UNIVERSITE BRETAGNE LOIRE / MATHSTIC

école normale supérieure rennes

THÈSE DE DOCTORAT DE

l'École Normale
Supérieure Rennes
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Hanwei ZHANG

«Deep Learning in adversarial context »

Thèse présentée et soutenue à Rennes, le
Unité de recherche : IRISA

Rapporteurs avant soutenance :

Michel Crucianu	CNAM, PR
Cecilia Pasquini	University of Trento, Assistant professor

Composition du Jury :

Président :	Patrice Quinton	ENS, PR émérite
Examinateurs :	Changbo Wang	ECNU, PR
	Yuan Xie	ECNU, PR
	Aimin Zhou	ECNU, PR
Dir. de thèse :	Laurent Amsaleg	CNRS, IRISA, Rennes, DR
Co-dir. de thèse :	Yannis Avrithis	Inria, IRISA, Rennes, ARP
	Teddy Furon	Inria, IRISA, Rennes, CR

ACKNOWLEDGEMENTS

I would like to give my sincere thanks to my supervisors, Laurent Amsaleg, Yannis Avrithis, and Teddy Furon, for their outstanding guidance and advice over three years. They not only inspired me, helped me to develop further on my academic journey but also provided personal support so that I went through the difficult moments. I will always remember what they taught during my PhD and carry them as a fortune for the rest of my life. I feel lucky to be a student of my supervisors, in addition to their scientific knowledge and curiosity, I will also never forget the kindness and supportiveness they have shown.

A special thanks to my reviewers and jury members, Patrice Quinton, Michel Crucianu, Cecilia Pasquini, Changbo Wang, Yuan Xie, and Aimin Zhou for evaluating my work and giving valuable feedback.

I would also like to thank the PRoSFER program between East China Normal University and École normale supérieure de Rennes so that I have the opportunity to study for my PhD in an international environment. I would like to thank Aimin Zhou who inspired me to go further in academics and supported me to join the PRoSFER program. Thank you Patrice Quinton for helping me find the PhD position and introducing me to Laurent. Besides, I would like to thank the Chinese Scholar Council for funding my thesis, as well as IRISA/INRIA for welcoming me to the Lab. Furthermore, I would like to thank Inria which supported me for an extra six months extension for my PhD due to the effect of Covid-19.

In particular, I would like to thank the ladies who have a superpower and help me to solve administrative issues. They are Deborah France-piquet, Aurelie Patier, Lucy Liu, and Xiaoling Liu. Thank you Deborah for welcoming me to ENS de Rennes, helping me to settle down in Rennes, and making me feel at home. Thank you Aurelie for everything. You make my life easier. Thank you Lucy and Xiaoling for helping me and caring about my study as well as my life.

I would like to thank all the former and current team members of LinkMedia not only for all the nice discussions and the time spent together but also for the care and supports for me. I would like to give a special thank to Oriane Simeoni and Marzieh Gheisari khorasgani. I feel lucky to go through my PhD with Oriane and Marzieh, who share their love and energy with me.

Last but not least, I would like to thank my family and friends who provided me with their support and unconditional love throughout my life.

TABLE OF CONTENTS

RÉSUMÉ EN FRANÇAIS

L'apprentissage artificiel (ou encore *Machine Learning (ML)* en anglais) est une branche de l'intelligence artificielle (ou *Artificial Intelligence (AI)* en anglais) qui apprend à partir de données pour identifier des modèles, faire des prédictions ou prendre des décisions avec un minimum d'intervention humaine. Toutes les techniques de ML prennent des données en entrée mais visent la réalisation de *tâches* différentes : *classification, régression, regroupement, réduction de la dimensionnalité et classement,* etc..

Grâce aux données présentes en masse et aux ressources de calcul importantes, de plus en plus d'applications faisant appel à de l'apprentissage artificiel apparaissent dans notre vie quotidienne. Ces applications libèrent les gens des tâches répétitives et compliquées et leur permettent d'acquérir facilement des informations utiles. Par exemple, les systèmes de reconnaissance faciale aident les humains à s'identifier et à obtenir des autorisations. Les moteurs de recherche rassemblent et organisent les informations liées à une requête donnée par l'indexation, la recherche et la mise en correspondance. Les applications de navigation recommandent le meilleur chemin vers leur destination pour un véhicule autonome.

