Generating Adversarial Examples Using Parallel Genetic Algorithm

Kira Chan
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
chanken1@msu.edu

***Abstract*—Deep Neural Network is a class of classifier that have had large success in the field of classification and image recognition. As such, they have been integrated into many practical application, including but not limited to safety critical systems where failure may cause serious harm, injury, or even death for humans. Adversarial examples are modified input images to deep neural network model that the model fails to classify correctly with respect to the ground truth. Traditionally, trustworthiness or robustness of deep neural networks against adversarial examples involve retraining the model on a set of adversarial inputs. This paper introduces a genetic algorithm and parallel genetic algorithm approach to generate new adversarial examples. A proof of concept implementation is provided to attest the feasibility of the approach. When applied to a set of input image originating from the CIFAR-10 dataset, the algorithm is able to produce adversarial examples that is misclassified by the model.**

I. Introduction

Deep Neural Networks (DNNs) are artificial neural networks with multiple layers of activation neuron used to predict a class label given an input. On object recognition tasks, DNNs can achieve an impressive object recognition results of up to 99.4% accuracy on the CIFAR-10 benchmark [1]. With its capability, DNNs have many real-world applications, such as malicious file detection, fraud detection, facial recognition, and virtual assistants (e.g., Siri or Cortana). In some application, commercially deployed DNNs may have significant consequences should the DNN fail, leading to potential for serious harm, injury, death, and financial loss. Such systems are commonly referred to as safety critical systems [2]. DNNs deployed in safety critical systems must be robust towards adversarial attacks. This paper proposes an evolutionary method to generate adversarial images that introduces small perturbations to the input images to cause the DNN to misclassify the input image.

In 2014, Szegedy et al. [3] discovered that machine learning models are vulnerable to *adversarial examples*. Adversarial examples are expected input images, often training or test data, with a small amount of perturbation introduced that the DNN fails to classify. For example, Eykholt et al. [4] successfully confused an autonomous vehicle's object detection algorithm from correctly detecting "STOP" signs by placing physical stickers in specific locations of the road sign. Figure 1 displays a perturbed stop sign that the detection algorithm fails to recognise. Thus, robustness against adversarial examples for DNN models is critical for ensuring safety for safety critical systems. When considering the security of DNNs, a number of research efforts have proposed to enhance the robustness of DNNs against adversarial examples through training the model against known attacks. Goodfellow et al. [5] introduced a Fast Gradient Sign Method (FGSM) to perturb the image using the sign of the gradient function. Deepfool introduced by Moosavi et al. [6] studied the decision boundaries between class labels in the data space and finds a path that the input image x can use to go past the decision boundary into a wrong class. Vidnerova et al. [7] introduced an evolutionary approach using Genetic Algorithm (GA) to generate adversarial example for various machine learning techniques.

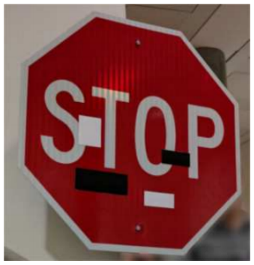


Fig. 1. Eykholt et al.'s attack on a stop sign [4]

This paper introduces a parallel evolutionary approach to generate adversarial examples as an extension to Vidnerova et al.'s [7] GA approach. By using a parallel Genetic Algorithm (pGA), we discover potential adversarial examples that can be used to fool the DNN from correctly classifying the input. A pGA approach can generate adversarial examples without access to the DNN model's underlying weights, gradient, or other critical information that adversaries typically do not have access to. The pGA approach only requires a completed DNN model used to query the image's current labels. As such, our approach is black-box and closely resembles that of the adversary capabilities in a real-world attack. Using pGA, multiple populations evolve in parallel. Migration between the populations occur every N generation. This approach prevents the population from prematurely converging, and thus allows for more sophisticated adversarial examples to be generated.

The proposed pGA approach evolves an adversarial example by random mutation and crossover operations between an image population. Since each run of the algorithm is stochastic, each execution of the algorithm with the same input image results in different perturbation. By comparing the distance to the original image, an adversary can choose the adversarial example with the least amount of change compared to the original image as the attack image. Our study shows that the pGA approach can generate adversarial example that causes DNNs to misclassify the input. Futhermore, the pGA approach was able to produce images that were closer to the original input than the GA approach.

As a proof of concept for our approach, we constructed an algorithm that modifies an input image using a GA and a pGA to assess the output adversarial example's label against the predicted label and ground truth. We demonstrate the effectiveness of the proposed approach by generating several adversarial examples that causes a misclassification with low and high confidence level. The remainder of this paper is organised as follows. First, we overview background material and review related work. Next, we describe the proposed pGA approach and the algorithms used to generate the adversarial examples. As a proof of concept, we implement the algorithm in python and use it to attack images from the popular image benchmark CIFAR-10. Several experiments are conducted to show the effectiveness of the attack. Finally, we conclude the paper and discuss future directions.

II. Background

This section describes background information for the paper. We overview the evolutionary pGA, current adversarial example generation techniques, and current theories on their existence.

