Journal of Dynamical Systems and Complexity (2026) vol.1(3): 11-22
eISSN: 3122-4326, DOI: 10.66131/JDSC12202611-23

Simple Gaussian Filter can Protect Images of Noise Injection in a Classification Task with Deep Neural Networks

Luis Gerardo de la Fraga, Christian Ruiz Hernández*

Cinvestav, Computer Science Department, Av. IPN 07460 Mexico City, Mexico

Abstract

We study the classification of a set of images under additive Gaussian noise and additive colored Pink noise. We observe that the very simple denoising Gaussian filter obtain the best results for a classification problem using a Deep Convolutional Network. We were able to generate an attack of noise injection to change the class of two images, and the correct class can be obtained applying also the very simple Gaussian filter.

Keywords— Convolutional Neural Networks, Deep Neural Networks, Noise Attack, Denoising

1 Introduction

When discussing Neural Networks or Convolutional Neural Networks (CNNs), it is often assumed that "all networks can tolerate some degree of noise," yet the specific amount or type of tolerable noise is rarely quantified. In this work, we explore this topic within the context of CNNs applied to classification problems. Specifically, we propose adding additive Gaussian noise to images to establish a reference point. We utilized a pretrained EfficientNet model, originally trained to classify eight shapes as described in the thesis [1].

The main motivation for researching this area is the cost of maintaining large convolutional neural network architectures. The big models for classification or detection use a minimal input size of 224 x 224. The industry tries to save computational power by using reduced architectures. We can see a clear reduction in the number of layers, number of neurons, or the use of small input sizes, resizing the input sizes to adapt in any situation. The use of limited neural networks for mass use in different fields is the reason why different weaknesses were found in convolutional neural networks on classification problems.

Currently, numerous CNN architectures exist, predominantly used for classification and detection. These models are widely deployed in critical applications, such as autonomous driving systems and medical diagnostics (e.g., detecting ailments via X-rays or magnetic resonance imaging). Given these high-stakes contexts, it is important to know their real weaknesses. One such vulnerability is noise tolerance, which raises the question of how effectively image noise can be mitigated using various preprocessing techniques.

It is important to note that our experiments employed a reduced version of the pretrained EfficientNet model, rather than the original complete architecture. This model was reduced in both image input size and the number of filters [1].

It is fundamental to identify the noise and the level of noise to create effective methods to increase the accuracy in models with different environmental noise. We work to find the most efficient method to mitigate the negative effect of noise over the images.

Here is the importance of this work. We want to find the levels of noise that the convolutional neural networks can support. Because if we want to make a solution of misclassification, we need to start identifying the noise. After that, we test the most simple methods to fix them, showing that it is not necessary to use more complex solutions like Generative Adversarial Networks or Autoencoders.

*Corresponding author: christian.ruiz@cinvestav.mx
Received: December 20, 2025, **Published:** October 30, 2026
Editor: J.M. Muñoz-Pacheco

This work was inspired by the article [2], they basically use a targeted noise using differential evolution (DE) to find the best noise to change the class where an image belongs. They coded the solution with two values (x, y) for the position of the pixel, and three values RGB to specify the color for each infected pixel, then each solution given by the DE heuristic contains $5n$ values, for n pixels of noise. This is a very important point because this implementation increases the complexity too fast if the number of pixels is high. We tested over models with an input size 64×64 while the article [2] uses an input models 32×32. In our test, we did not find solutions from 1 to 30 pixel attacks over images of 64×64 pixels, and this noise was not enough to change the class. We were able to attack images of size 16×16 pixels.

In the article [2], authors mention that on datasets like ImageNet where the images size is 227×227, then they resize the images to the input size of the convolutional neural network architecture, using the nearest pixel algorithm to resize the images. This is the simplest algorithm to assign the pixel value in an image resize operation. This idea was an interesting method to attack any model as a black box, like in a real scenario. Then the community continues investigating the attack. The article [3] made a comparison between the performance on methods the one-pixel attack, and brute force, using differential evolution, and genetic algorithms as search methods. They observe that the differential evolution is not the most efficient way and open the problem to use other search methods.

In the same way, we use Gaussian noise to contaminate the image models because it is another of the most commonly studied noises. For example, in the article [4], a CNN with Gaussian noise over chest X-ray images was trained as a binary classifier. They discover that this kind of trained model can increase the specificity while the general accuracy is maintained. Then, with this method to train with noised samples, a CNN is powerful, but they did not try to clean the samples during the test.

Another recent example is in the article [5], they use an anisotropic Gaussian filter to train a CNN that helps to denoise images that contain different levels of noise. Their work demonstrates that it is possible to use a preprocessed filter on combinations with CNN to learn to denoise images and reduce the weakness of the CNN.

