Geometric robustness of deep networks: analysis and improvement

Can Kanbak, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard
École Polytechnique Fédérale de Lausanne
{can.kanbak, seyed.moosavi, pascal.frossard} @epfl.ch

Abstract

Deep convolutional neural networks have been shown to be vulnerable to arbitrary geometric transformations. However, there is no systematic method to measure the invariance properties of deep networks to such transformations. We propose ManiFool as a simple yet scalable algorithm to measure the invariance of deep networks. In particular, our algorithm measures the robustness of deep networks to geometric transformations in a worst-case regime as they can be problematic for sensitive applications. Our extensive experimental results show that ManiFool can be used to measure the invariance of fairly complex networks on high dimensional datasets and these values can be used for analyzing the reasons for it. Furthermore, we build on ManiFool to propose a new adversarial training scheme and we show its effectiveness on improving the invariance properties of deep neural networks.[1]

1. Introduction

Although convolutional neural networks (CNNs) have been largely successful in various applications, they have been shown to be quite vulnerable to additive adversarial perturbations [25, 10, 18] which can negatively affect their applicability in sensitive applications such as autonomous driving [6]. Deep networks have also been shown to be vulnerable to rigid geometric transformations [7, 9], which are more natural than additive perturbations: they can simply represent the change of the viewpoint of an image. Therefore, invariance to such transformations is certainly a key feature in practical vision systems. In this paper, we focus on studying the robustness of deep networks to geometric transformations in the worst-case regime as these can be quite problematic for sensitive applications. We approach this problem by searching for minimal 'fooling' transformations, *i.e.*, transformations that change the decision of image classifiers, and we use these transformed examples to measure the invariance of a deep network. We further show that fine-tuning on such worst-case transformed examples can improve the invariance properties of deep image classifiers. Our main contributions are as follows:

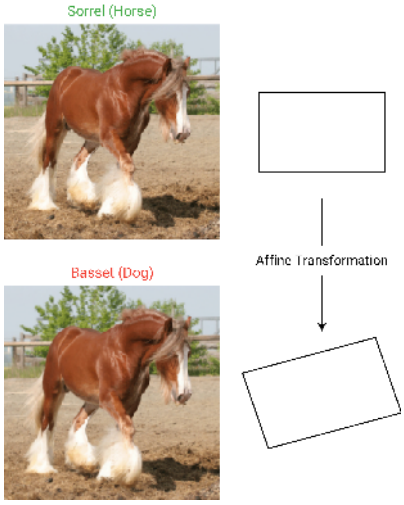


Figure 1: An example of a worst-case 'fooling' affine transform for AlexNet [15]. While the image on top is correctly classified as sorrel (a type of horse), a small transformation in the bottom image can cause it to be classified as basset (a type of dog), even though the change in the image is imperceptible.

- We propose a scalable algorithm, ManiFool, for finding small worst-case transformations and define a measure to compare the invariance properties of different networks.
- As far as we know, we perform the first quantitative study on the robustness of deep networks to geometric transformations that are trained on a large scale dataset, *i.e.*, ImageNet, and show that these networks are susceptible to small and sometimes imperceptible *transformations*.
- We use the ManiFool algorithm to perform adversarial training using geometric transformations and show that it actually improves the invariance of deep networks.

[1] To encourage reproducible research, the code of our method will be later published.

The adversarial examples are first introduced in [25]. Since then, many methods to find additive adversarial perturbations have been proposed such as [18, 10, 4]. Other types of adversarial examples are later found in [3, 20]. The work [25] also introduces the concept of adversarial training to increase the accuracy of networks. The authors in [10] later show that adversarial training can also be used for increasing the robustness of networks against adversarial examples constructed by additive perturbations.

The vulnerability of CNNs against geometric transformations, on the other hand, has been studied in [17] and [24] that analyze image and visual representations to find theoretical foundations of transformation invariant features. The work in [2] uses the information about human visual system to understand and improve the transformation invariance. In addition, several practical solutions have been suggested for improving the invariance characteristics. One approach is to modify the layers of the networks; *e.g.*, pooling layer[5] or convolutional layers [22]. Another method is to add modules to the network, like the spatial transformer networks [12]. Even though these works focus on improving the invariance, they do not offer methods for measuring invariance properties of classification architectures. This problem is the main focus of [9], where the invariance is measured by using the firing rates of neurons in the network for one dimensional transformations. On the other hand, the authors of [8] propose a probabilistic framework for estimating the robustness of a classifier by using a Metropolis algorithm to sample the set of transformations. Lastly, another approach is given by Manitest [7], where the invariance is measured using the geodesic distances on the manifold of transformed images. In this work, we also use a manifold-based definition of invariance and propose a new scalable algorithm for evaluating invariance in more complex networks and improving it by fine-tuning.

2. Preliminaries

In this section, we briefly introduce the mathematical tools that we will use to measure the robustness of deep networks to geometric transformations.

Let $\mathcal{T}$ be a Lie group of geometric transformations such as rotations or projective transformations. Then, let $\tau \in \mathcal{T}$ be a function $\tau : \mathbb{R}^2 \to \mathbb{R}^2$ in this group. For 2D images, τ can be seen as a bijection that maps the points of an image to the points of another image. More precisely, let an image I be defined as a square integrable function $I : \mathbb{R}^2 \to \mathbb{R}$. The action of τ on I can be represented as $I_\tau(x, y)$. This can also be seen as a function that maps the Lie group $\mathcal{T}$ to the image space, which we denote by $\psi^{(I)}(\tau) : \mathcal{T} \to L^2$, where L^2 is the space of square integrable functions. A transformation τ can be represented by as many parameters as the dimensionality of $\mathcal{T}$. For example, the rotation angle can be used to parameterize the rotation group. These parameters can be grouped in a vector $\boldsymbol{\theta}$, where each element represents one of the parameters of τ.