A. Evolutionary Algorithm

In an evolutionary algorithm, individuals of a population randomly co-evolve towards an optimum of the solution space. First, the algorithm maps a feature of the target as genes and initialises a randomly distributed population of individuals. A random initialisation allows for random distribution of members in the solution space, increasing the chance of converging at the global optimum. Next, the algorithm repeatedly evolve the population by random mutation and crossover operations between two chosen parents of the population. Parents are selected via a tournament selection, where a random subset of individuals compete against each other in a round-robin fashion. The selected p individuals reproduce to create offsprings, which are introduced to the population. The population is then reduced to the original size by another tournament selection. A fitness function is used to assess the performance of an individual. Finally, elitism ensures that the best performing individual always survives each generation. By simulating natural selection and evolution observed in nature, evolutionary algorithms are able to evolve optimal solutions for the defined problem.

B. Parallel Genetic Algorithm

Course-grained parallel genetic algorithm (pGA) introduced by [8] is a variation of the genetic algorithm. pGA evolves multiple "islands" of population in parallel similar to that of a tradition genetic algorithm. After every $\mathbf{i}$ generations, a set number of individuals migrate across populations, introducing diversity and preventing premature convergence. pGA explores the solution space of the defined problem better by allowing each population of an island to explore a separate region of the solution space independently. Migration cycles prevent the population from converging too quickly on a local optimum.

C. Adversarial Examples

Adversarial examples describe input images that are maliciously perturbed with noise to cause the DNN to misclassify the input. Figure 2 shows original input image, perturbation noise, and the resulting image that causes a failure in the DNN. The adversarial example resembles the original image closely, thus is not human distinguishable. The following subsections categorise the adversarial example attacks into different categories, according to the state-of-the-arts survey papers [9], [10].

1) Whitebox vs. Blackbox attacks: During the development of an adversarial attack, different assumptions for the adversary may exist. For example, the adversary may have different types of access and understanding

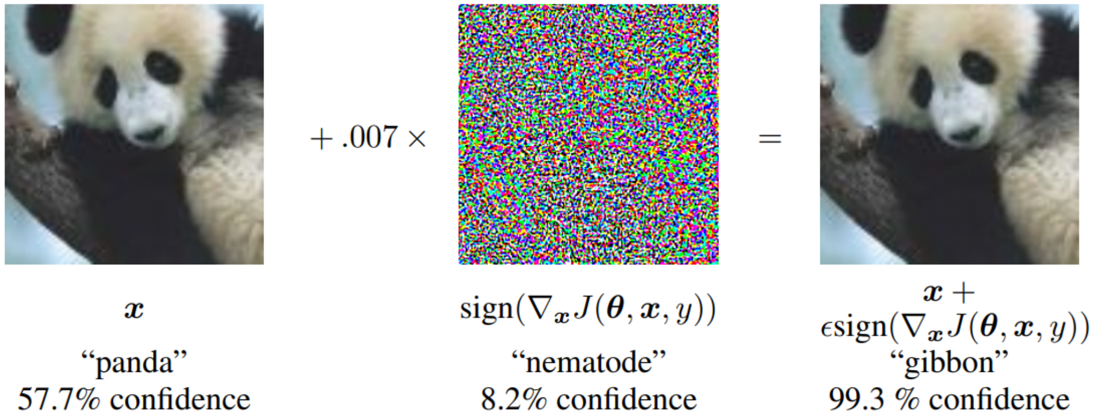

+ .007 ×
=
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Fig. 2. Example of an adversarial example [5]

on the underlying model to be attacked. In this paper, we defined a *whitebox* attack as an attack where the adversary has access to any information on the model to be attacked. For example, the adversary may have information of the type of model, weights, gradient information, or structure of the model. Traditional methods such as the FGSM [5] and BFGS [3] exploit gradient information of the model to be attacked and modify the image accordingly.

On the other hand, a **blackbox** attack assumes that the adversary has no prior knowledge on the model to be attacked. The adversary may query the model with an image and obtain the model output, but may not obtain the underlying weights and structure of the model. Thus, a blackbox attack closely resembles a real-world scenario where the development of the DNN model may be proprietary, and only the predictor is available to the adversary. In the two methods considered in this paper, we assume that the adversary only have access to the model via inputting an image and observing the output. No other information is provided for the adversary.

Tramer et al. [11] observed that adversarial examples have a transferability property. In a black-box attack, the adversary do not have access to the underlying model's gradient to create an attack. However, Tramer et al. observed empirically that adversarial examples occur in large, continuous regions. Thus, the adversarial examples are transfer from one machine learning model to a similar one. Using the transferability property, an adversary can train a similar DNN model to the victim DNN model, and use the new model's weights and gradient to derive an attack before applying it to the targeted victim model.