There are many ways to solve this problem on different publications. In the article [6], the authors mention the advantages to use genetic programming to solve classification problems, they showed a good result using a tree with different known descriptors like Gabor filter, Sobel, HOG and poolings (average, minimum and maximum). They modeled the classification problem like combinatorial optimization problem. Using a combination of these descriptors and another simple classifiers well known like random forest, Support Vector Machine and logistic regression. They fixed the classifiers in the architectures and evolve the combinations of descriptors.

Another study [7] suggests that feature extraction can be divided into regions of interest (ROIs), not that similar features frequently appear in specific areas within the same dataset. The authors utilize two geometric configurations for these regions: rectangles and squares. For rectangular regions, it is necessary to define the position as well as the height and width dimensions. In contrast, for square regions, only the position and a single side length are required to determine the dimensions. They modeled the problem of searching for the optimal combination of descriptors as a classification task, similar to the approach in [6]. In this work, they employ k-means as the sole classifier to evaluate and evolve various permutations of coordinates and sizes. The objective is twofold: to identify the best descriptor combinations and to discover the most common regions that yield superior features using both geometric shapes.

In the book [8], chapter 8, the author mentions that it is possible to use genetic programming as a feature extractor in classification problems by searching in parallel the regions of the most common places over the images to get the maximum number of features.

One objective on this work is the exploration of the most simple black box attack named one pixel attack [2]. Mention that it is possible to find some noised images modeling the problem as an optimization problem. Where the CNN is a black box function, and the problem is maximize the confidence of the different label class than the original. The authors use differential evolution to solve the optimization problem.

Another way to see the misclassify problem is like a white box or gray box. Where we know partial information about the CNN, or we can have full access to CNN model. This is the case explored in the article [9], the algorithm attack the images adding noise over all the image starting with a small perturbation to big perturbation, and stops the search when the image classification changes. Using information about the gradients of the models as a compass. The two differences are the way to search the noise and how it is applied over all the pixels.

This article is organized as follows. In Section 2, it is explained how the Gaussian noise and the one pixel attack noise are used. In Section 3, the results of our experiments are shown. In Section 4, some discussions about this work are given. Finally, in Section 5, some conclusions of this work are described.

2 Method

2.1 Gaussian additive noise

The Gaussian noise is the most common and well-studied noise. This kind of noise is very simple to generate and add to images because we only need to generate pseudorandom numbers with a normal distribution and only use two parameters, the mean and the standard deviation, to increase and decrease the level of noise.

For experiments, the pixel values of the grayscale input image are normalized to values from 0 to 1, and we propose 16 combinations of Gaussian noise with four means [0.2, 0.4, 0.6, 0.8] and four standard deviations [0.06, 0.12, 0.18, 0.24].

We create an image with these generated numbers and add it to the original image, using the clip function to limit the range of values to [0, 1]. The clip function sets values smaller than 0 to 0 and values greater than 1 to 1. In Figure 1, we visualize this process.

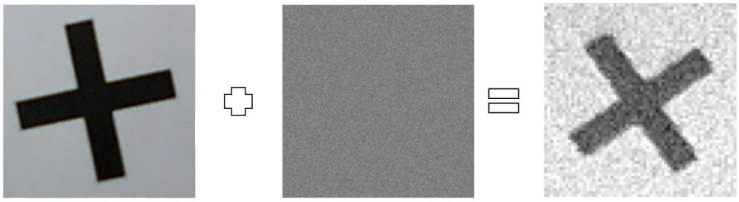


Figure 1: Illustration about how is added the Gaussian noise.

The main idea of this attack is to get a sample from the dataset and test by adding sixteen noise combinations. After that, we identify the noise combination that changes the classification label of the noisy image.

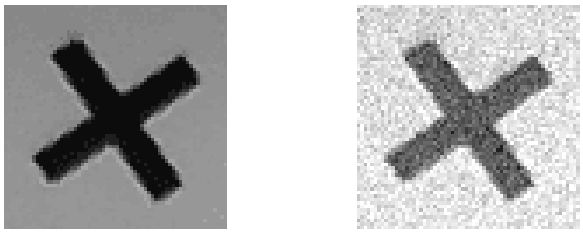


Figure 2: On the left cross figure clean and on the right noisy image with Gaussian noise using 0.2 as mean and 0.06 as standard deviation.

$$\mathbf{A}+\mathbf{B}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}+\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix} \tag{1}$$

