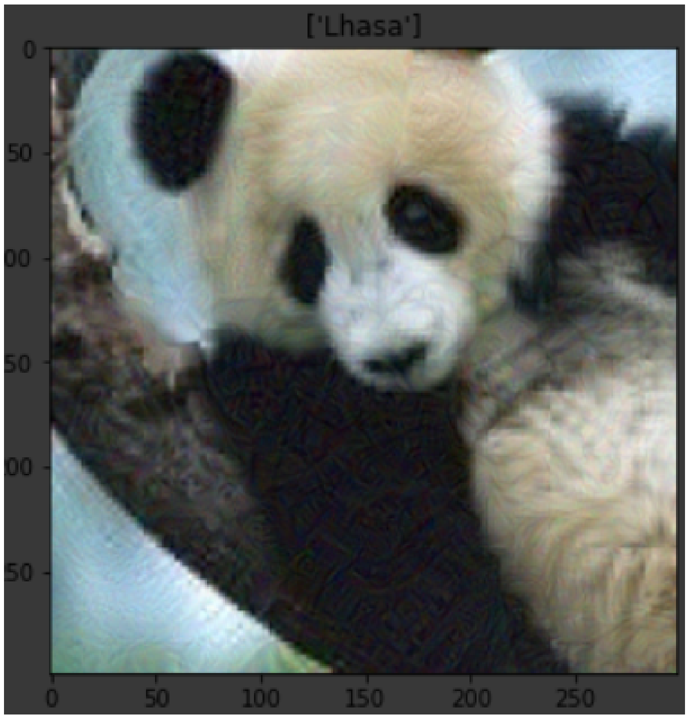
Figure 5.20: ITERLL's impact on sample CIFAR10 Input batch - [cat,ship,ship,plane], with predicted outputs



(a) Epsilon - 4/255



(b) Epsilon - 15/255



(c) Epsilon - 40/255

Figure 5.21: ITERLL's impact on sample ImageNet image of giantPanda, with predicted outputs

Analysis -

Like in the case of FGSM attacks, here as well, CIFAR10 is sensitive to the perturbation level which isn't the case with MNIST. ImageNet shows a constant of zero in all cases. But it has to be noted that here we are forcing the model to predict a specific output & it is not quite easy to achieve.

This attack does a fairly good job with respect to Image perturbation but the noise level is much more visible than PGD.

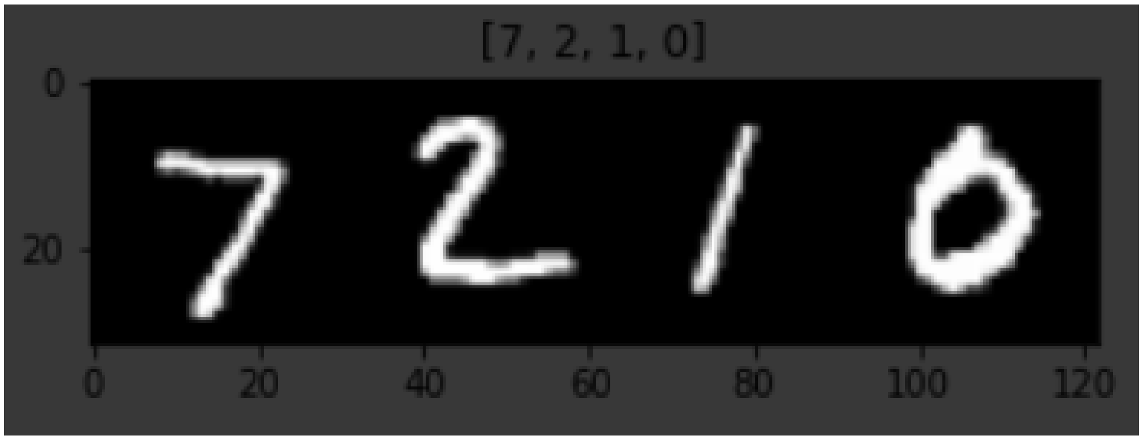
Attack Type - C&W

Model \ C, LR	5, 0.01	8, 0.5	10, 0.01
MNIST	98.0	98.0	98.0
CIFAR10	5.74	3.77	2.76
ImageNet	0.0	0.0	0.0

