2017

Adversarial Attacks On Vision Algorithms Using Deep Learning Features

Andy Michel
University of Central Florida

ADVERSARIAL ATTACKS ON VISION ALGORITHMS USING DEEP LEARNING FEATURES

by

ANDY MICHEL
B.S. Florida Atlantic University, 2014

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2017

ABSTRACT

Computer vision algorithms, such as those implementing object detection, are known to be susceptible to adversarial attacks. Small barely perceptible perturbations to the input can cause vision algorithms to incorrectly classify inputs that they would have otherwise classified correctly. A number of approaches have been recently investigated to generate such adversarial examples for deep neural networks. Many of these approaches either require grey-box access to the deep neural net being attacked or rely on adversarial transfer and grey-box access to a surrogate neural network.

In this thesis, we present an approach to the synthesis of adversarial examples for computer vision algorithms that only requires black-box access to the algorithm being attacked. Our attack approach employs fuzzing with features derived from the layers of a convolutional neural network trained on adversarial examples from an unrelated dataset. Based on our experimental results, we believe that our validation approach will enable designers of cyber-physical systems and other high-assurance use-cases of vision algorithms to stress test their implementations.

I dedicate my thesis work to my family and friends who in many ways have extended their support throughout this process. A special expression of gratitude to my parents who have thought me to be relentless and humble in the pursuit success. I would also like pay forth gratitude to the numerous mentors that I have had over the years, who have invested their time in me unconditionally.

ACKNOWLEDGMENTS

I wish to express the utmost gratitude to my advisor and committee chairman Dr. Sumit Jha for his unparalleled mentorship and countless number of hours invested in seeing the success of this research. In addition to his guidance, he also assisted financially with grants that he has been awarded to buy hardware for the experimental analysis conducted in this paper.

A special thanks to Dr. Leavens Gary, Dr. Thankachan Valliyil Sharma for being being generous with their time in agreeing to serve on my committe and Sunny Raj for providing his insight and assistance throughout this entire process.

Finally, I would like to acknowledge UCF College of Engineering and Computer Science for providing an environment that promotes growth and excellence.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

In recent years, the field of computer vision has attracted a number of enthusiasts both from the industry and the research community due to the breadth of its application. More specifically, it has brought together many experts from the engineering, medical disciplines and many more; all with the purpose of leveraging robust image recognition systems to solve complex issues related but not limited to medical diagnosis, autonomous driving and security/privacy.

Over the past 5 years, we have been witnesses to a paradigm shift in the car manufacturing world where giant automakers such as Ford and Toyota have utilized computer vision to help drivers park their cars and alert potential collisions on the road. Smart-phone makers have also taken advantage of the latest state of the art research in the field to render better object detection/tracking when taking pictures or recording live videos. An even more personal experience with vision algorithms is in the realm of social media where Facebook, one of the primary leaders in the space has shown the ability to automatically tag people in images. Fast-forward to 2017, the use of computer vision systems has evolved from solving elementary problems to more sophisticated tasks namely self-driving; an effort that is being led by the likes of Google, Tesla, Mercedes and BMW. Furthermore, computer vision systems have become an agent in the medical field to aid with biomedical research particularly *genetic classification*.

As we become more dependent on such systems for daily activities, going about proving the correctness of vision algorithms is no longer an afterthought but a rather important task that must solved. Recently, this area has garnered an enormous amount of interest especially in the industry where a wide range of companies attempt to achieve commercial success with products that depends on intelligent vision systems. Yet, the multifaceted task of testing computer vision models remain a great challenge, mainly due to the fact that correctness in this case is not a binary answer.

Adversarial Attacks

An *adversarial attack* is a deliberate attempt at deceiving a machine learning algorithm i.e., Convolutional Neural Network (CNN) or Deep Neural Network (DNN). It employs the use of malicious examples that are carefully crafted by an attacker to force a model to misclassify a particular input or simply fool the model by adding noisy perturbations to mask features relevant to said model. An example of such attacks, is in network traffic monitoring where an intruder can exploit weak passwords or encryptions to take control of a router. At first sight, a compromised router may behave similarly to other intractable routers in a network to: a) keep a low profile, therefore go undetected b) learn about transmission schemes and frequencies. Upon gathering statistical traffic behaviors such as low and high volume time-frames, load-balancing among routers and nearest service nodes; an attacker may start spoofing packets containing small samples of malicious data to a target endpoint in order to gain control. All the while masking its own *Internet Protocol* address with that of a legitimate user. An attack of this nature may go unnoticed to a model because of the pattern similarity between an uncompromised and a malicious traffic, moreover it becomes harder to detect from a human perspective.

Adversarial attacks have become a real threat to modern technologies that relies on trained machine learning models to automate certain tasks. Most recently, there has been ample attentions gravitated towards fraud detection in online systems. Industry leaders of the likes of Amazon have vested interest in preventing fraudulent activities in the form of credit card transactions or fake reviewers. Effective methods such as $rule - based$ fraud detection and $bound - based$ edge detection [2] on adversarial transaction strive to discern unusually activities and take actionable steps to prevent further development.