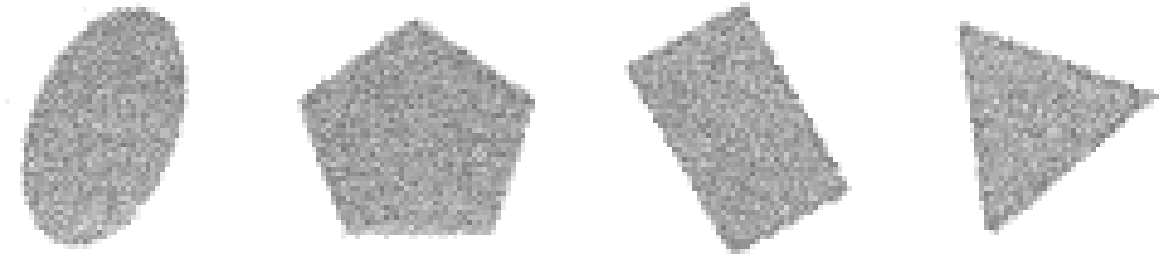
Equation (1) shows the operation to add the original image to the noise. The matrix A represents the original image, and the matrix B represents the noise matrix. Each element of matrix A is added to the corresponding element of matrix B at the same position to generate the output matrix, which is the noisy image.

With these different types of noise, we test the pretrained EfficientNet model and obtain the results shown in Table 1. We evaluate the combination with the level of noise with mean 0.6 and standard deviation 0.06 with an accuracy around 0.25; example images with this noise are shown in Figure 3.

Table 1: EfficientNet All noises

Different levels of noise		
Mean μ	Standard deviation σ	Accuracy
0.2	0.06	1.000
0.2	0.12	0.864
0.2	0.18	0.160
0.2	0.24	0.125
0.4	0.06	1.000
0.4	0.12	0.160
0.4	0.18	0.133
0.4	0.24	0.131
0.6	**0.06**	**0.254**

(a) Cross shape **(b)** Star shape **(c)** I shape **(d)** Line shape



(e) Oval shape **(f)** Pentagon shape **(g)** Rectangle shape **(h)** Triangle shape

Figure 3: Example of shapes with Gaussian noise (mean 0.6, std. dev. 0.06).

2.2 One pixel attack

The main idea of this attack is to assume that there exists a trained and unknown convolutional neural network (CNN) architecture, used as a black box, to classify images. The original authors modeled this problem as an optimization problem where the classification error is the target value to maximize. We then evaluate the CNN output as an indicator. We use only the case of a non-targeted attack, then in this variant is only necessary to minimize the confidence of the original class.

As we mentioned in the introduction, we test a one-pixel attack on our pretrained EfficientNet model. First, we test with original images of size 64×64 for the one-pixel non-targeted attack, which means we search for the best solution where the classification changes without searching for a specific target class. We adapted the attack to our pretrained EfficientNet model because it uses only one channel; therefore, we can reduce the search space by encoding the solution as [x,y,intensity] instead of [x,y,R,G,B].

With the original input size, we did not find a solution with this technique. Then, we resized the input image from 64×64 to 16×16 using the same nearest-neighbor filter mentioned in the article [2]. We tested with 30 and 40 pixels on rescaled images, and we found a satisfactory solution where the original pentagon shape was misclassified as a cross.

The optimization problem is described in (2). The classification function is f. The element $\mathbf{x}$ is the input image and the e function describes the noise process. The e function returns a vector with the same size of the input image. There is a constraint where the norm L_0 of e needs to be below or equals d, where d is the number of pixels attacked. That means the vector will contain 0 values where the values of pixels were not attacked to keep the same size of the image. Then the noise is a simple additive process. The scheme attack sees the f function as a black box and the e function is defined by the evolution process. It is assumed the classifier always return a vector with the confidences for each class. The objective is t maximize the class target confidence value.

$$\begin{aligned} \max_{e(\mathbf{x})^*} \quad & f_{adv}(\mathbf{x} + e(\mathbf{x})) \\ \text{subject to} \quad & \|e(\mathbf{x})\|_0 \leq d, \end{aligned} \tag{2}$$

The flowchart describing how the one-pixel attack works is shown in Fig. 4. The attack requires a method to encode each attacked pixel in a linear fashion. For each pixel, we must specify the x and y coordinates and the intensity value. Consequently, for grayscale images, it is necessary to optimize three values per pixel, whereas for RGB images, it is necessary to optimize five values per pixel.