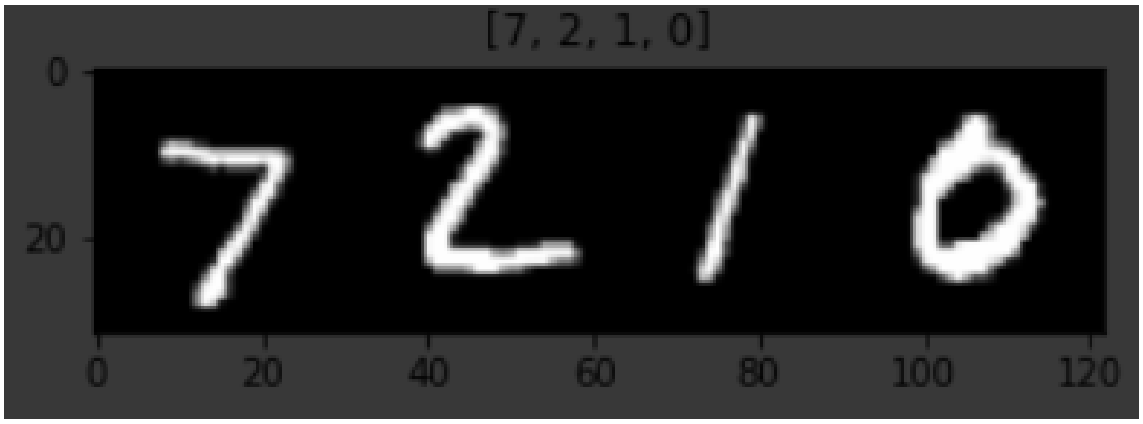
Table 5.8: C&W - Test Accuracy Percentage

NOTE - In the above table, 'C' denotes the perturbation level & 'LR' denotes the learning-rate which controls the C&W attack.

