# Dependable AI - Minor 1

# Report

**Mukul Shingwani**

**B20AI023**

---

**Research paper :** Towards Large yet Imperceptible Adversarial Image Perturbations with Perceptual Color Distance

**Github repo :** https://github.com/ZhengyuZhao/PerC-Adversarial

---

## Abstract

This paper evaluates image perturbations designed to fool image classifiers based on adversarial effect and visual imperceptibility. The conventional assumption is that perturbations should have tight Lp-norm bounds in RGB space, but this work proposes a **new approach** that **minimizes perturbation size with respect to perceptual color distance.** In this work there are **two approaches**, Perceptual Color distance C&W **(PerC-C&W)**, which extends the widely-used C&W approach and produces larger RGB perturbations while maintaining adversarial strength and contributing to imperceptibility and second, Perceptual Color distance Alternating Loss **(PerC-AL)** achieves the same outcome, but does so more efficiently by alternating between the classification loss and perceptual color difference when updating perturbations, are introduced that outperform conventional Lp approaches in terms of robustness and transferability, and add value to existing structure-based methods for creating image perturbations.

It has two kinds of attack modes: "targeted" or "non-targeted."

- If the attack is targeted, the code block optimizes the generated input to make the other class (i.e., the class that is not the target class) the most likely classification. The logit_dists tensor contains the difference between the logit (i.e., pre-softmax) outputs of the other class and the target class, and the torch.clamp function sets all negative values in the tensor to 0.
- If the attack is non-targeted, the code block optimizes the generated input to make the target class the least likely classification. The logit_dists tensor contains the difference between the logit outputs of the target class and the other classes, and the torch.clamp function again sets all negative values in the tensor to 0.

This figure motivates the use of perceptual color distance for creating adversarial images. Here, we have taken a solid color image (left) and added the same perturbations to the green channel (middle) and to the blue channel (right). Although both RGB channels were perturbed identically, **the perturbations**

**are only visible in the green channel.** The rea- son is that color as it is perceived by the human eye does not change uniformly over distance in RGB space. Relatively small perturbations in RGB space may correspond to large differences in perceptual color space. Conversely, relatively large changes in RGB space may remain unnoticeable if they lead to small perceived color differences. *(taken from paper)*

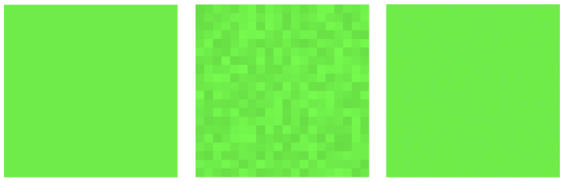


Figure 2: Left: Original image (a 20 × 20 8-bit RGB image patch with color (15,240,15)). Middle: Image perturbed by adding noise in the G channel, sampled from a uniform distribution in the range [-15,15]. Right: Image perturbed by adding the identical noise, but in the B channel. The B-channel perturbations are imperceptible (best viewed on screen).

**Dataset used:** 1000 images from Imagenet dataset.
**Image dim.** = 3x299x299

## Explanation of code blocks (understanding of code)

- Diff color functions:
    - This code block is a collection of functions used to **convert RGB (Red Green Blue) color space to LAB** (Lightness, a* and b*) color space. The LAB color space is a three-axis system (a, b and L) which is designed to approximate human vision. This conversion is required because the **LAB color space is more perceptually uniform than RGB color space,** the rgb2lab_diff() function is the main function which converts RGB color space to LAB color space
    - The hpf_diff(), dhpf_diff(), and ahpf_diff() functions are used to calculate differences between two colors in the LAB color space. These functions are used to compute the **CIEDE2000 metric.**
- PerC_CW:
    - a **method for creating adversarial images that introduces perceptual color distance into the joint optimization of C&W.**
    - This code block is **implementing** an attack method named **PerC_CW** (perceptual Color and White-box attack based on the C&W method). This is an adversarial attack on deep neural networks that tries to generate adversarial examples by **exploiting perceptual color differences between original and adversarial examples.**