2) Targeted vs. Non-targeted Attacks: Let F be the DNN model and data (x, y) represent the input image x and the ground truth label y. Then the problem of generating an adversarial example δ can be rewritten as

$$\delta_{min} = \operatorname*{argmin}_{\delta} ||\delta||_2 \text{ s.t. } F(x+\delta) \neq y$$

In a **non-targeted** attack, the adversary simply have to find a perturbation such that $F(x+\delta) \neq y$. In contrary, in a **targeted** attack, the adversarial finds a $F(x+\delta) \neq \hat{y}$ such that $\hat{y}$ is a targeted class. For example, an adversary may want to misclassify a truck into an airplane to force the corresponding system to misbehave by not avoiding the truck in front. In practice, generating an adversarial example for a non-targeted attack is easier than a targeted attack, since the targeted attack restricts the set of possible adversarial examples into a sub-region of all possible incorrect labels.

3) Poisoning vs. Evasion Attack: The attack objective of the adversary also forms a category for the method used. **Poisoning attack** refers to the attack(s) where the adversary aims to corrupt the model to be created. Poisoning attacks generally involve modifying the input training data set to cause the trained DNN to misclassify correct inputs. An example of a poisoning attack is where the adversary provides malicious training data to an open-source data collection. When machine learning models are trained using the malicious data, the resulting models may not predict malicious inputs correctly.

Evasion attack refers to the attack(s) where the adversary aims to provide a malicious input to a trained model. For example, an adversary may want to provide an image to the classifier that evades the correct prediction. As the attack developed in this paper involves causing the DNN to misclassify the input, they are considered to be evasion attacks.

4) On the Existence of Adversarial Examples: A number of researchers have proposed theories and hypothesis on the existence of adversarial examples. However, no general consensus have been reached on why they exist.

Initially, Gu and Rigazio [12] believed that adversarial examples lie in low-probability regions of the solution space. Thus, they are hard to reach by randomly sampling the solution space. Goodfellow et al. [5] proposed that the linear behaviours of the model in high-dimnsional spaces is sufficient to cause adversarial examples. To prove their theory, the authors introduced an attack method, FGSM, to exploit the linearity that quickly creates adversarial examples. Grosse et al. [13] proposed that adversarial examples are drawn from a different distribution than normal inputs. They introduced an statistical test approach to detect adversarial examples based on the theory. Finally, Tanay et al. [14] countered the previous arguments and suggested that the learned decision boundaries of the model are slightly tilted with respect to the data manifold. As a result, adversarial examples can be found where the input image crosses the boundary (i.e., into another classification label). The authors suggested that the existence of adversarial examples is due to over-fitting, and thus can be mitigated through regularisation.

D. Related Work

Many researchers have explored the production and effects of adversarial examples.

Szegedy et al. [3] searches for a minimally distorted input x' by optimising a loss function using the cross entropy. Goodfellow et al. [5] proposed an one step method called FGSM by searching for a point that has the minimum loss value to another label in the original x naught's ϵ-neighbor norm ball. The Basic Iterative Method (BIM) [15] extends the FGSM approach in an iterative manner to heuristically search for perturbed examples with the largest loss value in the l_∞ ball around the original input image.

Jacobian-based saliency map attack [16] introduces a greedy attack aimed to modify pixels with the most influence to the model's output until an adversarial example is produced. Through using a greedy approach, the Jacobian-based saliency map attack is able to quickly generate adversarial examples with little perturbations.

Next, Carlini and Wagner [17] introduced the C&W attack similar to that of Szegedy et al.'s [3]. However, instead of cross entropy loss, C&W optimises the objective based on the marginal loss. When C&W was introduced as an attack, it was able to bypass all state-of-the-arts detection methods at the time.

Finally, several researchers have explored using evolutionary approaches to generate adversarial examples. Su et al. [18] proposed an one-pixel attack using Differential Evolution (DE). This attack has several advantages. First, by only changing one pixel, the perturbation introduced is limited and hard to identify to the naked human eye. Next, the attack is black-box and produces solution that is likely close to the global optimum due to the properties of DE. However, changing only one pixel may also lead to easy detection (e.g., a sudden white dot in the image of a black curtain). Furthermore, if the image is compressed or normalised, then the effects of the attack is likely to disappear. Vidnerova et al. [7] introduces an evolutionary approach to find adversarial examples. Their approach uses a GA to evolve solutions for different machine learning methods. This paper is an extension to their approach by using GA and pGA to evolve adversarial examples. We show that by using pGA, we are able to produce adversarial examples with less l_2 norm perturbation than the GA produced solution.

III. Methodology

This section overviews the proposed methodology and describes how it evolves adversarial examples using both a GA and a pGA. We first introduce the genome representation used in the algorithm. Next, we describe the fitness criteria used for the algorithm. Finally, both implementation algorithm details are described.

A. Genome Representation

In evolutionary computing approaches, individuals to be evolved must be mapped to "genomes" correspondingly. Genomes are a set of parameters that represent the individual, thus the corresponding genome must be able to construct the image. In this paper, each individual (i.e., image) is represented by a three element list, denoting each of the RGB channel. Each value of the individual range from [0,1], mapping to [0, 255] (brackets denote inclusive). During the operation of the algorithm, the values in the genomes are modified to evolve new solutions.