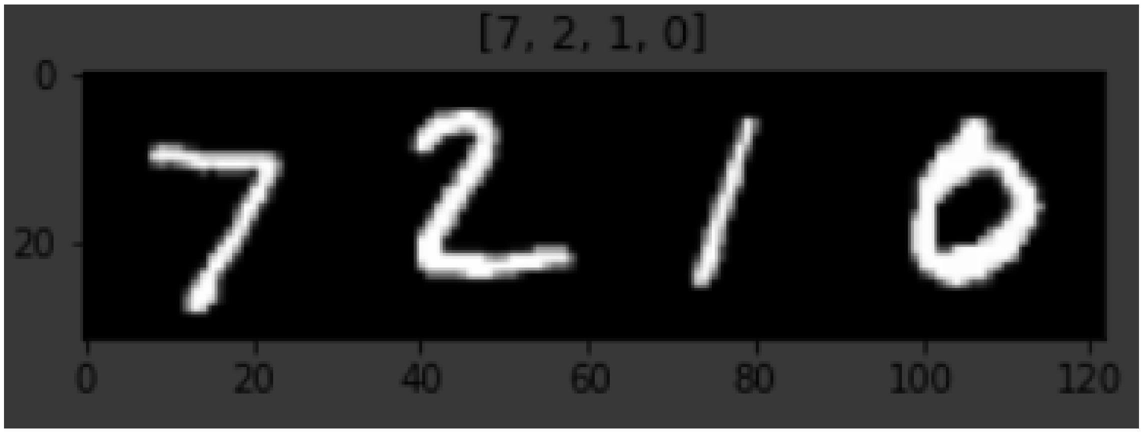
Perturbed Image -



(a) C-5

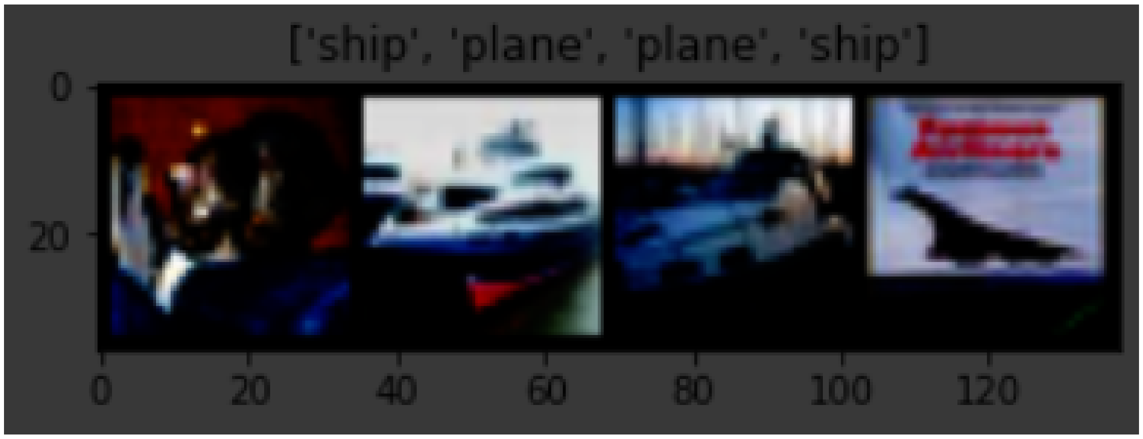


(b) C-8

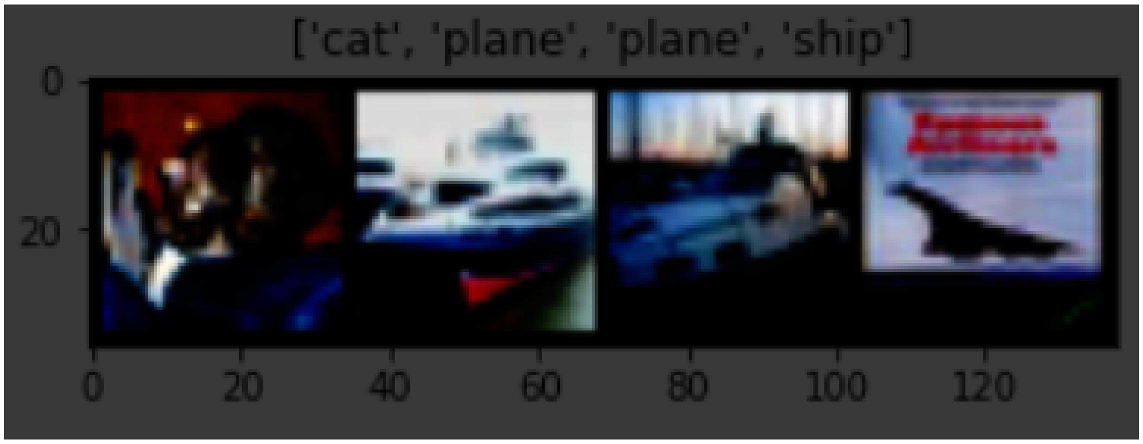


(c) C-10

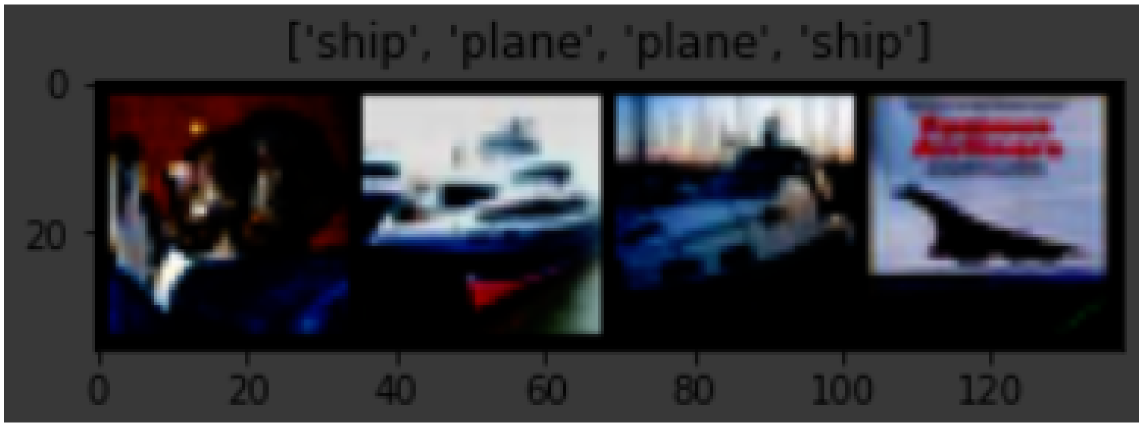
Figure 5.22: C&W's impact on sample MNIST Input batch - [7,2,1,0], with predicted outputs