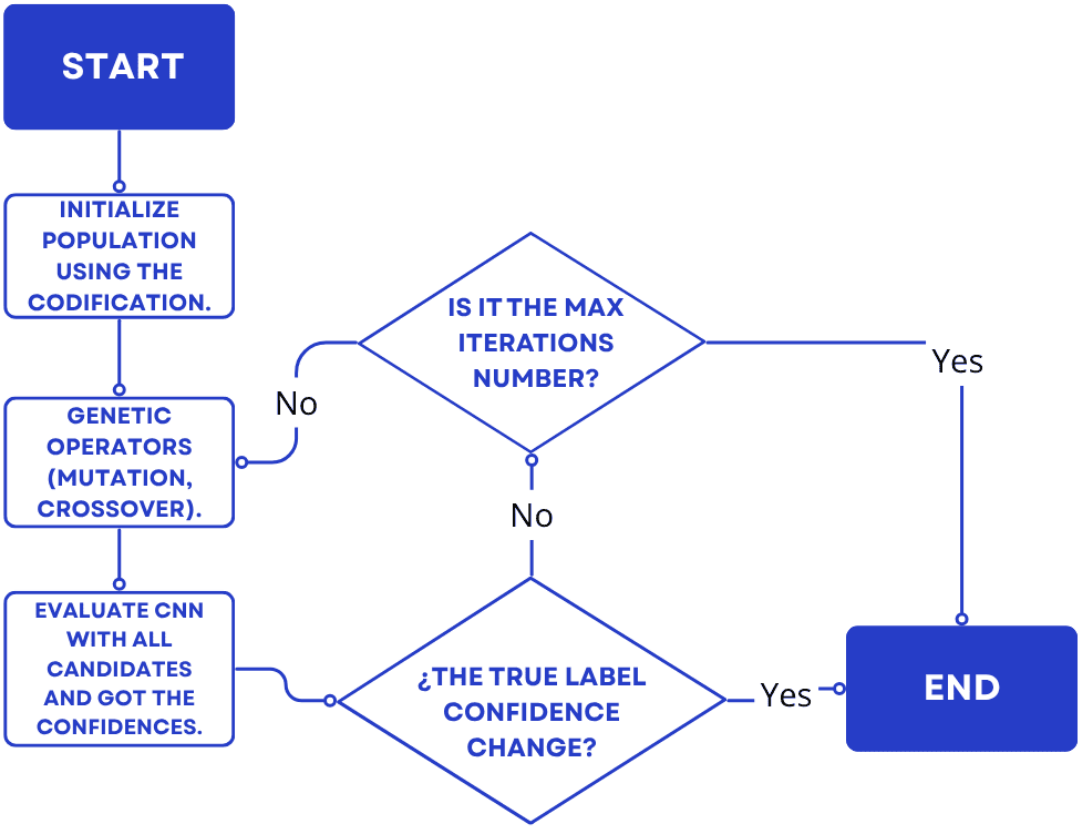


Figure 4: One pixel attack flowchart

3 Results

3.1 Testing additive Gaussian noise

We trained the EfficientNet model with 60,000 noisy images during 100 epochs with a batch size of 128. We then test the model on 480 random noisy images (60 per class) to validate it. Then we obtain the results shown in Tab. 2.

Table 2: EfficientNet retrained with noisy images with μ=0.6 and σ=0.06

Different levels of noise		
Mean μ	Standard deviation σ	Accuracy
0.2	0.06	0.795
0.2	0.12	0.691
0.2	0.18	0.370
0.2	0.24	0.150
0.4	0.06	0.968
0.4	0.12	0.870
0.4	0.18	0.800
0.4	0.24	0.325
0.6	**0.06**	**0.985**
0.6	0.12	0.547
0.6	0.18	0.337
0.6	0.24	0.314

The results indicate that the model learns about the noise very well but with less noise, with mean value less than 0.6, the model did not keep the accuracy.

3.2 Retraining classification model with Gaussian filter

This method consists of training the EfficientNet model with the same dataset of 60,000 noisy images, but using the Gaussian filter to denoise the image. In this case, we retrained that model to learn how to correctly classify images with that filter applied over noisy images. We obtained the results shown in Table 3, we can see that obtained accuracy is 1.0.

Table 3: Efficientnet using Gaussian filter to denoise summary results

Different levels of noise		
Mean μ	Standard deviation σ	Accuracy
0.2	0.06	1.000
0.2	0.12	1.000
0.2	0.18	1.000
0.2	0.24	0.950
0.6	**0.06**	**1.000**
0.6	0.12	0.981
0.6	0.18	0.502
0.6	0.24	0.268

3.3 Pink noise

We tested with another kind of noise named pink noise. This noise is one of the noises called colored noise. This kind of noise basically takes a random noise, applies the Fourier transform to change over the frequency domain, and applies a filter using Equation (3). Where f is the frequency. Different color noise changes the exponent, sometimes named alpha or beta in the literature, but in this case we try with pink noise, and this value is fixed in 1. After we calculate Equation (3) multiply with the noise values to generate the filtered. Finally, it is necessary to apply the inverse Fourier transform to obtain the noise that will be added over the input signal.

$$S(f) \propto \frac{1}{f^{\alpha}}, \tag{3}$$

in Equation (3), S is the power spectral density. If it is considered a 2D signal, 't' becomes to the x and y coordinates. Then, when the fast Fourier transform is applied, it will work with the magnitude of the frequency.