Adversarial attacks transcends the boundaries of traffic monitoring and fraud detection problems. According to [3], public facing machine learning models exposed via restricted application programming interfaces are just as susceptible to adversarial attacks even though these models may be

trained in secured environments and the network model is kept private. The task of identifying adversarial attacks remains a problem of great difficulty due to the craftiness of adversarial examples generated by fraudsters.

Recent Machine Learning Failures

Machine learning has been the driving force behind some of the most important milestones in the 21st century. The applications of intelligent systems are unbounded and aims at achieving expertise level of understanding in a wide range of areas. With the rise of smart assistants, we have noticed the mastering of mundane daily activities to harder tasks such as weather prediction in order to facilitate our lives. Nevertheless, there remain many areas for improvement in the field of machine learning.

Figure 1.1: Cases where computer vision fails

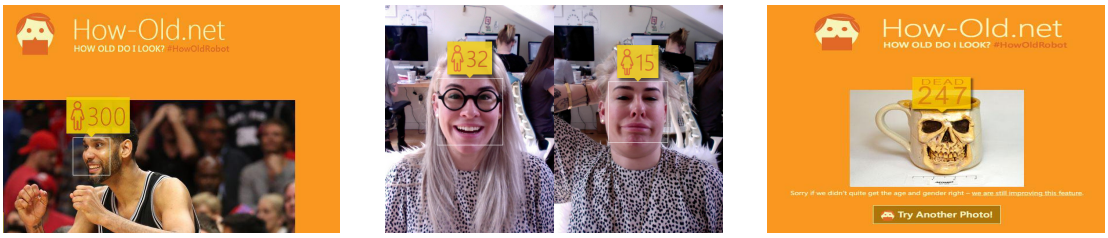


Figure 1.2: Age prediction failure

The above figures show the common cases where machine learning algorithms fail to render expected behaviors, we emphasize the fact that $correctness$ has become the staple of heterogeneous machine learning models. Once upon a time, defining test harnesses and rule based methodologies to evaluate correctness had been an industry standard but with the advent of neural networks it has become increasingly challenging to calibrate the notion of $correctness$.

Suppose we define $correctness$ as a measure by which a machine learning model i.e. classification or regression, is rewarded for making the right prediction or penalized for the inverse. Then a regression task of predicting housing values based on an array of parameters: number of rooms, year built, location and square footage; can be evaluated by the accuracy of the predictions from the model. Furthermore, testing the model on unseen inputs give us a more accurate understanding of the models behavior and a better sentiment about the idea of correctness in this context.

Assume a deep neural network is trained on a vast database of traffic images in which we hope the model to behave $like$ a human driver when different traffic signs are detected. At best the model consistently output the correct behavior with 99% accuracy, does the former definition of $correctness$ hold? If not how shall we define correctness?

We propose the following definition of $correctness$ and $accuracy$. Correctness in deep neural networks, is the correctness of the knowledge learnt whereas accuracy is the measure by which a model is rewarded based on analysis of its predictive ability. Based on this definition, we are able to test the correctness of the knowledge acquired by a DNN using counterexample inputs for which the model's accuracy meets a threshold.

Our Goal

The challenge in validating intelligent systems, especially in neural networks; can be attributed to their multivariate and nonlinear computational models. Consequently, the application of statistical

hypothesis to intelligent systems in order to determine their correctness does not yield reliable results. This phenomenon is credited to the nondeterministic nature of those systems. Our attempt in this research is to use a neural network to gather statistical data analytics in combination with deep learning techniques in computer vision to better understand the projection of data onto multi-dimensional spaces. Our hypothesis is that, from the generated statistical models; we will be able to derive perturbation filters that will allow us to create counterexample input sets for machine learning systems in order to determine their correctness. Moreover, we claim that using the noise filters extracted from a deep neural network trained over a database of perturbed images, will reveal adversarial properties that will perform better than random noise in black-box attacks.

Having defined our meaning of correctness in the realm of neural networks, we aim at evaluating a deep neural network D using clean and perturbed inputs. The first step in our approach is determining the model's accuracy based on a) *confusion matrix* [4] in the supervised learning phase on the AT& T face database and b) *matching matrix* in the unsupervised learning over the ImageNet 2012 database comprised of a variety of subjects. The second step, is evaluating correctness by counterexample, using crafted noise filters extracted from our deep neural network. We use the following criteria to better evaluate the model's understanding of its knowledge:

Figure 1.3: Our attempt at validating a model's correctness

- A model D fed clean images with the task of performing human detection will yield an accuracy of X which will remain consistent over testing sets and unseen instances.