(a) C-5

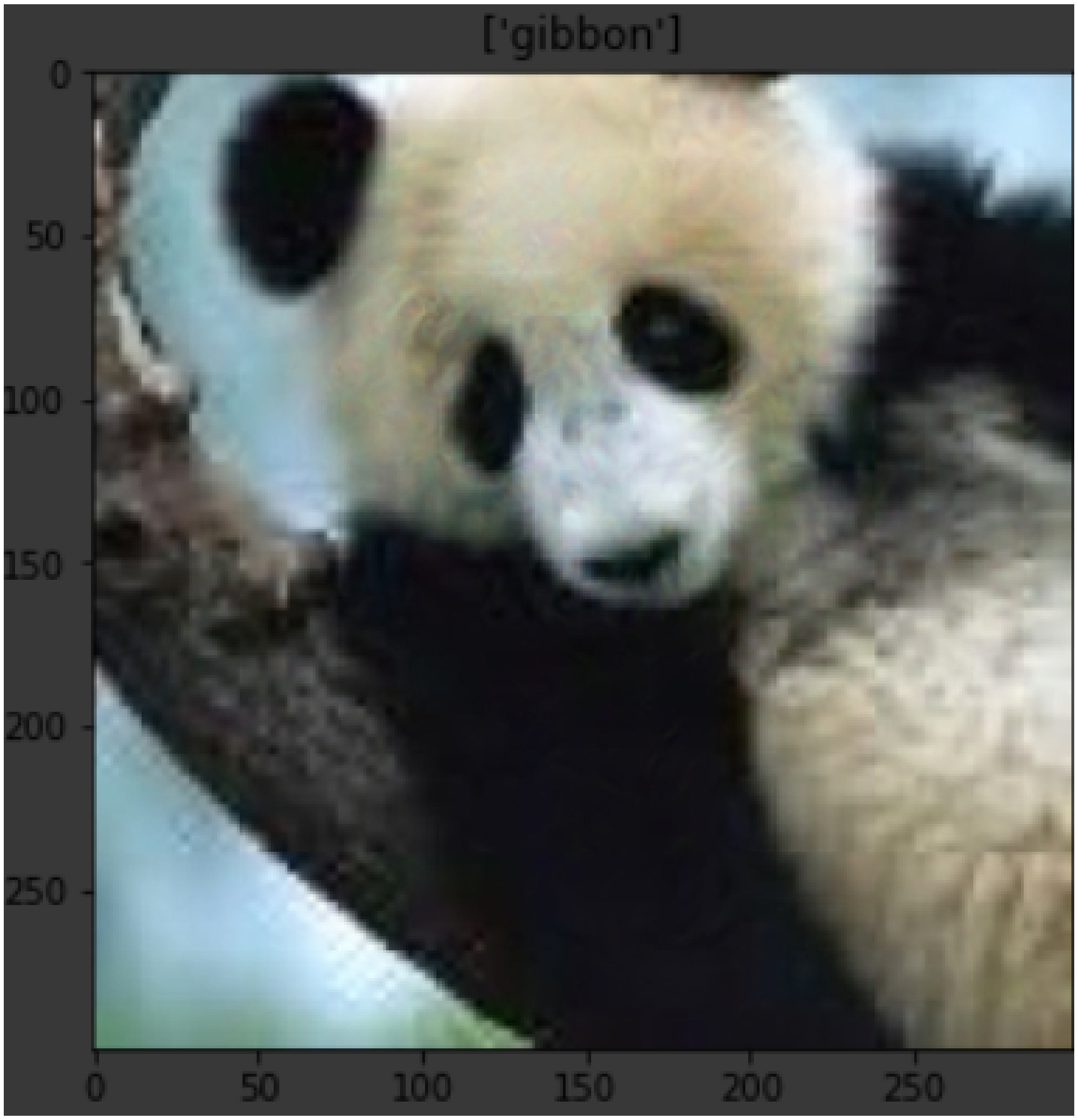


(b) C-8



(c) C-10

Figure 5.23: C&W's impact on sample CIFAR10 Input batch - [cat,ship,ship,plane], with predicted outputs



(a) C-5,8,10

Figure 5.24: C&W's impact on sample ImageNet Input image of giantPanda, with predicted outputs in all possible cases

Analysis -

The most-powerful attack among all the attacks we explored so far, can be attributed to 'C&W' attack as it has the capability to evade strong defense strategies. Here, both MNIST & ImageNet always maintains a constant accuracy but CIFAR10 shows an accuracy of less than 5% in all the cases. Although this is a powerful attack, the reason why it can't bring down the accuracy of the MNIST model can be pointed to tuning the hyperparameter. As opposed to previous attacks which has hardly 1 or 2 hyperparameters to tweak with, in this case the number is bit high. Thus, by setting the hyperparameters to right values, the strength of the attack can be increased.

This attack does an extremely good job with respect to perturbed images as it is evident from CIFAR10, in which despite being the attack accuracy less than 5% in all cases, there is no difference among the adversarial image & the true input image. Almost zero noise is visible here.

If we exclude one drawback associated with this attack which is high computation time, the title winner among all the 'White-box Non-Targeted' attack can be certainly attributed to 'C&W' attack.

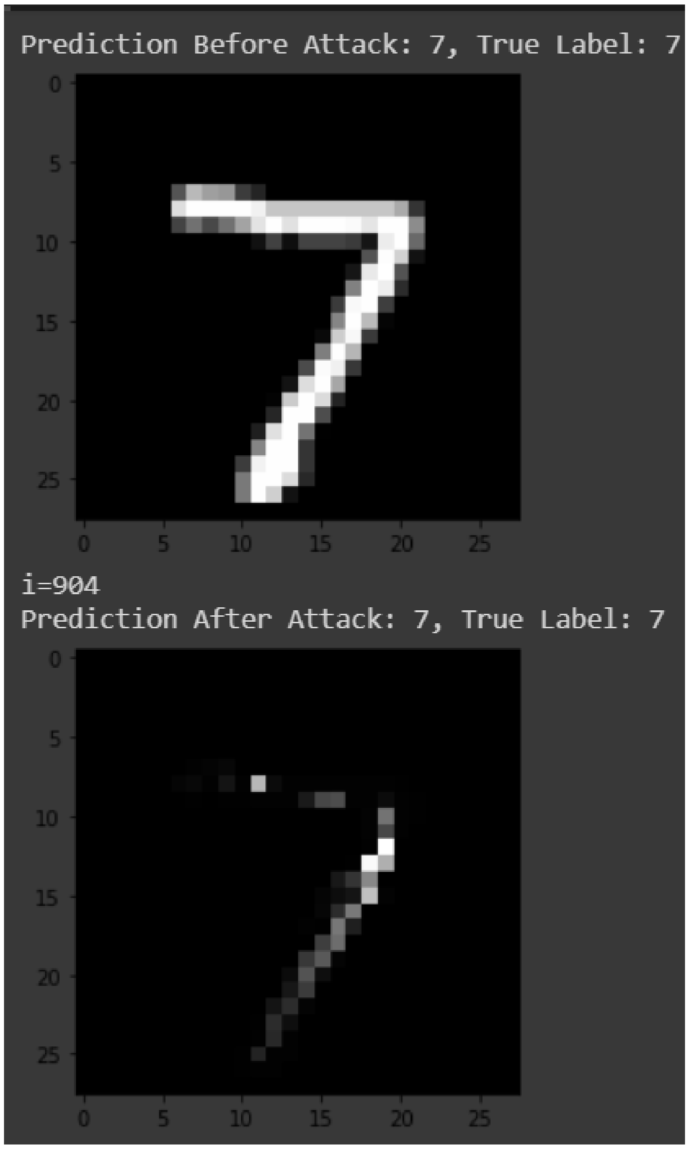
B. **White-Box Targeted-Attack - JSMA:**