Algorithm 1 Pink Noise

Require: A is a matrix of $n \times n$ size with random values and a uniform distribution.
Ensure: Pink noise on pixel values context, in range [0, 1]

$F \leftarrow \text{fftshift}(\text{fft2}(A))$ ▷ Apply fast Fourier transform 2D and move the zero frequency to the center
for each pixel (i, j) with the same size $n \times n$ **do**
  $f \leftarrow \sqrt{u_i^2 + v_j^2}$ ▷ Calculate the frequency
  $G(i, j) \leftarrow \frac{1}{\max(f,1)}$ ▷ Calculate the filter
end for
$F_{\text{filtered}} \leftarrow F \odot G$ ▷ Multiply the values of the filter with the original signal to obtain the filtered signal on the frequency domain.
$I_{\text{imaginary}} \leftarrow \text{ifft2}(\text{ifftshift}(F_{\text{filtered}}))$
$I_{\text{real}} \leftarrow \text{Real}(I_{\text{imaginary}})$ ▷ Only uses the real component
$I_{\text{pink}} \leftarrow \frac{I_{\text{real}} - \min(I_{\text{real}})}{\max(I_{\text{real}}) - \min(I_{\text{real}})}$ ▷ Apply a normalization a $[0, 1]$
return I_{pink}

This implementation was based on GitHub https://github.com/felixpatzelt/colorednoise and fallow the instructions written by Paul Bourke on his blog https://paulbourke.net/fractals/noise/ to adapt in two dimensions the pink noise. The implementation in Algorithm 1, uses the *numpy* library and its functions FFT2 and IFFT to calculate the fast Fourier transform and its inverse. It is visible in Algorithm 1, that the pink noise is a filter applied over the random noise in the frequency domain. Then pink noise is added over the images to obtain images as shown in Figure 5.