- The PerC_CW method **modifies the Carlini & Wagner (C&W)** attack, which is a state-of-the-art white-box attack that aims to generate adversarial examples for any deep neural network.
- It defines a class named PerC_CW, which contains the attack method, including the constructor and a method named adversary.
- The constructor has several parameters, including the bounds of the images, number of classes, confidence of the adversary (The "**confidence**" variable controls the magnitude of the perturbation applied to the input.) , learning rate, search steps, maximum iterations, initial constant of the adversary, and device to use for the adversary.
- The adversary method takes a PyTorch model, a batch of image examples, and their labels (Original labels if untargeted, else labels of targets.) and returns a batch of image samples modified to be adversarial.
- The attack method tries to find a new input that minimizes the difference between the original and adversarial examples while maximizing the difference between the logits (output of the neural network) of the original and adversarial examples. **It uses a binary search algorithm to find the optimal scale constant and a gradient descent algorithm to find the adversarial example.** The method also considers perceptual color differences between original and adversarial examples by adding a penalty term to the loss function.
- PerC-C&W, adopts the joint optimization of the well-known C&W, but replaces the original penalty on the L2 norm with a new one based on perceptual color difference. It can be formally expressed as:

$$\underset{\boldsymbol{w}}{\text{minimize}} \ \ \|\Delta E_{00}(\boldsymbol{x}, \boldsymbol{x}')\|_2 + \lambda f(\boldsymbol{x}'),$$

- PerC_AL:
  - **an efficient method that optimizes alternating loss (AL) functions, switching between classification loss and perceptual color difference**
  - This code **implements the PerC_AL algorithm**, which is an iterative optimization technique to generate adversarial examples. PerC_AL stands for "**Alternating Loss of Classification and Color Differences to achieve imperceptible perturbations with few iterations".** The algorithm uses two different types of loss functions: the classification loss and the perceptual color difference loss.

- The PerC_AL class is initialized with hyperparameters such as the maximum number of iterations, the learning rate for updating perturbations with respect to classification loss, the learning rate for updating perturbations with respect to perceptual color differences, the confidence of the adversary for **Carlini's loss**, and the device on which to perform the adversary.
- The "adversary" method performs the adversary of the given PyTorch model on the input examples. The method receives the model, inputs, and labels (Original labels if untargeted, else labels of targets) as arguments, where inputs and labels represent batches of image examples and their corresponding labels. **The "targeted" parameter indicates whether to perform a targeted adversary or not.**
- The method then initializes some variables and performs the optimization for the maximum number of iterations. In each iteration, the method **calculates the gradient** of the **classification loss** and the **perceptual color difference** loss with respect to the perturbation, and updates the perturbation accordingly. The **cosine annealing technique** is used to decrease the learning rate for both types of losses as the number of iterations increases. The method also **applies quantization** to the perturbed image and **clips it to the range of [0, 1].**
- If the "targeted" parameter is False and the "confidence" hyperparameter is not 0, then the method sets the confidence of the adversary for Carlini's loss. If the "targeted" parameter is True and the "confidence" hyperparameter is not 0, then the method returns an error message, as this approach only supports the confidence setting in an untargeted case.

**Algorithm 1** Perceptual Color Distance Alternating Loss (PerC-AL)

**Input:**
$\boldsymbol{x}$: original image, $t$: target label, $K$: number of iterations
$\alpha_l$: step size in minimizing classification loss
$\alpha_c$: step size in minimizing perceptual color difference

**Output:** $\boldsymbol{x}'$: adversarial image

Initialize $\boldsymbol{x}'_0 \leftarrow \boldsymbol{x}$, $\boldsymbol{\delta}_0 \leftarrow \mathbf{0}$
**for** $k \leftarrow 1$ to $K$ **do**
  **if** $\boldsymbol{x}'_{k-1}$ is not adversarial **then**
    $\boldsymbol{g} \leftarrow -\nabla_{\boldsymbol{x}} J(\boldsymbol{x}'_{k-1}, t)$
    $\boldsymbol{g} \leftarrow \alpha_l \cdot \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|_2}$
    $\boldsymbol{\delta}_k \leftarrow \boldsymbol{\delta}_{k-1} + \boldsymbol{g}$ ▷ Update $\boldsymbol{\delta}$ in the direction of $\boldsymbol{g}$
  **else**
    $C_2 \leftarrow -\|\Delta E_{00}(\boldsymbol{x}, \boldsymbol{x}'_{k-1})\|_2$
    $\boldsymbol{g}_c \leftarrow \nabla_{\boldsymbol{x}} C_2$
    $\boldsymbol{g}_c \leftarrow \alpha_c \cdot \frac{\boldsymbol{g}_c}{\|\boldsymbol{g}_c\|_2}$
    $\boldsymbol{\delta}_k \leftarrow \boldsymbol{\delta}_{k-1} + \boldsymbol{g}_c$ ▷ Update $\boldsymbol{\delta}$ in the direction of $\boldsymbol{g}_c$
  **end if**
  $\boldsymbol{x}'_k \leftarrow \text{clip}(\boldsymbol{x} + \boldsymbol{\delta}_k, 0, 1)$
  $\boldsymbol{x}'_k \leftarrow \text{quantize}(\boldsymbol{x}'_k)$ ▷ Ensure $\boldsymbol{x}'_k$ is valid
**end for**
**return** $\boldsymbol{x}' \leftarrow \boldsymbol{x}'_k$ that is adversarial and has smallest $C_2$