Jacobian Saliency Map Attack (JSMA) is the only pure form of Targeted attack (although *Iterative-least-likely-class* method roughly fall under this category as well), which in theory modifies the given input image in a least extent to approximately match with the targeted image, so that the machine learning model predicts the specific target instead of a random target. One must be cautioned that it is not a guarantee that this will force the model to make the wrong prediction.

Hypothesis -

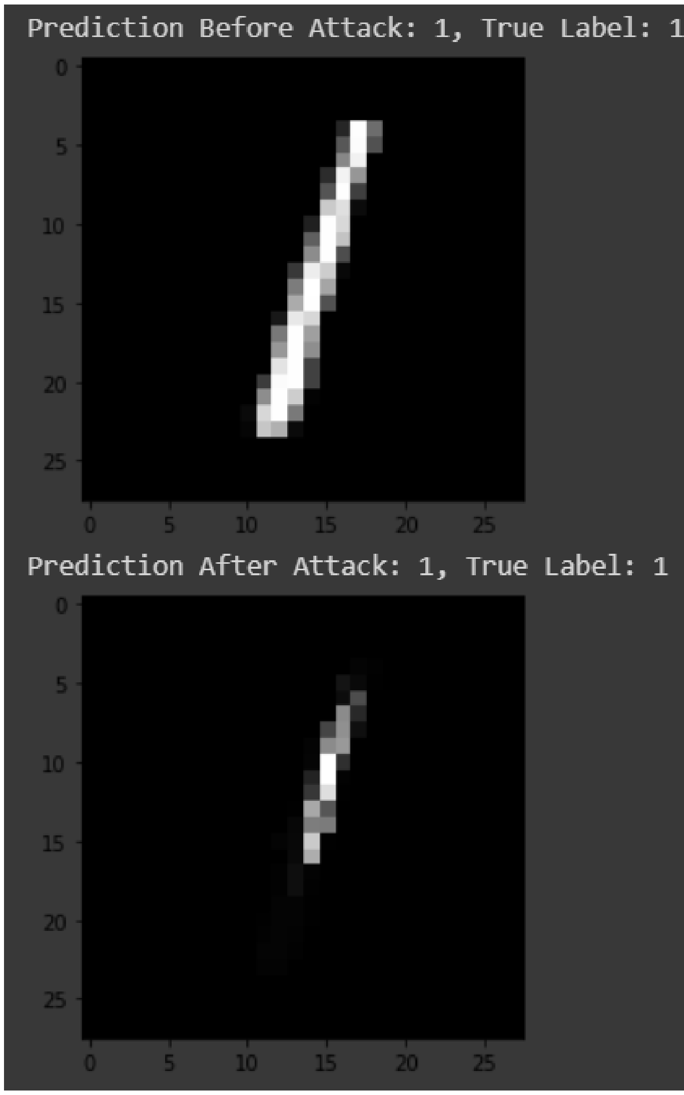
Lets set a Hypothesis that *"For a given MNIST image, JSMA can tweak it in visually indistinguishable way & fool the model to predict the specified target in all possible scenarios."*

Perturbed Images -



```
Prediction Before Attack: 7, True Label: 7
i=904
Prediction After Attack: 7, True Label: 7
```

(a) 7 -> 1 Attack



```
Prediction Before Attack: 1, True Label: 1
Prediction After Attack: 1, True Label: 1
```

(b) 1 -> 7 Attack

Figure 5.25: JSMA Attack with 2 Experiments. Experiment (a) signifies the input image as 7 & the target to be 1, while Experiment (b) is vice-versa case of (a).

Analysis -

From the very first impression, it is clearly evident that JSMA as it claimed is able to tweak the input image but the concern is on the extent of tweaking. Incase of Experiment (a), most of the pixels are blackened out but still one can confidentally claim the output as 7 by the looks of it whereas incase of Experiment (b), it has nearly distorted the input image & one can't be certain about the output in this case.