B. Fitness Criteria

As the individuals of a GA evolve, the algorithm must use a fitness evaluation to choose individuals for offspring selection and survivor selection. Individuals that are more fit (i.e., closer to the solution) are usually chosen to reproduce, as it evolves the entire population towards the objective. However, evolutionary approaches use randomness to retain less fit individuals to prevent one strong performing member from quickly overtaking the entire population. One member of the population overtaking the entire population leads to premature convergence on a local optimum. Traditional implementations of GAs, the fitness criterion is defined as

$$fitness = globalSolution - currentSolution$$

For example, in the eight-queens problem, a player places eight queens on a 8x8 chess board such that no queens intersect with each other (vertically, horizontally, or diagonally). The fitness criterion for the eight queens problem is thus the number of conflicting queens, where zero is a solution to the problem.

In the adversarial example solution space, we represent the fitness of the individual as

$$fitness = \text{argmax model.predict}(\delta) \text{ s.t. } \delta \neq \hat{y}$$

Essentially, we obtain the probability labels of the classification of the current individual using the model. Next, the highest probability in the class that does not belong to the ground truth (i.e., any other class) is used as the fitness of the individual. As such, we promote members that are classified incorrectly into other classes to reproduce in the algorithm.

C. Mutation and Crossover Operators

During each generation of the algorithm, individuals in the population undergo mutation and crossover operations. In the proposed implementation, each mutation operation involves choosing three random pixels in the individual and changing the a random RGB value by 10%. The mutation of a pixel may increase or decrease the value, chosen randomly. When individuals crossover to produce children, a random (x,y) location is chosen on the image. All subsequent pixels are then swapped between the two parent images to produce the children.

D. Evolving Adversarial Examples Using a GA

We first reproduce the results from Vidnernova et al's [7] approach using a simple GA. Algorithm 1 shows the process of generating an adversarial example using a GA.

Algorithm 1: Simple GA algorithm

```
initalise population;
for i to generation do
    mutate(population);
    for number of children do
        parent1, parent2 = tournamentSelection();
        child1, child2 = crossover(parent1,
          parent2);
        mutate(child1);
        mutate(child2);
        population.append(child1, child2);
    end
    calculateFitness(population);
    sort(population);
    survivorSelection(population);
    if bestMember == adversarial then
        return bestMember;
    end
end
return bestMember;
```

First, a population of individuals are initialised using the input image to be modified. Next, until the max number of specified generation is reached, the algorithm repeats the following. Each member of the population is mutated once (to promote change and diversity in the population). Then, two tournamentSelections are used to select two random individuals to reproduce. A tournamentSelection randomly samples the population for the tournamentSize number of individuals, then each individual competes in a round-robin tournament to select the most fit member in the sample for reproduction. Tournament selection prevents the most fit members of the population from quickly overtaking the entire population, leading to premature convergence on a local optimum. The resulting parent members then crossover to create two children. The two children undergoes a mutation operation before they are added to the rest of the population. At the end of each generation, the population is sorted and culled down to the original size via another tournamentSelection. However, the best few members are kept in every generation in a process known as elitism. Elitism ensures that the survival of the best performing members to promote convergence. If an adversarial example is found, then the algorithm stops and returns the image.

E. Evolving Adversarial Examples Using a pGA

In the pGA approach, multiple islands of individuals evolve independently to maintain population diversity. The algorithm is similar to that of a simple GA approach. However, during the initialisation stage, x islands of populations are created instead. During each generation, only individuals on the same island can reproduce with each other. After every N generation, a selected portion of each island "migrates" to another island in a circular fashion (i.e., individuals from island 1 migrate to island 2, individuals from island 2 migrate to island N, and individual from island N migrate to island 1). Through maintaining three independent islands, pGA maintains diversity in the global population as each island are randomly evolving in a different direction of the solution space. Migration between islands prevents the algorithm from prematurely converging on a sub-optimal solution. As such, the pGA is able to produce solutions that are globally optimal, or close to the global optima. Algorithm 2 shows the pGA implementation to produce an adversarial example.

Algorithm 2: Parallel GA algorithm

```
initalise x islands of population;
for i to generation do
    for each island do
        mutate(population);
        for number of children do
            parent1, parent2 =
              tournamentSelection();
            child1, child2 = crossover(parent1,
              parent2);
            mutate(child1);
            mutate(child2);
            population.append(child1, child2);
        end
        calculateFitness(population);
        sort(population);
        survivorSelection(population);
        getBestMemberFromEachIsland();
        if bestMember == adversarial then
            return bestMember;
        end
    end
    if i % migrationCycle == 0 then
        migrate();
    end
end
getBestMemberFromEachIsland();
return bestMember;
```

IV. Implementation

To attest the feasibility of the approach, we implement the GA and pGA image modification algorithm in python. A DNN of ResNet-20 model is used as the underlying model to be attacked. The python scripts used to train and run the model has been provided by the courtesy of Mick Langford[1].