- Main block for PerC_AL and PerC_CW:
    - The **code first sets a mode (untargeted or targeted)**, batch size, and initializes arrays to store the color differences, L2 norm, and L_inf norm for each adversarial image generated. Then, it loads input images in batches, normalizes the images, and converts them to the LAB color space using the rgb2lab_diff function.
    - The code then initializes the PerC_AL or the PerC_CW object as the adversary with its parameters, including the maximum number of iterations, alpha_l_init, alpha_c_init, and confidence. The adversary is then applied to the input images to **generate adversarial images** using the adversary function and by applying the **inception_v3 pretrained model.**
    - The code then **computes** the **perceptual color differences, L2 norm, and L_inf norm** between the original and adversarial images using the ciede2000_diff function, and stores these values in the previously initialized arrays. **Finally, the adversarial images are saved to a specified output directory.**
    - Both the targeted and untargeted attacks mode were used to test the attacks
    - Given below is the table showing related information as described above

| **Approach** | **Budget** | **Perturbation size** | | |
|---|---|---|---|---|
| | | **L2** | **L_inf** | **Perceptual color diff.** |
| PerC_CW | 9x1000 | 0.91 | 11.76 | 56.37 |
| PerC_AL | 1000 | 1.41 | 18.09 | 64.87 |

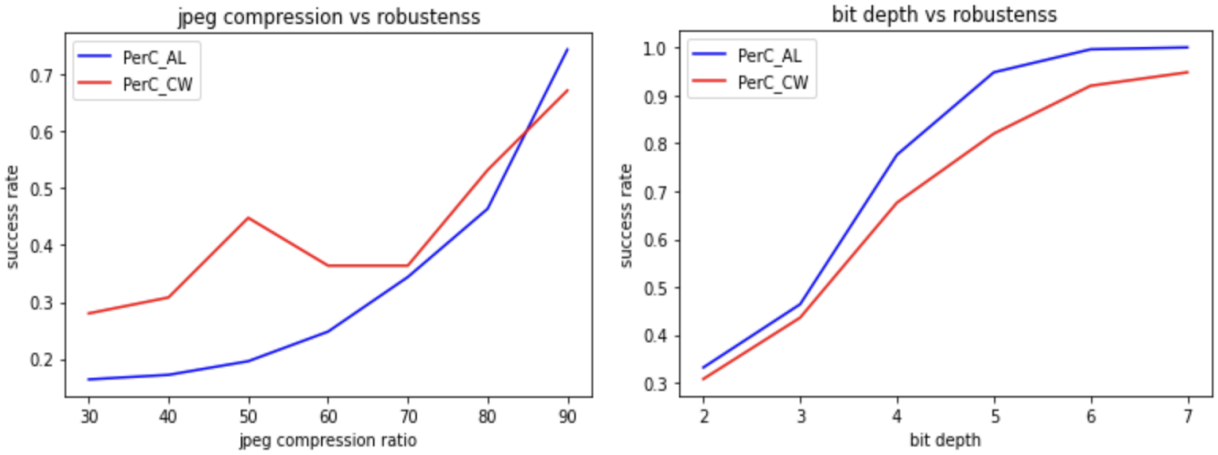
- Evaluation on Adversarial strength (success rate):
    - This **code is used to evaluate the success rate of the adversarial attacks**. The evaluation is done batch-wise.
    - The number of batches is calculated using the length of the image_id_list and batch_size.
    - For each element in the current batch, the original image (X_ori) is loaded and preprocessed using the trn function. Similarly, the adversarial image (X_adv) is loaded and preprocessed. The adversarial label (label_adv) is predicted using the preprocessed adversarial image.
    - If the **attack is untargeted**, the **original label (label_ori) is predicted** and compared with the **adversarial label.**