Coming to the prediction aspects, the given model is able to make the correct prediction in both the cases which means JSMA is unable to fool the given model. In experiment (a), **i=904**, signifies that JSMA attack process is stopped at that iteration although it is permissible to run for 10,000 iterations. This means there are no more valid pixels to be perturbed after this. Hence, the claimed hypothesis *"For a given MNIST image, JSMA can tweak it in visually indistinguishable way & fool the model to predict the specified target in all possible scenarios."* is **false.** This reflects the point that although 'JSMA is targeted attack, it is not the guaranteed one'.

C. **BLACK BOX ATTACK WITH PGD :**

Here, there are two models namely *substitute & victim*. The victim model is the actual model which is to be attacked & the substitute model is the one which is used for the sole purpose of generation of adversarial images. The generated adversarial images are stored & the victim model's accuracy is tested in 2 cases - one with clean input images & the other with the generated adversarial images.

ANALYSIS -

There is nothing new to comment about the quality of perturbed images as it was done before. But coming to the Test Accuracy of the victim model, with the clean input images, the victim model is able to achieve an accuracy of about **77.77%** but when the same victim model is feeded with the generated adversarial images its accuracy drops to **26.63%**. This clearly proves the **Transferable**

nature of adversarial attacks.

DEFENSE ASPECTS

D. ADVERSARIAL TRAINING :

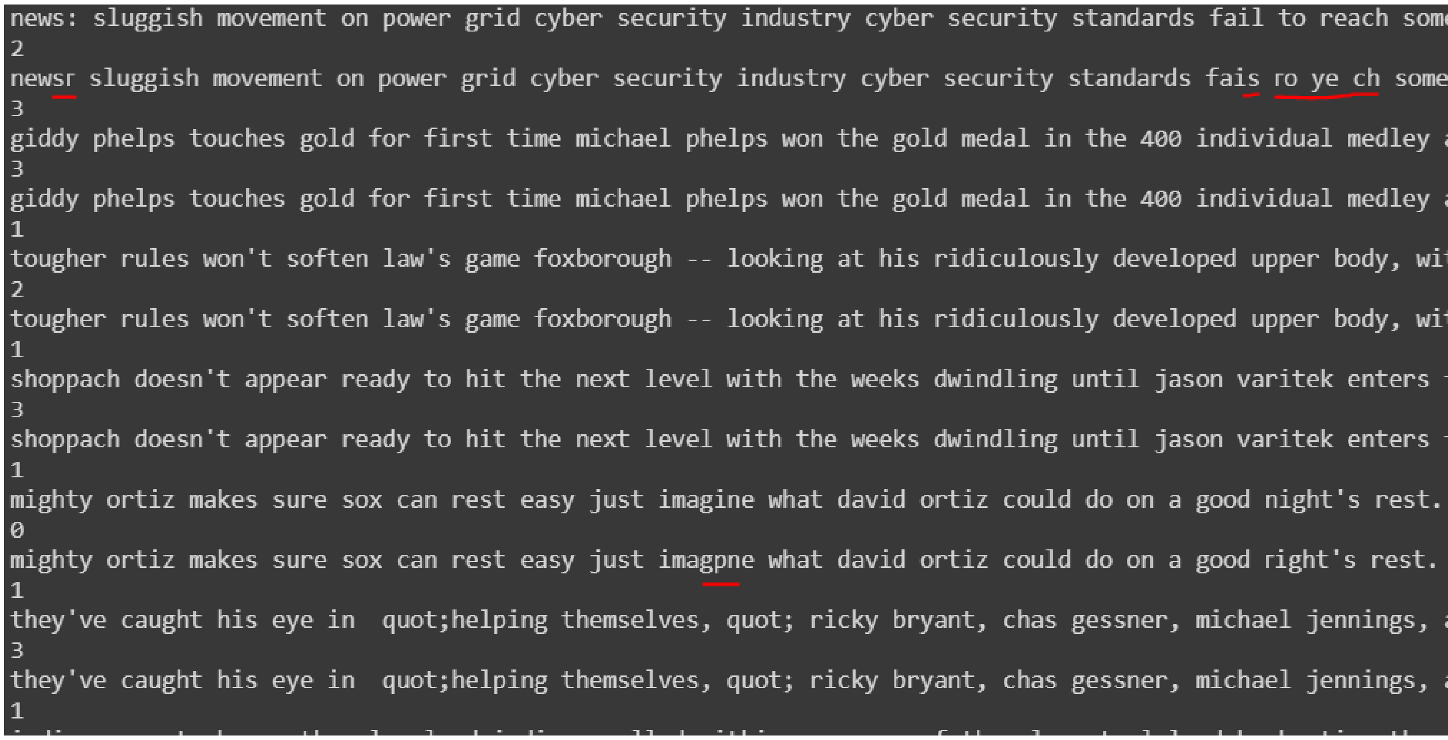
After completing the Adversarial Training of the model, it is tested by providing the clean images with zero noise & with the adversarial images generated from its very own parameters. Test-accuracy came out to be **99.10%** for *clean images* while **98.99%** for *adversarial images*. This clearly proves that Neural Network got hardened against adversarial examples but if the adversaries are generated from a substitute model & transferred to this model, it fails to resist against this which means they are unable to defend against **Black-Box** attacks. From this it can be inferred that this basically prevents/resist the attack strategies from exploiting out the model's gradients, which is a key ingredient when it comes to generation of adversaries.

