

BACHELOR THESIS  
COMPUTING SCIENCE



RADBOUD UNIVERSITY

---

**Increasing the Perceptual Image  
Quality of Adversarial Queries for  
Content-based Image Retrieval**

---

*Author:*

Sam Sweere  
s4403142

*First supervisor/assessor:*

Prof. Martha Larson  
[m.larson@cs.ru.nl](mailto:m.larson@cs.ru.nl)

*Second supervisor:*

MSc. Zhuoran Liu  
[z.liu@cs.ru.nl](mailto:z.liu@cs.ru.nl)

*Second assessor:*

Prof. Lejla Batina  
[lejla@cs.ru.nl](mailto:lejla@cs.ru.nl)

July 10, 2019

## **Abstract**

Adversarial queries are images that have been altered such that a content based image retrieval (CBIR) has trouble retrieving similar images while humans can still clearly perceive the contents of the adversarial query (the altered image). This thesis presents a new method to create adversarial queries for CBIR systems named CV-PIRE. This method takes human color perception into account, when altering the image, by reducing perturbations in low color variance areas and increasing perturbations in high color variance areas. CV-PIRE outperforms previous methods based on the perceptual difference between the original image en the adversarial query based on the SSIM score.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	6
1.1.1	Safety threats . . . . .	6
1.1.2	Privacy threats . . . . .	6
1.2	Roadmap . . . . .	7
<b>2</b>	<b>Preliminaries</b>	<b>8</b>
2.1	Artificial neural networks . . . . .	8
2.1.1	Single layer ANN . . . . .	8
2.1.2	Multi layer neural network . . . . .	11
2.1.3	Adaptive learning rate . . . . .	12
2.1.4	Kernels . . . . .	13
2.1.5	Convolutional neural networks . . . . .	14
2.2	Content based image retrieval . . . . .	15
2.2.1	General overview . . . . .	15
2.2.2	Local feature based CBIR . . . . .	18
2.2.3	CNN based CBIR . . . . .	19
2.3	Creating adversarial examples . . . . .	20
2.3.1	Intuition . . . . .	20
2.3.2	Adversarial examples for CBIR systems . . . . .	22
2.4	Recent work . . . . .	22
2.5	Image processing techniques and color perception . . . . .	25
2.5.1	Color spaces . . . . .	25
2.5.2	Human color perception . . . . .	26
2.5.3	Gaussian kernel . . . . .	27
<b>3</b>	<b>Approach</b>	<b>28</b>
3.1	Intuitive insight . . . . .	29
3.2	Pixel color variance . . . . .	30
3.2.1	Gaussian kernel . . . . .	30
3.2.2	Algorithm to calculate the pixel color variance . . . . .	31
3.2.3	Pixel color variance image . . . . .	32
3.3	Generating adversarial examples . . . . .	33

3.3.1	Saving adversarial queries . . . . .	35
3.3.2	Determining the hyperparameters . . . . .	35
3.4	Other algorithms for comparison . . . . .	36
3.4.1	SD-PIRE . . . . .	36
<b>4</b>	<b>Experimental results</b>	<b>38</b>
4.1	Evaluating adversarial queries . . . . .	38
4.1.1	Measuring the performance of CBIR . . . . .	38
4.1.2	Determining perceived image quality . . . . .	39
4.1.3	Data . . . . .	39
4.2	Determining the hyperparameters of CV-PIRE . . . . .	41
4.2.1	Pixel color variance hyperparameters . . . . .	41
4.2.2	Iterations . . . . .	42
4.2.3	Threshold multiplier . . . . .	42
4.3	Results . . . . .	44
4.4	Detailed comparison . . . . .	46
4.5	Comparison between CV-PIRE and SD-PIRE . . . . .	47
4.5.1	Visual differences . . . . .	51
4.6	Convergence rate . . . . .	52
<b>5</b>	<b>Conclusion</b>	<b>53</b>
5.1	Conclusion . . . . .	53
5.2	Discussion and outlook . . . . .	53
5.2.1	Robustness . . . . .	53
5.2.2	Different datasets . . . . .	53
5.2.3	Multiple CBIR systems . . . . .	54
5.2.4	SSIM and human color perception . . . . .	54
5.2.5	CIE94 instead of CIEDE2000 . . . . .	54
5.2.6	Speeding up the calculations . . . . .	54
<b>6</b>	<b>Acknowledgements</b>	<b>55</b>
<b>A</b>	<b>Appendix</b>	<b>59</b>
A.1	SSIM formula . . . . .	59
A.2	mAP compared to iterations for CV-PIRE . . . . .	60
A.3	More detailed results . . . . .	61
A.4	PIRE with a higher learning rate . . . . .	62

# Chapter 1

## Introduction

One of the biggest tech developments in the last decade is machine learning. A majority of this development is constituted by neural network frameworks and a part of this are convolutional neural networks (CNN). One of the tasks they perform exceptionally well is image classification. While these CNNs are inspired by the way the human brain processes images, they, however, seem to not achieve this task in a similar manner. In the article [1] it is argued that classifiers based on modern machine learning techniques are not learning the underlying properties that define what they are trying to classify. Instead it seems like these algorithms have found a way to be successful at classification without using the underlying properties that determine the correct output label. For example, a CNN classifier can be trained to classify trucks. Such a CNN can come to the conclusion that something can only be a truck if its red, since it has only seen trucks that were colored red. The moment it is shown an image of a green truck it will not classify it as a truck. While for humans color is not the defining property of a truck and they will, therefore, still clearly see a truck. This difference in perception makes these algorithms prone to so called “adversarial attacks”. By changing pixels in the image that are essential for these CNNs to perform their classification, but do not substantially change the image for humans, one can fool such an algorithm to classify the image as something else, while for a human the content of the image has not changed substantially. The paper [2] demonstrates this property, by shifting the colors of an image and in turn getting different output classifications, as is shown in figure 1.1.



Figure 1.1: Demonstration image from [2]. A low resolution sample image of a truck from the CIFAR10 dataset, on the left side the original image, on the right three color-shifted versions of the same image. All classified differently by the VGG16 [3] network (a CNN classifier) trained on the CIFAR10 dataset.

One field in computing science where machine learning has become an important factor is in content-based image retrieval (CBIR). Given a query image (an input image) a CBIR system can find images that have visually similar contents. In contrast to a CNN classifier, it is not mandatory for a CBIR system to have seen the contents of the query image before, since it does not classify the image, but instead finds images with visually similar content. This property could enable social media platforms to track users based on their uploaded images. These CBIR systems could retrieve similar images, using information connected to these retrieved images to pinpoint the location where the uploaded photo was taken, potentially violating the users privacy.

To protect the privacy of a user, he/she could change the image into an adversarial query, as demonstrated in [4]. This means altering the image in such a way that humans still perceive it clearly, but when presented to a CBIR system it returns the wrong results as it has difficulties retrieving the content. This idea is illustrated in figure 1.2. One way to create an adversarial query is by perturbing (altering) pixels in an image in such a way that a CBIR system performs really poorly [4]. However, the perturbation of pixels can be quite noticeable to the human eye and results in the image looking “noisy”. It is therefore interesting to see if there is a way to do these perturbations in such a way that they become less noticeable to the human eye, while still creating a good adversarial query for CBIR.

Query	Top Ranked Results		
Original Image Query  			
Adversarial Image Query  			

Figure 1.2: An image query (top left) and the corresponding adversarial query image (bottom left). On the right are the top 3 retrieved images by a CBIR system. For the original query image the CBIR correctly retrieved the top 3 images, as shown in green. The adversarial query image looks almost identical to the human eye but the CBIR incorrectly retrieved the top 3 images, as shown in red.

In this paper we research this problem and propose a solution. In total making the following contributions:

- We motivate why it is important to research adversarial attacks on CBIR systems.
- We propose a new method to determine the noticeability of a pixel perturbation within an image based on the color variance around a pixel.
- We propose a new method, named CV-PIRE, that is able to generate adversarial queries for CBIR systems while maintaining good perceptual image quality.

## 1.1 Motivation

### 1.1.1 Safety threats

In the paper “Robust Physical-World Attacks on Deep Learning Visual Classification” [5] it is shown that it is possible that physically altering road signs can generate adversarial queries for classification algorithms. Self driving cars using such a classification algorithm miss-classified the altered “*Stop*” sign to a “*Speed Limit 80*” sign and would therefore not stop. While for humans the altered stop sign was still clearly perceived as a stop sign. While in this research a classifier was used, it is imaginable that self driving cars will use some CBIR algorithms. For developers or researchers building these CBIR based implementations for self driving cars or other high risk applications, it would be crucial to know how vulnerable their implementation is to adversarial attacks in order to keep their system safe.

### 1.1.2 Privacy threats

One possible application of CBIR systems, as mentioned before, is to find and track people who have posted pictures online. When someone uploads a photo on for example a social media platform, he usually has the intention to share this with friends. Most likely it is not their intent to make himself vulnerable to being monitored or tracked by automated systems. When the uploaded photo contains a clear description or meta-data such as geolocation coordinates, it is relatively easy for the social-media platform to track this person with this data. The paper [6] demonstrated how they were able to retrieve private addresses of celebrities using photos from social media platforms including YouTube and Twitter. However, a person can choose not to include geolocation meta-data or clear descriptions. In this case the social media platform can make use of a CBIR system. Running the person’s photo through a CBIR system will result in photos that have visually similar content. If the content of these similar photos is known, for example through description or geolocation meta-data, the social media platform could assume that the photo was taken in the vicinity of the same location. This information can consequently be used to track and monitor the user, endangering his privacy. In [7] it is demonstrated how geolocation of videos shared on Flickr can be retrieved by extracting visual features from frames of the video and other metadata. The extraction of visual features in this research is comparable to how a CBIR system extracts information from an image.

If instead of uploading the original photo the user removes the geolocation metadata and generates an adversarial query of his photo, the performance of such a CBIR system in finding images with visually similar content could decrease to an unusable level, rendering it useless for tracking or monitoring purposes and thereby securing the user’s privacy.

## 1.2 Roadmap

This thesis is structured as follows:

- We discuss preliminary information on how content based image retrieval works and how to generate adversarial queries for these systems. As well as discussing image processing approaches, color spaces and human color perception (Chapter 2).
- We explain the intuition behind our proposed method CV-PIRE. Where pixels perturbed in low color variance regions seem to be more perceivable than perturbations in high color variance regions (Section 3.1).
- We propose a new method to determine the noticeability of a pixel perturbation within an image based on color variance (Section 3.2).
- We propose a new method that is able to generate adversarial queries for CBIR systems while maintaining good perceptual image quality (Section 3.3).
- We discuss ways to analyse the performance of adversarial queries (Section 4.1).
- We discuss and analyse different practical aspects of generating adversarial queries using our proposed method CV-PIRE (Section 4.2 and 4.6).
- We evaluate and compare the performance of adversarial queries generated by CV-PIRE compared to related methods (Section 4.3, 4.4 and 4.5).
- We conclude this thesis and give an outlook to further research (Chapter 5).

# Chapter 2

## Preliminaries

In this chapter some subjects that are required to understand the research in this thesis are discussed. First, a short overview of artificial neural networks including convolutional neural networks is given in section 2.1. Next, the working of content based image retrieval is discussed in section 2.2. In section 2.3 the creation of adversarial queries on neural network based systems is discussed. In section 2.4 recent work on content based image retrieval and adversarial queries that are relevant for this research are discussed. Finally, in section 2.5 image processing techniques and the perception of (digital) colors are discussed.

### 2.1 Artificial neural networks

This thesis is not about artificial neural networks (ANN), but systems that make use of them. Therefore, it is necessary that we explain the basics of ANN's first. A full overview on how artificial neural networks (ANN) work is beyond the scope of this thesis, we will therefore give a short overview of each element crucial for making ANN's work. If you are new to neural networks the book *An Introduction to Neural Networks* by Kevin Gurney [8] discusses almost every part of this section in detail.

#### 2.1.1 Single layer ANN

A fully connected single layer ANN (also known as a perceptron) consists of input nodes and output nodes that are fully connected, with every connection having its own weight  $w$  as illustrated in figure 2.1. These networks operate as a feed-forward network where the outcome  $y$  is generated by multiplying the weights  $w$  with the corresponding input  $x$  and applying an activation function  $g$ , thus  $y = g(w^T x)$ .

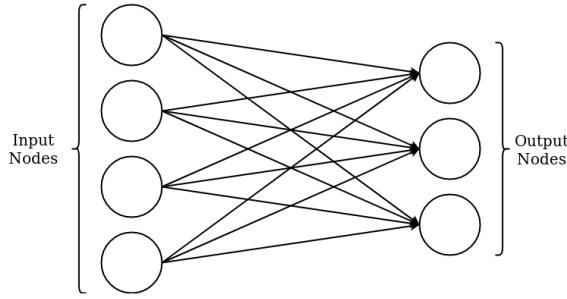


Figure 2.1: Illustration of a single layer ANN with 4 input nodes and 3 output nodes.

### Training goal

We want to find the optimal parameters  $\theta$  such that the network gives the desired output as often as possible, where  $\theta$  are modifiable parameters such as the weights  $w$  or the input  $x$ . To do this we need to determine a function that describes how good the output  $y$  of the network is compared to the known correct output  $t$ . We call this function the loss function  $l$ . With this loss function we can determine a cost function  $J$  that describes how well the network performs on multiple inputs  $x$ . The goal to train the network is done by finding the parameters  $\theta$  such that the cost function  $J$  is minimized. Or in other words the goal of learning is to find the optimal parameters  $\theta^*$  such that:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} J(\theta) \quad (2.1)$$

### Cost function

Before we can determine a cost function  $J$ , we need to determine a loss function  $l$ . One of the most common loss functions is the Euclidean loss given by:

$$l(y, t) = \frac{1}{2} \|y - t\|_2^2 \quad (2.2)$$

The resulting cost function, also called the mean squared error (MSE), is defined by:

$$J(\theta) = \frac{1}{2N} \sum_{n=1}^N l(y_n, t_n) \quad (2.3)$$

With  $N$  the number of presented inputs,  $y_n$  the prediction (output) of input  $n$  and  $t_n$  the known target output of input  $n$ .

## Gradient descent

Now that we know what our learning goal (equation 2.1) is given a cost function  $J$  we need a way to update the parameters  $\theta$  such that the learning goal is achieved. We want to find the minimum of the cost function, the way we do this is by looking at the slope of the cost function  $J$  in respect to the parameters  $\theta$ . By following the slope downwards in small steps we could eventually reach a (local) minimum. This procedure is called gradient descent. The gradients of the cost functions are found by:

$$\nabla J = \left( \frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_k} \right)^T \quad (2.4)$$

Where  $k$  is the number of parameters the network contains.

To apply gradient descent we need to calculate  $\nabla J$  and therefore  $\frac{\partial J}{\partial \theta_i}$ . To demonstrate the application we use a single layer fully connected ANN with an arbitrary activation function  $g$ , weights  $w$  and inputs  $x$ . Where the goal is to train the network, thus to update the weights  $w$  (thus  $\theta = w$ ). The outcome of such network is given by:

$$y = g(w^T x) \quad (2.5)$$

Choosing the MSE cost function and using the sum and product rules we get:

$$\frac{\partial J}{\partial w_i} = \frac{1}{N} \sum_n \frac{\partial l}{\partial w_i} \quad (2.6)$$

Choosing the Euclidean loss function and using the chain rule this becomes:

$$\frac{\partial J}{\partial \theta_i} = \frac{1}{N} \sum_{n=1}^N (y_n - t_n) g' (w^T x_n) x_{i,n} \quad (2.7)$$

Where  $g$  is the activation function,  $y_n$  the outcome of the network for input  $x_n$ ,  $t_n$  is the known desired outcome and  $N$  is the total amount of inputs. Using this we can update the parameters in a gradient descent step:

$$\theta \leftarrow \theta - \epsilon \nabla J \quad (2.8)$$

Where  $\epsilon$  is called the learning rate that determines how big the steps are.

## Argmax function

The process of finding optimal parameters  $\theta$  to maximize a function is often referred to as *argmax*. This can be implemented when having a network, for example a NN, where the only part of the network that changes is the parameters  $\theta$ . Usually these parameters are the weights  $w$  of a network or

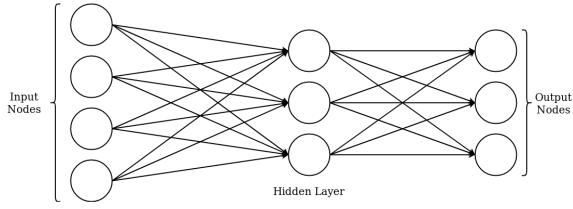


Figure 2.2: Illustration of a multi layer NN with 4 input nodes, one hidden layer with 3 hidden nodes and 3 output nodes.

the inputs  $x$ . Using the theory previously given for ANN's the goal becomes to find optimal parameters  $\theta^*$  such that:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} J(\theta) \quad (2.9)$$

Instead of using gradient descent we can now use gradient ascent, since we are trying to find the maximum. This algorithm is exactly the same as gradient descent except the gradient step, this becomes:

$$\theta \leftarrow \theta + \epsilon \nabla J \quad (2.10)$$

Here we use a  $+$  instead of a  $-$  since we want to ascent.

### 2.1.2 Multi layer neural network

So far, we have discussed single layer neural networks. For more complex tasks we can combine multiple single layer neural networks to create a multi layer neural network, in figure 2.2 such a multi layer NN is shown. In this example the network only has one hidden layer, but this can of course be increased to any amount. These multilayer NN's can be trained using the backpropagation algorithm, which essentially does gradient descent over multiple layers.