La vision par ordinateur (ou Computer Vision (CV) en anglais) étudie comment les ordinateurs peuvent acquérir, traiter, analyser et comprendre les images numériques. Les progrès réalisés dans le domaine de l'apprentissage artificiel facilitent le développement d'algorithmes s'appuyant sur de la vision par ordinateur pour notamment réaliser la *tâche de classification d'images*. Lorsque cette tâche est mise en oeuvre par des réseaux neuronaux profonds, alors on arrive à reconnaitre automatiquement et avec un très haut degré d'exactitude le contenu visuel d'images. Pour cela, il faut entrainer le système en lui faisant tout d'abord analyser des milliers d'images d'animaux, de lieux, de personnes, de plantes, etc.

Apprentissage profond

Au cours des dernières décennies, les Deep Neural Network (DNN)s se sont développés rapidement dans le domaine de la classification d'images. *Convolutional Neural Network (CNN)* [LBBH98] obtient des caractéristiques visuelles utiles et sémantiques. Un Convolutional Neural Network (CNN) profond typique possède de nombreuses couches et des archi-

tectures complexes, comme AlexNet [KSH12], Inception [SVI+16], Deep Residual Network (ResNet) [HZRS16a], DenseNet [HLVDMW17] et autres. Ce sont là quelques exemples bien connus de Deep Learning (DL) ou de DNNs.

Au cours des dernières décennies, les réseaux profonds se sont développés rapidement dans le domaine de la classification d'images. En particulier, les réseaux profonds convolutionnels [LBBH98] réussissent à extraire des caractéristiques visuelles utiles à la classification et à l'interprétation sémantiques des images. Ces caractéristiques existent dans un espace représentationnel de grande dimension. Cet espace est analysé par les algorithmes qui calculent des gradients permettant de séparer les images en classes différentes.

Sur cette base, des réseaux aux nombreuses couches et aux architectures complexes, comme AlexNet [KSH12] et Inception [SVI+16], ont été proposés et sont particulièrement performants. Ils réussissent à classer presque correctement toute image donnée au réseau, avec une probabilité de confiance élevée. Les performances de classement sur *ImageNet* [RDS+15], un jeu de données difficile et réaliste, sont proches de celles des humains.

La performance des réseaux profonds est souvent en relation avec leur profondeur : plus le réseau est profond, meilleure est la performance. Cependant, la complexité élevée de ces réseaux due à l'empilement d'un grand nombre de couches pour acquérir des caractéristiques visuelles sémantiques entraîne des difficultés d'apprentissage. Cela pourrait être dû à la disparition des gradients pendant la rétropropagation, appelée *problème de disparition des gradients*. Il existe de nombreuses variantes de l'architecture originale des réseaux profonds qui ainsi tentent de contourner ce problème. Cela inclut les unités résiduelles de ResNet [HZRS16a] et les *Transformer*s [VSP+17]. Les unités résiduelles court-circuitent certaines couches pendant le processus d'entrainement, qui est ainsi plus simple, accéléré et offre la possibilité d'explorer un plus grand espace de caractéristiques. Cependant, cela rend également ResNet plus vulnérable aux perturbations. Les *Transformer* [VSP+17] utilisent le concept d'auto-attention qui aide les réseaux à se concentrer sur les caractéristiques importantes. Dans l'ensemble, toutes ces avancées augmentent les performances de tâches telles que la classification. Les réseaux profonds réussissent non seulement à traiter des données d'image, mais aussi à traiter des images contenant du bruit, des occultations ou d'autres artefacts visuels.

Exemples adversaires

En 2013, des chercheurs ont découvert qu'une légère modification des images conduisait les classificateurs à faire des prédictions erronées [SZS+13]. La grande surprise était que ces

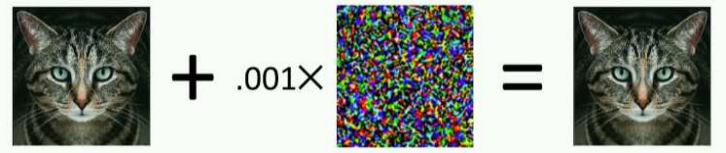


Figure 1 – Cette image provient de la présentation *Attacking Machine Learning : On the Security and Privacy of Neural Networks* de Nicholas Carlini. Elle montre que lorsqu'une perturbation adverse est ajoutée à l'image d'un chat, cette image est ensuite classée comme étant un chien.

modifications étaient d'une faible amplitude et étaient presque imperceptibles à l'œil humain. Cette découverte a révélé la vulnérabilité des réseaux profonds. C'est le début de l'étude des phénomènes adverses.