E. DEFENSIVE DISTILLATION :

The most powerful defense strategy is perhaps the 'Defensive Distillation', as it can defend against any type of attack except for C&W attack & Transferable attacks. But the results were not that great with the present implementation as the model predicts the output value as '7' every time irrespective of the input. There is a suspicion if the 'KLDivergenceLoss' is best loss function candidate under this setting or if there is some problem in the implementation logic.

NLP Attack - Real World Datasets

F. AG_NEWS Dataset:

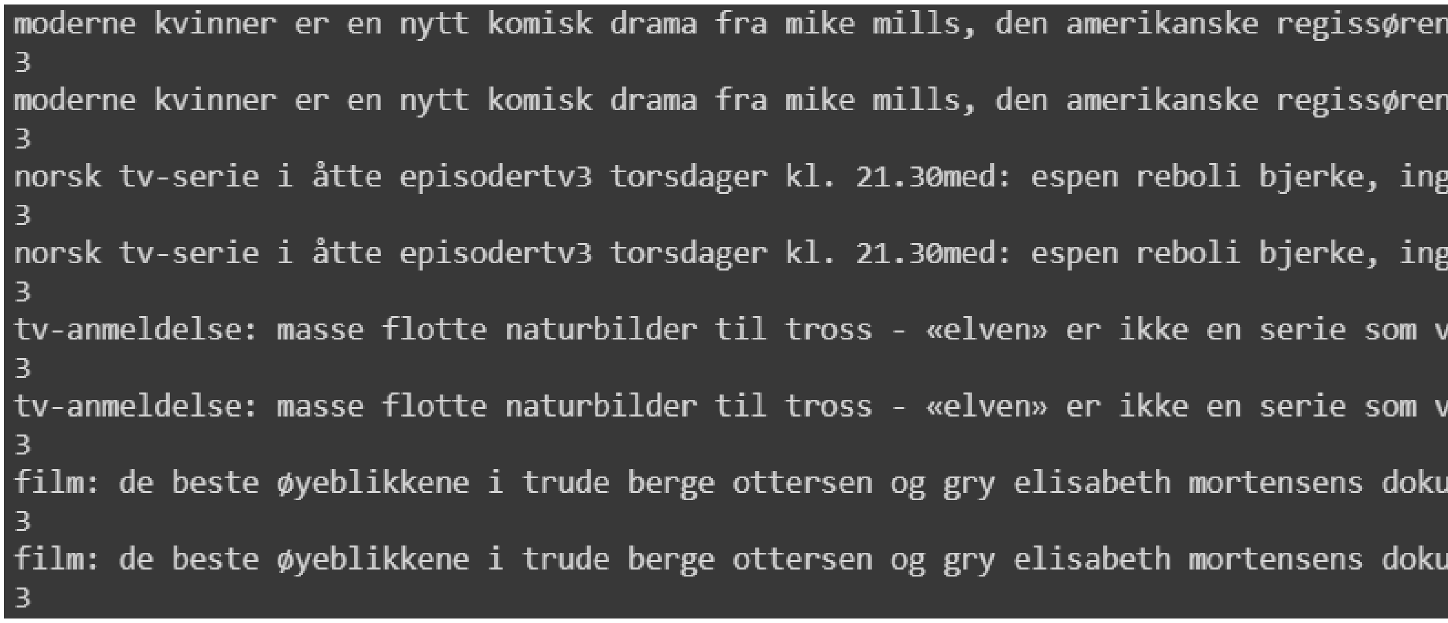


```
news: sluggish movement on power grid cyber security industry cyber security standards fail to reach some
2
newsr sluggish movement on power grid cyber security industry cyber security standards fais ro ye ch some
3
giddy phelps touches gold for first time michael phelps won the gold medal in the 400 individual medley a
3
giddy phelps touches gold for first time michael phelps won the gold medal in the 400 individual medley a
1
tougher rules won't soften law's game foxborough -- looking at his ridiculously developed upper body, wit
2
tougher rules won't soften law's game foxborough -- looking at his ridiculously developed upper body, wit
1
shoppach doesn't appear ready to hit the next level with the weeks dwindling until jason varitek enters t
3
shoppach doesn't appear ready to hit the next level with the weeks dwindling until jason varitek enters t
1
mighty ortiz makes sure sox can rest easy just imagine what david ortiz could do on a good night's rest.
0
mighty ortiz makes sure sox can rest easy just imagpne what david ortiz could do on a good right's rest.
1
they've caught his eye in  quot;helping themselves, quot; ricky bryant, chas gessner, michael jennings, a
3
they've caught his eye in  quot;helping themselves, quot; ricky bryant, chas gessner, michael jennings, a
1
```

Figure 5.26: Output for Ag_News dataset

As shown in the result, the Black-box mode NLP attack on the *ag_news* dataset is pretty good & the result shows that the original input sentence with the predicted class which is followed by the *adversarially modified sentence* with its prediction alongside. The 'Red' underline shows the *transformed characters* which the scoring algorithm has identified & it is visually indistinguishable for a human eye to pick that up.