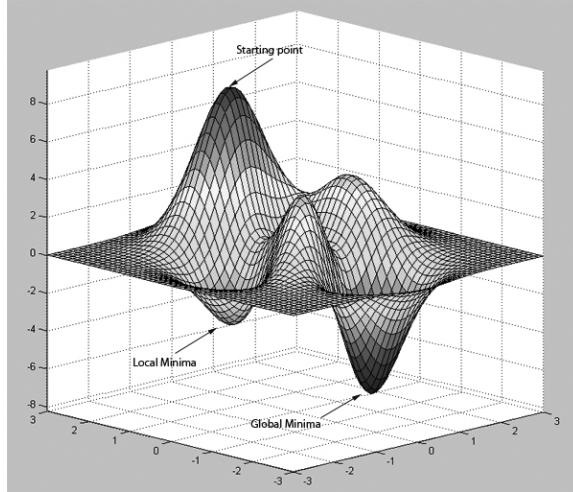


Figure 2.3: Example of a non-convex surface of a cost function.<sup>1</sup>

### 2.1.3 Adaptive learning rate

For gradient descent and backpropagation we need to set a learning rate. This learning rate determines how big the parameter updates are after each iteration. The goal is to find a minimum of the cost function that is as low as possible, ideally the global minimum.

For single layer NN's we are guaranteed to find the global minimum using gradient descent, this is because it is a convex problem, that is, the downward gradient always points to the global minimum. In this scenario choosing a bigger learning rate could decrease the amount of iterations needed to find the global minimum. However, with a high learning rate it could also step over the minimum. In this case we want to have a smaller learning rate to do more precise steps.

For multi layer NN's finding the global minimum using backpropagation is not guaranteed, since it is a non-convex problem (see figure 2.3). In that case it is possible to get stuck in a local minimum. To increase the chance we find the best possible minimum we want to have a learning rate that is not too big, since it could step over the minimum, but also not too small, since it could get stuck in a undesirable local minimum.

One way to tackle this problem is by changing the learning rate while training with the use of an adaptive learning rate function. Such a function could decrease the learning rate based on the gradient such that at the beginning the steps are big enough such that calculation is faster, but at the end the learning rate is small enough such that a minimum can be reached. One of the most popular algorithms for adaptive learning rate is ADAM [9].

---

<sup>1</sup>Image from: <https://www.codeproject.com/articles/175777/financial-predictor-via-neural-network>

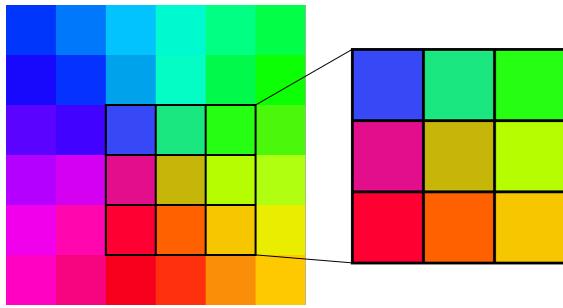


Figure 2.4: Example of a 3x3 kernel.

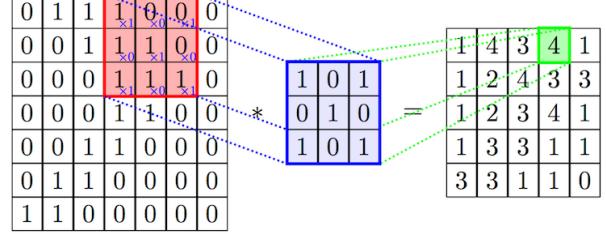


Figure 2.5: Example of the application of a 3x3 convolutional kernel.<sup>2</sup>

#### 2.1.4 Kernels

In image processing a kernel is a small matrix that is applied to a certain part of the image, for example in figure 2.4 a 3x3 kernel on a 6x6 image is shown. This kernel is used to do calculations around a specific pixel. Such a kernel can be scanned over the image such that this calculation is done on every pixel in the image.

In CNN'S these kernels are used to do convolutions between a kernel and an image. These kernels contain values that are multiplied with the image at the location of the kernel. The values of the outcome of the multiplication are stored in a new image as demonstrated in figure 2.5. Depending on the values and size of the kernel this can highlight certain patterns in the image, such as lines or contrast changes.

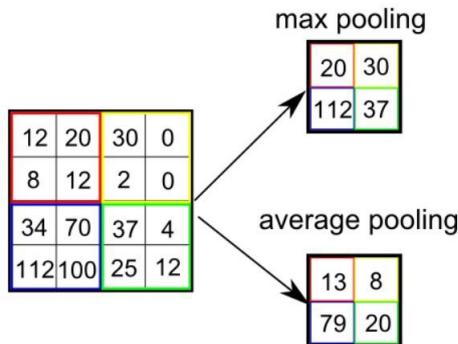


Figure 2.6: Example of Max-Pooling and Average-Pooling.<sup>3</sup>

---

<sup>2</sup>Image from: <https://www.saama.com/blog/different-kinds-convolutional-filters/>

<sup>3</sup>Image from: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

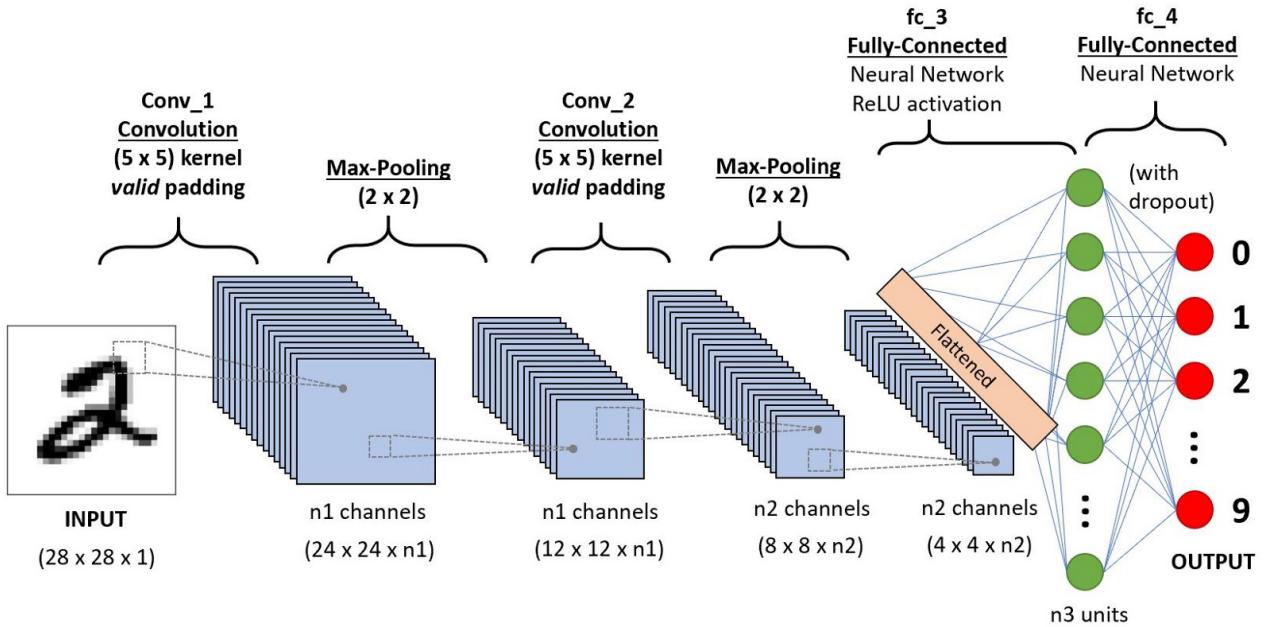


Figure 2.7: Illustration of a convolutional neural network.<sup>4</sup>

### 2.1.5 Convolutional neural networks

The kernels described in section 2.1.4 can be used to create convolutional layers. Convolutional layers learn and apply these kernels as demonstrated in figure 2.5. By applying such a kernel for every possible location in a matrix we get a new matrix. We can apply multiple different kernels on one input matrix to get a feature map. The feature map represents specific features, for example lines or corners. Since these kernels are trained, the network will find the kernels and thus features that more valuable for predicting the correct outcome. Convolution also allows for features to be detected independent of their location in the image. Usually in a convolutional neural network (CNN) we use multiple kernels, after which there is a feature map consisting of the same number of matrices as there were kernels. Usually we want to do multiple convolutions after each other, since this enables the network to detect more complex features. In order to reduce the complexity of the network and increase the generalization of feature detection we want to do pooling between two convolutions. Pooling reduces the dimensions of a matrix by taking the average (Average-Pooling) or maximum value (Max-Pooling) over a region as demonstrated in figure 2.6. Finally, after the convolution layer and or pooling layers we can connect a number of fully

---

<sup>4</sup>Image from: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

connected layers. These are the same as a previously described multilayer neural network with as input the convolution or pooling layer. An example of such a CNN making use of convolution layers, pooling layers and fully connected layers is shown in figure 2.7. The first CNN with backpropagation was introduced by [10].

## 2.2 Content based image retrieval

Image search aims to efficiently and accurately retrieve images from a large database given a textual description or a visual example. Traditionally this has been done using the metadata that is attached to the images, like titles and tags. However, textual information can be inconsistent or absent; therefore, content based image retrieval (CBIR) could be more reliable. CBIR retrieves content information from the images themselves. This content information can substantially be used for the image search. This technique has made advancements in the last decade. In this section we will give an overview of how CBIR works and the different approaches that can be taken.

### 2.2.1 General overview

The goal of content based image retrieval is to get the best matching image based on content from a background database given a query. The challenge that needs to be solved is thus how to compare the images in the database to the query. To do image comparison we need to have a way to transform the image and query into a feature space that can be compared, i.e., an image representation. This feature space could represent visual features of the image, such as textures, color layout, etc .. The goal is to minimize the impact of the parts of the image that are not likely to be important for responding to a query and image transformations (rotation, translation, resizing, illumination, etc...) while having the visual content of the image that could be important when responding to a query well represented in the feature space.

When we say that two images have the same content we, as humans, usually mean the semantic content. For example, if we see two images containing very different looking chairs we could still say that they contain the same content. But for a CBIR system, it can be hard to group these together since the chairs themselves look very different, it therefore has to learn a notion of what a chair is to successfully pair them together (similar to the truck example in chapter 1). This is part of a more general challenge in machine learning, namely the “semantic gap” problem, where there is a difference in having a system that is good at statistically predicting the outcome versus understanding the what the outcome should be. Currently, the best performing CBIR solutions do not directly determine and use the semantics of an image but use visual feature matching methods instead.

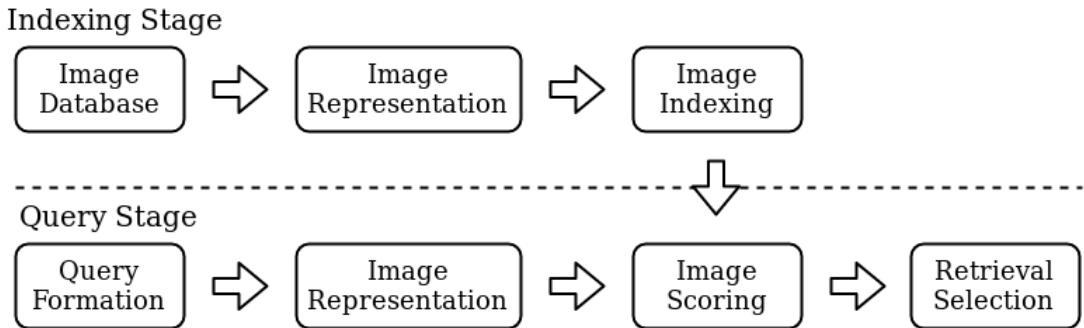


Figure 2.8: General framework of content-based image retrieval. Inspired by [11].

### General flowchart

The core flowchart of CBIR is illustrated in figure 2.8. There are two main parts, the indexing stage and the query stage. The indexing stage starts with having a background image database, from which images will be retrieved. To be able to retrieve images, they have to be represented into a feature space, usually in the form of a vector. In order to increase the comparison speed, done at the image scoring phase, this vector has to be of a fixed-length and be compact, from now on we will refer to it as the compact vector. Reducing the size of the feature space, in the form of a compact vector, could also be beneficial for accuracy of the retrieved images, since with a reduced feature space less important features are not taken into account.

The query stage starts with the query formation, the query describes what we want to retrieve. This query is transformed to a compact vector with the same structure as the compact vectors used in the indexing stage. Next, this compact vector is compared to the compact vectors of images in the background database. Based on the similarities a score is calculated. Finally, we can select the desired retrieved images based on the score. If we want to find the most similar images we select the images with the highest score.

### Image query

The query describes what we want to retrieve. It can be formulated in different ways, such as keywords, an image example [12] or a sketch [13] as illustrated in figure 2.9. In this research, we will use query by example, where given an image query we want to find images that have similar content in them. From now on we will assume the CBIR makes use of query by example image.

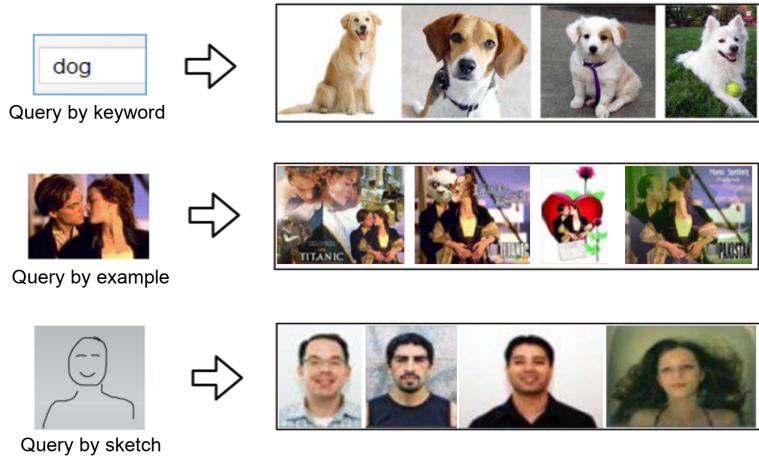


Figure 2.9: Illustration of the query by keyword, query by example and a query by sketch. Adapted figure from [11].

### Feature extraction

As mentioned before, we need to represent the image in a feature space such that it can be compared. Usually, this is done by comparing visual features, and first, the visual features are extracted from the image. Early CBIR algorithms commonly used global features to describe the image content. They combined properties like color, shape, texture and structure to a single representation [11]. These methods worked well for finding duplicate images in large databases [14], but generally do not work well when background clutter is present. This changed in 2003 with the introduction of SIFT [15]. From 2003 to 2012 the big breakthroughs in CBIR were achieved using SIFT-based feature extraction, which we will discuss in section 2.2.2. This changed in 2012 when the classifier AlexNet [16] outperformed the previous best results by a large margin using a deep learning based method. Since then, research on CBIR has mainly focused on using deep learning based methods for feature extraction [17], we will call these CNN based CBIR systems from now on and they will be discussed in section 2.2.3. In figure 2.10 the pipelines of SIFT and CNN based CBIR are illustrated.

### Image retrieval

With the features extracted from the query image it is time to compare them to the features of the images in the database. However, one problem is that the image database can be quite large. When we want to retrieve an image it could be infeasible to compare all visual features, therefore we need to extract a limited number of features and encode those in a smaller representation (usually a fixed sized compact vector). This smaller representation

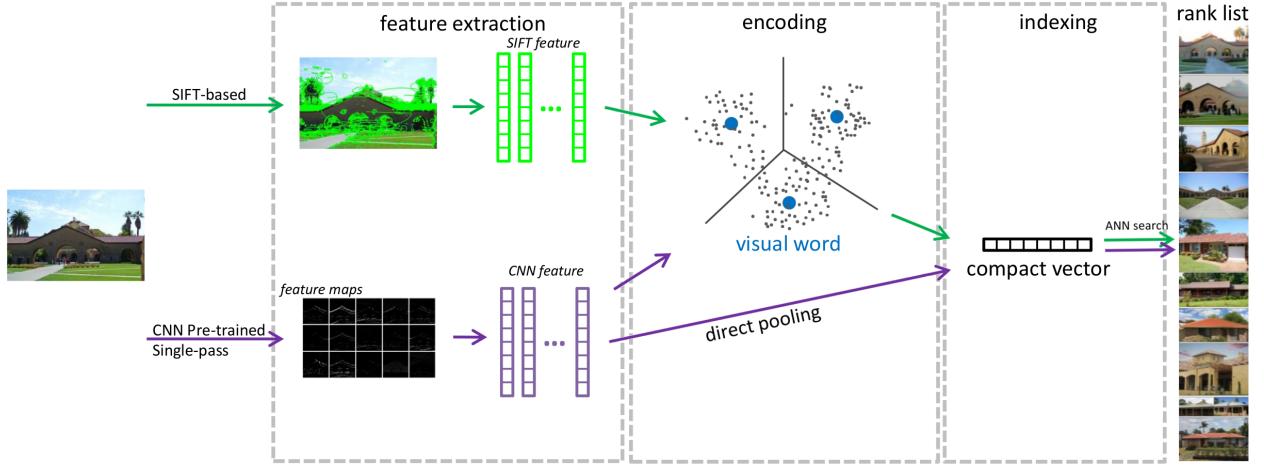


Figure 2.10: Illustration the SIFT and CNN pipelines. Adapted figure from [17].

also causes that only the more important features remain present, resulting in better generalization of retrieval process. Next, this smaller representation is used to do the image comparison, based on the similarity, a score is calculated using an assignment function. One assignment function that is often used is approximate nearest neighbor (ANN) [18]. When we want to retrieve the most similar images we select the images with the highest score.