Several experiments were conducted to compare the two approaches. First, we generate five adversarial examples by running the algorithm on the same input image five times. The 2-norm distance (sum of pixel difference values) is used as the metric. The same image is used to compare the effectiveness of the two approaches. Next, we use another input image to demonstrate that the algorithm works on any input image from the CIFAR-10 dataset. We show that we can generate examples with low perturbation where the DNN classifier fails to classify the image correctly. Finally, we show that we can strengthen the attack to force the DNN to predict incorrectly with a confidence interval of $\geq 90\%$.

The experiment can be reproduced using the source code located at https://github.com/EvilJuiceBox/parallelGAadversarialexample. The results and images of the experiment is stored in the folders experiment1 and experiment2. First, the DNN is trained using the script **train_cifar10_dnn.py**. The resulting model is saved in the file **cifar10_dnn.h5**. The experiments are written in the file **main.py**. A python virtual environment has been prebuilt in the source directory with the required support libraries to run the experiment.

A. Experiment 1: Generating Adversarial Input

In the first experiment, we use a simple GA as described in section III-D to generate several adversarial examples. Table I shows the evolutionary parameters used for the simple GA implementation. Figure 3 shows the original image before the perturbation is added.

Configuration	Value
Number of Generation	Up to 1000 generations
Population Size	25 individuals
Number of Children	3
Tournament Size	3 comparisons
Crossover Point	Random
Mutation Rate	100%
Elitism	True

TABLE I

Evolutionary Parameters for the simple GA experiment

The GA implementation is executed on an arbitrary image taken from the test set of the CIFAR-10 dataset. Note that while the algorithm is allowed to run for 1000 generation, the algorithm was able to produce a solution usually before the 100th generation. Figure 4 shows the

[1] Mick Langford, Michigan State University.

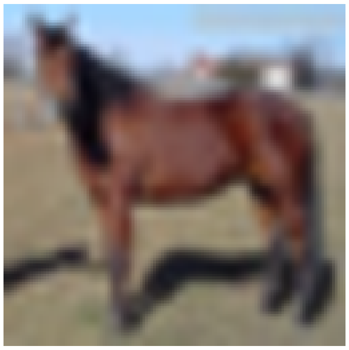


Fig. 3. Original horse image used before modification

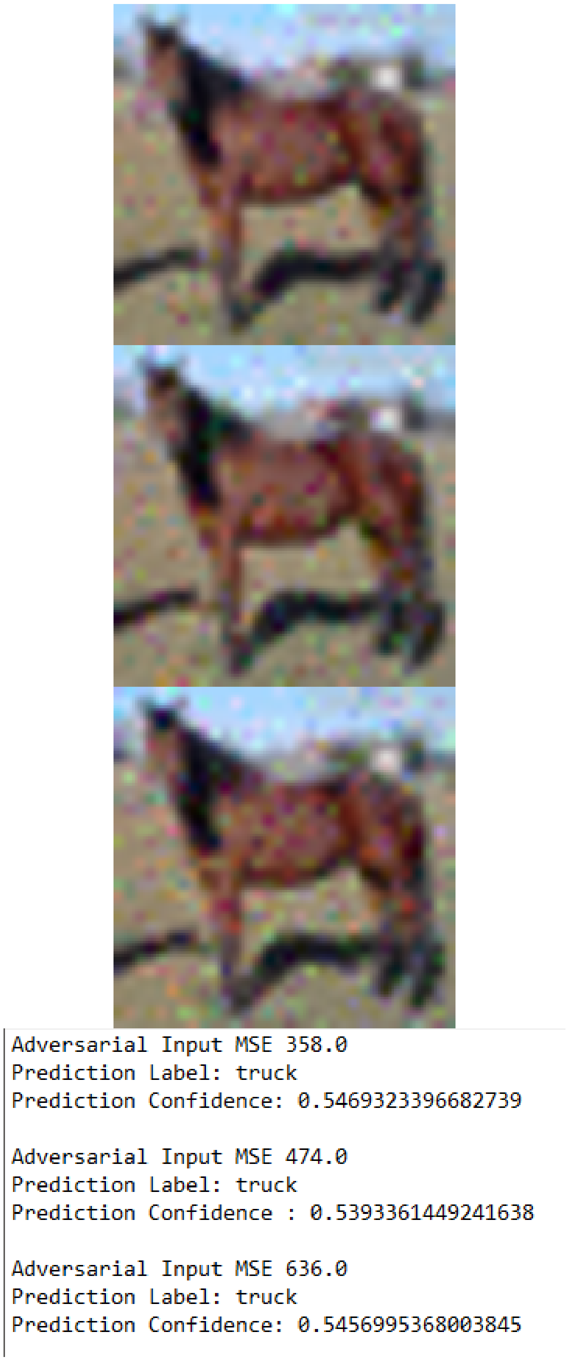
best three resulting images from the experiment, sorted by Mean Square Error (MSE). The MSE is calculated as the sum of the l_2 euclidean distance between the two images. The last subfigure maps the corresponding image to their incorrectly predicted output, with the corresponding confidence intervals.