For a given Lie group $\mathcal{T}$, we need to define a metric $d(\tau_1, \tau_2) : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ to measure the actual effect of transformations on images. A naive metric would consist in measuring the ℓ^2 distance between the parameter vectors of τ_1 and τ_2. This however is not a useful metric since it does not take into account the different nature of the transformation parameters such as rotation angle or scale. The metric should rather depend on the image as well as the transformations. However, another metric such as the squared L^2 distance $d_I(\tau_1, \tau_2) = \|I_{\tau_1} - I_{\tau_2}\|^2_{L^2}$ still does not fully capture the properties of the transformation even if it depends on the image(see [7] for an illustrative example). Thus, a better metric should be able to capture the intrinsic geometric structure of the transformed images.

One such metric is to the length of the shortest curve between $\tau_1, \tau_2 \in \mathcal{T}$, *i.e.*, the *geodesic distance*. This metric, however, requires a Riemannian metric to be defined for $\mathcal{T}$. In this case, the Riemannian metric can be acquired by mapping $\mathcal{T}$ to the set of transformed versions of image I, *i.e.*, $\mathcal{M}(I) = \{I_\tau : \tau \in \mathcal{T}\}$. This set forms a differentiable manifold called the image appearance manifold (IAM) following the works of [27, 13] and it inherits a Riemannian metric from its ambient space, L^2. From [7], it can be seen that the Riemannian metric on $\mathcal{T}$ can be chosen accordingly, such that the length of a curve on $\mathcal{T}$, $\gamma(t) : [0, 1] \to \mathcal{T}$, will be equal to the length of the mapped curve on $\mathcal{M}(I)$, $I_{\gamma(t)} : [0, 1] \to \mathcal{M}(I)$. Thus, the geodesic distance between $\tau_1, \tau_2 \in \mathcal{T}$ is equal to the geodesic distance between $I_{\tau_1}, I_{\tau_2} \in \mathcal{M}(I)$. Then, for $\tau_1, \tau_2 \in \mathcal{T}$, the transformation metric can be finally defined as

$$d_I(\tau_1, \tau_2) = \min_{\gamma:[0,1]\to\mathcal{M}(I)} L(\gamma) \quad \text{s.t.}\, \gamma(0) = I_{\tau_1}, \gamma(1) = I_{\tau_2}, \tag{1}$$

where $L(\gamma)$ is the length of the curve γ. This metric both depends on the transformed image and also takes into account the geometric properties of the transformation set $\mathcal{T}$. In order to measure the action of a transformation τ, we therefore use the distance in Eq. (1) –between the transformation and the identity transformation e, *i.e.*, $d_I(e, \tau)$. To compare the effect of transformations on different images, we normalize $d_I(e, \tau)$ by the norm the image, that is

$$\tilde{d}_I(e, \tau) = \frac{d_I(e, \tau)}{\|I\|_{L^2}}. \tag{2}$$

3. Robustness to geometric transformations

As we have now chosen our metric as (2), our approach to measure robustness of classifiers can be formalized as follows. Let k be the given classifier, I an image and $\mathcal{T}$ the set of transformations we are interested in. As with [7],

we define the invariance metric as the minimal normalized distance between the identity transformation and a transformation that leads to misclassification. Hence, the invariance measure is denoted as

$$\Delta_{\mathcal{T}}(I,k) = \min_{\tau \in \mathcal{T}} \tilde{d}(e,\tau) \quad \text{subject to} \quad k(I) \neq k(I_\tau), \quad (3)$$

Note that this definition is similar to those used in works on adversarial perturbations such as [18][9]. However, in our case, instead of looking at minimal fooling additive perturbations, we are interested in the minimal or worst-case geometric transformation. Thus, for a probability distribution μ on the set of images, the global invariance score of the classifier k to transformations in $\mathcal{T}$ is defined as

$$\rho_{\mathcal{T}}(k) = \mathbb{E}_{I\sim\mu} \Delta_{\mathcal{T}}(I,k). \quad (4)$$

As the underlying probability distribution of the images is generally unknown, the invariance score of the classifier is calculated using the empirical average of the estimated invariance over a set of images:

$$\hat{\rho}_{\mathcal{T}}(k) = \frac{1}{m} \sum_{j=1}^{m} \tilde{d}_{I_j}(e,\hat{\tau}). \quad (5)$$

One can also define the invariance of a classifier to a random transformation by again using the metric $\tilde{d}_I(e,\tau)$. In this case, we use the probability of misclassification of transformed images under random transformations with given geodesic scores to measure the invariance. This can be defined it as

$$r_{\mathcal{T}}(k) = \min r \text{ s.t. } \mathop{\mathbb{P}}_{I,\tau}(k(I_\tau) \neq k(I) \mid d_I(e,\tau) = r) \geq 0.5. \quad (6)$$

Note that 0.5 is chosen as a threshold here, but other thresholds can be used for defining the invariance to random transformations.

In practice, $\hat{\rho}_{\mathcal{T}}(k)$ in (5) is computed using the algorithm described in Section 4 to find a small transformation $\hat{\tau}$ that can fool the image and using the geodesic distance of this transformation, $\tilde{d}_I(e,\hat{\tau})$, to estimate $\Delta_{\mathcal{T}}(I,k)$. The expectation can then be calculated using multiple images sampled from a dataset. On the other hand, the estimation of $r_{\mathcal{T}}$, $\hat{r}_{\mathcal{T}}(k)$, is computed by sampling the set $\mathcal{M}_r = \left\{ I_\tau : \tau \in \mathcal{T},\ \tilde{d}_I(e,\tau) = r \right\}$ for increasing r and computing the misclassification percentage of the corresponding transformed samples.