- If the **attack is targeted,** the target labels are extracted from the label_tar_list and compared with the adversarial label. In both cases, the number of successful adversarial attacks (cnt) is incremented.
- Finally, the success rate is calculated as the ratio of successful adversarial attacks to the total number of images in the dataset. Given below is the table

| Attack Technique | Mode | Success Rate |
|---|---|---|
| PerC_AL | targeted | 100% |
| PerC_AL | untargeted | 100% |
| PerC_CW | targeted | 100% |
| PerC_CW | untargeted | 100% |

- Evaluation based on robustness:
  - This is used for **evaluating the robustness** of a machine learning model against image perturbations. The code evaluates **how much the accuracy of the model is affected by JPEG compression** of different quality levels **and also based on bit depth.** Here is how it works:
  - **bit_depth_red** is a function that takes an image tensor X_before and a depth parameter, and **reduces the bit depth of the image to the given depth level by quantizing the pixel values**.
  - **JPEG_compression** is a function that takes an image tensor X_before and a quality parameter, **saves the image in JPEG format with the given quality level**, and then loads the image back into a tensor X_after.
  - The code defines a list of levels for JPEG compression. For each level, it iterates over the input images, loads them into tensors, and evaluates the model accuracy before and after applying the JPEG compression at the current level.
  - num is a tensor that keeps track of the number of misclassifications for each level of compression. It is initialized to all zeros, and then the number of misclassifications for each level is added up.

- The result of the evaluation is the num tensor divided by the total number of input images. **It gives the proportion of misclassifications for each level of compression.**
- Given below are the **comparative plots to see the performance of two attacks** based on two parameters : jpeg compression and bit depth




On higher values of quality (i.e. **lower compression ratios**), **success rate should be low** because the images have been compressed with a high quality that preserves more information from the original image, and therefore, the model is more likely to correctly classify the image.

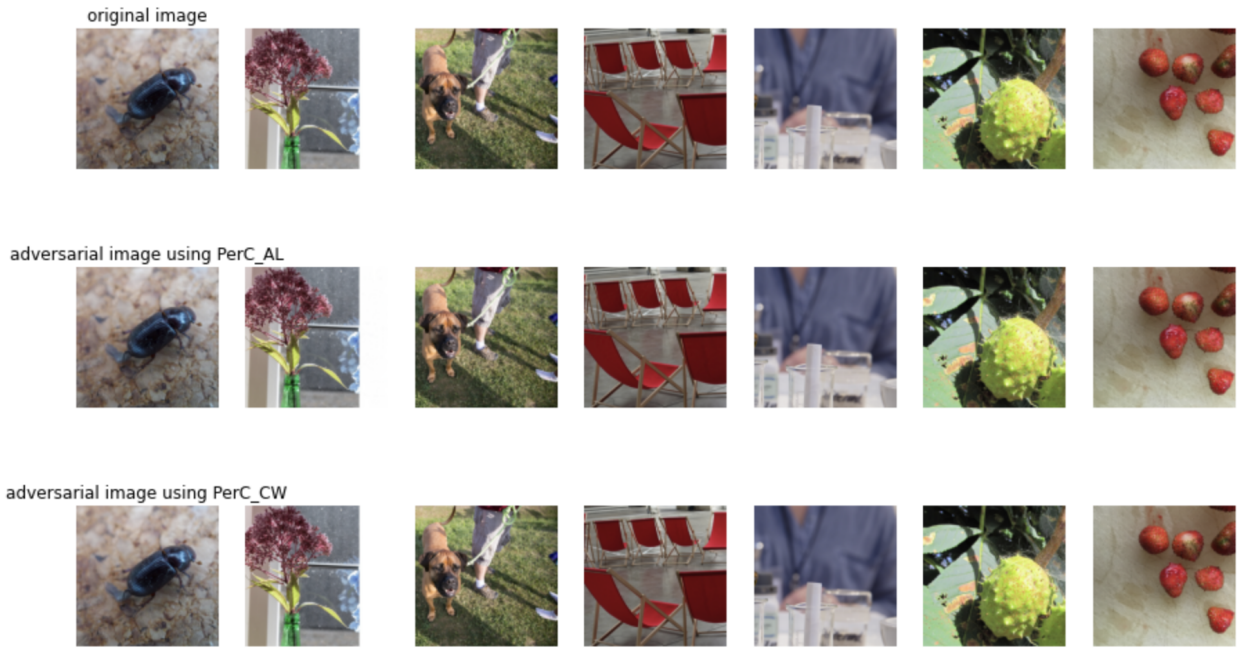
On the other hand, on lower values of quality (i.e. **higher compression ratios), success rate should be high** because the images have been compressed with a low quality that removes more information from the original image, and therefore, the model is more likely to misclassify the image.