The results from the first image indicate that the algorithm has successfully evolved a solution that the DNN fails to classify. Empirically, the GA induces noise to the image that modifies the image appearance. However, to the human naked eye, the object in the image can still be clearly identified as a horse. Note that while all three of the produced images are classified as "truck", the label of the image may not always be a truck. For example, the fourth and fifth images produced are of label "deer" and "frog" respectively. We hypothesise that for this particular solution space, the subspace of the horse is close to that of a truck, thus the perturbations with low MSE is likely to reside in the intersection of horse and truck.

Using the configuration from Table II, we create adversarial examples using the same input image. The best three adversarial examples produced by the algorithm is shown in Figure 5. Comparing the two approaches, the generation count required to produce a solution in the pGA approach requires on average twenty less generations than the simple GA implementation. Furthermore, the MSE score for the solutions produced are lower in the pGA approach, implying that the pGA approach produces adversarial examples that are closer to the original input image.

Configuration	**Value**
Number of Generation	Up to 1000 generations
Population Size	15 individuals
Number of Islands	3 islands
Migration Cycle	Every 10 generations
Number of Children	3
Tournament Size	3 comparisons
Crossover Point	Random
Mutation Rate	100%
Elitism	True

TABLE II
EVOLUTIONARY PARAMETERS FOR THE pGA EXPERIMENT

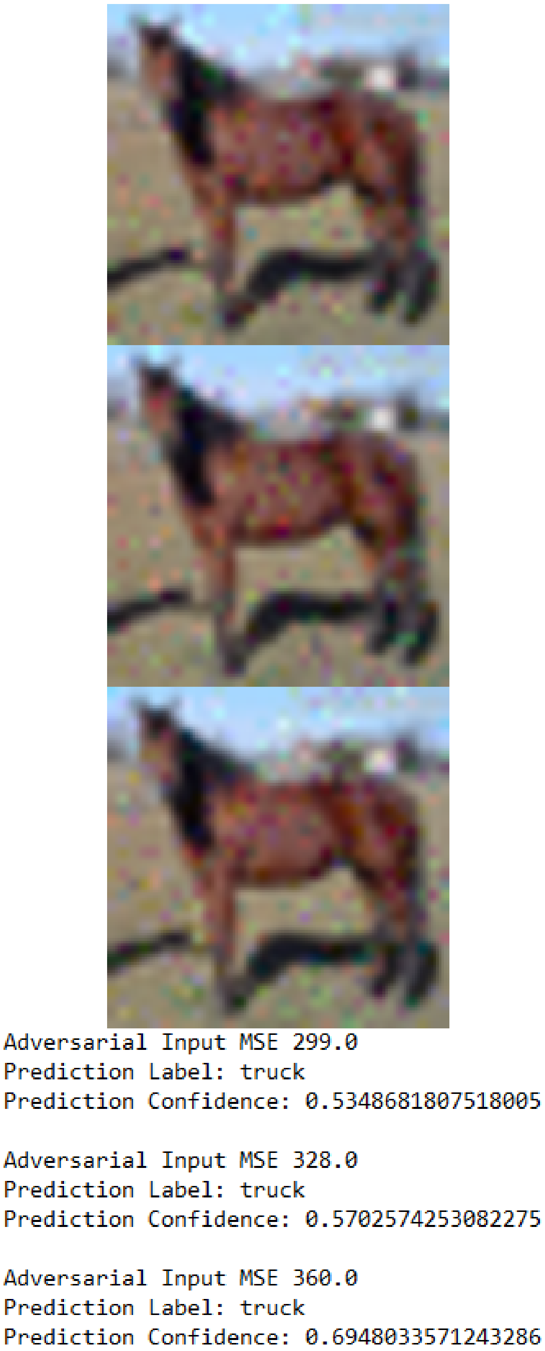


```
Adversarial Input MSE 358.0
Prediction Label: truck
Prediction Confidence: 0.5469323396682739

Adversarial Input MSE 474.0
Prediction Label: truck
Prediction Confidence : 0.5393361449241638

Adversarial Input MSE 636.0
Prediction Label: truck
Prediction Confidence: 0.5456995368003845
```

Fig. 4. The best three performing adversarial examples from the GA algorithm



```
Adversarial Input MSE 299.0
Prediction Label: truck
Prediction Confidence: 0.5348681807518005

Adversarial Input MSE 328.0
Prediction Label: truck
Prediction Confidence: 0.5702574253082275

Adversarial Input MSE 360.0
Prediction Label: truck
Prediction Confidence: 0.6948033571243286
```

Fig. 5. The best three performing adversarial examples from the pGA algorithm

B. Experiment2: Parallel Genetic Algorithm - High Confidence Attacks

In the second experiment, we attempt to increase the confidence of the resulting adversarial examples. We first produce a set of adversarial examples using a different input image. The original input image used is shown in Figure 6. The evolutionary parameters used in this experiment is the same from experiment 1. The algorithm terminated before the 30th generation for every image produced.