4. ManiFool

In this section, we first introduce the ManiFool algorithm to find a small fooling transformation $\hat{\tau}$ for binary and multiclass classifiers. We then present a method to measure the geodesic distance $\tilde{d}_{I_j}(e,\hat{\tau})$ to compute the invariance score in (5). The main idea of the ManiFool algorithm is simply to iteratively move from an image sample towards the decision boundary of the classifier where the classification decision changes, while staying on the transformation manifold. Each iteration is then composed of two steps: choosing the movement direction and mapping this movement onto the manifold. The iterations will continue until the algorithm reaches the decision boundary and finds a fooling transformation example. An illustration of the algorithm can be found in Figure 2 and a more detailed description of the algorithm is given in the following section.

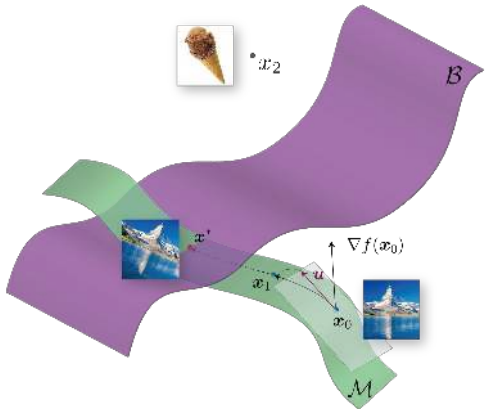


Figure 2: Illustration of Algorithm 1. Assume $\mathcal{M}$ is the manifold of transformed images for the input $\boldsymbol{x}_0$ and $\mathcal{B}$ is the decision boundary of classifier f. The algorithm iteratively moves towards the decision boundary. The first iteration is shown where the movement direction $\boldsymbol{u}$ is found by projecting ∇f to the tangential space of $\mathcal{M}$ and the next image $\boldsymbol{x}_1$ is found by mapping u back onto the manifold

4.1. ManiFool for binary classifiers

Since a multiclass classifier can be thought as a combination of multiple binary classifiers, we first start from the binary classifier case. We consider from now on discrete versions of images, denoted as $\boldsymbol{x} \in \mathbb{R}^n$ where n is the number of pixels of $\boldsymbol{x}$. The binary classifier is defined as $k(x) = \text{sign}(f(x))$, where $f : R^n \to \mathbb{R}$ is an arbitrary differentiable classification function. Without loss of generality, we can assume that the original label of the image is 1.

Let $\boldsymbol{x}^{(i)}$ be the image at the start of iteration i and let iterations start with $i = 0$. The first step for any iteration consists in finding the movement direction. Since we assumed that the original label is 1, $f(\boldsymbol{x}^{(0)}) > 0$ for the input image $\boldsymbol{x}^{(0)}$. Thus, to reach the decision boundary where $f(\boldsymbol{x}) = 0$ while following the shortest path, we need to choose the direction which maximally reduces $f(\boldsymbol{x})$, which is the opposite of the gradient of f, $-\nabla f(\boldsymbol{x})$. However, since we want to stay on the set of transformed images, we restrict the classifier to the image appearance manifold,

Algorithm 1 ManiFool for binary classifiers

Input: Image $\boldsymbol{x}$, classifier f
Output: Transformation $\hat{\tau}$

Initialize with $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $f(\boldsymbol{x}^{(i)}) > 0$ **do**
  $\hat{\boldsymbol{u}} \leftarrow -\boldsymbol{J}^{+}_{\boldsymbol{x}^{(i)}} \nabla f(\boldsymbol{x}^{(i)})$
  $\boldsymbol{u}^{(i)} \leftarrow \lambda_i \frac{\hat{\boldsymbol{u}}}{\|\hat{\boldsymbol{u}}_{l_n}\|} + \gamma \boldsymbol{u}^{(i-1)}$
  $\tau_i \leftarrow \exp\left(\sum_j u_j G_j\right)$
  $\boldsymbol{x}^{(i+1)} \leftarrow \boldsymbol{x}^{(i)}_{\tau_i}$
  $i \leftarrow i + 1$
end while
return $\hat{\tau} = \tau_0 \circ \tau_1 \circ \dots \tau_i$

$\mathcal{M}(\boldsymbol{x})$, as $f_{|\mathcal{M}} : \mathcal{M}(\boldsymbol{x}) \rightarrow \mathbb{R}$, and use its gradient $\nabla f_{|\mathcal{M}}$. At point $\boldsymbol{x}^{(i)}$, this gradient can be acquired simply by projecting $\nabla f(\boldsymbol{x}^{(i)})$ onto the tangent space of $\boldsymbol{x}^{(i)}$, $T_{\boldsymbol{x}^{(i)}}\mathcal{M}$ [1]. This projection is done using the pseudoinverse operator as

$$\boldsymbol{u} = -\boldsymbol{J}^{+}_{\boldsymbol{x}^{(i)}} \nabla f(\boldsymbol{x}^{(i)}) = -(\boldsymbol{J}^{T}_{\boldsymbol{x}^{(i)}} \boldsymbol{J}_{\boldsymbol{x}^{(i)}})^{-1} \boldsymbol{J}^{T}_{\boldsymbol{x}^{(i)}} \nabla f(\boldsymbol{x}^{(i)}). \tag{7}$$

where $\boldsymbol{J}_{\boldsymbol{x}^{(i)}}$ is the Jacobian matrix, whose columns form the basis of the tangential space and $\boldsymbol{u} \in T_{\boldsymbol{x}^{(i)}}\mathcal{M}$ is the projection of $\nabla f(\boldsymbol{x}^{(i)})$, *i.e.*, our movement direction for this iteration.