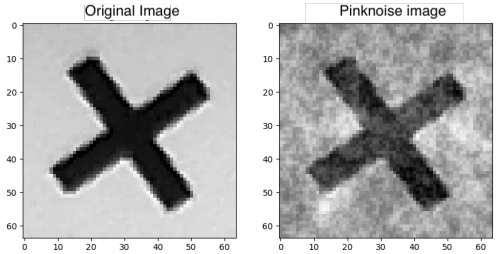


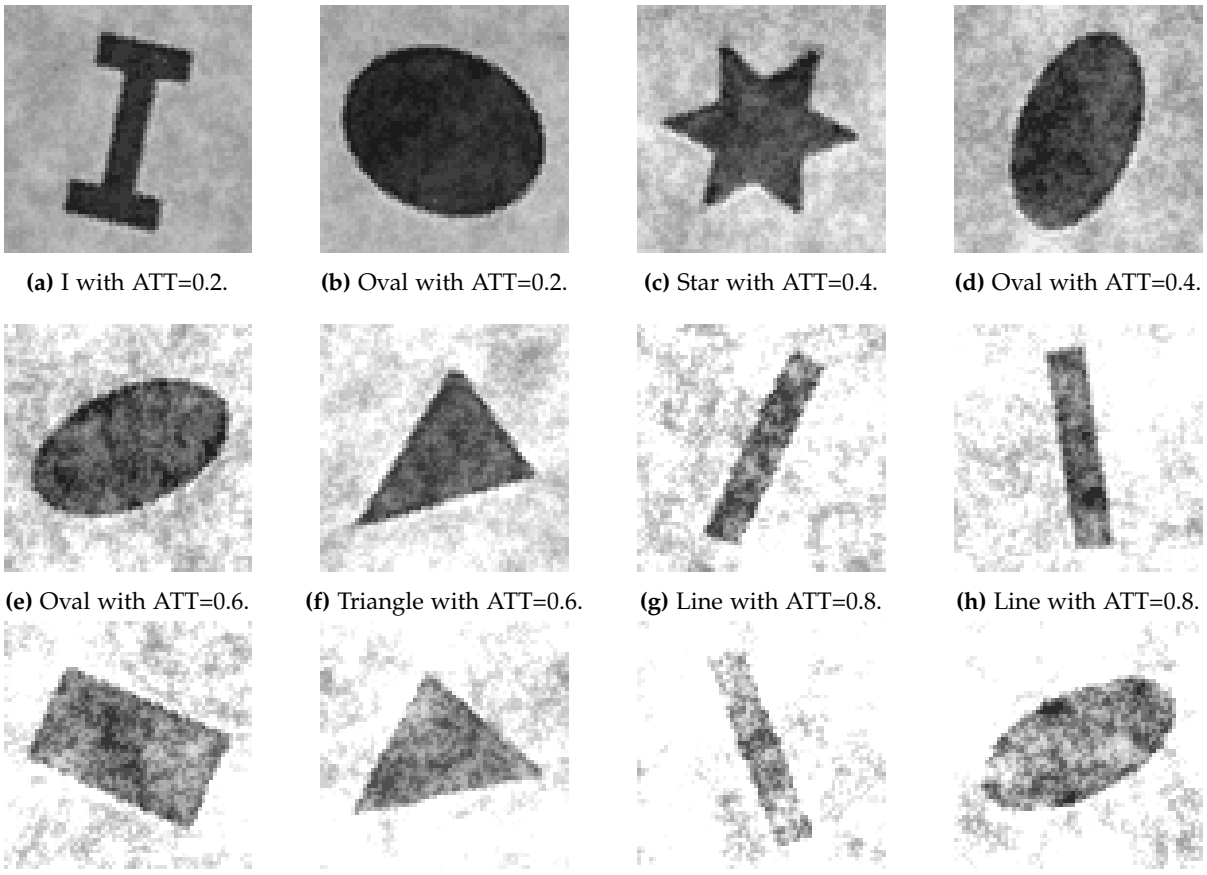
Figure 5: Sample Pink Noise

For using similar parameters, we test with 5 different attenuation levels of noise, [0.2, 0.4, 0.6, 0.8, 1.0]. Where each number of attenuation in Table 4 represents the percentage of intensity for each pixel that will be added. For example, if the pixel intensity of the pink noise is 0.4 and the attenuation factor 0.4, then we only added 0.16 as intensity over the original image.

Table 4: Results using pink noise, tested over EfficientNet retrained and preprocessed with Gaussian filter.

Different levels of noise	
Attenuation	Accuracy
0.2	1.0
0.4	1.0
0.6	**0.9**
0.8	0.5
1.0	0.2

If you consider the Gaussian noise generated values near the mean, then it is equivalent to attenuate the signal. Extrapolating this concept to pink noise we consider test with the same mean as attenuate noise. With this method is possible to see a consistent behavior between the Gaussian noise and the pink noise where using the same attenuation of 0.6 as it is shown on Table 4, we obtain better with the same attenuation values employing our model retrained with the Gaussian noise and Gaussian filter as a preprocessing. It is interesting to observe that, despite not being trained specifically on pink noise at the same noise levels, the model exhibited a similar behavior when using this re-training and pre-processing approach with a Gaussian filter.



(a) I with ATT=0.2. **(b)** Oval with ATT=0.2. **(c)** Star with ATT=0.4. **(d)** Oval with ATT=0.4.

(e) Oval with ATT=0.6. **(f)** Triangle with ATT=0.6. **(g)** Line with ATT=0.8. **(h)** Line with ATT=0.8.

(i) Rectangle with ATT=0.8. **(j)** Triangle with ATT=0.4. **(k)** Line with ATT=1.0. **(l)** Oval with ATT=1.0.

Figure 6: Example of shapes with pink noise with different attenuation levels.

In Figure 6 is shown how the pink noise in two dimensions generate noise similar to clouds. In high levels the distortion is worst than the Gaussian noise because the pink noise generate biggest regions with similar values. This similar regions deform the original figure.

3.4 Testing one pixel attack

We tested with different population sizes: the initial population size was 15 (the default for the differential evolution heuristic in the SciPy library). Subsequently, we increased the population size to 100, and finally to 1000. We performed these increments to investigate whether the number of function evaluations was a bottleneck affecting the trade-off between solution quality and execution time. All tests were conducted on a server with 20 cores. We run the test on a server with 20 cores and 20 threads, then we did these evaluations using all the power with 40 threads.

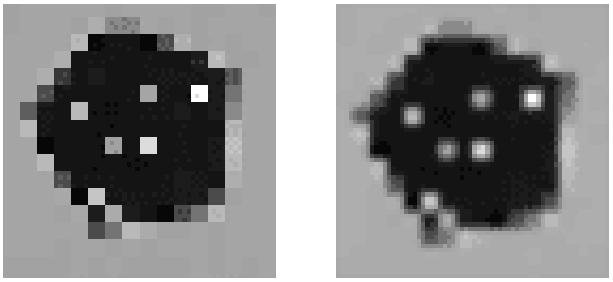


Figure 7: On the right is the noisy image solution of 30 Pixel Attack and on left pentagon shape with Gaussian filter .

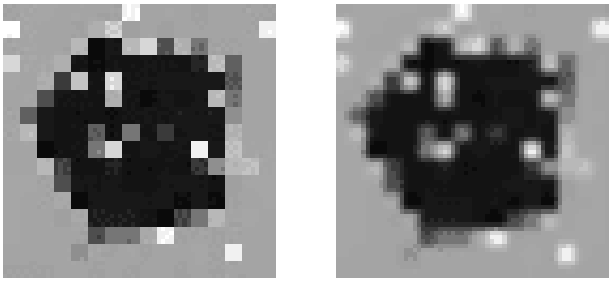


Figure 8: On the right is the noisy image solution of 40 Pixel Attack, and on the left is the pentagon shape with a Gaussian filter .

We found the solution using 100 population size, and for the 40-pixel attack, it took around 8 hours using one thread, evaluating entirely on the CPU, and 2 hours using 20 threads to find the solution for the 30-pixel attack with the same evaluation on the CPU.