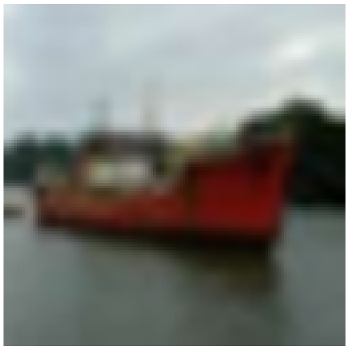
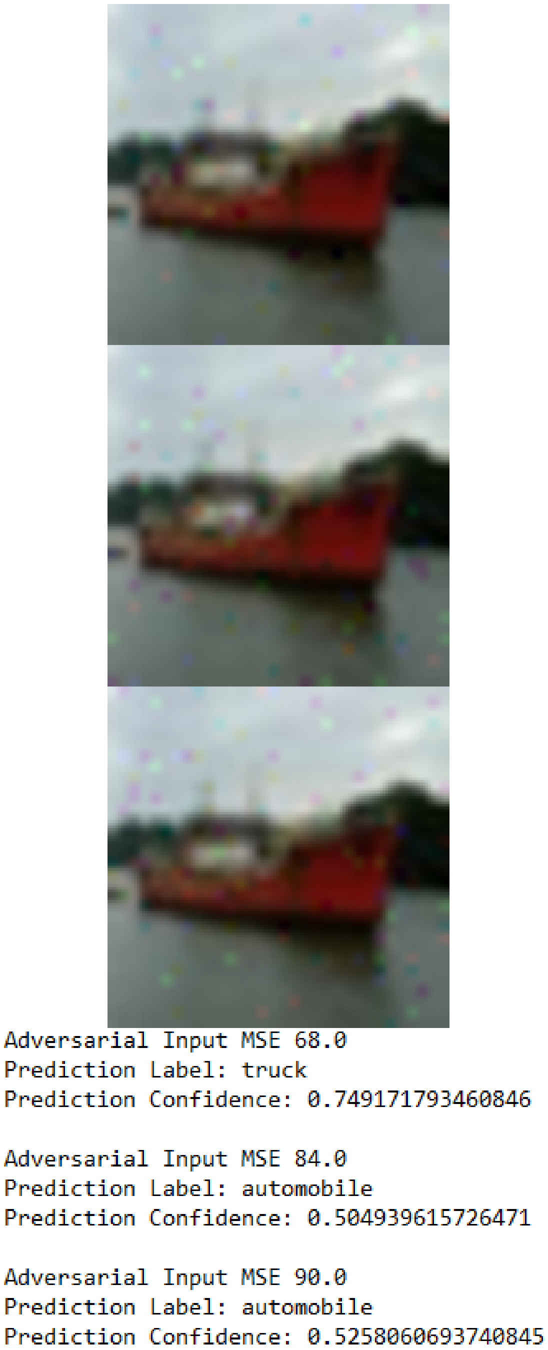


Fig. 6. Original image used in figure two. The correct label is ship.

Figure 7 shows the best three results of the pGA algorithm with the lowest MSE to the original image. Compared to the horse image, the ship image is easier to perturb, as the MSE compared to experiment 1 is overall lower. The first subfigure shows the adversarial example with the lowest MSE that causes the image to be misclassified. The difference in the image and the original are barely perceptible by the human eye.

To show that our approach can create strong adversarial examples that cause the DNN to misclassify the input image with a confidence interval of more than 90%, we introduce a new criteria to the fitness evaluation of the algorithm. Figure 8 shows a collection of images generated by the algorithm that are misclassified by the DNN model. The MSE of the images that cause a misprediction with high confidence is notably higher than normal. If we closely observe the perturbation added to the images, then we can identify patterns of pixels added by the algorithm. In a safety critical application, if an adversary can force the DNN to mispredict with high confidence, then the potential damage may be significant.