### 2.2.2 Local feature based CBIR

Local feature extraction consists of interest point detection and local region description. Interest points are points that can be found even after the image has been altered using various transformations. There are multiple methods to find interest points, where in the previous decade SIFT [15] has been most popular and successful since the local features it finds are generally more invariant to translation and resizing than previous methods.

The first step of a local feature based CBIR is to extract the local features. This process could yield thousands of these features, resulting in a feature vector that would be too long to do retrieval on a large image database since the comparison of the feature vectors between two images would take to much processing time. Also a long feature vector would not generalize well. To tackle this problem the features can be transformed into a fixed-length compact vector. One way is by the Bag-of-Visual-Words (BoVW) model [12]. This is usually implemented using a form of k-means clustering, where the k defines the size of the compact vector. Using this implementation, keypoints that are similar are in the same cluster and are regarded to be the same. Finally, the resulting vector is compared to the vectors of the images in the database using for example the approximate

nearest neighbor [18] method.

### 2.2.3 CNN based CBIR

In the last few years, CNN based CBIR methods have received the most research attention and are slowly replacing SIFT based methods [17]. There are multiple configurations possible for CNN based CBIR, we will discuss pre-trained CNN models since this is the same as the CBIR model used in this research.

#### Pre-trained CNN Models

When looking at the structure of CNN's they can be viewed as a set of layers that do convolution and pooling and at the end connected to a fully connected NN (see figure 2.7 and section 2.1.5). These convolution layers are sensitive to certain visual patterns as demonstrated in [19]. On the lower layers, these patterns are more primitive, like edges, squares and circles. On the higher level layers, more elaborate patterns start to appear, like dog faces or complex brick patterns. These lower layers can be compared to SIFT based methods, since they observe the same kind of patterns. However, the complex patterns that the higher level layers track is something SIFT based methods are unable to do. This enables CNN based CBIR methods to outperform SIFT based methods.

When working with CNN's that are used for similar tasks, there is some representations of these convolution layers, i.e., the patterns used for classification can be used for feature extraction. Thus instead of starting with a completely untrained CNN you can start with a CNN that is already trained for a different but similar task and then retrain the network to do image retrieval. By fine-tuning of the network. Popular pre-trained networks are AlexNet [16], VGG [3] and ResNet [20] trained with ImageNet [21].

#### Feature extraction

One way is to add a fully connected NN layer to a CNN (similar to figure 2.7), the outcome of the fully connected NN layer can than be used as the feature representation [16]. This fully connected NN layer is connected to the last convolution layers, combining all the information from these patterns to form a compact feature vector.

More recent proposed CNN based CBIR methods also use lower level convolution layers [22] [23] [24]. Here, a consideration can be made what layers to use. Lower convolution layers contain patterns similar to SIFT while higher convolution layers contain more complex patterns.

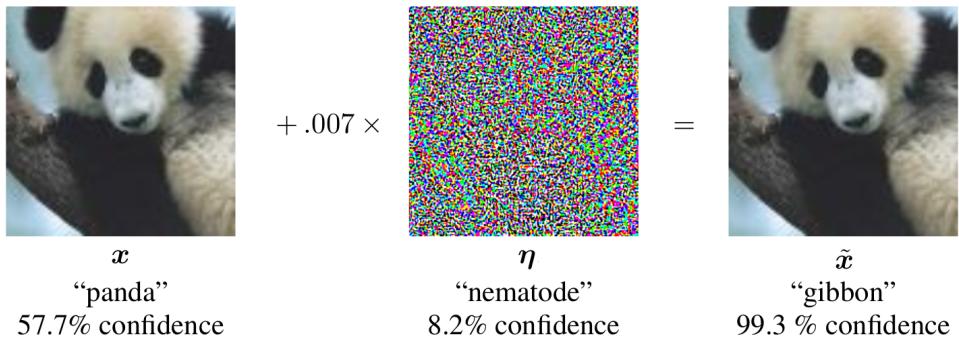


Figure 2.11: An adversarial example applied to GoogLeNet [26]. Altered figure from [1].

### Image retrieval

Finally, the feature information has to be encoded to a short fixed sized compact vector that can be used for the comparison. There are generally two ways this is done. The first is by encoding. This is done similarly to how SIFT features where encoded Bag-of-Visual-Words (BoVW), but instead of using the keypoints, the values of the last convolution layer or pooling layer of the CNN feature extractor are used. The second option is to use a pooling layer [22] [24] to go from the found features straight to the compact vector, this has the added benefit that they are less sensitive to the location of the found features compared to BoVW, since the convolutional layers find patterns in certain areas of an image.

## 2.3 Creating adversarial examples

In 2014 Szegedy *et al.* [25] made the discovery that several machine learning models were vulnerable to “adversarial examples”. In chapter 1 this term is already introduced, for completion we will repeat it here. Adversarial examples are examples (such as images) that are only slightly altered but result in different outcomes when analysed by the machine learning model. Further in this research the term adversarial query is mostly used, this is an synonym for adversarial example. They also found that the same adversarial examples are often misclassified when tested on different machine learning classifiers that use different methods. Thus it looks like there is something intrinsic about how most modern machine learning algorithms work that make them vulnerable to adversarial examples.

### 2.3.1 Intuition

What makes these machine learning methods vulnerable to adversarial examples? In chapter 1 a crude example was given in how changing the color

of a truck can create an adversarial example (figure 1.1), with the intuition that in the training set of such classifier there were probably more red trucks than green ones. However, this example is a bit too simple and does not explain how smaller changes to an image that do not change the color of the whole image can still generate an effective adversarial example. In [1] two explanations to why adversarial examples work are described. One for linear machine learning models and one for deep learning methods.

### Adversarial examples on linear machine learning models

For simplicity we consider a linear machine learning model consisting out of a fully connected ANN (see figure 2.1 and section 2.1). Let us consider small perturbations  $\eta$  (small changes) that are made to the inputs  $x$  creating an adversarial input  $\tilde{x} = x + \eta$ . Now consider what happens when this perturbation  $\eta$  is propagated through the network with weights  $w$ :

$$w^T \tilde{x} = w^T x + w^T \eta \quad (2.11)$$

The activation in one node of the output layer of the NN will grow by  $w_i^T \eta$ , where  $w_i$  are the weights connected to that node. We can increase this activation by choosing the perturbations to be  $\eta = \epsilon \text{sign}(w_i)$ , where  $\epsilon$  is the relative size of the perturbation. This will cause every weight connection to the specific output node to grow positively. If the NN has a small number of input nodes, thus  $n$  is small, all the combined maximised perturbations  $\eta$  will not change the activation of the output node  $i$  substantially. However, this activation grows linearly with  $n$ , thus with more input nodes the activation will grow while the perturbations  $\eta$  are still very small. This shows that a sufficiently large NN is potentially vulnerable to adversarial examples. An implementation of this technique is demonstrated in figure 2.11, where a picture of a panda is perturbed creating a picture that is classified as a “gibbon”, whilst these perturbations are so small that they are barely perceivable. In [1] they also give a reason why these adversarial examples can seem counter-intuitive: *“We live in three dimensions, so we are not used to small effects in hundreds of dimensions adding up to create a large effect.”*

### Adversarial examples on deep neural networks

For deep neural networks there is not a simple explanation why they are vulnerable to adversarial examples, it could be argued that theoretically they should be able to resist adversarial examples, since the universal approximator theorem [27] guarantees that a neural network with at least one hidden layer can represent any function as long as the hidden layer is sufficiently large, i.e., there could exist a function that is implemented as a

multilayer NN that is resistant to adversarial examples. This, however, assumes that the human perception of adversarial examples is representable by a function. The universal approximator theorem also does not guarantee that a network can be trained to find such a function. Despite this, it has been shown that a multi layered NN can be trained to be more resistant to adversarial examples [25][28]. This can be done by, for example, including adversarial examples into the training data. There are multiple methods that can be used to create adversarial examples for deep neural networks. One method is to use a similar technique as previously described for linear machine learning models [29], by looking how input data propagates through the network. Using this knowledge we can find how to precisely alter the input such that a specific outcome is achieved. This method, however, needs full access to the internals of the network.

Another way is by changing the input and only looking at the output of the network, one can then observe how different inputs change the output. This information can then be used to create adversarial examples [30]. This is called the “whitebox” method, since one does not have to know the internal workings of the network, but does need to have access to the network in order to generate adversarial examples. When a method does not need any access to a network to generate adversarial examples, it is called a “blackbox” method.

There are two different goals when making adversarial examples, targeted and un-targeted. Targeted adversarial examples aim to get a specific outcome when presented to a network, i.e. they want to “steer” the outcome. Un-target adversarial examples aim to get any outcome except the correct one, generally the amount of perturbations needed to achieve un-targeting is less.

### 2.3.2 Adversarial examples for CBIR systems

Generating adversarial examples for CBIR based systems is a new field of study. This research is based on (one of the) first published techniques for creating adversarial examples for neural based CBIR systems [4] named PIRE (perturbations for image retrieval error). In section 2.4, we will explain how PIRE creates adversarial examples for CBIR systems.

## 2.4 Recent work

This section elaborates on work that is related to this research. The work is about a state of the art CBIR system that we will use, a method to create adversarial queries for CBIR systems and reducing the noticeability of perturbations when generating adversarial queries.

### **Fine-tuning CNN image Retrieval with No Human Annotation**

In [24] a CNN image retrieval model is proposed that achieves state of the art performance. This model starts with a pre-trained network like ResNet101 [20] or AlexNet [16]. Next, the network is fine-tuned using a method called structure-from-motion (SfM). This method takes unordered images and attempts to create all possible 3D models. If it can successfully create a 3D model with a set of images they are considered to be matching. This enables a better selection of matching images and non-matching images without needing any human annotation. Finally, the final convolutional layers of the fine-tuned CNN are connected to a novel pooling layer. This pooling layer consist out of a generalized-mean (GeM) with trainable pooling parameter. They show that this outperforms non-trainable pooling layers and that the results of their CBIR system outperform previous state-of-the-art approaches.

### **Towards imperceptible and robust adversarial example attacks against neural networks**

In [30] a method is described to generate adversarial examples for CNN classifiers that are less perceptible to the human eye and are more robust under transformations like JPEG compression, Gaussian blurring, contrast and brightness changes. In this work, they describe a targeted attack, where they want to generate an adversarial example that will classify to a predetermined outcome. To do this they use a whitebox technique, that is, the CNN classifier and the image query are available, but the contents of these are not known. Their approach works as follows, first they calculate what the impact is when a specific pixel in the image is perturbed, next they sort these and select the pixels that have the most impact. The impact is defined by how big the adversarial effect is versus the human perception of perturbing that pixel, which they want to minimize. They perturb the pixels with the most impact and repeat all the steps until the desired adversarial effect has been reached. The human perception of how noticeable a perturbation would be is determined by the standard deviation around the to be perturbed pixel.

### **Who’s Afraid of Adversarial Queries? The Impact of Image Modifications on Content-based Image Retrieval**

In [4] a whitebox method to create un-targeted adversarial queries for CBIR based systems is proposed. Our solution CV-PIRE is build upon this technique. In this section we will only discuss the general idea of PIRE, we will discuss the details of PIRE in section 3.3 since our method CV-PIRE is based on this. PIRE makes use of an unsupervised learning method, where the goal is to maximize the distance of the compact vector representation

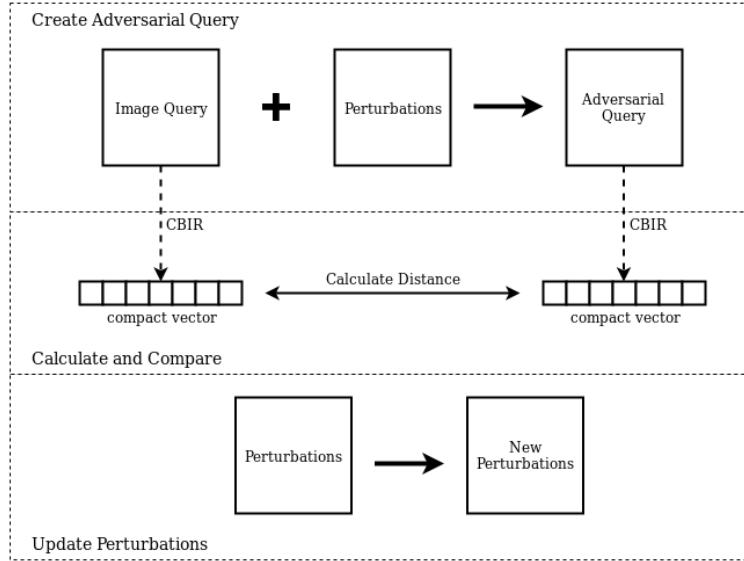


Figure 2.12: Illustration of one iteration of the PIRE [4] pipeline.

generated by the CBIR system between the image query and adversarial query. This is done as follows:

- PIRE starts with generating a random vector with the same dimensions as the image query, this vector contains the perturbations that cause the adversarial effect. Since the perturbations will become quite noticeable when the perturbations are too big, every perturbation has to be within a perturbation vector range that limits the maximum perturbation.
- The next steps are iterated  $T$  times, the steps that are repeated are illustrated in figure 2.12.
- Create Adversarial Query: The perturbation vector is added onto the image query to create the adversarial query.
- Calculate and Compare: The compact vector of the image query and the adversarial query is determined by the CNN of the used CBIR system and the distance between these two compact (feature) vectors is calculated.
- Update Perturbations: Based on the distance, the perturbations are updated using *argmax* (see section 2.1), such that the distance between the compact vectors is maximized.

After  $T$  iterations the adversarial query is finished. Using this approach the authors of [4] were able to reduce the performance (mAP) of the GeM [24] CBIR from 74.42% to 2.31% on the Oxford-5k [31] dataset.

## 2.5 Image processing techniques and color perception

### 2.5.1 Color spaces

A color space specifies a way to numerically describe colors. Digital images consist of an array of pixels. These pixels each have their own color, these colors have to be numerically represented in color spaces.

#### RGB color space

The RGB color space is the most used color space in modern computers. This color space describes the red, green and blue values of a pixel. In the standard format, 8-bytes are dedicated to saving one of these colors values, thus the values range from 0 to 255. The format of these colors is in the form of  $(R, G, B)$ . Since computer screens work with light the colors are additive. Thus  $(255, 255, 255)$  is white,  $(0, 0, 0)$  is black and  $(255, 0, 0)$  is red. The reason the RGB color space is the standard when working with digital colors is that digital screens and cameras work with these three values. For every pixel in a screen there is a red, green and blue element and for every pixel in a camera sensor there is also a red, green and blue element. However, this does not mean that it is the best color space for every application. One downside of the RGB color space is that it is not visually uniform for humans, i.e., equal distances between two colors are not always perceived similarly.

#### CIELAB color space

To tackle this problem the CIE (International Commission on Illumination) introduced in 1979 the CIELAB color space [32]. This color space uses three variables to save the color data,  $L^*$ ,  $a^*$  and  $b^*$ . Where  $L^*$  represents the lightness of the color,  $a^*$  green-red component and  $b^*$  blue-yellow component. The distance between two colors in the CIELAB color space is closer to the human perception of the color difference than the distances in the RGB color space.

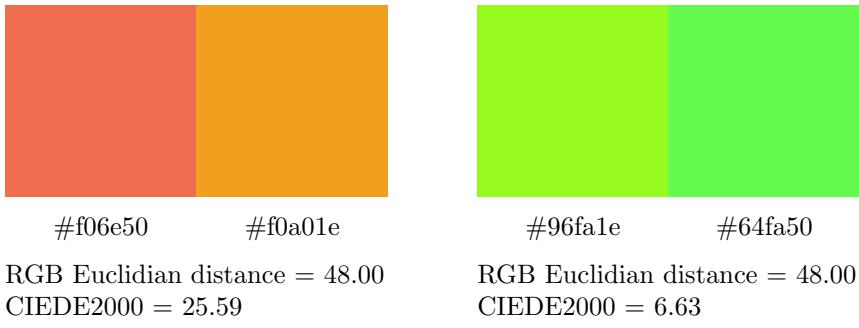


Figure 2.13: Comparison between RGB Euclidean color difference and CIEDE2000 [33] on two different color pairs. The values are linearly normalized, where 100 is the maximum color difference.

### 2.5.2 Human color perception

There are different methods to save color values of pixels within digital images. These methods are also called color spaces. The most common color space for digital images is RGB (see section 2.5.1). When we want to compare the color difference between two pixels within the RGB color space, we calculate the Euclidean distance between the Red, Green and Blue values of the two pixels, see equation 2.12.

$$distance = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2} \quad (2.12)$$

However, this does not consistently correspond with the human perception of color difference. In figure 2.13 two color comparisons are shown. When calculating the color difference using the Euclidean distance within the RGB color space we get the same score. While clearly for humans the color difference between the red and orange colors is greater than the color difference between the two shades of green.

To get a numerical color distance that is more closely related with how humans perceive color difference CIELAB introduced the CIELAB color space [32] (section 2.5.1). Throughout the years improvements on the CIELAB color space were made, notably the CIEDE2000 algorithm [33] that improved on the performance of blue colors and a better scaling factor for grey colors, CIEDE2000 outperformed CIELAB by a large margin, therefore it has been adopted to be the international standard for color-difference calculation. Note that CIEDE2000 is not a color space but an algorithm that given two colors represented in the CIELAB color space it returns the human perceived numerical color difference, where 0 is no color difference and 100 is the maximal perceivable color difference.