After finding the movement direction, the next step is to map $\boldsymbol{u}$ onto $\mathcal{M}(\boldsymbol{x})$. This step depends heavily on the transformation set. As we want to minimize the geodesic distance, the natural choice of mapping would be to use the exponential map for $\mathcal{M}(\boldsymbol{x})$ since it follows the geodesics [26]. If an exponential map is readily available for $\mathcal{M}$ and does not have high computational complexity, it can be used. However, for most transformation sets, this does not hold and a retraction is used instead. Here, we will talk about one such retraction for the set of projective transformations and its subsets. Different retractions can be defined for other Lie groups.

The retraction in our implementation uses the matrix representation of projective transformations, where the matrix exponential forms a map from $T_e\mathcal{T}$ to $\mathcal{T}$. Let $\boldsymbol{y} \in \mathcal{M}(\boldsymbol{x})$, $\boldsymbol{u} \in T_{\boldsymbol{y}}\mathcal{M}(\boldsymbol{x})$ and G_i be the basis of $T_e\mathcal{T}$, which are also called the generators of $\mathcal{T}$. The retraction we use can be summarized as mapping $\boldsymbol{u}$ to $T_e\mathcal{T}$ by using the generators, then mapping it to the matrix Lie group $\mathcal{T}$ and lastly, mapping it back to $\mathcal{M}(x)$ by using $\psi^{\boldsymbol{x}^{(i)}}$. More formally, the retraction at the point $\boldsymbol{y}$, $R_{\boldsymbol{y}} : T_{\boldsymbol{y}}\mathcal{M} \rightarrow \mathcal{M}$ can be written as

$$R_{\boldsymbol{y}}(\boldsymbol{u}) = \psi^{(\boldsymbol{y})}\left(\exp\left(\sum_j u_j G_j\right)\right). \tag{8}$$

The image for the next iteration of the algorithm is thus written as

$$\boldsymbol{x}^{(i+1)} = R_{\boldsymbol{x}^{(i)}}(\boldsymbol{u}) = \boldsymbol{x}^{(i)}_{\tau_i}, \tag{9}$$

where τ_i is the transformation represented by

$$\tau_i = \exp\left(\sum_j u_j G_j\right). \tag{10}$$

Lastly, the label of the generated image is checked. If $k(\boldsymbol{x}^{(i+1)}) = 1$, the algorithm continues with the next iteration, this time starting from $\boldsymbol{x}^{(i+1)}$. Otherwise, the algorithm has finished successfully and the transformation that generated $\boldsymbol{x}^{(i+1)}$ is found as

$$\hat{\tau} = \tau_0 \circ \tau_1 \circ \dots \tau_i. \tag{11}$$

The algorithm is summarized on Algorithm 1. Overall, it should be noted that our algorithm is closely related to manifold optimization techniques, particularly to line-search methods. The convergence analysis of such methods can be found for example in [1]. Using this analogy, choosing the movement direction has changed by including line-search and momentum terms, since it has been seen empirically that they improve the accuracy and reduce the chance of converging to a local minimum. This new direction term can be written as

$$\boldsymbol{u}^{(i)} = -\lambda_i \frac{\boldsymbol{J}^{+}_{\boldsymbol{x}^{(i)}} \nabla f(\boldsymbol{x}^{(i)})}{\|\boldsymbol{J}^{+}_{\boldsymbol{x}^{(i)}} \nabla f(\boldsymbol{x}^{(i)})\|} + \gamma \boldsymbol{u}^{(i-1)}, \tag{12}$$

where λ_i is a step size term that is chosen using line search to maximize the decrease in f in each step and γ is the constant momentum parameter.

4.2. ManiFool for multiclass classifiers

The most common scheme used in multiclass classifiers is one-vs-all, which serves as a basis for our method. In this scheme, the classifier function has c outputs where c is the number classes. Thus, the function is defined as $f : \mathbb{R}^n \rightarrow \mathbb{R}^c$ and the classification is performed as:

$$k(\boldsymbol{x}) = \arg\max_k f_k(\boldsymbol{x}), \tag{13}$$

where f_k is the output of f that corresponds to the k^{th} class. Let $l_x = k(\boldsymbol{x}^{(0)})$, where $\boldsymbol{x}^{(0)}$ is the input image to the algorithm. Then we can define $c-1$ binary classifiers for $l \neq l_x$ as:

$$g_l(\boldsymbol{x}) = f_{l_x}(\boldsymbol{x}) - f_l(\boldsymbol{x}). \tag{14}$$

Since $l_x = \arg\max_k f_k(\boldsymbol{x}^{(0)})$, $f_{l_x}(\boldsymbol{x^{(0)}}) > f_l(\boldsymbol{x^{(0)}})$ for all $l \neq l_x$ and thus $g_l(\boldsymbol{x}^{(0)}) > 0$.

As we now have $c-1$ binary classifiers, we can get $c-1$ examples of fooling transformations by using each binary classifier as input to the binary ManiFool from Algorithm 1. When the algorithm is used with g_l, it stops iterating

Algorithm 2 ManiFool for multiclass classifiers

Input: Image $\boldsymbol{x}$, classifier f
Output: Transformation $\hat{\tau}$

Initialize with $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{x}$, $l_x \leftarrow k(\boldsymbol{x}^{(0)})$.
for $l \neq l_x$ **do**
  $g_l \leftarrow f_{l_x} - f_l$
  $i \leftarrow 0$
  while $k(\boldsymbol{x}^{(i)}) = l_x$ **do**
    $\hat{\boldsymbol{u}} \leftarrow -\boldsymbol{J}^{+}_{\boldsymbol{x}^{(i)}} \nabla g_l(\boldsymbol{x}^{(i)})$
    $\boldsymbol{u}^{(i)} \leftarrow \lambda_i \frac{\hat{\boldsymbol{u}}}{\|\hat{\boldsymbol{u}}_{l_n}\|} + \gamma \boldsymbol{u}^{(i-1)}$
    $\tau_i \leftarrow \exp\left(\sum_j u_j G_j\right)$
    $\boldsymbol{x}^{(i+1)} \leftarrow \boldsymbol{x}^{(i)}_{\tau_i}$
    $i \leftarrow i + 1$
  end while
  $\hat{\tau}_l \leftarrow \tau_0 \circ \tau_1 \circ \ldots \tau_i$
end for
$l_{min} \leftarrow \arg\min_{l \neq l_x} \tilde{d}(e, \tau_l)$
return $\hat{\tau} = \tau_{l_{min}}$