- An unaltered model D attacked with adversarial inputs derived from the deepest layer of our network shall yield an accuracy Y under similar constraints as X.
- A pre-defined boundary for which X and Y remains consistent over a series of testing epochs.

CHAPTER 2: RELATED WORK

The area of adversarial attack is actively being researched, much of the interest can be dissected in multiple categories: non-targeted, targeted and defense against adersarial attacks. The algorithms grouped under each category have their respective missions and the focus of each is orthogonal to the choice of the technique used to attack and defend machine learning models. It has been shown that the use of counterexamples can be an effective mean to reinforce [5] accuracy in networks that aims at solving classification and regression problems but it can also be used for violation attacks. Hence, adversarial examples in themselves have the potential to be dangerous, especially targeted attacks where a system can be fooled to falsely report the truthfulness of a particular condition or presence of a specific attritube. Therefore futher efforts is being invested in building robust models able to whistand adversarial attacks.[6].

Inversely, a variety of proposed methodologies to training $DNNs$ have been put forth to further improve the success rate of threat models. One of which [3] suggests that an adversarial model *A* trained with no insight of the inner workings of a target network *T* can be very effective using synthetic inputs. The premise is that during training, network *A* although not exposed to *B's* structure is fed labeled outputs generated by *B* which in turns creates a similar decision boundary compared to the target network. As a byproduct, counterexamples that subverts *A* is likely work against *B*.

Adversarial Networks

Adversarial networks are machine learning models that are trained to generate counter inputs to deceive other intelligent systems. These systems can be used for the purpose of validation, penetration testing, malicious attacks and more. In the case of computer vision, suppose we have a

model M that has been trained on millions of images to perform a classification task i.e labeling objects in an image. With an input image I that contains many objects, the network returns a set of labels L i.e $M(I)$ returns L_{dog} L_{cat} L_{human} L_{boat}. An adversarial network M' is tasked to attack model M with adversarial inputs, a counterexample could take the form of $M'(I) = F_{\text{generic}}$ where F is a noise filter generated from the inputs. A successful adversarial attack could influence M in two possible ways:

1. $M(I + F)$ yields L_{unknown} L_{unknown} L_{unknown} L_{unknown} where M cannot detect any objects
2. $M(I + F)$ yields L_{pen} L_{speaker} L_{mug} L_{building} where is fooled into mislabeling objects

Where $I + F$ denotes pixel by pixel perturbations taken from the filter F and applied to the source image I. The use of adversarial networks is popular and extends beyond the scope of computer vision. Some other areas where similar networks have proven to be very effective are in training spambots, malware detection and intrusion systems. One of the most interesting properties of adversarial networks is in their transferability properties or generic perturbation filters. In some instances, a noise filter may contain patterns that is effective against multiple trained models even though the intensity of the noise may vary from one model to the other. One of the most important aspects in building adversarial networks is the training, there have been interesting methods that shows the efficacy of supervised and unsupervised learning. Supervised learning is well suited for scenarios where internal knowledge of the targeted network is known, it is also referred to as white-box training whereas features learnt from unsupervised learning tend to show patterns for which the neurons of the network are mostly activated. Those patterns are more generic in nature and may prove to be very useful in black-box perturbations where for either security or privacy reasons, the target network's model is not accessible.

It has been shown that white-box adversarial examples are more effective than random perturbation by at least an order of magnitude [7]. Experiments conducted using the Caffe deep learning

regression model for object classification, trained on the ImageNet database; revealed vulnerabilities in certain deep neural networks.

Black-box attack on the other hand is more challenging due to the fact that the training dataset is not curated for a specific network, the effectiveness of the adversarial network is measured by its transferable properties given the fact that the targeted network's structure is hidden. Therefore, perturbations attack performed using the black-box technique tend to be stronger and also take longer. In the context of adversarial image generation, the added distortion can be heavier. Our work is based on the premise that an adversarial network trained on a large dataset of adversarial images will reveal import underlying properties that can be used for general black-box attacks and the learnt features will contain strong transferable attributes.

Defense Against Adversarial Attacks

Although the focus of this paper is to present our approach to adversarial attacks, failing to mention the potential techniques to defend against them would not be wise. Adversarial attacks can be used to help train more robust machine learning models but it can also be used as a mean to carry out malicious intentions. With well crafted adversarial examples, one can deliberately subvert a system's expected behavior. Therefore defending against potential threats becomes a mandate. There exists a dichotomy in using adversarial examples to improve a system's accuracy while also defending against attacks. There are several techniques to train network models to defend in white-box and black-box setups.