In figure 2.13 it is demonstrated how the CIEDE2000 color difference score better resembles human color perception compared to the Euclidean distance in the RGB color space.

0.077847	0.123317	0.077847
0.123317	0.195346	0.123317
0.077847	0.123317	0.077847

Figure 2.14: 3x3 Gaussian kernel with a standard deviation  $\sigma = 1$ .

### 2.5.3 Gaussian kernel

A Gaussian kernel is a kernel that consists of a Gaussian distribution. Pixels closer to the origin are given more weight than pixels further away (see section 2.1.4 on how kernels work). The distribution of such a Gaussian kernel is calculated with equation 2.13.

$$Gauss(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2.13)$$

Where  $x$  and  $y$  are the pixel distance from the origin and  $\sigma$  is the standard deviation. In figure 2.14 such a Gaussian kernel with  $\sigma = 1$  is shown. In this case, the dimensions of the kernel are 3x3 ( $n = 3$ ). In the ideal scenario the kernel size is as large as the image. However, practically, smaller kernel sizes ( $n = 3, 5, 7, \dots$ ) are chosen for computational reasons. For a standard deviation  $\sigma \geq 1$  the Gaussian distribution approximates 0 for pixels further away. For example, using equation 2.13:  $Gauss(x = 3, y = 3, \sigma = 1) \approx 0.00002$ , this is the same value of the corners for a kernel with size  $n = 7$ . Therefore choosing larger kernel sizes when  $\sigma \geq 1$  will usually not significantly impact the results.,

# Chapter 3

## Approach

In this chapter we will explain all the different elements that contributed to the creation of our solution CV-PIRE for generating adversarial queries. First, we will outline the intuitive insight behind our solution (section 3.1). Based on this, an algorithm is proposed that calculates the how noticeable a perturbation on a specific pixel in the image will be (section 3.2). This algorithm is then used to determine the perturbation threshold (how much any single pixel may be perturbed), this is then used to generate the adversarial query (section 3.3). Finally, we will discuss the methods that are going to be used to evaluate our solution CV-PIRE (section 4.1).

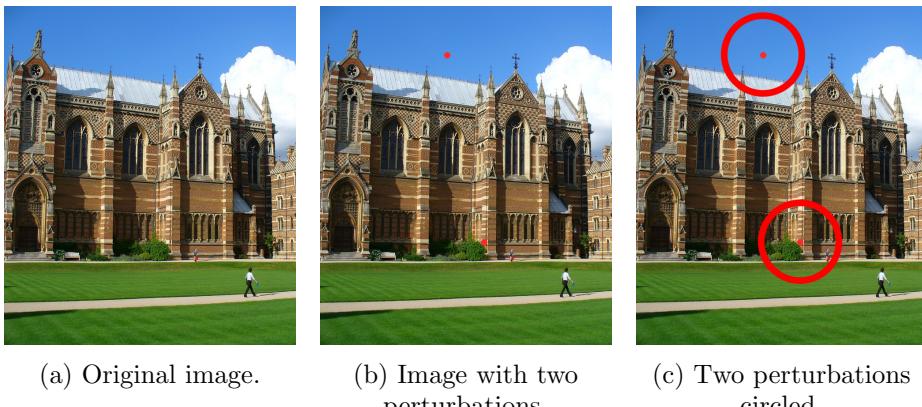


Figure 3.1: A cropped image query from the Oxford-5K [31] data set. On the left side the original query image is shown, the middle image contains two of the perturbations of the same magnitude, in the right image the perturbations are circled.



(a) Adversarial query generated by  
PIRE [4].  
(AP = 1.63, SSIM = 0.775)

(b) Adversarial query generated by our  
color variance solution CV-PIRE.  
(AP = 0.465, SSIM = 0.883)

Figure 3.2: Adversarial queries generated by PIRE and our solution. The image originates from the Oxford-5K [31] dataset. The AP and SSIM score are added for completeness (these definitions will be discussed in section 4.1).

### 3.1 Intuitive insight

In figure 3.1 three images are visible. In the middle image, two identical perturbations are added. We would argue that the perturbation in the sky is more noticeable than the perturbation on the building. This could be explained by the difference in color variance, i.e., the amount of color difference around a pixel, as this is much lower in the sky than on the building. This results in the perturbations in the sky being more noticeable than the perturbations on the building. We see a similar effect when an adversarial example is created from the same image query using PIRE [4] as shown in figure 3.2a. Even though the maximum perturbation per pixel is the same for the whole image, we would argue that people would notice the perturbations in lower color variance regions like the sky the most. But the perturbations in higher color variance regions like on the wall of the building are almost imperceptible. The paper [30] discussed in section 2.4 also works with the concept that perturbations in higher color variance regions are most likely less perceptible to the human eye, however, they do not consider color perception.

This leads to a new set of possibilities regarding the creation of adversarial query images for CBIR systems. What would happen if the perturbations in low color variance regions are reduced and the perturbations in high color variance regions are magnified? What if this could be done in such a way that the adversarial query image performs as well or even better than the ones generated with PIRE [4] (figure 3.2a), while at the same time being less noticeably altered to the human eye. With this question as a starting

point we developed a method that generates adversarial queries that are less noticeable to the human eye. An example is shown in figure 3.2b. Our method is called CV-PIRE<sup>1</sup> (Color Variance based Perturbations for Image Retrieval Error).

## 3.2 Pixel color variance

In order to take the color variance within an image query into account we first need a way to calculate these. In this section the method to calculate the color variance around every pixel in an image query is explained.

### 3.2.1 Gaussian kernel

When generating adversarial examples every pixel has a perturbation threshold, the maximum amount the pixel may be perturbed (see section 3.3). When perturbing a pixel its surrounding pixels are of importance for the human perception (see section 3.1). Therefore, we take the human perceived color difference of the surrounding pixels into account. However, if we would only do this for the directly neighboring pixels we could get the situation where two pixels next to each other have extremely different pixel color variance and thus could have very different perturbations, this could be quite noticeable. For example, if there is a bird far away in the sky in an image query. This bird is so far away that it only resembles one black pixel in the image. However, this one black pixel has a color difference compared to its surrounding blue pixels (the sky). When taking only the direct neighboring pixels into account the pixel color variance compared will be quite a bit bigger than the color variance of the surrounding pixels. This could lead to the scenario where the one black “bird” pixel is perturbed a lot more than its surroundings and therefore become quite noticeable. Therefore we want to take more of the surrounding pixels into account. This results in the situation that two pixels next to each other cannot have extremely different pixel thresholds, reducing extreme perturbations compared to its neighbouring pixels. The amount of surrounding pixels we take into account when calculating the pixel threshold we call kernel size  $n$ . For example,  $n = 5$  means that we use a  $5 \times 5$  kernel, thus taking the pixels with maximum distance of 2 around the origin pixel into account. To keep detail in the adversarial query image we want to prioritize the pixels closer to the origin, therefore we use a Gaussian kernel (explained in section 2.5.3). Exploratory experiments confirmed the need for the Gaussian kernel.

---

<sup>1</sup>Source code and results available at: <https://github.com/SamSweere/CV-PIRE>

---

**Algorithm 1:** Pixel color variance “without edge cases”

---

**Input:** Image query  $img$ ; Pixel location  $(x_i, y_i)$  ; Kernel size  $n$ ;  
Standard deviation  $\sigma$ ;

**Output:** Pixel color variance for the pixel on location  $(x_i, y_i)$ ;  
Calculate the max distance;  
 $d = \lfloor \frac{n}{2} \rfloor$ ;  
 $x = x_i - d$ ;  
 $y = y_i - d$ ;  
 $totalVar = 0$ ;  
Access every surrounding pixel within max distance  $d$ ;  
**while**  $x \leq x_i + d$  **do**  
    **while**  $y \leq y_i + d$  **do**  
        Calculate the Gaussian distribution contribution;  
         $g = Gauss(|x - x_i|, |y - y_i|, \sigma)$ ;  
        Calculate the CIEDE2000 pixel difference;  
         $diff = CIEDE2000(P_{x,y}, P_{x_i,y_i})$ ;  
        Calculate the total color variance, taking the root mean square;  
         $totalVar = totalVar + (diff * g)^2$ ;  
         $y = y + 1$ ;  
      
     $x = x + 1$ ;  
Complete the root mean square calculation;  
 $pV = \sqrt{\frac{totalVar}{n^2}}$ ;  
Return the pixel color variance;  
**return**  $pV$ ;

---

### 3.2.2 Algorithm to calculate the pixel color variance

All the elements to formulate an algorithm to calculate the color variance around a pixel have now been introduced, and brought together in algorithm 1. In this algorithm  $img$  represents the image query,  $(x_i, y_i)$  represents the origin pixel location of which we want to know the color variation compared to its surrounding pixels based on the kernel size  $n$  and  $\sigma$  represents the standard deviation of the Gaussian distribution. The algorithm loops through every pixel that is within the kernel range. For each of these pixels it calculates the color difference compared to the origin pixel (the pixel on  $(x_i, y_i)$ ) using the CIEDE2000 algorithm [33] (see section 2.5.2) and multiplies this number with the Gaussian distribution value based on the distance. Finally, of all these values the root mean square is calculated, this is the final pixel color variance  $pV$  for the pixel on location  $(x_i, y_i)$ . Note that at one point in algorithm 1 the color difference of the origin compared to itself is calcu-

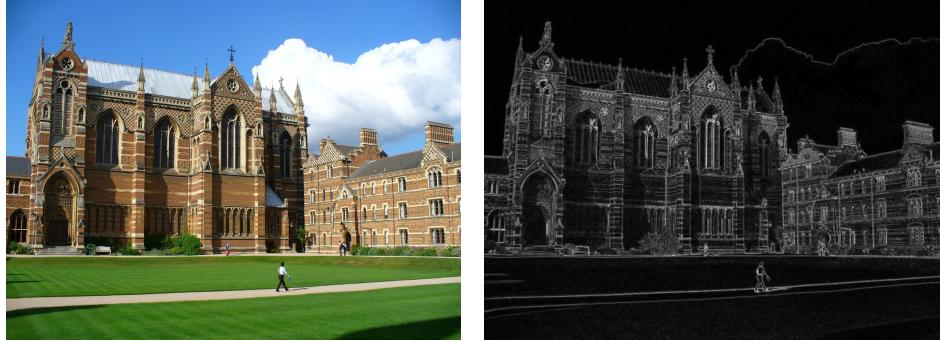


Figure 3.3: An image query from the Oxford-5K [31] data set (left) has pixel color variance (right). The pixel color variance was calculated with  $n = 5$  and  $\sigma = 1$ .

lated ( $CIEDE2000(P_{x_i,y_i}, P_{x_i,y_i})$ ). One might be alarmed that calculating the difference with itself would cause problems. But since we are comparing the perceived difference between the same pixel the outcome of CIEDE2000 is zero. Therefore, it does not contribute to the accumulated color variance  $pV$ .

Algorithm 1 only works when the pixel location  $(x_i, y_i)$  has a minimal distance of  $d = \lfloor \frac{n}{2} \rfloor$  from the edge of the image. To calculate the color variance closer to the edge of the image the same steps are performed, but when a pixel on location  $(x, y)$  is out of bounds we skip this pixel. Note that if a pixel is skipped the final root mean square calculation becomes  $pV = \sqrt{\frac{totalVar}{c}}$ , where  $c$  is the number of pixels that contributed to the  $totalVar$  (that were not out of bounds).

### 3.2.3 Pixel color variance image

Using algorithm 1 we can now calculate the pV (pixel color variance) for every pixel in an image query. When we have all the pV's we linearly normalize these values, where 1.0 is the biggest pV of the image query and 0.0 is no pV at all. The reason for normalization is that in an image query with little color differences we still want to do perturbations, if we would not normalize the pixel color variance values could become too small to do any significant perturbations resulting in a weak adversarial query. After calculating all the pV's of an image query and normalizing them we can visualise the outcome as an image as shown in figure 3.3, this also helps in getting some insight on where the perturbations will take place.

---

**Algorithm 2:** Color Variance based Perturbations for Image Retrieval Error (CV-PIRE)

---

**Input:** Image query  $\mathbf{img}$ ; Color variance matrix  $\mathbf{cV}$ ; CBIR feature function  $f$ ; Threshold multiplier  $m$ ; Iterations  $T$ ;

**Output:** Generate adversarial image query;  
Get the final per pixel threshold matrix;  
 $\mathbf{pT} = \mathbf{cV} \cdot \frac{m}{255}$ ;  
Get the width and height of the image;  
 $w = \text{width}(\mathbf{img})$ ;  
 $h = \text{height}(\mathbf{img})$ ;  
Generate a random matrix between  $-1$  and  $1$  with the same dimensions as the image and multiply it with the pixel threshold;  
 $\mathbf{v} = \text{random}(-1, 1, h, w) * \mathbf{pT}$ ;  
Update  $i = 0$ ;  
**while**  $i < T$  **do**  
    Determine the distance between the perturbed image  $\mathbf{img} + \mathbf{v}$  and the original image  $\mathbf{img}$  according to the CBIR feature function  $f$ ;  
     $v = \underset{\mathbf{v}}{\operatorname{argmax}} \|f(\mathbf{img}) - f(\mathbf{img} + \mathbf{v})\|_2^2$ ;  
    Make sure the perturbation  $\mathbf{v}$  stays within the pixel threshold  $\mathbf{pT}$ ;  
     $\mathbf{v} = \text{clip}(\mathbf{v}, -\mathbf{pT}, \mathbf{pT})$ ;  
     $i = i + 1$ ;  
Return the perturbed image as the adversarial query;  
**return**  $\mathbf{img} + \mathbf{v}$ ;

---

### 3.3 Generating adversarial examples

The method to generate the adversarial queries is presented in algorithm 2, this algorithm is build on PIRE [4] (see section 2.4). The algorithm starts by calculating the pixel threshold matrix  $\mathbf{pT}$ . This 2-d matrix contains the maximum perturbation values for every pixel in the image. The dimensions of the pixel threshold matrix  $\mathbf{pT}$  is the same as the image query  $\mathbf{img}$ . The pixel threshold matrix  $\mathbf{pT}$  is calculated using:

$$\mathbf{pT} = \mathbf{pV} \cdot \frac{m}{255}$$

Where  $\mathbf{pV}$  is the color variance matrix for the image query  $\mathbf{img}$ . This is obtained by calculating the pixel variance  $\mathbf{pV}$  for every pixel in the image query  $\mathbf{img}$  using algorithm 1 and linearly normalizing these values as discussed in 3.2.3.  $m$  is the threshold multiplier, this variable scales the pixel threshold, the larger this variable the bigger the  $\mathbf{pT}$  values, thus resulting in bigger perturbations. Next, a matrix  $\mathbf{v}$  with the same dimension as the

image query  $\mathbf{img}$  containing random values in the range of  $\{\mathbf{pT}, -\mathbf{pT}\}$  is generated. This matrix  $\mathbf{v}$  contains the perturbations for every pixel in the image.  $\mathbf{v}$  is iteratively optimized using the objective functions :

$$\underset{\mathbf{v}}{\text{maximize}} \quad \|f(\mathbf{img}) - f(\mathbf{img} + \mathbf{v})\|_2^2 \quad (3.1)$$

Where the absolute values of perturbation matrix  $\mathbf{v}$  have to be pairwise smaller or equal to the pixel matrix  $pT$ . The optimization process will stop after  $T$  iterations.  $f$  is the CBIR feature extraction function. This function returns the CBIR specific vector representation of an image query (see section 2.2). Our solution is designed assuming that the CBIR model is available. In this research we are using the GeM model [24] (see section 2.4), GeM is a state of the art CNN image retrieval model. The specific structure we are using was based on the ResNet101 [20] pre-trained on ImageNet [34] and fine-tuned on 120k Flickr images by using a structure-from-motion pipeline. The  $\text{argmax}$  function works by first calculating the loss function:

$$l = \|f(\mathbf{img}) - f(\mathbf{img} + \mathbf{v})\|_2^2 \quad (3.2)$$

This results in the cost function:

$$J(\mathbf{v}) = \frac{1}{2} \|f(\mathbf{img}) - f(\mathbf{img} + \mathbf{v})\|_2^2 \quad (3.3)$$

The cost function results from applying the mean squared error (MSE) given in equation 2.3 in section 2.1 with  $N = 1$ ,  $y = f(\mathbf{img} + \mathbf{v})$  and  $t = f(\mathbf{img})$ . Since the CBIR feature function  $f$  and the input image  $\mathbf{img}$  are fixed we can use gradient ascent (see section 2.1). To do this we first calculate the cost gradient  $\nabla J$  of the cost function 3.3 based on  $\mathbf{v}$ . Knowing the gradient of the cost function  $\nabla J$  in respect to  $\mathbf{v}$  we can update  $\mathbf{v}$  using:

$$\mathbf{v} \leftarrow \mathbf{v} + \epsilon \nabla J \quad (3.4)$$

Where  $\epsilon$  is the learning rate. To increase the computation time and precision the adaptive learning rate algorithm ADAM [9] is used (see section 2.1.3). After updating the perturbation matrix  $\mathbf{v}$  we have to make sure that all new perturbation values are still within the maximum pixel threshold  $pT$ . Therefore we clip the perturbation matrix using:

$$\mathbf{v} = \text{clip}(\mathbf{v}, -\mathbf{pT}, \mathbf{pT}) \quad (3.5)$$

When working with a neural based CBIR model like GeM [24] the image query needs to be normalized to the mean and standard deviation of the dataset the model was trained on to get the best results. Finally, when the adversarial image query is generated this image has to be de-normalized to its original mean and standard deviation.