when $k(\boldsymbol{x}^{(i)}) \neq l_x$, instead of stopping when $g_l(\boldsymbol{x}^{(i)}) < 0$, because the classifier we are trying to fool is k, and not g_l. Let τ_l be the output transformation when g_l is used as the input to the binary ManiFool. As we now have a fooling transformation example for each class, the class with the smallest transformation by using the geodesic score from (2) can be chosen as

$$l_{\text{min}} = \arg\min_{l \neq l_x} \tilde{d}_{\boldsymbol{x}^{(0)}}(e, \tau_l), \tag{15}$$

and the algorithm will output the corresponding transformation, $\tau_{l_{\text{min}}}$.

The complexity of the algorithm depends on multiple factors, including the properties of the input classifier, input image and parameters such as γ. For example, as a separate single target ManiFool is run for each output class, the complexity depends heavily on c, the number of classes. Thus, to reduce complexity, only the most probable $\hat{c}$ classes are used in the algorithm, which are the $\hat{c}$ classes with highest $f_l(\boldsymbol{x}_0)$ excluding l_x. The complexity is also highly dependent on the number of iterations N_{it} for each single target ManiFool. Although these cannot be known exactly before running the algorithm, they are bounded by the maximum number of iterations N_{max}. Thus, the complexity increases linearly with N_{max}, which should be chosen carefully in order not to unnecessarily increase the complexity.

4.3. Measuring robustness to geometric transformations

Although we have utilized geodesic distance $d_{\boldsymbol{x}}(e, \tau)$ in the above methods, we have not shown how it can be computed. One possible method is to use Fast Marching Method (FMM), which progressively calculates the distance of points on a grid on the manifold from a reference point [21]. However, the complexity of this algorithm increases exponentially with the manifold dimension, and may rapidly become too complex. Thus, we propose a different method, by assuming that the geodesic path to the target node is direct, and we estimate the geodesic distance using this direct path. Let $p_0, p_1 \in \mathcal{M}$, and $v_1 = R_{p_0}^{-1}(p_1)$ where R is a retraction. Then, the direct path from p_0 to p_1 can be defined as

$$\gamma(t) = R_{p_0}(tv_1), \quad t \in [0, 1]. \tag{16}$$

For a chosen step-size η, v_1 can be divided into parts as $\hat{v} = \eta \frac{v_1}{\|v_1\|}$. Then for $N = \lfloor \frac{\|v_1\|}{\eta} \rfloor$, the distance can be estimated as

$$\hat{d}(p_0, p_1) = \sum_{i=1}^{N} \|R(i\hat{v}) - R((i-1)\hat{v})\|_{L^2} + \|p_1 - R(N\hat{v})\|_{L^2}, \tag{17}$$

which is the sum of L^2 distances on the sampled direct path. An illustration is given on Figure 3. In our case, we estimate $d_{\boldsymbol{x}}(e, \tau)$ as $\hat{d}(\boldsymbol{x}, \boldsymbol{x}_\tau)$, since from (1), the distance between the transformations on $\mathcal{T}$ is equal to the distance between their transformed counterparts.

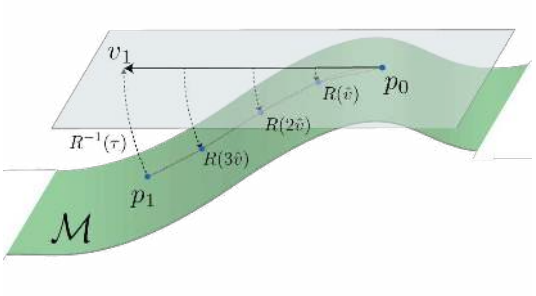
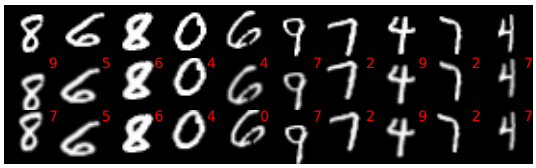


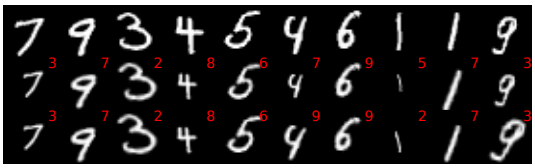
Figure 3: Illustration of the distance measurement. Assume $\mathcal{M}$ be a manifold and $p_0, p_1 \in \mathcal{M}$. We estimate the geodesic distance between these points using the direct path, by first mapping p_1 to the tangential space of p_0 as v_1 vector, dividing v_1 into smaller vectors and remapping these back onto the manifold. Then, $\hat{d}(p_0, p_1)$ from (17) is given by the sum of distances between these points, as the length of the grey curve.