In the case of deep neural networks, the lack of understanding of their inner workings further complicates the task of shielding them from attacks. Moreover, unlike the well known Denial Of

Service attack, the goal of adversarial examples is not to take down a system - although that may be the intent of an attacker - rather its purpose is to make a system render a response that benefits the intruder. In this context, a susceptible system is not necessarily one that crashes but one that allows their predicted results to be skewed especially in the case of recurrent networks. Suppose a recurrent model trained to solve classification problems uses kernel methods to project features extracted after *Principal Component Analysis* (PCA) onto an $n - dimensional$ space. *PCA* is a procedure that transforms correlated variables into linearly uncorrelated variables, it can be used to perform dimensionality reduction by evaluating the distance between data points. Suppose, the initial structure is a set of clusters that are created to group together closely related features and through feedback loops the classification accuracy is above an acceptable threshold i.e. 98%. A live version of this model could succumb to black-box perturbations as it re-adjusts the centroid of its clusters to feedback from adversarial examples, hence live learning models are the most susceptible to fool even in a closed setting where the model's structure is not exposed.

The [8] and [6] publications, show the efficacy of modifying the *Rectified Linear Units* (ReLU) activation function to truncate the forward propagation of heavy input signals. *ReLU* is a non-linear activation function in the hidden layers of deep neural networks that filters layer by layer neuron activations. It has been proven [9] to speedup training by a considerable margin due to its simplicity. In a DNN, increasingly deeper layers depends for input on the output of their predecessors; the *ReLU* function annihilates neurons that emits negative signals during propagation but forwards the raw value of positive ones to the next layer. The proposed method introduces *Bounded ReLU* (BRELU) that uses an upper boundary cut-off parameter to eliminate positive signal with abnormally strong values caused by perturbation from prpagating to the next layer.

Valentina et al. [6] proposed a two step approach, that in addition to the cost of configurations, does not incur an additional tax to the training phase of a model. The technique aims at strengthening weak points of deep neural networks and the agnostic nature of the method exhibits similar

behavior to the transferability properties of adversarial networks.

Papernot et al. [10] present defense distillation that aims at minimizing the success rate of sampled adversarial attacks on neural networks. Interesting data gathered in this study shows distillation weakens adversarial attacks' effectiveness from 95% to 0.5%, moreover it further reveals that on average; the number of features that needs to be disturbed by adversarial attacks increased by 800%. This attack agnostic method requires two models to be trained. The *softmax* layer of the first classification model is leveled by division with a constant value T and the second model is trained on the same training set but uses the output - probability vectors - of the last layer of the first model as labels instead , then the second model is used for practical purposes.

CHAPTER 3: BACKGROUND

Opencv

OpenCV is a state of the art computer vision library originally written in C but has been extended with C++ APIs that aims to achieve computational efficiency while providing an extensive tool-set to manipulate computer graphics. It has been built to leverage multicore processors and is widely used for real-time applications particularly in robotics . Additionally, it provides Java and Python bindings and also supports different operating systems.

Tensorflow

TensorFlow is a machine learning library developed by the Brain team at Google. It stands out when it comes to running at scale and performing resource intensive operations the likes of numerical calculations. With its pluggable architecture, it separates operations and data into a graph where the nodes denotes the mathematical operations and the edges are multidimensional arrays otherwise known as tensors.

Pixelhex

PixelHex is our generative adversarial neural network implemented using TensorFlow's Python API. Its principal objective is to find an adversarial filter from the ImageNet ILSVRC 2012 database where the images have been perturbed prior to training. This model can be queried for a series of operations such as the visual representation of what the network has learnt at the l^{th} where l represents the number of layers in the network.

Maya

Maya is a tool that has the primary objective of applying perturbation to given images in order to create adversarial counterexample. It uses an error model extracted from features of a convolutional neural network. The error model can be described as a multidimensional structure that contains scalar values which are applied to an image using statistical hypothesis. For an image *I* that has shape in the form of $h \times w \times d$ where:

- *h* is the number of rows in the structure
- *w* is the columns
- *d* is the depth where 1 corresponds to grayscaled images and 3 to RGB

Maya explores all sub-spaces of the image to apply perturbation, this method has proven to be 20 times faster than random pixel by pixel perturbations [7].

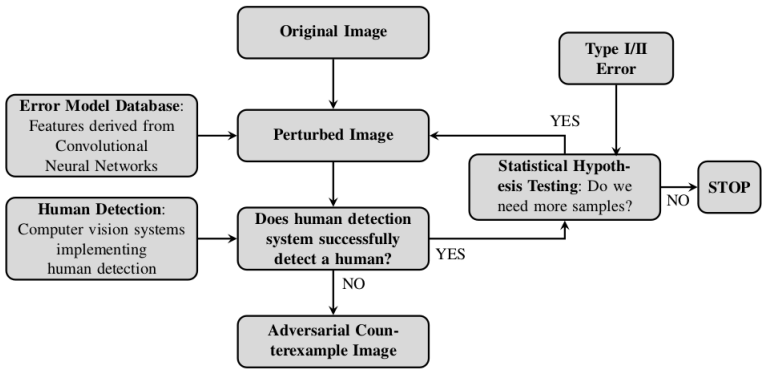


Figure 3.1: Testing of human detection algorithms using statistical hypothesis testing and the error models derived from a convolutional neural network