### 3.3.1 Saving adversarial queries

Some pixels of the resulting adversarial query can have very small perturbations. When saving such an image query these small perturbations may disappear due to a limited number of bits used in the image format. However, it is crucial that the adversarial queries remain adversarial when saved. Therefore we can enlarge the small perturbations in such a way that they remain present when saved. In this research, the adversarial queries are saved in an uint8 JPEG format, this means that for every pixel 8 bits are used. If a perturbation is so small that it would be rounded away when saved in this format, we enlarge it just enough such that it is retained when saved. In this case we change the return statement of algorithm 2 to:

```
return img + enlargeToImage(v);
```

Where the *enlargeToImage()* is the function that enlarges the perturbations such that they remain.

### 3.3.2 Determining the hyperparameters

In order to implement the CV-PIRE method there are hyperparameters that have to be chosen. For the pixel color variance ( $\mathbf{pV}$ ) (algorithm 1) the kernel size  $n$  and standard deviation  $\sigma$  have to be chosen. We are going to first determine the standard deviation  $\sigma$ . This influences how big the importance of the pixels further away from the origin is when calculating the pixel color variance. Next, the kernel size  $n$  will be determined, this has to be sufficiently large such that there is no significant difference with an even bigger kernel size. This is because the Gaussian distribution factors (determined by  $\sigma$ ) make pixels that are sufficiently far away from the origin practically zero, thus there are no benefits for having a bigger kernel size.

For CV-PIRE (algorithm 2) the hyperparameters are the threshold multiplier  $m$  and iterations  $T$ . The threshold multiplier  $m$  determines how big every perturbation can maximally be. If the perturbations are bigger, we expect that the adversarial query will perform better at fooling the CBIR system, however, with bigger perturbations we also expect that the perceived image quality will be lower, since bigger perturbations probably are easier to spot for the the human eye. In this research, the goal is to hide the actual content in the image query for the CBIR system. Therefore we will choose the threshold multiplier  $m$  such that the adversarial queries of a specific dataset will perform equally bad on the CBIR system as random guessing. If we would choose the threshold multiplier  $m$  so large that the resulting adversarial queries will perform worse than random guessing, someone who knows that the images are adversarial could look at the images that are deemed least similar by the CBIR and retrieve information this way, since in this scenario on average the adversarial queries will score worse than random guessing.

Finally, we have to choose the iterations  $\mathbf{T}$ , we will set this hyperparameter such that the performance of the adversarial query has converged, after which there is no use in doing more iterations.

One hyperparameter that also has to be chosen but is not present in algorithms 1 and 2 is the starting learning rate of the ANN (used in the *argmax* function). This has to be chosen before determining the threshold multiplier  $\mathbf{m}^v$  and iterations  $\mathbf{T}$ . The learning rate influences how big the perturbations are in each iteration. The consideration that has to be made is to set it sufficiently large such that not too many iterations  $\mathbf{T}$  have to be done, but that the image quality of the adversarial queries is still as good as with a smaller learning rate. Having to do a lot of iterations can be impractical due to the increased computing time.

### 3.4 Other algorithms for comparison

When evaluating the adversarial queries generated by CV-PIRE it is important to compare it to other methods that generate adversarial queries for CBIR systems. In this research we will compare CV-PIRE with PIRE [4] (see section 2.4) and SD-PIRE.

#### 3.4.1 SD-PIRE

For comparison we developed another method to generate adversarial queries named SD-PIRE (Standard Deviation based Perturbations for Image Retrieval Error). This method is motivated by [30] (see section 2.4), in which they suggest to use the standard deviation to determine how noticeable a perturbation on a pixel will be. In this method, instead of the color variance  $cV$  (see algorithm 2) the variance of a pixel using the standard deviation of its surroundings is calculated. First, the image is converted to greyscale, most commonly the luminance conversion from RGB to greyscale is used. This conversion takes into account that humans do not perceive all colors in the same intensity. For our implementation we used the luminance conversion values as recommended by the International Telecommunication Union (ITU) [35], this results in equation 3.6.

$$Y = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B \quad (3.6)$$

Where  $Y$  is the luminance and  $R, G$  and  $B$  are the red, green and blue channels of the RGB image. For a pixel  $x_i$  the standard deviation  $SD(x_i)$  is calculated using equation 3.7.

$$SD(x_i) = \sqrt{\frac{\sum_{x_k \in S_i} (x_k - \mu)^2}{n^2}} \quad (3.7)$$

Where  $S_i$  are the pixels in the  $n * n$  kernel with as origin pixel  $x_i$ ,  $\mu$  is the average value of the pixels in the region and  $n$  is the size of the kernel. This method has one parameter, namely the kernel size  $n$ . Since this method does not take into account how far away pixels are from its origin  $x_i$ , the resulting adversarial image becomes quite blurred when choosing a large  $n$ . Therefore, a kernel size  $n$  has to be chosen such that SD-PIRE generates sufficiently good adversarial queries while still keeping the perceived image quality as good as possible.

# Chapter 4

# Experimental results

## 4.1 Evaluating adversarial queries

The goal of this research is to create effective adversarial queries with the highest possible perceptible image quality. Thus the adversarial queries need to be evaluated by:

- A metric that measures the accuracy of the retrieved images by the CBIR. Ideally a retrieved image has the same chance of being relevant as a randomly chosen image from the background dataset. This would make the information retrieved by the CBIR useless and thus protecting the privacy of the owner of the image.
- A metric that measures the perceptual image quality of the adversarial query, we want this to be as high as possible.

### 4.1.1 Measuring the performance of CBIR

To measure the performance of a CBIR on a set of image queries, mean average precision (mAP) is usually used. In order to calculate the mAP, for all the image queries in our testing dataset, the relevant images that the CBIR can retrieve have to be known. When a CBIR processes an image query, it tries to find the images in the dataset that are most similar. When we know the relevant images of the image query, we can check if the images that the CBIR system retrieved are relevant. When evaluating the performance on a single image query we use average precision (AP). AP makes use of both precision and recall. Precision is the fraction of the retrieved images that are relevant. Recall is the fraction of the retrieved images that are relevant compared to all the relevant images. AP is calculated using equation 4.1.

$$AP = \frac{\sum_{k=1}^n P(k) \cdot corr(k)}{R} \cdot 100 \quad (4.1)$$

Where  $P(k)$  is the precision of the top  $k$  retrieved images,  $corr(k)$  is a binary indication if the top  $k$  image is of the correct category, where  $corr(k) = 1$  if the category is correct,  $corr(k) = 0$  otherwise,  $n$  is the number of retrieved images and  $R$  is the total number of images of the same category as the query image.

To determine the performance of multiple image queries, the mean of the average precision (mAP) is taken. The mAP score gives an indication how well a CBIR performed on a set of image queries.

#### 4.1.2 Determining perceived image quality

When determining the perceived image quality we want to compare the original image with its adversarial counterpart. The perceptual difference between two images can be quantified using structural similarity.

##### Mean structural similarity (mSSIM)

Structural similarity (SSIM) is a method for quantifying perceived image quality by measuring the similarity between two images, introduced by [36]. In contrast to other image quality metrics like mean square error (MSE) and peak signal-to-noise ratio (PSNR) that both calculate absolute errors (independent of pixel location), SSIM focuses on perceptual difference by taking luminance, contrast and structure into account. The score of SSIM is between 0.0 and 1.0, where 1.0 is the score given when the same image is compared. The mean structural similarity (mSSIM) is obtained by calculating the mean SSIM of multiple images. See section A.1 for the full formula of SSIM.

#### 4.1.3 Data

This research is focused on privacy, this privacy is violated when a user is tracked based on their images. Tracking can be done when outdoor environments and buildings that have a distinct location can be retrieved. Based on this we chose two industry standard datasets to test the performance of our adversarial technique.



Figure 4.1: Six query images from the Oxford-5K [31] dataset.

### Oxford buildings dataset (Oxford-5k)

The Oxford Buildings Dataset <sup>1</sup> [31] consists of 5062 images sourced from Flickr by searching for 17 text queries of buildings in Oxford. The dataset also contains distractor images. These images are not related to any of the queries. For this dataset 55 query images are used for evaluation. In figure 4.1 a few of the query images are shown.



Figure 4.2: Six query images from the Paris-6k [37] dataset.

### Paris dataset (Paris-6k)

The Paris Dataset<sup>2</sup> [37] consists of 6412 images sourced from Flickr by searching for 12 text queries of landmarks in Paris. The dataset also contains distractor images. These images are not related to any of the queries. For this dataset 55 query images are used for evaluation. In figure 4.2 a few of the query images are shown.

---

<sup>1</sup>Available at: <http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/>

<sup>2</sup>Available at: <http://www.robots.ox.ac.uk/~vgg/data/parisbuildings/>

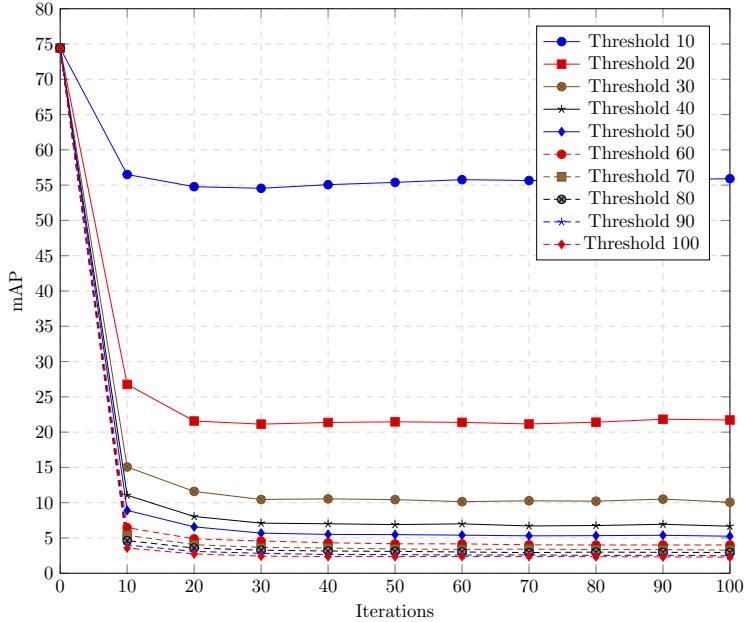


Figure 4.3: Performance (mAP) of neural-feature-based CBIR (GeM [24]) on adversarial queries generated by CV-PIRE on the Oxford-5K [31] dataset for different thresholds and iterations. CV-PIRE used  $n = 5$  and  $\sigma = 1$ .

## 4.2 Determining the hyperparameters of CV-PIRE

When implementing the CV-PIRE method there are some hyperparameters in algorithms 1 and 2 that have to be determined as explained in section 3.3.2.

### 4.2.1 Pixel color variance hyperparameters

When determining the hyperparameters for the pixel color variance  $pV$  (algorithm 1), recall that we first choose the standard deviation  $\sigma$  and then the kernel size  $n$ , since the effective size of the kernel is dependent on the standard deviation. When having a bigger standard deviation the Gaussian distribution (see section 2.5.3) is more localized around the origin. In this case we can use a smaller kernel, since the Gaussian distribution factor further from the origin becomes practically zero. The opposite is true when having a smaller standard deviation, in this case the Gaussian distribution is more spread out and therefore we need to go further away from the origin before the Gaussian distribution factor becomes practically zero, thus needing a bigger kernel size. For the standard deviation we chose  $\sigma = 1.0$ , this was experimentally chosen such that it gives enough spread such that the

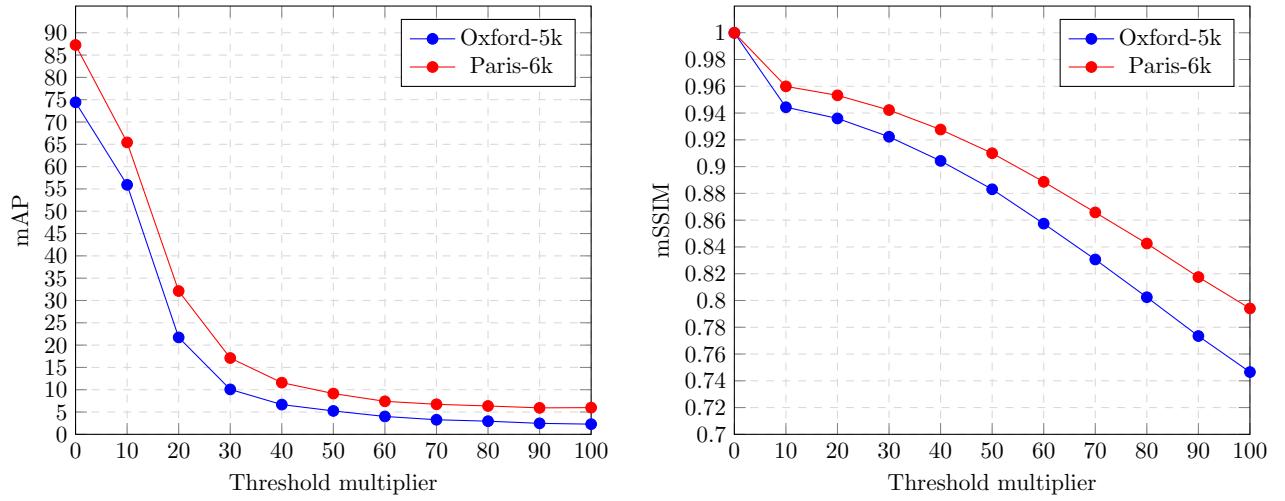


Figure 4.4: Performance (mAP) and perceptual image quality (mSSIM) of neural-feature-based CBIR (GeM [24]) on adversarial queries generated by CV-PIRE from the Oxford-5K [31] and Paris-6k [37] dataset for different thresholds. CV-PIRE used  $n = 5$ ,  $\sigma = 1$  and  $T = 150$  iterations.

pixel color variance is not too localized but it still contains enough detail. For the kernel size we chose  $n = 5$ , since we did not observe any substantial change when testing a bigger kernel size.

#### 4.2.2 Iterations

In figure 4.3 the mAP score of the adversarial queries generated by CV-PIRE on the Oxford-5k [31] dataset at every 10 iterations for different threshold multipliers is visible (see the similar figure A.1 for the Paris-6k [37] dataset). We can see that after 20 iterations the mAP score already looks to converge, it only slightly improves afterwards. Since the convergence of CV-PIRE is dependent on the image queries we chose to use  $T = 100$  iterations to be on the safe side.

#### 4.2.3 Threshold multiplier

Finally, we have to choose the threshold multiplier  $m$ , this is the hyperparameter that scales how big every perturbation can maximally be. As explained in section 3.3.2 we want to choose this threshold multiplier  $m$  such that the retrieved images have the same chance of being the correctly retrieved as randomly choosing images. In practice this means that we choose  $m$  such that the mAP score is similar to an mAP score we would get when randomly retrieving images from the background dataset.

We can estimate this random mAP score by randomly shuffling the ranking of the retrieved images for every image query of the test dataset and taking the average of the resulting mAP scores. To get a good estimate we repeat this process until the random mAP score converges. Using this method the random mAP score for the Oxford-5k[31] dataset is approximately 1.11. For the Paris-6k [37] dataset the random mAP score is approximately 2.69.

In figure 4.4 the mAP and mSSIM score for different thresholds on the Oxford-5K [31] and Paris-6k [37] datasets is shown. We can see that around a threshold multiplier value of 50 the mAP is already near the convergence point, however, when increasing the threshold multiplier the mSSIM score decreases. We chose to use a threshold multiplier of  $m = 50$ . Note that the mAP score is higher than the ideal random mAP score, for Oxford-5k the mAP score for  $m = 50$  is 5.07 instead of the ideal 1.11 and for Paris-6k the mAP is 8.90 instead of the ideal 2.69. Although these mAP scores are bigger they are arguably still effective.

Looking at how the mAP is calculated (equation 4.1) we can make an estimation at how many of the top retrieved images we have to look to get on average one relevant image for a given mAP. Assuming that  $P(k) = 1$ , which is the same as assuming that every retrieved image is a perfect match based on the compact feature vector, and  $\text{corr}(k)$  is only once 1, since we want to have only one relevant retrieved image, we can calculate  $R$  given an  $mAP$ . In the case of  $mAP = 5.07$  (Oxford-5k) this becomes  $R = 19.72$ , meaning that on average only one relevant image will be present in the top 19.72 retrieved images. And in the case of  $mAP = 8.90$  (Paris-6k) this becomes  $R = 11.24$ . Since we assumed  $P(k) = 1$ , which is highly unlikely to happen, the actual  $R$  value will be even higher. Imagine wanting to retrieve images with similar content and only 1 in 11.24 images of the top retrieved images are actually similar. Based on this we argue that these mAP scores are still good enough to consider it an effective adversarial query. However, in edge cases, where for example we have a lot of image queries that we know have similar content, we could extrapolate information from all the retrieved images.