5. Experimental Results

We now test our algorithm on convolutional neural network architectures. In these experiments, the invariance score for minimal transformations, defined in (5), is calculated by finding fooling transformation examples using ManiFool for a set of images, and computing the average of the geodesic distance of these examples. On the other hand, to calculate the invariance against random transformations, we generate a number of random transformations with a given geodesic distance r for each image in a set and



(a) Translations



(b) Similarity transformations

Figure 4: Examples of MNIST images transformed using outputs of ManiFool and Manitest [7] for translation and similarity sets. Top rows show the original images, the middle rows show the outputs from ManiFool and the bottom row shows the output of Manitest. The red numbers indicate the new output labels of the transformed images.

calculate the misclassification rate[2] of the network for the transformed images. A random transformation is created by sampling the unit sphere of $T_e\mathcal{T}$ and increasing the magnitude of this vector until the corresponding transformation has score equal to r, *i.e.*, $d(e, R(\alpha v)) = r$ where $v \in T_e\mathcal{T}$ is the sampled vector with $\|v\| = 1$ and $\alpha > 0$ is a scaling factor. A more detailed explanation of how the random transformation is sampled can be found in the supplementary material. The misclassification rate is calculated for different r to see the performance of the network on different levels of perturbation and to get $\hat{r}_{\mathcal{T}}$ from (6). In every case, the discrete images after transformation are obtained using bilinear interpolation; they further have the same size as the original image with zero-padding boundary conditions when necessary.

5.1. Performance of ManiFool

The first experiment compares the ManiFool algorithm with Manitest [7], to evaluate the performance of the algorithm in terms of speed and accuracy. For this comparison, on top of calculating the invariance score using ManiFool with $N_{\max} = 50$, we also do the same thing with Manitest, *i.e.*, we use it to find fooling transformation examples for the set of images and use these transformations to measure invariance. The comparison is done using 1000 images from the MNIST [16] training dataset and a baseline CNN with two 5×5 layers with 32 and 64 feature maps respectively with ReLU nonlinearity and 2×2 max pooling. In both cases, the geodesic distance is calculated using (17) to be comparable. Some of the transformed images generated during the experiment can be seen in Figure 4 and Table 1 reports the invariance score $\hat{\rho}_{\mathcal{T}}$ and the running time for both methods.

Manitest uses the fast marching method to find the transformation that changes the label. This requires measuring the distance of all the transformed images on a grid over parameters and evaluate the classifier until it reaches a fooling transformation. Thus, it is guaranteed to find the minimal transformation in the discretized search space and it is more accurate than ManiFool in this regard. However, as it can be seen on Table 1, it is more complex than ManiFool, especially as the dimension of the manifold is increased. Thus, it is not scalable for high dimensional manifolds. On the other hand, while not being as accurate, ManiFool is less complex in these situations. For example, Table 1 shows that the complexity of ManiFool does not closely depend on the dimensionality of $\mathcal{M}(\boldsymbol{x})$. Thus, ManiFool can be used for measuring the invariance of more complex networks such as the state-of-the-art architectures used with ImageNet database. Because of this, we have only used ManiFool to compare networks in the following experiments.

	ManiFool		Manitest	
Transformation	$\hat{\rho}_{\mathcal{T}}$	time	$\hat{\rho}_{\mathcal{T}}$	time
T ($d = 2$)	1.68	2.6 s	1.54	2.7 s
R+T ($d = 3$)	1.40	3.6 s	1.33	23.9 s
S+T ($d = 3$)	1.41	6.2 s	1.32	34.6 s
T+R+S ($d = 4$)	1.26	3.1 s	1.25	29.5 s

Table 1: Comparison of Manitest and ManiFool for different transformation sets on MNIST dataset. In the table, T, R and S stand for translation, rotation and scaling respectively; while d represents the number of dimensions of the transformation groups. The time column lists the average time required to compute one sample. The experiment was done using a baseline CNN with 2 convolutional layers. These times are computed on a server with 2 Intel Xeon CPU E5-2680 v3 without GPU support.

5.2. Invariance score of different architectures

The second experiment uses ManiFool to quantitatively compare the invariance properties of different networks. For this purpose, we have computed the invariance score of AlexNet [15], ResNet [11] and VGG [23] pre-trained models, using 5000 random images from ILSVRC2012 validation dataset [19]. Some examples that are created during this experiment can be seen on Figure 5, which shows that these networks are quite vulnerable against geometric transformations. In addition, we also compute the invariance of these networks to random transformations for $r \in [0, 10]$ using 5000 images from ILSVRC2012 validation dataset with 10 random transformations each. The misclassifica-

[2] As we consider the invariance of a network, we define misclassification as a change in label, *i.e.*, if $k(\boldsymbol{x}) \neq k(\boldsymbol{x}_\tau)$ for the transformed image $\boldsymbol{x}_\tau$

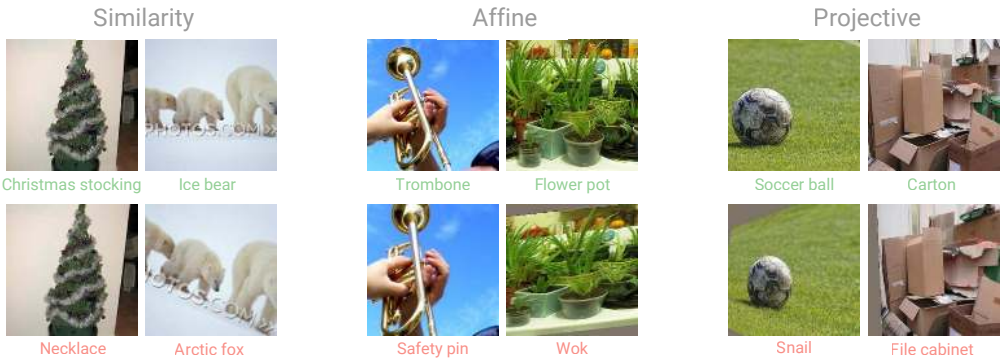
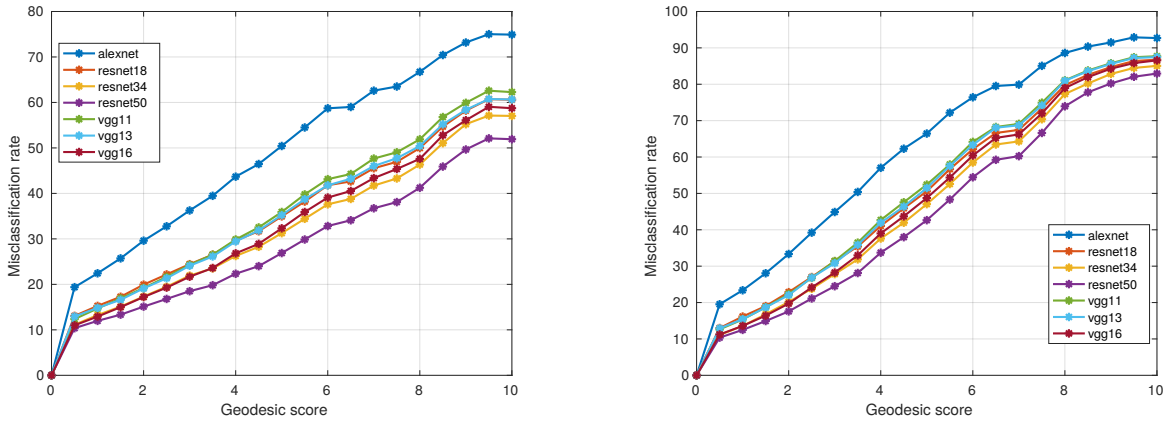