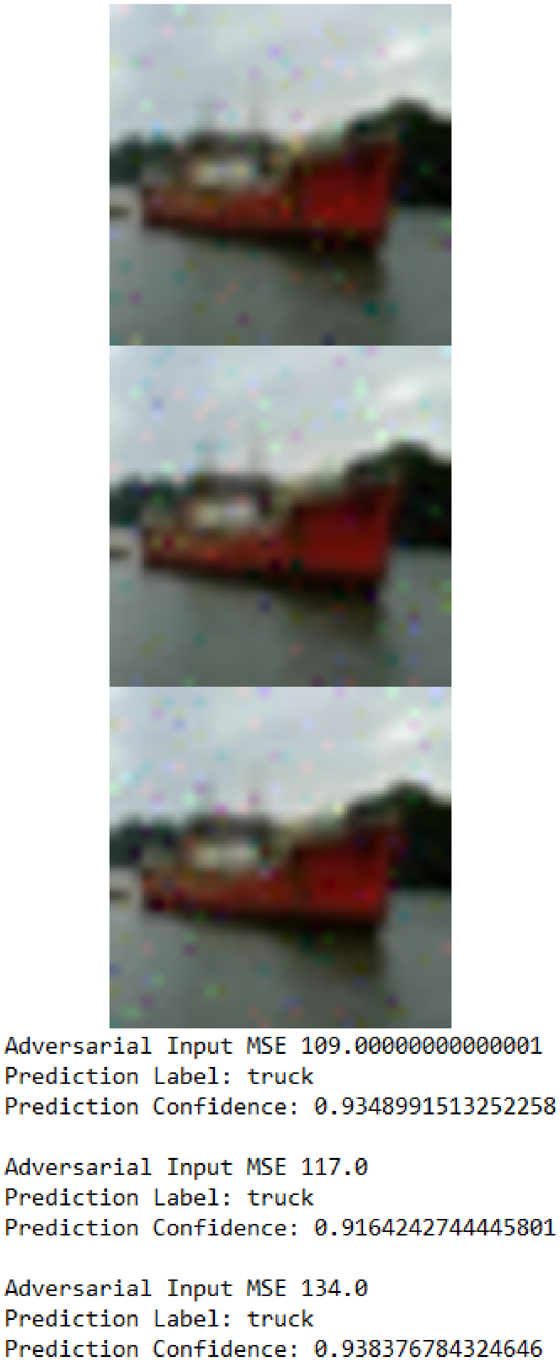


```
Adversarial Input MSE 68.0
Prediction Label: truck
Prediction Confidence: 0.749171793460846

Adversarial Input MSE 84.0
Prediction Label: automobile
Prediction Confidence: 0.504939615726471

Adversarial Input MSE 90.0
Prediction Label: automobile
Prediction Confidence: 0.5258060693740845
```

Fig. 7. The best three performing adversarial examples ship modification



```
Adversarial Input MSE 109.00000000000001
Prediction Label: truck
Prediction Confidence: 0.9348991513252258

Adversarial Input MSE 117.0
Prediction Label: truck
Prediction Confidence: 0.9164242744445801

Adversarial Input MSE 134.0
Prediction Label: truck
Prediction Confidence: 0.938376784324646
```

Fig. 8. The best three performing adversarial examples ship modification with high confidence

V. Threats to Validity

The results in this paper are limited to adversarial examples generated with the GA and pGA implementations on a DNN with ResNet-20 architecture. The effectiveness of the approach may vary with each run, as evolutionary algorithm relies on non-determinism to evolve solutions. To ensure the feasibility of the approach, multiple runs of the algorithm used in this paper on different sample inputs have been tested.

VI. Conclusion

Adversarial examples are malicious modified input images that a Deep Neural Network fails to classify. This paper has introduced a parallel genetic algorithm approach to generate adversarial examples. We have demonstrated that by using the proposed approach, we can construct input images that the Deep Neural Network misclassifies, where a human can still clearly identify the object. Future research will explore potential optimisation to reduce the difference between the original input image and the adversarial example. Additionally, other image data sets with higher resolution (e.g., ImageNet) may be explored to study the effects and magnitudes of the perturbation. Finally, robustness of the underlying DNN model can be potentially improved by retraining the model with adversary examples generated with this method.

References

[1] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," *arXiv preprint arXiv:1912.11370*, vol. 6, no. 2, p. 8, 2019.

[2] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pp. 547–550, 2002.

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[4] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1625–1634, 2018.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

[7] P. Vidnerová and R. Neruda, "Vulnerability of classifiers to evolutionary generated adversarial examples," *Neural Networks*, vol. 127, pp. 168–181, 2020.

[8] Shyh-Chang Lin, W. F. Punch, and E. D. Goodman, "Coarse-grain parallel genetic algorithms: categorization and new approach," in *Proceedings of 1994 6th IEEE Symposium on Parallel and Distributed Processing*, pp. 28–37, 1994.

[9] H. Xu, Y. Ma, H.-C. Liu, D. Deb, H. Liu, J.-L. Tang, and A. K. Jain, "Adversarial attacks and defenses in images, graphs and text: A review," *International Journal of Automation and Computing*, vol. 17, no. 2, pp. 151–178, 2020.

[10] A. Serban, E. Poll, and J. Visser, "Adversarial examples on object recognition: A comprehensive survey," *ACM Comput. Surv.*, vol. 53, June 2020.

[11] F. Tramer, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "The space of transferable adversarial examples," 2017.

[12] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014.

[13] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," 2017.

[14] T. Tanay and L. Griffin, "A boundary tilting persepective on the phenomenon of adversarial examples," *arXiv preprint arXiv:1608.07690*, 2016.

[15] A. Kurakin, I. Goodfellow, S. Bengio, *et al.*, "Adversarial examples in the physical world," 2016.

[16] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European symposium on security and privacy (EuroS&P)*, pp. 372–387, IEEE, 2016.

[17] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, pp. 39–57, IEEE, 2017.

[18] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.