	Oxford-5k		Paris-6k	
	mAP	mSSIM	mAP	mSSIM
Original	74.42	1.00	87.26	1.00
CV-PIRE ( $T = 100$ )	5.07	0.916	8.90	0.911
SD-PIRE ( $T = 100$ )	4.69	0.908	8.70	0.904
PIRE ( $T = 500$ )	4.66	0.793	9.12	0.783

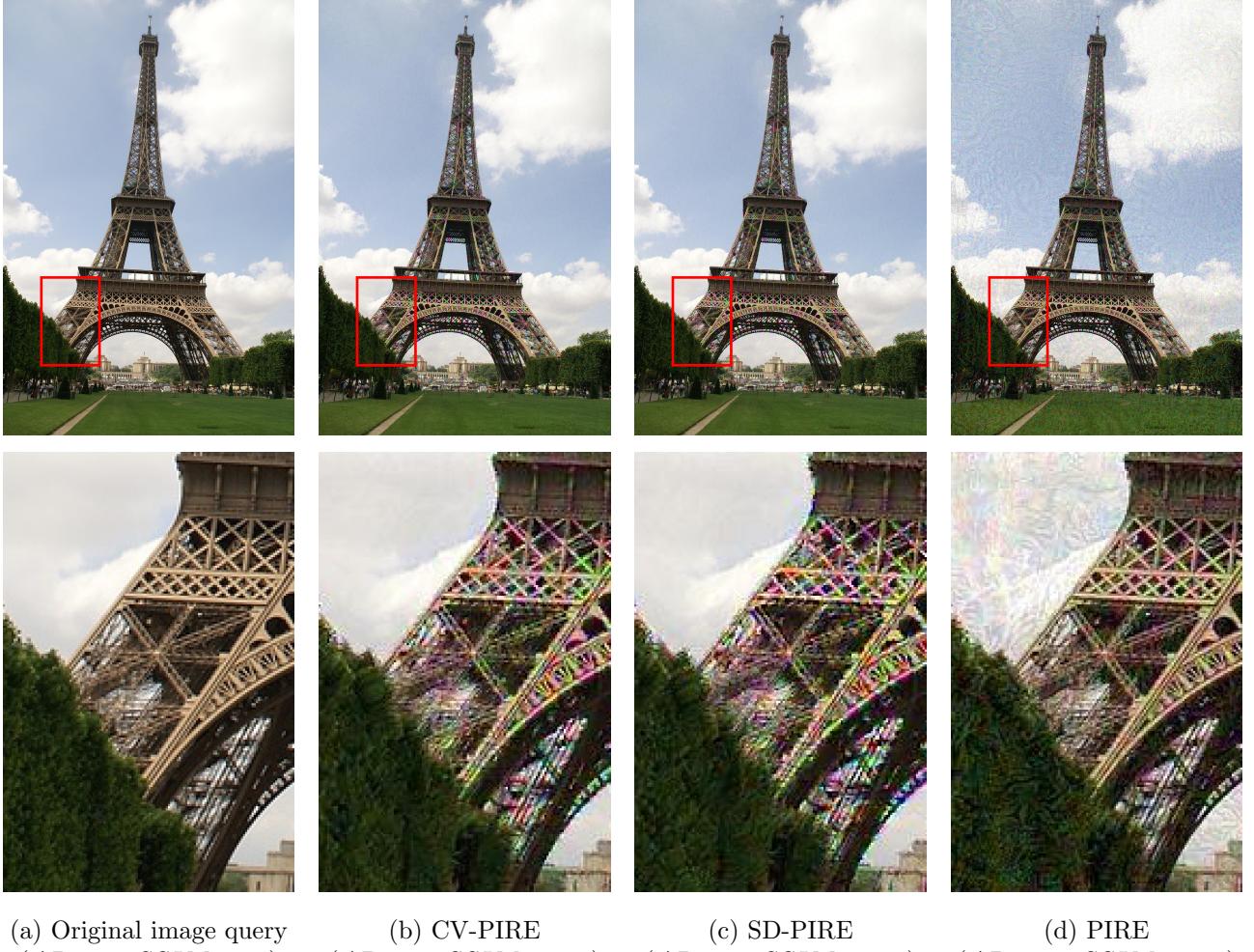
Table 4.1: Performance comparison between CV-PIRE, SD-PIRE and PIRE for the Oxford-5k and Paris-6k datasets. CV-PIRE is using  $m = 50$  and  $n = 5$ , SD-PIRE is using  $m = 50$  and  $n = 3$ .

### 4.3 Results

The final results are shown in table 4.1 and a random sample of image queries in figure 4.5. We can clearly see that CV-PIRE and SD-PIRE outperform PIRE [4] based on the image quality mSSIM score with a similar mAP. The performance of CV-PIRE and SD-PIRE is almost the same. This is not too surprising considering that the methods use comparable techniques to generate adversarial queries and use the same learning rate, iterations and threshold multiplier. The only difference between CV-PIRE and SD-PIRE is in how they determine the pixel threshold. However, this does result in subtle differences that are in favor of both methods, we discuss this in detail in section 4.5.



Figure 4.5: Randomly selected image queries from the Oxford-5k [31] and the Paris-6k [37] datasets. On the left side the original image query is shown, on the right side the corresponding adversarial queries generated by CV-PIRE, SD-PIRE and PIRE.



(a) Original image query  
 $(AP=83, SSIM=1.0)$

(b) CV-PIRE  
 $(AP=16, SSIM=0.95)$

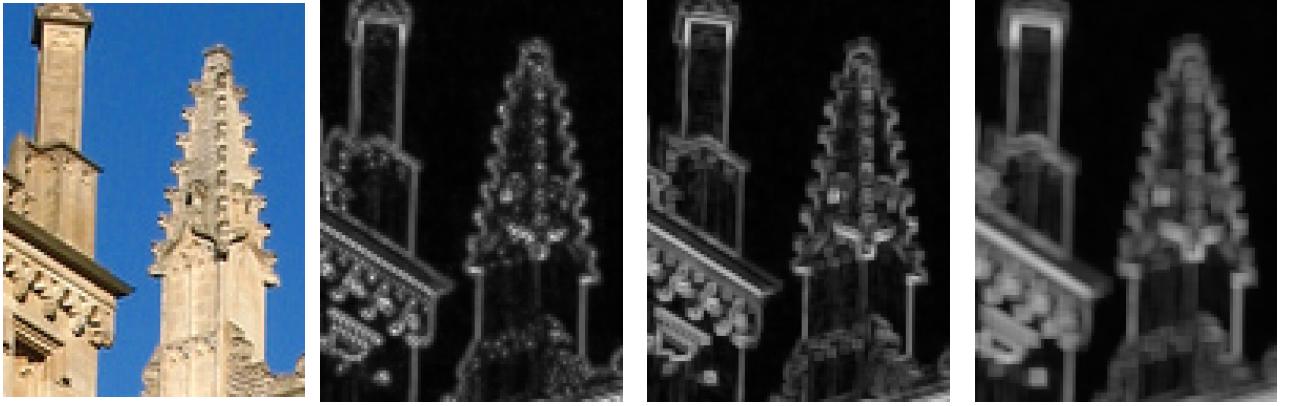
(c) SD-PIRE  
 $(AP=16, SSIM=0.94)$

(d) PIRE  
 $(AP=8.7, SSIM=0.78)$

Figure 4.6: An image query (left) and the corresponding adversarial queries using different methods (right). On top the full size images are shown, on the bottom the 5 times enlarged images at the location of the red rectangle. The image query is hand picked from the Paris-6K [37] dataset. The AP and SSIM scores are based on the full size image.

#### 4.4 Detailed comparison

In figure 4.6 we see adversarial queries generated with different methods. When seeing the 5 times enlarged closeups we can clearly see how the adversarial query using the color variation to determine a threshold for every pixel (CV-PIRE figure 4.6b) is different from a method that uses the same threshold for every pixel (PIRE figure 4.6d). Note how CV-PIRE perturbs the pixels higher color variation regions more (in this case, on the steel beams of the Eiffel tower) while PIRE perturbs all pixels, but the pixels on the steel beams are perturbed to a lesser extent on average. When looking at the full size adversarial queries of CV-PIRE (4.6b) and PIRE (4.6d), we would argue that the perturbations of CV-PIRE are less noticeable than the perturbations of PIRE. The differences we find are most clearly visible in the low color variance regions like the blue sky and the green grass. The differences between CV-PIRE (4.6b) and SD-PIRE (4.6c) are again quite small. More detailed examples are shown in section A.3.



(a) Original image. (b) CV-PIRE ( $n=5, \sigma=1$ ). (c) SD-PIRE ( $n = 3$ ). (d) SD-PIRE ( $n = 5$ ).

Figure 4.7: An 8 times magnified part of an image query from the Oxford-5K [31] dataset and the corresponding pixel variance images.

## 4.5 Comparison between CV-PIRE and SD-PIRE

The main difference between CV-PIRE and SD-PIRE is the way the variance around a pixel is calculated. Where CV-PIRE uses the color difference in combination with a Gaussian kernel to give more importance to pixels that are closer to the source, SD-PIRE calculates the standard deviation around a pixel based on the pixel greyscale values (see section 3.4.1). The resulting difference can be visualised using the pixel variance images as shown in figure 4.7. We see that SD-PIRE (figure 4.7c and 4.7d ) has a more blurry look compared to CV-PIRE (figure 4.7b). This makes sense since SD-PIRE does not use the difference of the pixel values compared to the center pixel, but instead calculates the standard deviation of the whole area. Conversely it is also visible that CV-PIRE has more detail in the pixel variance image, compared to the standard deviation model, even with a bigger kernel size ( $n = 5$ ). This is also why we choose to use the  $n = 3$  kernel size for the SD-PIRE method, since a bigger kernel size reduces the detail too much and would therefore be an unfair comparison, as can be seen in figure 4.7d.

		Oxford-5k		Paris-6k	
	Threshold	mAP	mSSIM	mAP	mSSIM
CV-PIRE	50	5.07	0.916	8.90	0.911
SD-PIRE	50	4.69	0.908	8.70	0.904
	49	4.48	0.909	9.08	0.906
	48	4.63	0.911	9.01	0.908
	47	4.72	0.913	9.13	0.910
	46	4.82	0.916	9.25	0.911
	45	4.84	0.917	9.72	0.913
	44	5.28	0.919	9.76	0.915
	43	5.33	0.921	9.82	0.917
	42	5.90	0.924	10.9	0.920

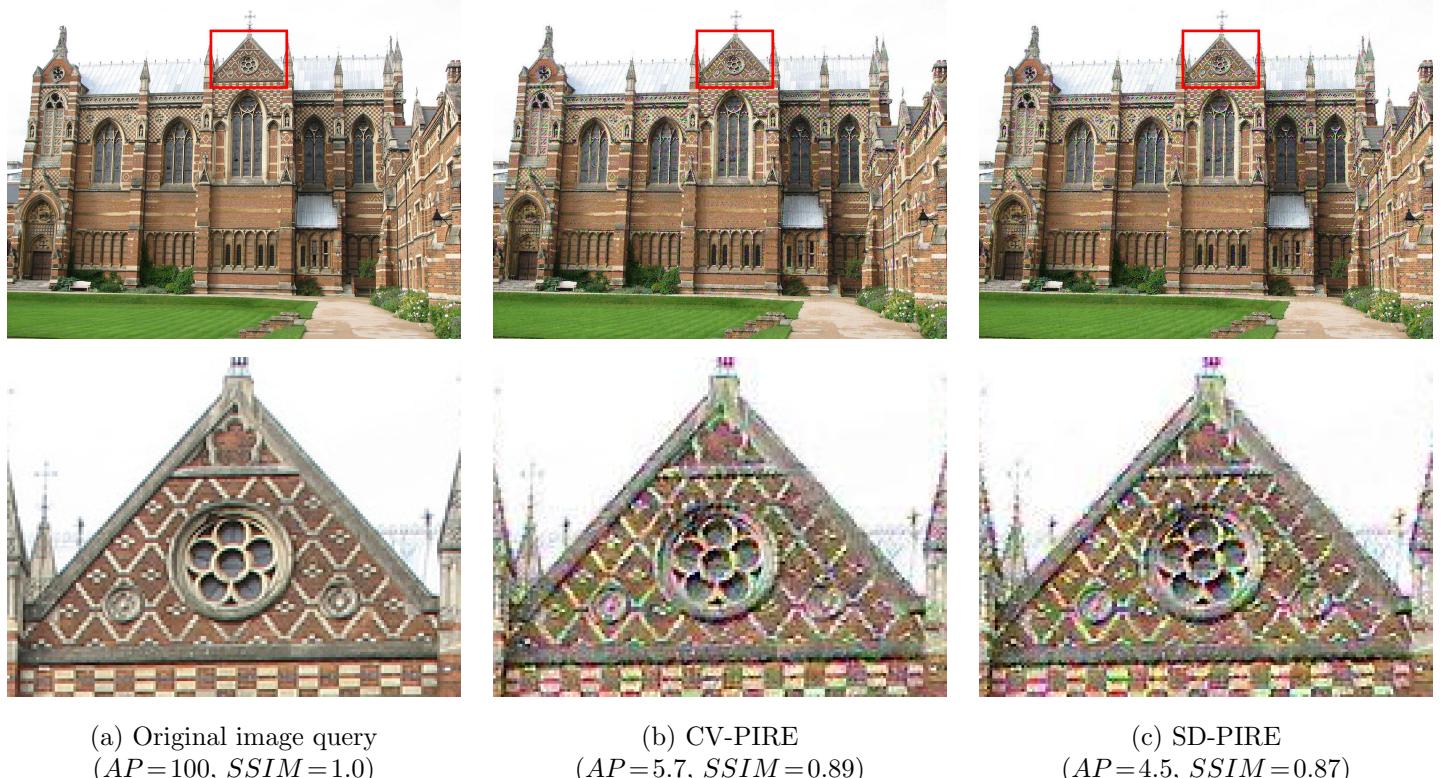
Table 4.2: Performance comparison between CV-PIRE and SD-PIRE for multiple thresholds after  $T = 100$  iterations. The random guessing mAP score for the Oxford-5k dataset is 5.88 and for the Paris-6k dataset is 8.33. CV-PIRE is using  $m = 50$  and  $n = 5$ , SD-PIRE is using  $m = 50$  and  $n = 3$ .

In general we expect that having more detail in the pixel variance image will positively affect the SSIM (image quality) score. However, a downside of having more detail in the pixel variance image is that generally the total pixel variation values are lower, this results in having less overall perturbations which could negatively impact the AP score. When we look at the overall performance of both methods in table 4.1 it is noticeable that indeed CV-PIRE has a bit of a higher mAP score and a bit of a lower mSSIM score compared to SD-PIRE with the same threshold. Since in table 4.1 for the SD-PIRE method for both datasets the mAP value is lower we can look at what happens to the mAP and mSSIM score if we decrease the threshold compared to CV-PIRE. The results are shown in table 4.2. We see that for a similar mAP for the Oxford-5k dataset SD-PIRE has a higher mSSIM score, however, at the same threshold the mAP score for the Paris-6k dataset has now increased more than the mAP for the Oxford-5k dataset has. On the other hand, when SD-PIRE has the same mAP score for the Paris-6k dataset as CV-PIRE the mSSIM is lower than that of CV-PIRE. From this information it is thus inconclusive which one is the better method.



(a) Original image. (b) CV-PIRE ( $n = 5, \sigma = 1$ ). (c) SD-PIRE ( $n = 3$ ).

Figure 4.8: An 8 times magnified part of an image query from the Oxford-5K [31]. CV-PIRE has  $AP = 11.2$  and  $SSIM = 0.933$ , SD-PIRE has  $AP = 12.3$  and  $SSIM = 0.927$  both calculated on the full size image.

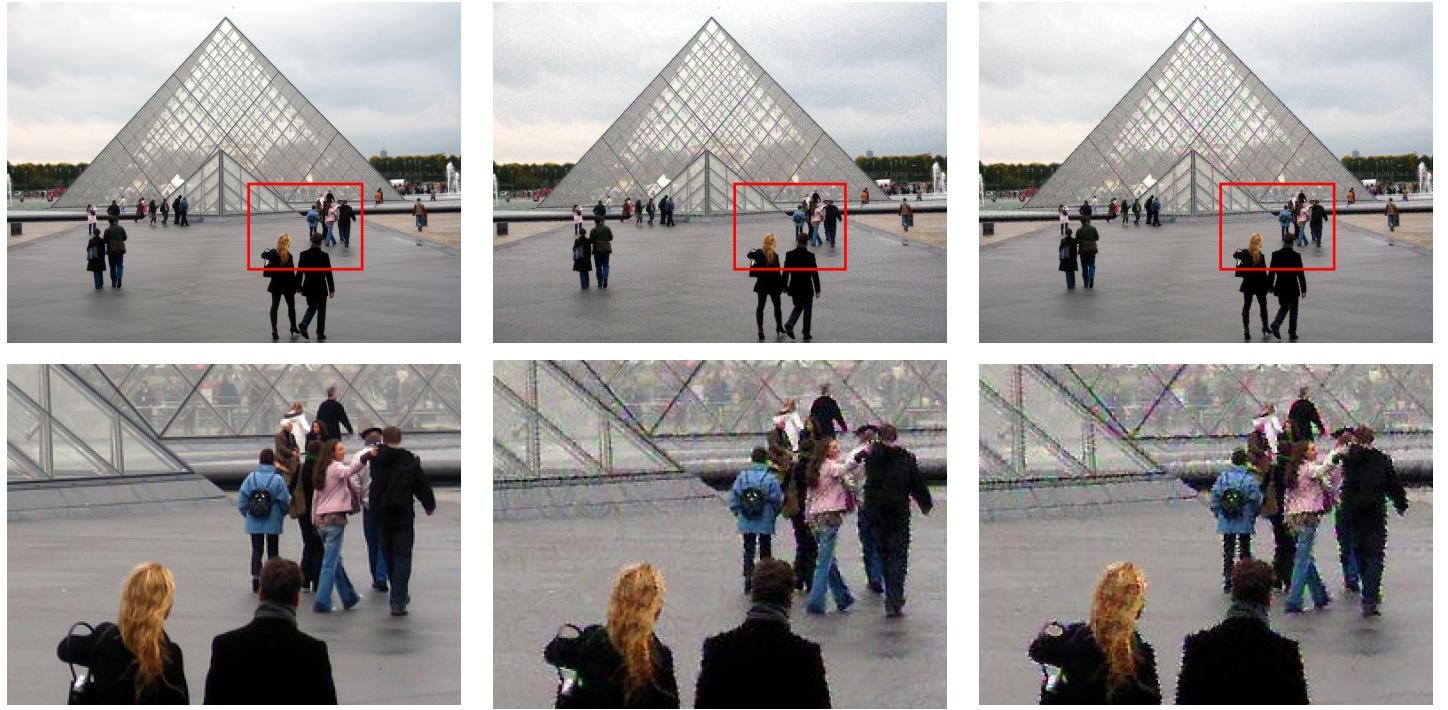


(a) Original image query  
( $AP = 100, SSIM = 1.0$ )

(b) CV-PIRE  
( $AP = 5.7, SSIM = 0.89$ )

(c) SD-PIRE  
( $AP = 4.5, SSIM = 0.87$ )

Figure 4.9: An image query (left) and the corresponding adversarial queries generated by CV-PIRE and SD-PIRE (right). On top the full size images are shown, on the bottom the 6 times enlarged images at the location of the red rectangle. The image query originates from the Oxford-5K [31] dataset. The AP and SSIM scores are based on the full size image.