Figure 5: Examples of ILSVRC2012 images transformed using outputs of ManiFool using ResNet18 for similarity, affine and projective sets.Top row shows the original image while the bottom row shows the transformed image. The texts bottom show the output labels of the images at top and bottom respectively. More examples are found in the supplementary material.



(a) Similarity transformations

(b) Affine transformations

Figure 6: Misclassification rates of different networks with respect to the geodesic score of the transformation of the input images. The rates are calculated using 5000 images from ILSVRC2012 validation dataset with 10 different transformations for each image.

tion rates of the networks for each tested r is seen on Figure 6. Figure 7 reports the invariance scores for all of the networks. It can be seen that, for the same type of networks (*e.g.*, VGGs and ResNets), the invariance increases with the number of layers in each set of transformations. This result is in agreement with the previous empirical studies on smaller datasets such as [9, 7], but we have shown here that this also holds for deeper, more complex networks. On top of this, we can also see that neither the number of parameters nor the depth of the networks are the only decisive factors: ResNet-18 is less invariant than VGG-16, even though it is deeper and VGG-16 has more parameters than ResNet-50, yet it is less invariant. Similar results can be observed for the invariance to random transformations, *e.g.*, the invariance again increases with depth. In fact, Figure 7 shows that there certainly is a correlation between these two invariance values. However, interestingly, the ordering is not exactly the same as can be seen on Figure 6. For example, ResNet-34 and ResNet-50 perform better against random transformations compared to VGG networks.

	Original	Minimal	Random	Baseline
$\hat{\rho}_{\mathcal{T}}$	1.13	1.78	1.55	1.10

Table 2: The invariance to affine transformations of ResNet18 on CIFAR10 before and after the first epoch of fine tuning. Invariance score is calculated using 5000 images from CIFAR10 test set. 'Minimal', 'Random' and 'Baseline' stand for the extra epoch done using the transformed dataset created using ManiFool, the dataset created using random transformations and the training set respectively.

5.3. Adversarial Training using ManiFool

As our last experiment, we have fine-tuned a network for CIFAR10 [14] classification by performing 5 additional epochs with a 50% decreased learning rate using images that were transformed by label changing affine transformations generated using ManiFool. To be complete, we also performed 5 extra epochs using the original data and 5 extra epochs using randomly transformed images. For these randomly transformed images, the score of the transformations were equal to the median geodesic score of the dataset generated with ManiFool. We used ResNet-18 for

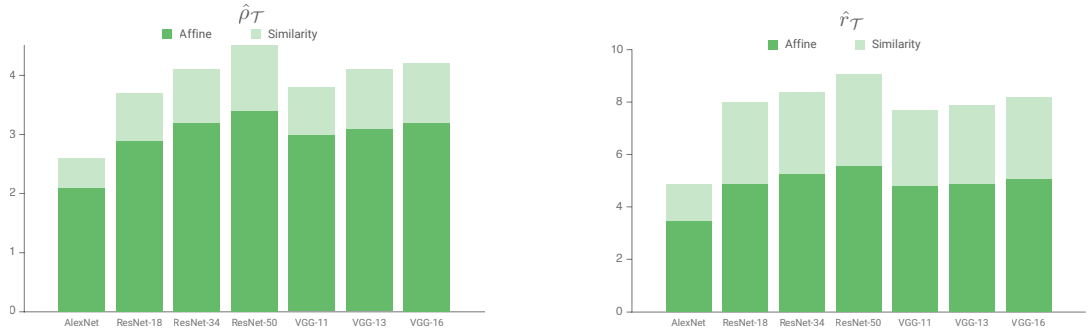


Figure 7: Invariance scores of different networks against similarity and affine transformations. For $\hat{\rho}_{\mathcal{T}}$, the invariance scores are calculated using 5000 images from ILSVRC2012 validation dataset and for $\hat{r}_{\mathcal{T}}$, they are calculated again using 5000 images with 10 different transformations for each image.