Les phénomènes adverses affectent largement les algorithmes d'apprentissage artificiel. Ils ont un impact sur différents médias, tels que l'image [SZS+13, GSS14, TPG+17], l'audio [CW18, YS18, YLCS18], et le texte [RDHC19, ZSAL20, ASE+18]. En outre, les attaquants ne se contentent pas de produire des exemples adversaires sauvegardés sous forme de données numériques, des images *e.g.*, dans un ordinateur, mais ils créent également des exemples adversaires dans le monde physique, tels que les patchs adversaires [TVRG19]. Il s'agit d'images imprimées et d'objets $3D$ [KGB16, SBBR16], capturés par des capteurs visuels comme les caméras, et qui affectent les applications qui les utilisent.

Une *Perturbation adverse* (ou encore adversaire) est une perturbation invisible qui incite les réseaux profonds à classer l'entrée perturbée dans une catégorie incorrecte. Par exemple, grâce à des perturbations adverses, on peut amener le classificateur à classer un chat comme un chien, comme dans Figure 1 [1]. En outre, les phénomènes adverses se transmettent entre les classificateurs. Les attaques qui tirent parti d'une certaine vulnérabilité d'un réseau particulier peuvent tromper d'autres réseaux, quels que soient l'architecture ou l'ensemble d'entraînement qu'ils utilisent – on parle alors de la propriété de *transférabilité* de ces exemples adversaires.

Modifier un contenu visuel en un autre est un gros problème [EEF+18, TVRG19, TRC19a, YLDT18, GSS14]. Un attaquant dont le but est de tromper le classificateur pour qu'il prenne des décisions inappropriées peut facilement perpétrer une perturbation adversaire. Ceci est inquiétant et dangereux, surtout lorsque les décisions du réseau mettent des vies en jeu. Par exemple, le fait de mettre des petits morceaux de papier d'une forme et d'une couleur particulières sur certains panneaux de signalisation empêche de les reconnaître [BMR+17]. Porter un tissu décoré d'un médaillon avec une texture particulière rend une personne invisible pour l'algorithme qui vise à détecter la présence de piétons [XZL+20]. Compte tenu de tous ces risques potentiels, il est

1. Source : `https://nicholas.carlini.com/slides/2019_rsa_attacking_ml.pdf`

crucial de comprendre les problèmes fondamentaux des exemples adversaires pour s'assurer que les algorithmes traitent les contenus de manière équitable et correcte. Les tâches de recherche typiques en matière d'exemples adversaires comprennent *les attaques et les défenses*. Les chercheurs étudient ces deux tâches afin i) d'apporter des contributions pratiques et ii) de comprendre ce phénomène.

Attaques. Les attaques visent à créer des perturbations adverses leurrant un réseau cible. Elles formalisent l'invisibilité et la mauvaise classification comme un problème d'optimisation. La difficulté des attaques dépend du fait que les attaquants connaissent ou non l'architecture des réseaux. Le cas de base est que les attaquants ont accès à l'architecture et aux paramètres des réseaux, ce qui est une *configuration en boîte blanche*. Ils profitent de ces informations pour élaborer des perturbations adverses.

L'attaque d'un réseau sans en connaître l'architecture et les paramètres, on parle alors de configuration en boîte noire, est un cas plus complexe. La notion de *transférabilité* implique que les échantillons adversaires se généralisent très bien sur différents réseaux et sur différents modèles d'apprentissage automatique [GSS14, TPG+17]. Cela indique que les échantillons adverses générés pour tromper un classificateur connu ont une certaine probabilité de tromper également un classificateur inconnu. Cela fournit un outil pour attaquer des réseaux dans un cadre de boîte noire.

Les attaques existantes parviennent à créer des perturbations adverses même si la contrainte est stricte. De façon surprenante, par exemple, une attaque d'un pixel [SVS19] change la prédiction des réseaux en modifiant seulement un pixel de l'image d'entrée. La perturbation universelle [MFFF17, HD18] révèle qu'une perturbation particulière est suffisante pour entraîner une mauvaise classification sur chaque image d'un ensemble de données considéré.