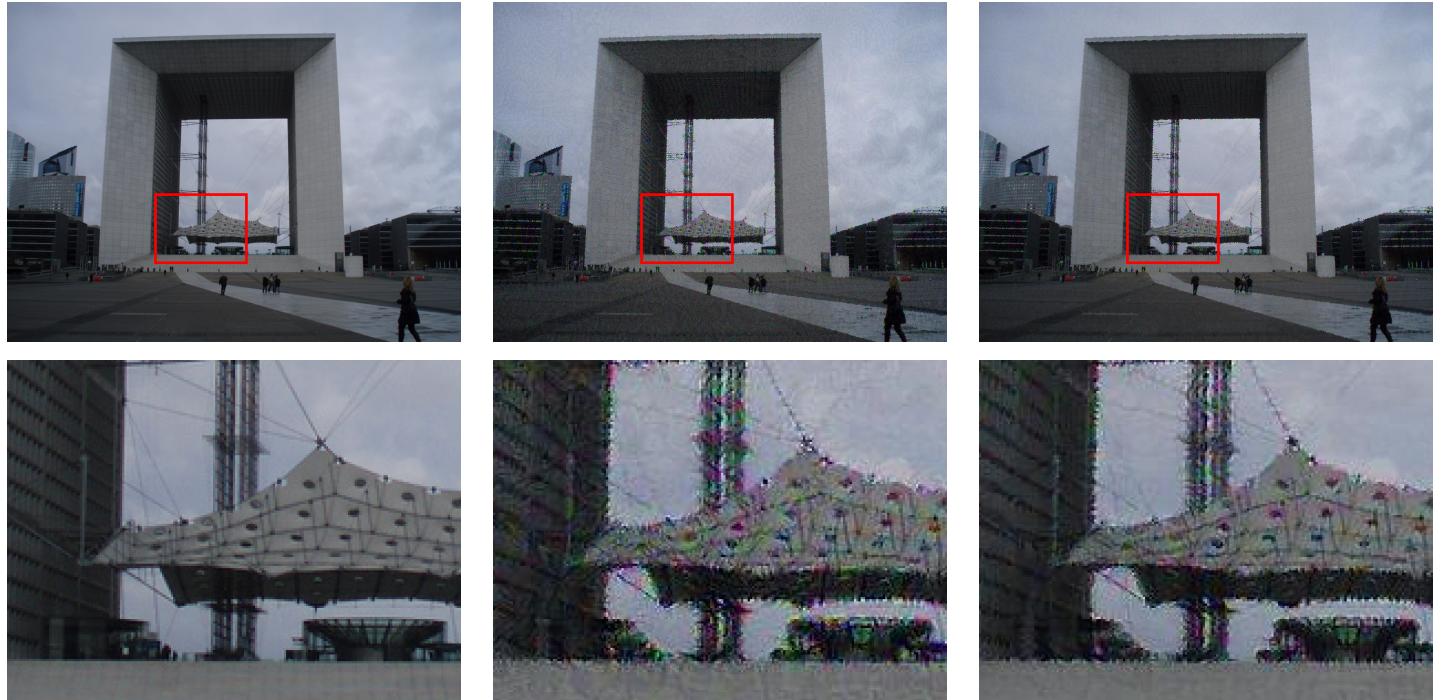


(a) Original image query  
( $AP=37, SSIM=1.0$ )

(b) CV-PIRE  
( $AP=4.9, SSIM=0.91$ )

(c) SD-PIRE  
( $AP=8.3, SSIM=0.92$ )

Figure 4.10: An image query (left) and the corresponding adversarial queries generated by CV-PIRE and SD-PIRE (right). On top the full size images are shown, on the bottom the 4 times enlarged images at the location of the red rectangle. The image query originates from the Paris-6k [37] dataset. The AP and SSIM scores are based on the full size image.



(a) Original image query  
( $AP=85, SSIM=1.0$ )

(b) CV-PIRE  
( $AP=1.4, SSIM=0.88$ )

(c) SD-PIRE  
( $AP=1.5, SSIM=0.94$ )

Figure 4.11: An image query (left) and the corresponding adversarial queries generated by CV-PIRE and SD-PIRE (right). On top the full size images are shown, on the bottom the 5 times enlarged images at the location of the red rectangle. The image query originates from the Paris-6k [37] dataset. The AP and SSIM scores are based on the full size image.

#### 4.5.1 Visual differences

When we look at the adversarial queries generated by CV-PIRE and SD-PIRE we can spot a few differences. In figure 4.8, 4.9, 4.10 and 4.11 some of these differences are shown. These images are cherry-picked, we tried to pick them as objectively as possible. When looking at the whole image in most cases the difference between the adversarial queries generated by CV-PIRE and SD-PIRE were almost indistinguishable. However when we look closer, we can see that in figure 4.9 and 4.10 a little more detail in the edges and shapes is preserved and the perturbations look a little less smeared out. This makes sense when considering the difference in the color variance (see figure 4.7). We found that this was generally the case for practically all images in the Oxford-5k [31] and Paris-6k [37] test sets, with the exception of one image shown in figure 4.11. The adversarial query generated by CV-PIRE looks a bit more noisy compared to the adversarial query generated by SD-PIRE. When looking more closely at the original image a possible explanation can be found, it looks like the original image has some kind of fine noise all over the image, since CV-PIRE is more sensitive to local changes this fine noise increases the calculated color variance and thus the pixel threshold. This results in bigger perturbations all over the image, resulting in a more noisy looking adversarial query.

Thus far, we have compared the image quality using SSIM, however, it is debatable how good of an image quality metric this is, since it does not take accurate human color perception into account. We discuss this further in section 5.2.4.

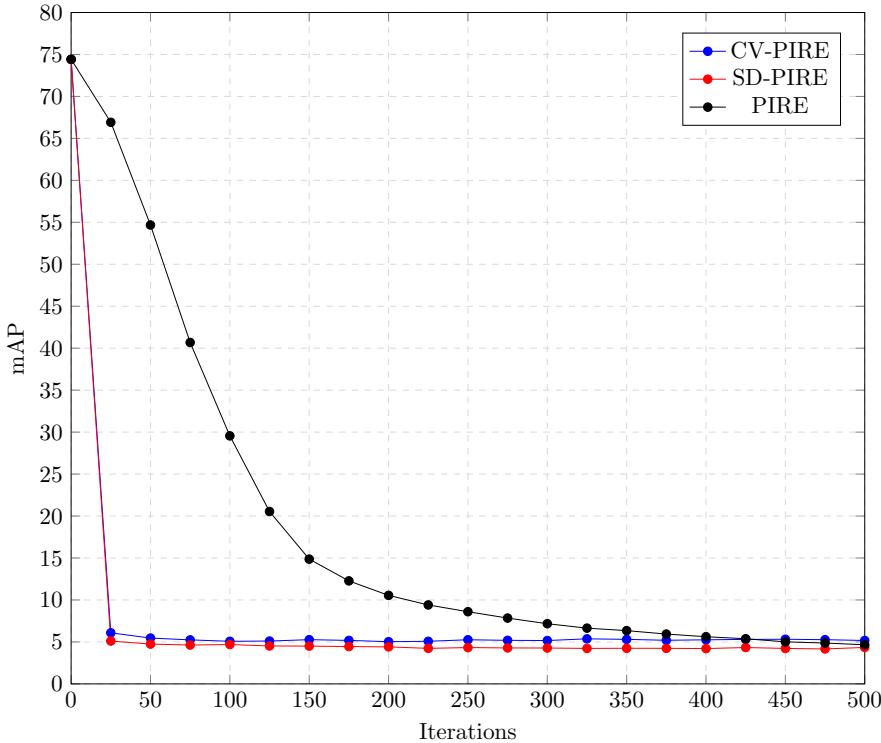


Figure 4.12: Performance (mAP) of neural-feature-based CBIR (GeM [24]) on Oxford-5K [31] dataset for CV-PIRE, SD-PIRE and PIRE. CV-PIRE used  $n = 5$  and  $\sigma = 1$ . SD-PIRE used  $n = 3$ .

## 4.6 Convergence rate

One other benefit of CV-PIRE and SD-PIRE compared to PIRE is the convergence rate as shown in figure 4.12. After only 25 iterations CV-PIRE and SD-PIRE already approached the convergence point. However, PIRE needs 500 iterations to get there. The main reason for this is the difference in the learning rate, the learning rate used for CV-PIRE and SD-PIRE is significantly higher ( $5 \cdot 10^{-2}$ ) than used for PIRE ( $2.5 \cdot 10^{-4}$ ). The reason for this is that if the learning rate is increased too much the image quality of the adversarial queries generated by PIRE suffers quite a bit (for an example see figure A.4). However, with CV-PIRE and SD-PIRE this is less the case. Therefore we chose to use a higher learning rate and  $T = 100$  for the amount of iterations, since the mAP score stabilized at this point. The reason for the possible difference in convergence is that CV-PIRE perturbs the high color variance regions more. These regions might be more important for the CBIR system when determining the retrieval vector. A low color variance region like a blue sky in the image is probably less important for retrieval compared to higher color variance regions such as brick patterns on a building wall. This could be for the simple reason that a lot of images have a blue sky, therefore it is not a good segment of the image to do retrieval on. Therefore, we think that the color variance regions highlight a lot of the retrieval keypoints. Perturbing these keypoints more than lower color variance regions leads to a better adversarial query with less overall perturbations. Since the high color variance regions are usually quite local, the image quality is not affected much when increasing the learning rate. Using a higher learning rate has the advantage that less iterations are needed to have the mAP score converge, saving computing time ( $\pm 100$  seconds per image for  $T = 500$  iterations and  $\pm 20$  seconds per image for  $T = 100$  on a Nvidia Tesla T4 ).

# Chapter 5

## Conclusion

### 5.1 Conclusion

In this research we worked on the problem of generating adversarial queries for CBIR systems such that the perturbations that enable the adversarial effect are less noticeable to the human eye. We proposed a new method called CV-PIRE to create adversarial queries on CBIR systems that makes use of human color perception. This is done by taking the color variation around a pixel into account when determining the maximum amount a pixel can be perturbed. These adversarial queries are thus less perturbed in low color variance regions and more in high color variance regions when compared to previous works. We also proposed a method called SD-PIRE that was inspired by [30] the use of the standard deviation of the pixel values to determine the maximum perturbation of a specific pixel. CV-PIRE and SD-PIRE averages a higher SSIM score than PIRE when tested on a neural based CBIR system on datasets containing buildings. The differences between CV-PIRE and SD-PIRE are much smaller, where CV-PIRE creates adversarial queries with a bit more detail around edges and shapes, but in scenarios where the image query contains fine noise SD-PIRE generates better adversarial queries. Having a higher SSIM score compared to previous works could indicate that the image quality is higher and therefore the perturbations are less noticeable to human perception.

### 5.2 Discussion and outlook

#### 5.2.1 Robustness

Since part of the perturbations done by CV-PIRE are small they could be sensitive to image compression and or filtering, this could mitigate the adversarial effect. However, at the same time the perturbations that are done in high color variance areas are relatively big. Therefore, further research could be done in how robust the adversarial queries created by CV-PIRE are. Further research could also be done in how to change CV-PIRE such that it is less vulnerable to compression and filtering.

#### 5.2.2 Different datasets

In this research we only tested CV-PIRE on datasets containing buildings (Oxford-5k [31] and Paris-6k [37]). It would be interesting to see how CV-PIRE performs on datasets that contain different kinds of settings that could be used to possibly track someone, for example nature landscapes or more iconic content like logos and paintings.

### 5.2.3 Multiple CBIR systems

CV-PIRE generates and evaluates the adversarial queries by using only one neural based CBIR system [24]. It would be interesting to see how well CV-PIRE generalizes when tested on different CBIR systems. CV-PIRE could also be altered such that it generates adversarial queries by making use of multiple CBIR systems to possibly create more general adversarial queries.

### 5.2.4 SSIM and human color perception

Structural similarity (SSIM) [36] is the default method in computer vision research to determine image quality compared to the original unaltered image. It does this by taking luminance, contrast and structure into account. However it does not take human color perception into account. In section 2.5.2 we demonstrated that the influence of human color perception can be quite substantial. In this research we only used SSIM to compare different methods in terms of perceptual image quality, but since SSIM does not take human color perception into account, it possibly makes it a flawed method. This can especially be seen when comparing CV-PIRE and SD-PIRE (section 4.5). There the SSIM score did not always seem to reflect human perception of image quality.

### 5.2.5 CIE94 instead of CIEDE2000

CV-PIRE uses the CIEDE2000 [33] algorithm, however, for large images the computational time to calculate the pixel color variance becomes troubling for practical use. It takes around 4.5 minutes per Oxford-5k [31] dataset image query while utilizing all 8 cores, 16 threads on a CPU. For this reason it could be more beneficial to use the CIE94 [38] algorithm instead when implementing CV-PIRE in a practical application. This algorithm does the same as CIEDE2000 but with a lower perceptual uniformity, but the calculations are less complex and thus faster. The final adversarial queries will probably only differ by a small amount when using CIE94 [38], since it only slightly changes how the pixel color variation is calculated. Since our research was focused on generating the best possible adversarial queries we did not explore how the results would differ when using the CIE94 algorithm.

### 5.2.6 Speeding up the calculations

In this research, the focus was put into creating adversarial queries with the best possible perceived image quality. This, however, resulted in long computation times. To give an impression, 4.5 minutes to calculate the color variation in one image using 8 cores and 16 threads on a modern CPU and 1.5 minutes to generate the adversarial query on a Nvidia T4 GPU. It could be feasible to drastically improve the computation time by using for example CIE94 (as discussed in section 5.2.5) and reducing the amount of iterations. We can see in figure 4.3 that even after  $T = 10$  iterations the mAP is already nearing the convergence point, this is 10 times less than was used in this research. These speed up methods will most likely reduce the perceived image quality, but to what extend could be minimal.

## **Chapter 6**

# **Acknowledgements**

This research was partially conducted on the Dutch national e-infrastructure with the support of SURF Cooperative.

# Bibliography

- [1] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations (ICLR)*, 2015.
- [2] H. Hosseini and R. Poovendran, “Semantic adversarial examples,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1614–1619, IEEE, 2018.
- [3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations (ICLR)*, 2015.
- [4] Z. Liu, Z. Zhao, and M. Larson, “Who’s afraid of adversarial queries? the impact of image modifications on content-based image retrieval,” in *ACM International Conference on Multimedia Retrieval (ICMR)*, ACM, 2019.
- [5] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1625–1634, IEEE, 2018.
- [6] G. Friedland and R. Sommer, “Cybercasing the joint: On the privacy implications of geo-tagging..,” in *USENIX Conference on Hot Topics in Security (HotSec)*, pp. 1–6, USENIX, 2010.
- [7] M. Larson, M. Soleymani, P. Serdyukov, S. Rudinac, C. Wartena, V. Murdock, G. Friedland, R. Ordelman, and G. J. Jones, “Automatic tagging and geotagging in video collections and communities,” in *Proceedings of the 1st ACM international conference on multimedia retrieval*, p. 51, ACM, 2011.
- [8] K. Gurney, *An Introduction to Neural Networks*. Bristol, PA, USA: Taylor & Francis, Inc., 1997.
- [9] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, 2014.
- [10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [11] W. Zhou, H. Li, and Q. Tian, “Recent advance in content-based image retrieval: A literature survey,” *arXiv preprint arXiv:1706.06064*, 2017.
- [12] J. Sivic and A. Zisserman, “Video google: A text retrieval approach to object matching in videos,” in *IEEE International Conference on Computer Vision (ICCV)*, p. 1470, IEEE, 2003.