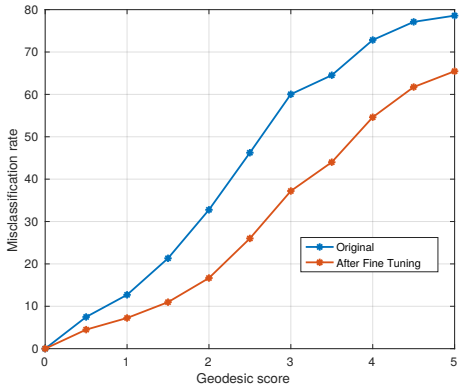


Figure 8: Misclassification rate of ResNet18 on CIFAR10 before and after fine tuning using adversarial geometric transformations with respect to the geodesic score of the random affine transformations of the input images. The rates are calculated using 5000 images from the test set with 20 transformations for each image.

this experiment, which was trained using stochastic gradient descent with softmax loss. The invariance score against minimal transformations, $\hat{\rho}_{\mathcal{T}}$ is then calculated for the networks after each epoch using 5000 images from CIFAR-10 test set. To see the effect of fine tuning on the invariance against random transformations, we also computed the misclassification rate for a transformation distance of $r \in [0, 5]$ using 5000 images from CIFAR10 test set with 20 transformations each. The results of these experiments are seen on Figure 8 and Table 2. It can be observed in Table 2 that fine tuning with adversarial examples has increased the invariance score significantly, even after only one extra epoch. This is in line with previous works on additive adversarial perturbations, where adversarial training was shown to improve robustness against the examined additive perturbation[10][18], but this effect is now seen for transformations as well. We can also see that its effect is greater than using only randomly transformed images. More interestingly, we can also see in Figure 8 that fine tuning using the worst-case examples has increased the robustness against *random transformations* considerably. For example, for random transformations with $\tilde{d}(e, \tau_l) = 2.5$, the misclassification rate has decreased more than 20% after fine tuning. Although there is also a small penalty in accuracy of the network (0.6% reduction on test set), this shows that choosing the worst-case transformation examples for fine tuning can increase invariance in both worst-case and random regimes.

6. Conclusion

In this work, we have presented a new constructive framework for computing the invariance score of deep image classifiers against geometric transformations. We proposed an algorithm, ManiFool, for finding small fooling transformation examples. The simple idea behind it is to perform gradient descent on the manifold of geometric transformations, in other words, it iteratively moves towards the class decision boundary while staying on the manifold to generate adversarial examples. Using this method, we have studied the robustness of networks trained on ImageNet against worst-case and random transformations. We also showed that adversarial training using ManiFool can be used as a way to improve the robustness of deep networks against both worst-case and random transformations and leads to more invariant networks. In the future, we believe this process can be used for empirical analysis of neural networks under geometric transformations and thus provide a better understanding of invariance to non-additive perturbations and the properties of different network architectures. Also, the ManiFool algorithm can be useful for generating new and practically relevant types of adversarial examples by using wider types of natural transformations.

Acknowledgments

We thank Alhussein Fawzi for the fruitful discussions. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research. This work has been partly supported by the Hasler Foundation, Switzerland.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, N.J. ; Woodstock, 2008. OCLC: ocn174129993. 4

[2] A. Bakry, M. Elhoseiny, T. El-Gaaly, and A. Elgammal. Digging Deep into the layers of CNNs: In Search of How CNNs Achieve View Invariance. In *International Conference on Learning Representations(ICLR)*, 2016. 2

[3] S. Baluja and I. Fischer. Adversarial Transformation Networks: Learning to Generate Adversarial Examples. *CoRR*, abs/1703.09387, Mar. 2017. 2

[4] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium On*, pages 39–57. IEEE, 2017. 2

[5] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable Convolutional Networks. In *International Conference on Computer Vision(ICCV)*, 2017. 2

[6] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust Physical-World Attacks on Deep Learning Models. *arXiv:1707.08945 [cs]*, July 2017. 1

[7] A. Fawzi and P. Frossard. Manitest: Are classifiers really invariant? In *British Machine Vision Conference (BMVC)*, 2015. 1, 2, 6, 7

[8] A. Fawzi and P. Frossard. Measuring the effect of nuisance variables on classifiers. In *British Machine Vision Conference (BMVC)*, pages 106.1–106.13, 2016. 2

[9] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng. Measuring invariances in deep networks. In *Advances in Neural Information Processing Systems*, pages 646–654, 2009. 1, 2, 3, 7

[10] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations(ICLR)*, 2015. 1, 2, 8

[11] 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(CVPR)*, pages 770–778, 2016. 6

[12] M. Jaderberg, K. Simonyan, A. Zisserman, and others. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 2

[13] E. Kokiopoulou and P. Frossard. Minimum Distance between Pattern Transformation Manifolds: Algorithm and Applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(7):1225–1238, July 2009. 2

[14] A. Krizhevsky and G. Hinton. *Learning Multiple Layers of Features from Tiny Images*. Master's thesis, University of Toronto, Department of Computer Science, 2009. 7

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. 1, 6

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov. 1998. 6

[17] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *IEEE Conference on Computer Vision and Machine Learning(CVPR)*, 2015. 2

[18] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016. 1, 2, 3, 8

[19] 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. *Int J Comput Vis*, 115(3):211–252, Dec. 2015. 6

[20] S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet. Adversarial Manipulation of Deep Representations. In *International Conference on Learning Representations(ICLR)*, 2016. 2

[21] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996. 5

[22] X. Shen, X. Tian, A. He, S. Sun, and D. Tao. Transform-Invariant Convolutional Neural Networks for Image Classification and Search. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 1345–1354, New York, NY, USA, 2016. ACM. 2

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations(ICLR)*, 2015. 6

[24] S. Soatto and A. Chiuso. Visual Representations: Defining Properties and Deep Approximations. In *International Conference on Learning Representations(ICLR)*, 2016. 2

[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. 1, 2

[26] L. W. Tu. *Differential Geometry*, volume 275 of *Graduate Texts in Mathematics*. Springer International Publishing, Cham, 2017. 4

[27] M. B. Wakin, D. L. Donoho, H. Choi, and R. G. Baraniuk. The multiscale structure of non-differentiable image manifolds. In *Optics & Photonics 2005*, pages 59141B–59141B. International Society for Optics and Photonics, 2005. 2