Both solutions shown in Figures 7 and 8 change the original class from pentagon to cross. After we see that the solution change the original class to another one. We test the use of the Gaussian filter to clean the image, we obtain two results of the noise attack shown in Figures 7 and 8, and the filtered images are shown on the right of both Figures. The shapes are classified correctly after the application of the Gaussian filter. In this case, using the Gaussian filter on the input image, we can obtain the original class pentagon.

We repeated this experiment with different attack sizes using the same population size (400) as reported in the original article [2]. The results are shown in Table 5. For the single-pixel attack, we obtained an attack success rate of 0.54. After applying Gaussian filtering, this rate was reduced to 0.45. However, with a 20-pixel attack, Gaussian filtering proved ineffective at reducing the attack success rate. We attribute this behavior to the fact that 20 attacked pixels may exceed the filter's ability to recover the original signal.

3.5 One-pixel attack solved using simulated annealing

During testing of this attack, we evaluated different optimization algorithms as alternatives to differential evolution. We found that simulated annealing seek solutions faster than differential evolution. We got better performance using simulated annealing, from minutes per image to seconds per image. In the worst case, the maximum execution time was only 94 minutes.

The Figure 9 shows examples of adversarial images generated using differential evolution and simulated annealing. Both examples were attacked with 20 pixels. Given this significant performance difference, we expanded the test to 40 images with varying numbers of pixels and two different input sizes. The results are presented in Table 6.

Table 5: Results using one pixel attack with different number of pixels.

One pixel attack				
Num of Pixels	pop Size	Time Per Image	Success	Success after Gaussian filter
1	400	97 minutes	0.54	0.45
3	400	18 minutes	0.26	0.19
5	400	48 minutes	0.41	0.33
10	400	98 minutes	0.71	0.58
20	400	512 minutes	1.0	1.0

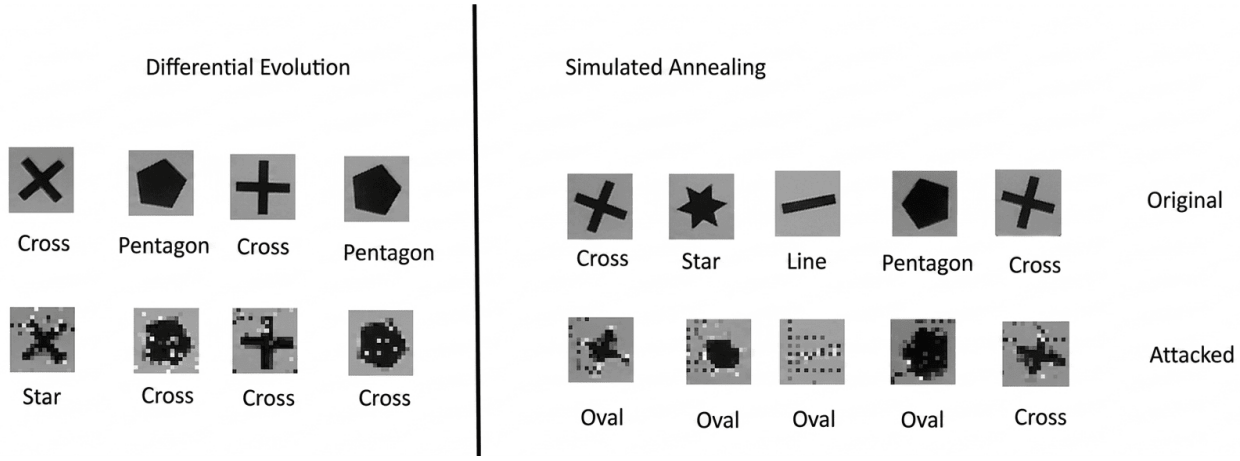


Figure 9: Examples of attacked images. On the left, the original implementation using differential evolution as an optimizer. On the right side are examples using simulated annealing as an optimizer.

Table 6: Results using one pixel attack with different number of pixels.

One pixel attack using simulated annealing as an optimizer.						
	64 × 64 rescaled to 16 × 16			32 × 32 rescaled to 16 × 16		
No.pixeles	Time	Attack Success	Gaussian filter	Time	Attack Success	Gaussian filter
20	94.04	0.70	**1.0**	84.74	0.84	**0.82**
10	46.19	0.63	**0.97**	41.63	0.76	**0.86**
5	22.95	0.43	**0.98**	20.64	0.67	**0.85**
1	4.52	0.08	**1.0**	4.09	0.21	**0.83**

4 Discussion

Perhaps our implementation needs to use a GPU to accelerate the evaluations, but the computational cost to search each solution is too high in comparison to solving using only a Gaussian filter to denoise the input images. In the case of one pixel attack noise, it is important to say that lot of details are loss when the original images are resized. The resizing algorithm is the nearest pixel value. We did not find any solution using a one pixel attack with images with the original size of 64×64 pixels.