CHAPTER 4: OUR APPROACH

PixelHex is our contribution in the realm of adversarial neural networks, it presents a novel and practical approach to generate adversarial examples that can be used for the purposes of DNN training and defense. Different methods have been proposed on how to use perturbation matrices in order to aid deep networks to fend off attacks as well as smooth decision functions. In this paper, we present our findings based on experiments that simulates real-world scenarios and benchmarks gathered from *random* and *controlled* noise filters generated by our neural network. We make the following hypothesis in context of adversarial attacks:

- During training, a model that is fed training data in conjunction with small crafted samples of adversarial examples will perform better against adversarial attacks in regression and classification tasks. Let's say a model M is trained on a dataset D comprised of D_{clean} targeted samples that makes up about 90% of the training data and D_{adv} the remainder. Chunks of D_{adv} is issued to the input layer at each phase over multiple epochs, as the perturbations are cascaded down to deeper layers; we expect them to activate the wrong neurons. The activated bits while in low intensity, should be discernible when projected in high dimensional spaces which would facilitate clustering i.e *SVM*. The potential benefits can manifest themselves in two possible ways: 1) weaker biases during training and 2) stronger decision boundaries.

- In adversarial attacks, *controlled* samples rendered from adversarial inputs by our neural network will carry a soft transitivity property. A network N_{adv} geared to attack N_A and N_B where both A and B have not been exposed to adversarial cases. Let's say network A has been compromised by N_{adv} we expect the same attacks against B to be at least marginally effective on the same input in a black-box environment. Mainly due to the fact that an image I and its counter I' represented by $n\ x\ m\ x\ c$ matrices, I' is only valid if it falls within a certain range

when evaluating the *peak signal-to-noise ratio*. The inconsequential difference between the original and the adversarial image should not be perceptible by the human eye.

Network Structure

The architecture of the PixelHex is symmetrical in that, each input layer has a matching output layer and the deepest layer otherwise known as coding is the internal representation of input layer. By building a *stacked* network, we aim to reconstruct the inputs and have each output layer match its corresponding input as closely as possible. Using unsupervised learning, the *PixelHex* model was trained on a database containing 43,000 thousand adversarial images. In order to minimize the possibility of over-fitting, the weights of the decoding layers were tied with that of the encoding layers. Suppose we have N layers and for each layer we have L_n that represents the weights W at the n^{th} layer then the decoding layer's weights can be expressed as: $WxN - L + 1$ where the coding layer is equal to the total number of layers divided by 2. An additional benefit we gain from this technique stems from the fact that we halves the weights which inherently speeds up training.

Our training environment was an Ubuntu Zesty, release 17.04 operating system running on the kernel 4.10 with 16GB RAM, an Intel i7 processor and an 11GB Nvidia 1080 Ti GPU. The accompanying Nvidia driver is version 375.82 which is the most recently supported driver at the time of this writing. Due to hardware limitation, the images from our database were down-sampled which resulted in perturbation information loss. More concretely, for every image a total number of 154,587 pixels multiply by 100 images per batch could not be held in memory for a forward and backward pass. Therefore we shrunk each image from 227 x 227 x *c* where *c* is equal to the number of channels to 50 x 50 x *c*.

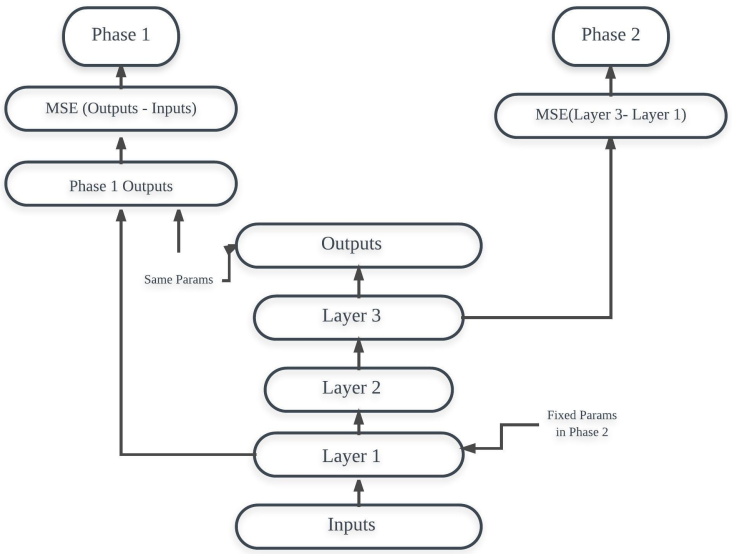


Figure 4.1: Network Architecture [1]

Adversarial Filter Analysis

The network was trained in two phases, during phase 1 only the input and output layers are activated. In phase 2, the output of the first input layer a 4-dimensional *tensors* is fed to the next hidden layer. Initial versions of the model was trained on the AT& T face database that contains 40 distinct subjects with close up frontal images taken from different angles. The dataset in total has 400 grey scaled images of size 112 by 92, where each image has a varying set of attributes and expressions. We use Maya's statistical hypothesis and error models to perturb the AT& T database with a mean $PSNR$ of 36.7.