Défenses. Les défenses visent à améliorer la robustesse des réseaux contre les attaques adverses. Elles ajoutent un composant supplémentaire pour défendre les réseaux contre les attaques adverses ou améliorent la robustesse intrinsèque des réseaux.

Les défenses introduisant un composant supplémentaire conservent les réseaux inchangés. L'application d'un prétraitement aux images est une défense particulière dans cette catégorie. Là, ces défenses traitent les perturbations adverses comme un type particulier de bruit et tentent de les supprimer par le biais de transformations [MC17, GRCvdM17, STL+19]. Les défenses qui considèrent les exemples adverses comme des données malveillantes utilisent un détecteur pour reconnaître les exemples adverses et les rejeter ou les corriger [XEQ17, LLS+18]. Ces défenses

sont bon marché et s'adaptent facilement à des réseaux donnés, mais elles sont généralement vulnérables dans les paramètres de la boîte blanche [ACW18].

Les défenses améliorant la robustesse intrinsèque tentent de rendre plus robuste la phase d'entrainement [GSS14, MMS+17], d'augmenter la robustesse de l'architecture [PMW+16], ou d'agir sur la fonction de perte [HXSS15, MMS+17, TKP+17]. L'entrainement adversaire [GSS14, MMS+17], comme défense typique dans cette catégorie, améliore la méthode d'entrainement en incluant des exemples adversaires comme partie des données de d'entraînement. L'hypothèse derrière cette défense est que la vulnérabilité de DNN est due à l'insuffisance des données d'entraînement. Ces défenses ont des performances correctes en termes de robustesse et de précision, mais elles sont généralement coûteuses car elles nécessitent d'entrainer les réseaux à partir de zéro.

Aperçu et contributions

Dans cette thèse, nous tentons de comprendre les phénomènes adversaires. Nous explorons à la fois comment générer des exemples adversaires et comment s'en défendre. L'analyse des multiples facettes de l'adversité donne les éléments clés à étudier :

— **Vitesse**. La rapidité est importante tant pour les attaques adverses que pour les défenses. Bien que les processus chronophages, comme l'optimisation de la création d'une perturbation adverse et l'entraînement d'un modèle, produisent des résultats de haute qualité, il n'est pas envisageable de générer un exemple adverse, de vérifier les entrées ou de construire un modèle robuste si cela prend un temps extrêmement long.

— **Invisibilité.** L'ampleur de la distorsion est largement utilisée comme mesure d'invisibilité mais elle n'est pas équivalente à l'invisibilité. L'invisibilité indique que la perturbation est imperceptible pour les humains d'un point de vue neurologique et psychologique. La mesure de l'invisibilité en informatique reste une question ouverte.

— **Distorsion.** L'être humain perçoit à peine les perturbations lorsque l'ampleur est faible. L'ampleur de la distorsion est également importante pour les défenses. Normalement, les défenses contre les perturbations adverses avec une plus grande distorsion sont plus robustes contre les effets adverses. Il s'agit d'une mesure importante tant pour les attaques adverses que pour les défenses.

— **Transférabilité.** La transférabilité décrit la possibilité que des exemples adverses générés pour tromper un réseau cible réussissent à tromper d'autres réseaux. La transférabilité est

cruciale pour les attaques dans le cadre de *configurations en boîte noire*, c'est-à-dire que les attaquants ne peuvent acquérir que les paires d'entrée-sortie des réseaux.

Nos travaux sont motivés par les concepts de *vitesse*, *distorsion* et *invisibilité*. Nous testons la *transférabilité* de nos perturbations adverses. Pour améliorer la qualité des perturbations adverses, nous travaillons dans deux directions : produire des perturbations adverses invisibles et créer efficacement des perturbations adverses de faible ampleur. Pour se défendre contre les attaques, nous proposons un algorithme léger qui permet d'obtenir des performances décentes en termes de robustesse et de précision. Nous mettons l'accent sur la vitesse et la performance.

Pour permettre aux lecteurs de mieux comprendre nos travaux, nous donnons d'abord un aperçu du contexte adversaire dans le cadre de l'apprentissage artificiel profond dans chapitre 2. Cela inclut 1) une présentation succinte de ce que sont l'apprentissage artificiel et les réseaux profonds, 2) la définition de base du problème adverse et 3) l'examen de haut niveau des travaux connexes existants sur la génération de perturbations adverses et l'augmentation de la robustesse contre les attaques.