Table 7 presents a comparison of our work with the original and the most recent work. Three key differences are that we test with a different input size, we use a dataset of well-formed shapes, and we employ a modern, well-established CNN architecture, EfficientNet.

Table 7: Comparison with related works

One pixel attack					
Work	Search method	Denoise Method	Better success rate	Better denoise rate	Image Size
Our work	Gaussian noise and one pixel attack (DE, SA)	Gaussian filter	0.54	0.86	32×32 and 64×64
[3]Most recent	GA,Random and DE	-	0.42	-	32×32
[2] First	DE	-	0.56	-	32×32

5 Conclusion

In conclusion, after we saw three methods to add noise in the images, the additive Gaussian noise, additive pink noise, and the one pixel attack, we found that the best way to denoise images is using simple filters like the Gaussian filter to enhance the classification model. One pixel attack is very effective, attacking architectures where the input size is too small, around 24×24 pixels, where the search field is small, but if the input size is too big it is necessary to resize the images to reduce the complexity of the search problem.

The Gaussian filter is a powerful and simple tool for denoising images when used as a preprocessing step with convolutional neural networks.

We observed that the usual attacks always use models with a reduced input image size. In this experiment, we test whether the problem is in the convolutional neural networks or in the images. That is why we tested a convolutional neural network that was trained with simple shapes in a reduced input size. Although the papers always test their attacks with original models with limited input sizes and resized inputs, we think the resize operation on the image produces information loss that helps enhance the attack. The problem is not the weakness of the convolutional neural networks; the problem is the use of reduced input size models trying to classify upscaled images. If the attack is added, it is probable that the model misclassifies. That was shown in the experiments, because we did not find any solution if we did not use a resized input image. The original implementations of the attacks used resized images and models with small input size images. In the same way, the problem increases where the search space is larger and the shapes are closer to the original size; the time to find a solution increases.

It is interesting that a simple Gaussian filter can be used as a preprocessing step to enhance accuracy. This suggests exploring the application of this filter to other types of noise. Additionally, it would be valuable to test these black-box attacks on more widely used CNN architectures, such as YOLO, and to investigate whether real-time black-box attacks are feasible.

For denoising images, trying alternative image preprocessing methods while keeping the processing lightweight for other kinds of noise would be valuable, because the methods explored in the references use complex methods to clean images.

References

[1] B. García-Morgado. "Análisis comparativo de algoritmos de clasificación de imágenes basados en aprendizaje automático tradicional y aprendizaje profundo". MA thesis. Cinvestav, 2024.

[2] J. Su, D.V. Vargas, and K. Sakurai. "One pixel attack for fooling deep neural networks". In: IEEE Transactions on Evolutionary C 23.5 (2019), pp. 828–841.

[3] Luana Clare, Alexandra Marques, and João Correia. "A Comparative Analysis of Evolutionary Adversarial One-Pixel Attacks". In: Applications of Evolutionary Computation. Ed. by Stephen Smith, João Correia, and Christian Cintrano. Cham: Springer Nature Switzerland, 2024, pp. 147–162. ISBN: 978-3-031-56855-8.

[4] M.W. Kusk and S. Lysdahlgaard. "The effect of Gaussian noise on pneumonia detection on chest radiographs, using convolutional neural networks". In: Radiography 29.1 (2023), pp. 38–43. DOI: 10.1016/j.radi.2022.09.011. URL: https://doi.org/10.1016/j.radi.2022.09.011.

[5] Teresa Kwamboka Abuya, Richard Maina Rimiru, and George Onyango Okeyo. "An Image Denoising Technique Using Wavelet-Anisotropic Gaussian Filter-Based Denoising Convolutional Neural Network for CT Images". In: Applied Sciences 13.21 (2023). ISSN: 2076-3417. DOI: 10.3390/app132112069. URL: https://www.mdpi.com/2076-3417/13/21/12069.

[6] Y. Bi, B. Xue, and M. Zhang. "Genetic Programming-Based Evolutionary Deep Learning for Data-Efficient Image Classification". In: IEEE Transactions on Evolutionary Computation 28.2 (2024), pp. 307–322. DOI: 10.1109/TEVC.2022.3214503.

[7] Y. Bi, M. Zhang, and B. Xue. "Genetic Programming for Automatic Global and Local Feature Extraction to Image Classification". In: 2018 IEEE Congress on Evolutionary Computation (CEC). 2018, pp. 1–8. DOI: 10.1109/CEC.2018.8477911.

[8] G. Olague. Evolutionary Computer Vision. Springer Science+Business Media, Jan. 2016. DOI: 10.1007/978-3-662-43693-6. (Visited on 11/12/2024).

[9] 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 (CVPR). June 2016, pp. 2574–2582.