Figure 4.2: Sampled examples from the AT& T face database that has been strongly perturbed for the purpose of training our deep network

Although the dataset was relatively small to train our network, it served as a mean to test the initial hypothesis of whether it was possible to detect patterns from a database of perturbed images. The results although not readily apparent to the naked eye were very promising, after training our network; certain pattern started to emerge.

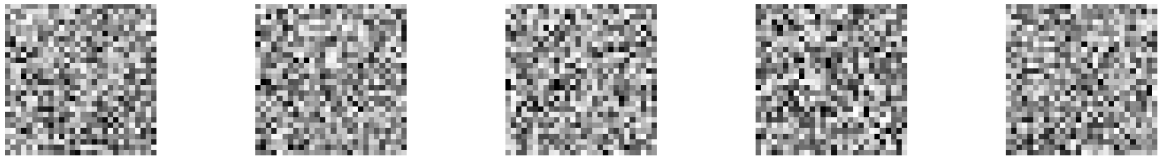


Figure 4.3: Network Weights

Driven by the success of our initial experiments, we opted for a larger dataset comprised a wide range of adversarial images i.e. dogs, cats, humans, trees, birds. The extracted noise filter showed the most recurring noise pattern from the database.

Figure 4.4: Visualization of noise filter from PixelHex's network. The adversarial pattern derived from the DNN was a 3D tensor of size $50 \times 50 \times 50$ with RGB values.

We believe the weights derived from our network is only one instance of all possible filters that could be extracted from the same dataset. Mainly due to the fact that different weights, optimizers and initialization functions may activate stronger signals in the forward propagation of neurons from one layer to the other. There are other methods that could potentially reveal important features in our dataset for example, researchers at Microsoft [11] proposed the *He* initialization function, based on *PReLU* networks that surpasses human perception in recognition tasks. Moreover, *PReLU* has been successful in reducing model overfitting at no additional cost.

CHAPTER 5: WEB CYBER INFRASTRUCTURE

Complimentary with our study, we have built a web interface to demonstrate the practical use of adversarial image generation. PixelHex's web layer allows one to upload images that gets perturb using different adversarial methods and finally proposes an adversarial image. The adversarial module is implemented using OpenCV's HOG and Haar human detection to generate counterexamples. *Gaussian blur* is applied to the image in small increments as we mask the features normally extracted by human descriptors in vision algorithms. During our experiment, we tested multiple frameworks to validate against the noise filter obtained from PixelHex's neural network.

Top-down Level Architecture

The front-end is single-page architecture (SPA) that is separated from the back-end server. The *SPA* design allows the client interface, the application layer and the database application to be hosted on respective containers. Given the resource intensive nature of machine learning tasks, we benefit two-fold from this paradigm: a) we can replicate each layer of the application and perform load-balancing b) using parallel and distributed algorithms, we can decompose expensive operations using a relation algebra tree and spread each operation to different servers. This lightweight architecture allows PixelHex to scale up and out without the cost of refactoring. When a user performs an action, a *User Interface* (UI) component fires an event that gets intercepted by the hidden UI services. Upon processing the event, the front-end service submits the request to the available back-end servers then polls for a response.

The application server provides a series of endpoints with different access control i.e. *who can see what*, a successfully authenticated user's request is relayed to the next layer for further processing.