- Evaluation on transferability:
  - The code loads three pre-trained models from PyTorch's model zoo, namely **GoogleNet, VGG16, and ResNet152**, and sets their `requires_grad` attribute to `False`. This attribute tells PyTorch to **not update the weights of the model during the training process**. Then, the code moves these models to the `cuda` device, which is used for GPU computation.
  - This code generates a list of eligible images for transferability. Then, for each batch, the code loads the images and converts them into PyTorch tensors. The pre-trained models are then used to predict the classes of the images. **A mask is generated by multiplying the predictions of each model for each image in the batch.** If all models predict the same class, the mask value is 1; otherwise, it is 0. This mask is appended to the eligible_list2.

- All three models are set to be in evaluation mode using the eval() method.
- The for loops that follow set the requires_grad property of all parameters of each model to False. This ensures that the parameters are not updated during training.
- Then, the code initializes the batch_size and the num_batches to compute the transferability of the images.
- It creates an empty list eligible_list2 to store the eligible images for transferability.
- The first for loop iterates over all the batches of images and creates tensors for each image. For each tensor, it performs four forward passes using the models model, model2, model3, and model4.
- It then applies a mask to the predicted labels of the four models to determine if an image is eligible for transferability.
- The eligible_list2 appends the computed masks.
- The next block of code iterates over all the batches of eligible images to calculate their transferability.
- It initializes X_ori and x_adv tensors and creates tensors for each image in a batch.
- For each tensor, it performs forward passes using the four models model, model2, model3, and model4.It then computes the success rates of the four models for the adversarial images.
- The code computes the success rate and prints the result, **given below is the transferability rate obtained.**

| Approach | Inception_v3 | GoogleNet | VGG16 | ResNet152 |
|---|---|---|---|---|
| PerC_CW | 100% | 5.2% | 3.45% | 1.72% |
| PerC_AL | 100% | 5.17% | 10.34% | 5.16% |

Following are the **image comparison of original vs the attacked images**, as can be seen they are visually imperceptible.




---

**New Dataset:** Visual Genome dataset (download here)

---

It contains more than 108,000 images with object annotations, object attributes, and object relationships with images generally being resized to 299x299 pixels along with 1000 classes for classification.

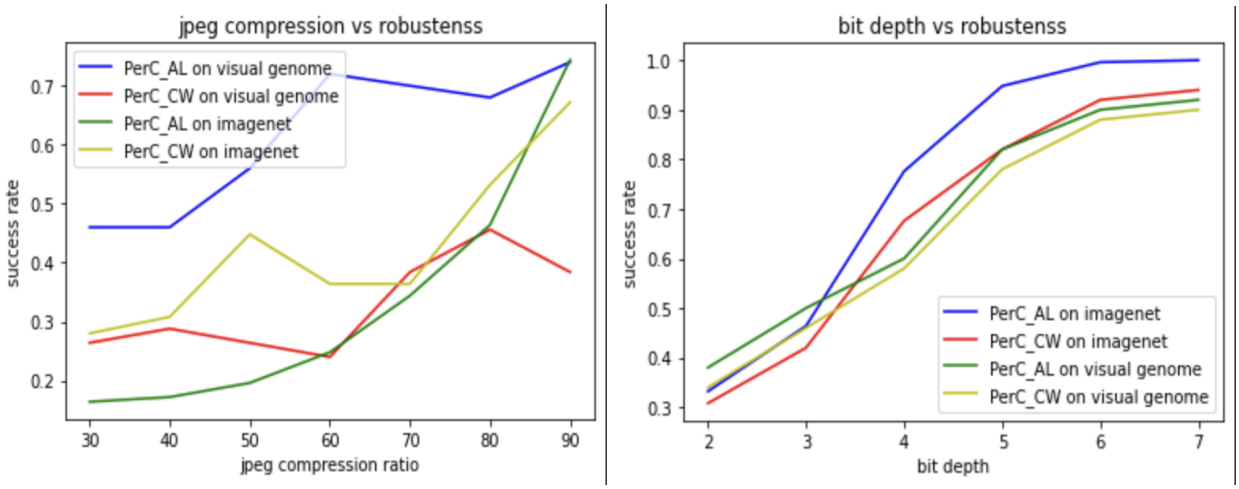
- the **perceptual color differences, L2 norm, and L_inf norm** between the original and adversarial images using the ciede2000_diff function were as follows on this dataset

| **Approach** | **Budget** | **Perturbation size** | | |
|---|---|---|---|---|
| | | **L2** | **L_inf** | **Perceptual color diff.** |
| PerC_CW | 9x1000 | 0.87 | 7.84 | 27.29 |
| PerC_AL | 1000 | 1.54 | 16.43 | 83.29 |