G. **NOREC Dataset:**



```
moderne kvinner er en nytt komisk drama fra mike mills, den amerikanske regissøren
3
moderne kvinner er en nytt komisk drama fra mike mills, den amerikanske regissøren
3
norsk tv-serie i åtte episodertv3 torsdager kl. 21.30med: espen reboli bjerke, ing
3
norsk tv-serie i åtte episodertv3 torsdager kl. 21.30med: espen reboli bjerke, ing
3
tv-anmeldelse: masse flotte naturbilder til tross - «elven» er ikke en serie som v
3
tv-anmeldelse: masse flotte naturbilder til tross - «elven» er ikke en serie som v
3
film: de beste øyeblikkene i trude berge ottersen og gry elisabeth mortensens doku
3
film: de beste øyeblikkene i trude berge ottersen og gry elisabeth mortensens doku
3
```

Figure 5.27: Output for norec,Norwegian dataset

Although the same model with its hyperparameter is used as that of 'agnews' dataset, it isn't functioning properly when it comes to attacking the Norwegian dataset.Despite of the fact that irrespective of the dataset's language, it is converted to numerical tensor, it produces the output class as '3' in most-of the cases which leaves with a possibility that model should receive the same input for that to happen. Is the activation function turning the input to zero? If yes, then how does it work for agnews dataset is the interesting point which has to be researched further.

Chapter 6

Conclusion

By this Thesis, we completely explored about the working nature of the popular adversarial attacks & defense techniques with their relative strength & weakness alongside. An in-depth analysis about the Image perturbation level by different attack methods were also done & explained why some attacks can come up with fine perturbations while other methods stick with high perturbations. Although basic demonstration of all these techniques were successful, in few cases the quality of results has the possibility to improve further. Similarly, we also explored about how the NLP is different from an image attack with the analysis of challenges/constraints invlolved & then came up with the attack method, which adhered with the constraints.

In Future work, i would like to contribute myself towards the machine-learning security by working on Defensive Distillation in *unsupervised or reinforcement* learning environment, thereby identifying the potential opportunity to improvise it further. Also, i would like to invest more time to explore if there is a possibility to defend C&W attack & propose a new technique for it.

This work lays the foundation for the paper "The possibility for Universal Defense against Adversarial attacks" which briefs about the probability to devise a new defense strategy which is capable to defend against all types of attacks possibly in all types of environments namely *supervised, unsupervised, reinforcement*.

Bibliography

[1] Christian Szegedy, Ian J.Goodfellow Jonathon Shlens; *Explaining and Harnessing Adversarial Examples*. Google Inc, Mountain View, CA, 2015.

[2] Ian J.Goodfellow, Christian Szegedy, Rob Fergus et al; *Intriguing properties of a Neural Network*. February 2014.

[3] Zhengli Zhao, Dheeru Dua, Sameer Singh; *Generating Natural Adversarial Examples*. February 2018.

[4] Nicolas Papernot, Ian Goodfellow, Ananthram Swami et al; *Practical Black-Box attacks against Machine-Learning*. March 2017.

[5] Xi Wu, Nicolas Papernot, Ananthram Swami et al; *Distillation as a Defense To Adversarial Perturbation against Deep Neural Networks*. March 2016.

[6] Stepan Komkov, Aleksandr Petiushko; *ADVHAT:REAL-WORLD ADVERSARIAL ATTACK ON ARC FACE FACEID SYSTEM*. Aug 2019.

[7] Alexey kurakin, Ian J Goodfellow, Papernot et al; *Adversarial Examples in the physical world*. Feb 2017.

[8] Aleksander Madry, Aleksandar Makelov, Papernot et al; *Towards Deep Learning Models Resistant to Adversarial Attacks*. May 2017.

[9] Nicholas Carlini, David Wagner et al; *Towards Evaluating the Robustness of Neural Networks*. August 2016.

[10] Rey Wiyatno, Anqi Xu et al; *Maximal Jacobian-based Saliency Map Attack*. August 2018.

[11] Florian Tramèr, Alexey Kurakin et al; *Ensemble Adversarial Training: Attacks and Defenses*. May 2017.

[12] Ji Gao, Jack Lanchantin et al; *Black-box Generation of Adversarial Text Sequences to Evade DeepLearning Classifiers*. 2018.

[13] Xiaoyong Yuan, Pan He et al; *Adversarial Examples: Attacks and Defenses for Deep Learning* . 2018.