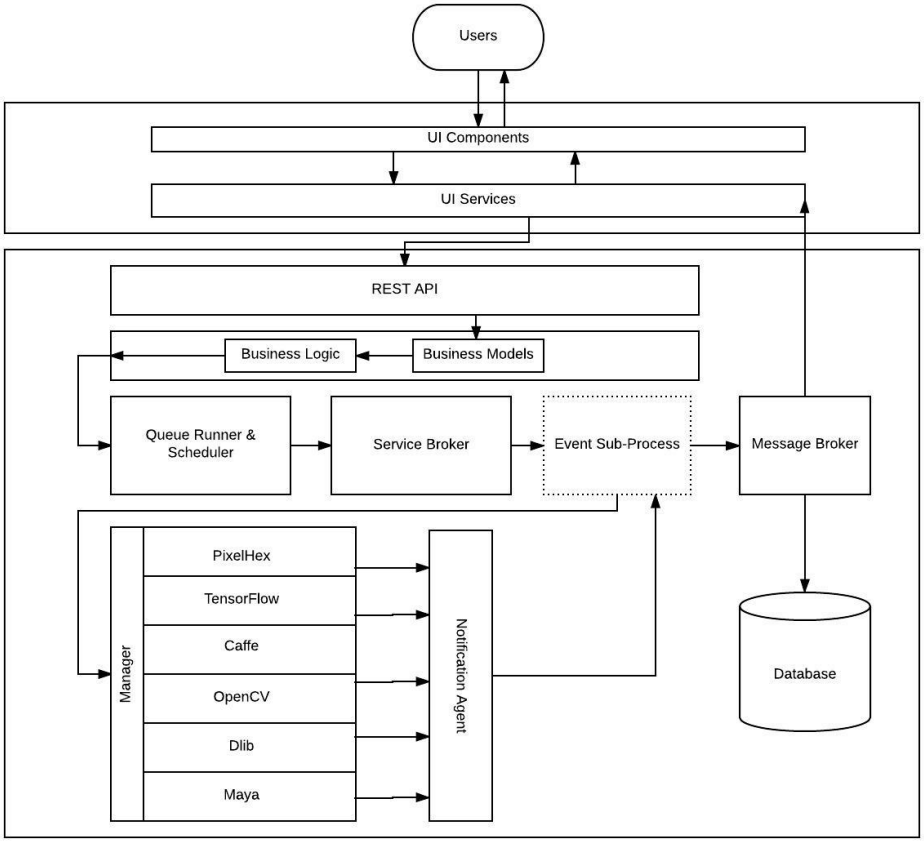


Figure 5.1: Web Architecture

At the heart of the application resides two modules that plays critical roles in managing resources and speeding up execution. First, a *Qeue Runner & scheduler* (QRS) that enqueues and dequeues requests in a first-in first-out order to guarantee fairness. Although the system is fair, starvation may occur as a result of lengthy operations that stems from a high rejection rate of perturbed candidate images. To alleviate the potential resource contention and starvation on server-side, we introduce the *service broker* (SE) that pulls jobs from the queue and evaluates resource availability in order to parallelize the execution of jobs up to the maximum amount of threads allowed by the operating system.

Each job is propagated to the *Event Sub-Process* (ESP) module that attaches a notification action on the request then forwards it to the *Manager*. The Manager is responsible for passing an action to the respective sub-process. An action is composed of the submitted image from the user, the perturbation parameters and the vision algorithm to attack. An example action will resemble the following: *attack* the *tensorflow* vision algorithm using the *random* perturbation model with a given list of *parameters*. An action is broken down into two distinct steps:

1. Craft an adversarial example
2. Subvert the requested vision algorithms with the counterexample generated in step 1.

Upon completion, the *Notification Manager* signals the polling *ESP* which dispatches the event type to the *Message Broker* (MB). The *MB* interprets the event and fires a) response to the client b) an action to persist statistical information in the database.

PixelHex's web interface provide a set of features that allows experimental evaluations of the noise filter presented in this paper. The main page is *Experiment* that contains a step-wise approach to perform black-box attack against a pre-defined sets of computer vision algorithms.

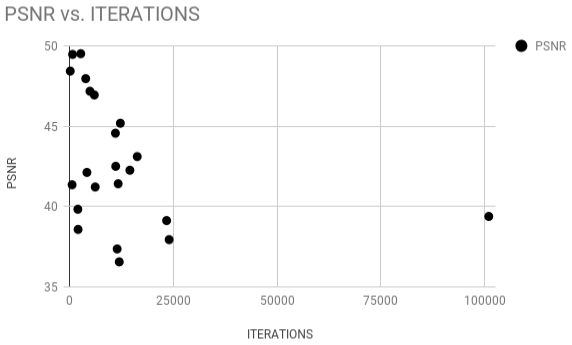
1. Upload a picture
2. Provide perturbation parameters i.e. intensity, probing length *PSNR*, type of attack
3. The type of algorithm to attack

Upon completion, a window is rendered showing the original and adversarial images and an icon that indicates whether the adversarial image fooled the targeted vision algorithm or not.

CHAPTER 6: EXPERIMENTAL RESULTS

Sample Results

To evaluate the noise filter generated from the PixelHex's network, we used the Histogram of Oriented Gradients feature descriptor contained in OpenCV's library. Our setup was comprised of the weights derived the neural network as our perturbation structure, the HOG human detection algorithm and the equation in Figure 6.1(b) to calculate the peak signal-to-noise ratio of a source image I and its adversarial I'. As to simulate a practical scenario, we captured the $PSNR$ and the number of perturbations applied to the adversarial image; our results are depicted in Figure 6.1(a).