- [13] Y. Cao, H. Wang, C. Wang, Z. Li, L. Zhang, and L. Zhang, “Mindfinder: interactive sketch-based image search on millions of images,” in *Proceedings of the 18th ACM international conference on Multimedia*, pp. 1605–1608, ACM, 2010.
- [14] B. Wang, Z. Li, M. Li, and W.-Y. Ma, “Large-scale duplicate detection for web image search,” in *2006 IEEE International Conference on Multimedia and Expo*, pp. 353–356, IEEE, 2006.
- [15] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [16] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “Cnn features off-the-shelf: An astounding baseline for recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 806–813, IEEE, 2014.
- [17] L. Zheng, Y. Yang, and Q. Tian, “Sift meets cnn: A decade survey of instance retrieval,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 5, pp. 1224–1244, 2018.
- [18] P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, ACM, 1998.
- [19] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*, pp. 818–833, Springer, 2014.
- [20] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [22] G. Tolias, R. Sicre, and H. Jégou, “Particular object retrieval with integral max-pooling of cnn activations,” in *International Conference on Learning Representations (ICLR)*, 2016.
- [23] L. Zheng, Y. Zhao, S. Wang, J. Wang, and Q. Tian, “Good practice in cnn feature transfer,” *arXiv preprint arXiv:1604.00133*, 2016.
- [24] F. Radenović, G. Tolias, and O. Chum, “Fine-tuning cnn image retrieval with no human annotation,” *IEEE transactions on pattern analysis and machine intelligence*, 2018.
- [25] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *International Conference on Learning Representations (ICLR)*, 2014.
- [26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
- [27] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

- [28] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *stat*, vol. 1050, p. 9, 2017.
- [29] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: A simple and accurate method to fool deep neural networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2574–2582, IEEE, 2016.
- [30] B. Luo, Y. Liu, L. Wei, and Q. Xu, “Towards imperceptible and robust adversarial example attacks against neural networks,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [31] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, “Object retrieval with large vocabularies and fast spatial matching,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, IEEE, 2007.
- [32] A. R. Robertson, “Historical development of cie recommended color difference equations,” *Color Research & Application*, vol. 15, no. 3, pp. 167–170, 1990.
- [33] M. R. Luo, G. Cui, and B. Rigg, “The development of the cie 2000 colour-difference formula: Ciede2000,” *Color Research & Application*, vol. 26, no. 5, pp. 340–350, 2001.
- [34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [35] R. I.-R. BT.709-6, “Parameter values for the hdtv standards for production and international programme exchange,” June 2015.
- [36] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [37] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, “Lost in quantization: Improving particular object retrieval in large scale image databases,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2008.
- [38] R. McDonald and K. J. Smith, “Cie94-a new colour-difference formula,” *Journal of the Society of Dyers and Colourists*, vol. 111, no. 12, pp. 376–379, 1995.

# Appendix A

## Appendix

### A.1 SSIM formula

Structural similarity (SSIM) [36] is a method for quantifying perceived image quality by measuring the similarity between two images. The SSIM score is calculated by taking multiple windows of both images. We are measuring the SSIM between two windows  $x$  and  $y$  with size  $N \times N$ , where these windows are on the same location of the two images. SSIM takes luminance  $l$ , contrast  $c$  and structure  $s$  into account. The functions for these parts are given by:

$$l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \quad (\text{A.1})$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \quad (\text{A.2})$$

$$s(x, y) = \frac{\sigma_{xy} + \frac{c_2}{2}}{\sigma_x\sigma_y + \frac{c_2}{2}} \quad (\text{A.3})$$

With:

- $\mu_x$  the average of  $x$
- $\mu_y$  the average of  $y$
- $\sigma_x$  the covariance of  $x$
- $\sigma_y$  the covariance of  $y$
- $\sigma_x^2$  the variance of  $x$
- $\sigma_y^2$  the variance of  $y$
- $\sigma_{xy}$  the covariance of  $x$  and  $y$
- $c_1$  and  $c_2$  a variable defined by  $c_1 = (k_1 L)^2$  and  $c_2 = (k_2 L)^2$ , with  $L$  the dynamic range and  $k_1, k_2$  constants (by default  $k_1 = 0.01$  and  $k_2 = 0.03$ )

These parts can be combined using a weighted combination to get the SSIM score:

$$\text{SSIM}(x, y) = [l(x, y)^\alpha \cdot c(x, y)^\beta \cdot s(x, y)^\gamma] \quad (\text{A.4})$$

Where the weights  $\alpha$ ,  $\beta$  and  $\gamma$  are by default all set to 1. Finally, the mean is taken of all the windows to get the final SSIM score.

## A.2 mAP compared to iterations for CV-PIRE

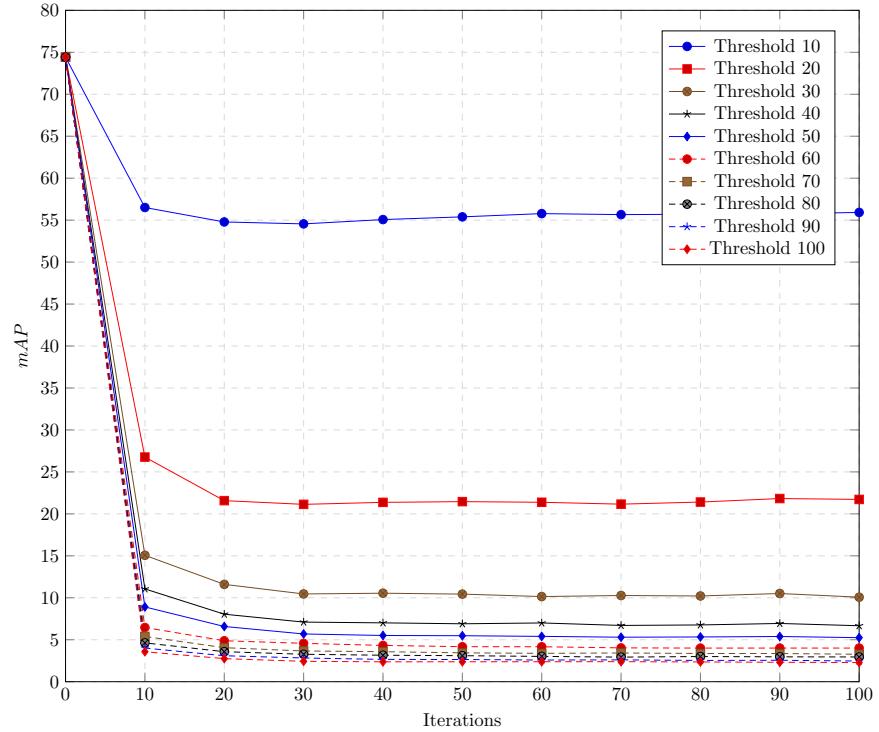


Figure A.1: Performance (mAP) of neural-feature-based CBIR (GeM [24]) on adversarial queries generated by CV-PIRE from the Paris-6K [37] data set for different thresholds and iterations. CV-PIRE used  $n = 5$  and  $\sigma = 1$ .

### A.3 More detailed results

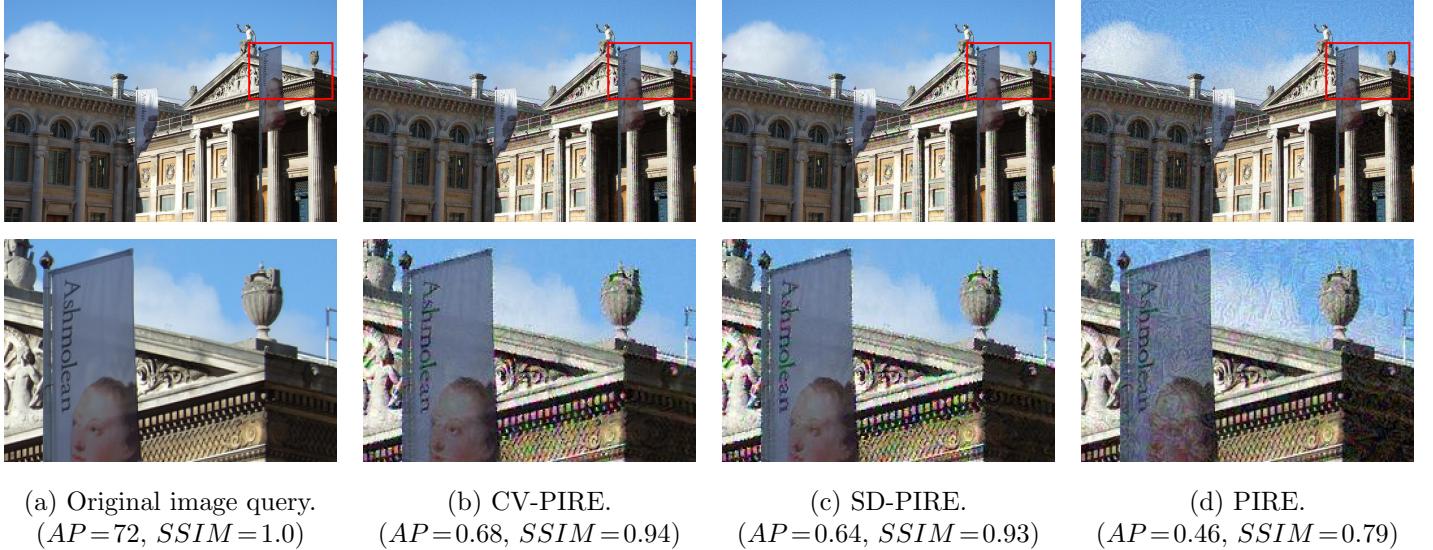


Figure A.2: An image query (left) and the corresponding adversarial queries using different methods (right). On top the full size images are shown, on the bottom 4 times enlarged images at the location of the red rectangle. The image query originates from the Oxford-5K [31] dataset. The  $AP$  and  $SSIM$  scores are based on the whole image.

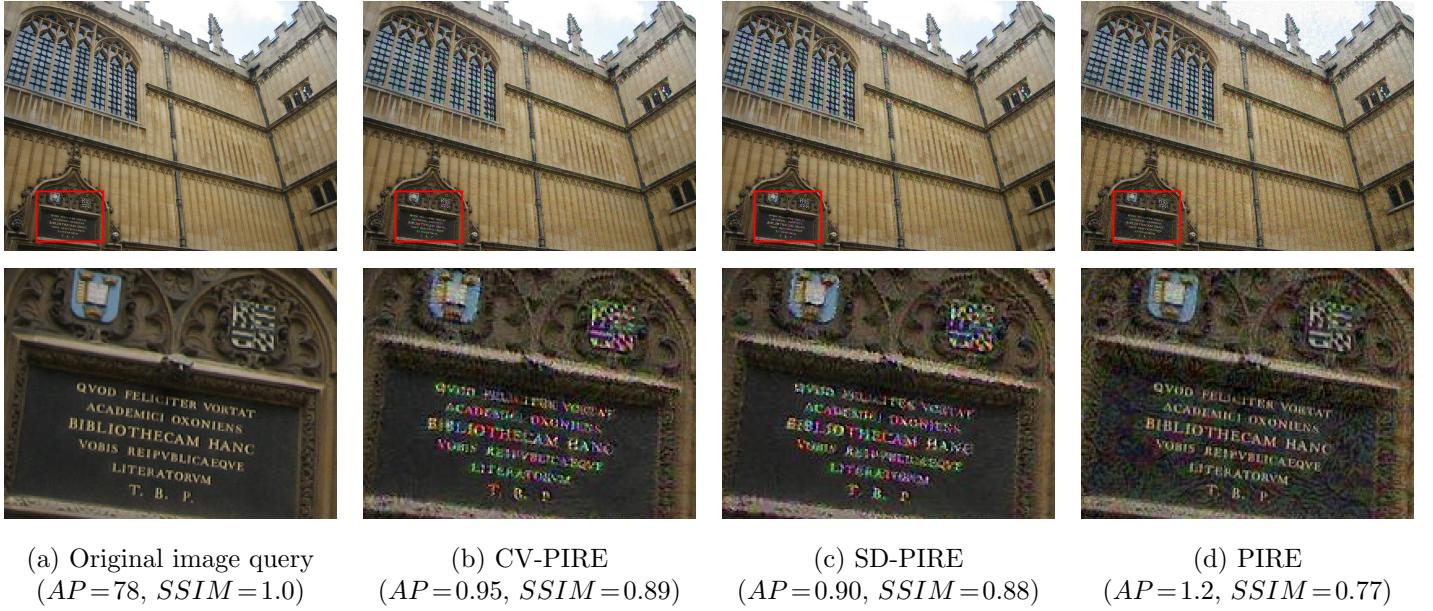
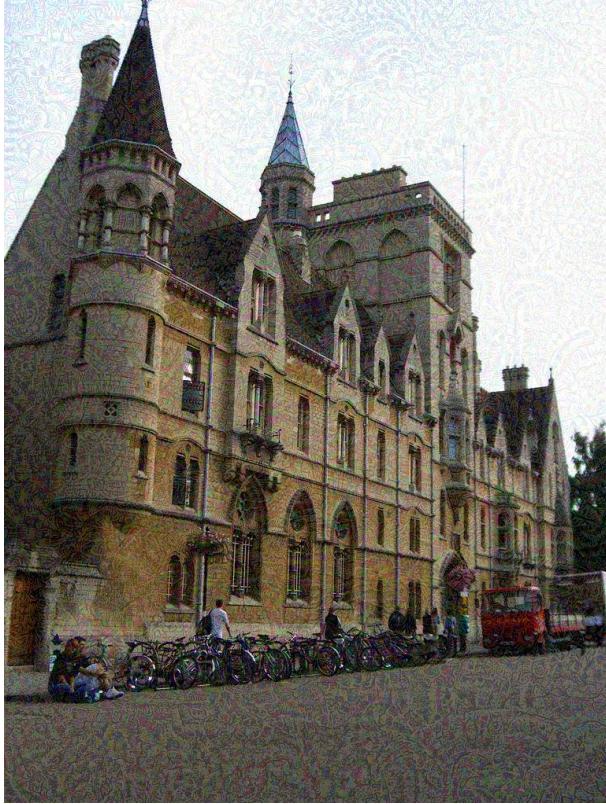
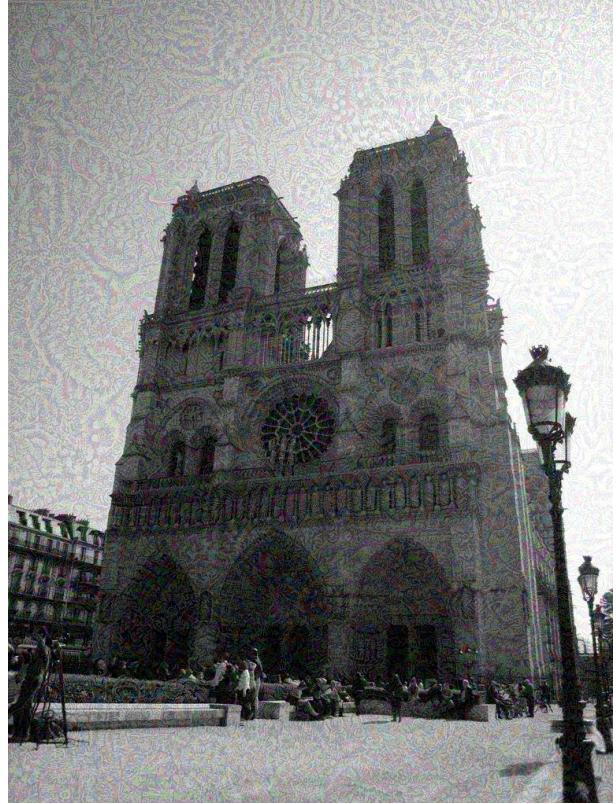


Figure A.3: An image query (left) and the corresponding adversarial queries using different methods (right). On top the full size images are shown, on the bottom the 5 times enlarged images at the location of the red rectangle. The image query originates from the Oxford-5k [31] dataset. The  $AP$  and  $SSIM$  scores are based on the full size image. The text is readable in the original image and in the adversarial query generated by PIRE, however in the adversarial queries generated by CV-PIRE and SD-PIRE the text in the image substantially harder to read.

## A.4 PIRE with a higher learning rate



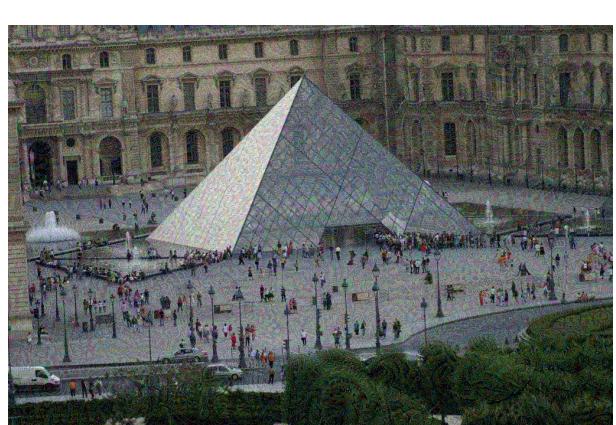
(a) ( $AP=0.29$ ,  $SSIM=0.69$ )



(b) ( $AP=4.2$ ,  $SSIM=0.62$ )



(c) ( $AP=0.13$ ,  $SSIM=0.59$ )



(d) ( $AP=2.5$ ,  $SSIM=0.65$ )

Figure A.4: Four adversarial queries generated by PIRE [4] with a learning rate of  $5 * 10^{-2}$  (the same learning rate used by CV-PIRE and SD-PIRE) after  $T = 500$  iterations. The left two images were randomly sampled from the Oxford-5k [31] dataset and the right two images were randomly sampled from the Paris-6k [37] dataset.