- Now, when the PerC_CW and PerC_AL attacks were used on this following was the success rate (**evaluation of adversarial strength**)

| Attack Technique | Mode | Success Rate |
|---|---|---|
| PerC_AL | untargeted | 70% |
| PerC_CW | untargeted | 80% |

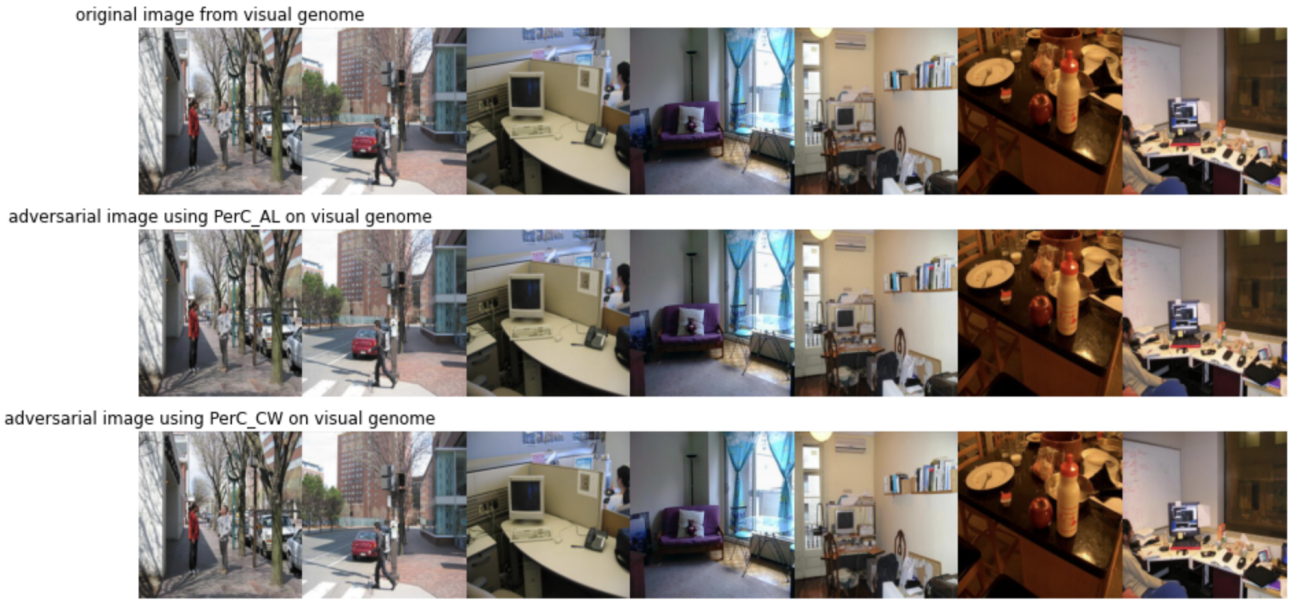
- These are the plots for **comparing the robustness** of a machine learning model against image perturbations involving jpeg compression and bit depth.




- **Transferability rate** obtained on visual genome dataset

| Approach | Inception_v3 | GoogleNet | VGG16 | ResNet152 |
|---|---|---|---|---|
| PerC_CW | 21.05% | 42.11% | 36.84% | 31.58% |
| PerC_AL | 73.68% | 47.37% | 36.84% | 31.58% |

- Following are the **original and adversarial images for the visual genome dataset**, as can be seen they are visually imperceptible




## Conclusion

This research paper illustrated how perceptual color distance can be used to generate substantial yet imperceptible adversarial image changes. PerC-C&W and PerC-AL are two ways for creating adversarial images that we have presented. The experimental testing of these approaches demonstrates that perceptual color distance can increase imperceptibility, particularly in smooth, saturated regions. We demonstrate that both approaches produce perturbations with greater RGB Lp norms than approaches that directly perturb in RGB space. This impact translates into adversarial strength, or the perturbations' capacity to deceive a classifier.

Their effects were recreated and tested on the 1000 imagenet samples as used in the original research paper along with that on few images from the visual genome dataset, although the success rate was a bit low on this new dataset, but that can be improved further by fine tuning, since we were only using the pre-trained models as of now due to limitations on time and resources.

---