(a) Results of image dissimilarity over number of perturbations applied

$$PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right)$$

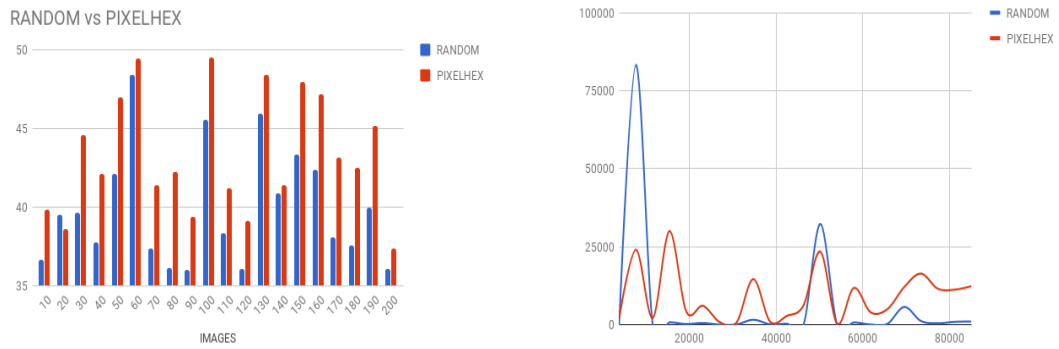
(b) PSNR equation

Figure 6.1: Scatter plot of similarity measurements over the number of perturbations

The results from our sampled experiments showed that the noise filter from our neural network yielded an average $PSNR$ of 42.819 with a dissimilarity threshold set at 90%. Normal values between 30 and 50 indicates small differences and the closer the upper boundary is better. In order to generate a counterexample, the source image is transformed into a 3 dimensional *bin* and

each sub-bin is visited at least once to determine whether it contains relevant features pertaining to human recognition. If a candidate bin is irrelevant it is ignored otherwise it is captured as a region of interest. The region of interest is a multi-dimensional space made up of pixels with the shape of: $height \times width \times depth$, we apply a sliding window over each pixel and up-sample their values with a constant T from the error model extrapolated from the neural network. The random attacks are applied in a similar manner to the each image but differs from targeted attacks in the error model. The random error structure is a normalized scalar tensor in the shape of a 3D RGB matrix, where the constants are bounded by the min and max weights derived from the PixelHex neural network.

Comparison to Random Attacks



(a) PSNR comparison between random and targeted perturbations from PixelHex

(b) Probing length between random and PixelHex's weights

Figure 6.2: Performance Analysis

According the performance analysis between random and targeted probes, we noticed that median probing length, when using the weights from the DNN is 56% higher than random. While on average, the similarity differences between clean and adversarial images generated from PixelHex's

weights is 87% whereas random is 79% which results in an 8% improvement.

Figure 6.3: Original test images from the Penn-Fudan Database for Pedestrian Detection and Segmentation

Figure 6.4: Adversarial Images generated by the weights from the PixelHex's network

The figures above show two sets of images from the Penn-Fudan Pedestrian database, where the first one shows the original images and the second the adversarial images generated using the noise filter from our network. As noted in prior sections, we aimed at achieving state-of-the art performance in adversarial attacks, with our results; we have fooled two publicly accessible computer vision frameworks. The *Dlib* frontal face detector and *OpenCV* Haar Cascade classifier both yielded a failure rate of 48% and 76% to our black-box adversarial attacks. Failure in this environment represents the inability of the computer vision algorithm to effectively classify the image or incapable of detecting the human.

CHAPTER 7: CONCLUSION

In conclusion, this study revealed important adversarial attributes in images generated by the PixelHex network that are effective in black-box attacks against different types of computer vision algorithms. Furthermore, it shows that, there exists an inherent filter that can be extrapolated from large sets of perturbed images to subvert practical machine learning models. The result of our experiments further strengthens our initial hypothesis which at the onset suggested that; there must exist a threat error structure in counterexamples that carries soft transitivity properties. According to our definition of correctness in machine learning models, similar filters can be used to both calibrate the predictive accuracy of deep neural networks as well as gaining better insight in a model's inner representation of what it has learnt. Even though our approach showed promising results, we were not able to fool other vision algorithms such as Caffe and Google's convolutional neural networks. Using the perturbation filter from PixelHex, we reduced the labeling percentage by 10-12% on average but not low enough to force a misclassification. Finally, our contribution is a deep neural network that closes the dissimilarity gap between original and counter images. We hope the result of this study, can help further advance the effort in validating network models using adversarial attacks.

LIST OF REFERENCES

[1] Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'REILLY, 2017.

[2] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. 2016.

[3] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*, 2016.

[4] Sofia Visa, Brian Ramsay, Anca Ralescu, and Esther van der Knaap. Confusion matrix-based feature selection.

[5] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[6] Ambrish Rawat Valentina Zantedeschi, Maria-Irina Nicolae. Efficient defenses against adversarial attacks. *arXiv:1707.06728v2*, 2017.

[7] Sumit Kumar Jha. Statistical hypothesis testing using cnn features for synthesis of adversarial counterexamples to human and object detection vision systems. 2017.

[8] Shan Sung Liew, Mohamed Khalil-Hani, and Rabia Bakhteri. Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems. *Neurocomputing*, 216(Complete):718–734, 2016.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks.

[10] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv:1511.04508*, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.