Le taux de réussite des attaques et l'ampleur de la *distorsion* sont deux critères standard pour mesurer la qualité d'une perturbation adversaire. Dans la chapitre 3, nous présentons l'évaluation standard des perturbations adversaire, y compris le jeu de données, les réseaux et les mesures d'évaluation. De plus, dans chapitre 3.3, nous proposons nos métriques d'évaluation qui permettent une comparaison équitable entre les attaques *ciblées distorsion* et les attaques *ciblées succès*.

Nous étudions deux algorithmes effectuant des attaques afin de gagner en compréhension dans *invisibilité* (voir chapitre 4) et la création *vitesse* (voir chapitre 5).

Chapitre 4 : Smooth adversarial perturbation. Dans le chapitre 4, nous étudions la définition de l'invisibilité et la formulons comme une fonction de contrainte afin qu'elle puisse être ajoutée de manière simple aux attaques existantes. Nous conjecturons que les perturbations adverses sont invisibles lorsque la similarité entre les pixels des perturbations et leurs pixels voisins est analogue au graphe de similarité de leurs images originales. Nous réussissons à générer des *perturbations adverses lisses* d'une petite magnitude de *distorsion*. Ces perturbations adverses lisses sont invisibles à l'œil nu, même si les exemples adverses sont artificiellement agrandis.

Chapitre 5 :Boundary Projection (BP) attack. Pour accélérer les attaques sans dégrader la qualité des exemples adverses, nous améliorons l'algorithme d'optimisation avec la connaissance

spécifique de la perturbation adverse. Dans le chapitre 5, nous proposons *Boundary Projection (BP)* une attaque qui change les directions de recherche en fonction de la solution actuelle. Lorsque la solution actuelle n'est pas adversaire, l'attaque BP recherche longuement la direction des *gradients* afin de diriger la solution actuelle pour traverser la *frontière* des réseaux. Lorsque la solution actuelle est adversaire, l'attaque BP cherche le long de la frontière pour diriger la solution actuelle afin de diminuer la magnitude de la *distorsion*. Ainsi, BP évite de gaspiller des calculs sur l'oscillation causée par le fait de suivre uniquement les gradients. Cela permet à l'attaque BP de gagner en *vitesse*. Les expériences montrent que l'attaque BP réussit à créer une perturbation adverse de très faible amplitude mais avec un taux de réussite élevé et rapidement.

Pour avoir une compréhension plus complète des problèmes adversaires, nous étudions ensuite les stratégies de défense. Le chapitre 6 présente une défense par remplacement de patch.

Chapitre 6 : Patch replacement. Les réseaux profonds sont plus robustes au bruit aléatoire qu'aux perturbations adverses. Pour en comprendre la raison, nous étudions la transition de la magnitude de *distorsion* (bruit aléatoire/perturbation adverse) à travers les couches du réseau. Inspirés par le comportement différent du bruit aléatoire et des perturbations adverses à l'intérieur du réseau, nous proposons une défense réactive appelée *remplacement de patch* dans le chapitre 6. Le remplacement de patchs tente d'éliminer les effets adverses lors de l'inférence en remplaçant les patchs d'entrées suspectes (images/caractéristiques) par leurs voisins les plus similaires dans les données d'entraînement légitimes. L'utilisation de données d'entraînement augmente la complexité des attaques même si les attaquants sont conscients de la défense par remplacement de patch. Comme nous ne prenons pas seulement en compte les images mais aussi les *features* intermédiaires des réseaux, le remplacement de patch est plus robuste que les autres défenses basées sur la transformation des entrées. Un inconvénient est que l'empoisonnement de l'ensemble de données au moment de la formation perturbe la stratégie de remplacement de patch. Cela permet d'accéder à des portes dérobées adverses.

Enfin, nous donnons la conclusion et nous proposons quelques perspectives dans le chapitre 7. En bref, nos contributions à la compréhension des problèmes d'apprentissage adverses sont i) de définir l'invisibilité d'un autre point de vue et de proposer une approche pour produire une perturbation adversaire lisse selon notre définition ; ii) de proposer un algorithme pour générer rapidement des exemples adversaires avec un taux de réussite élevé et une faible distorsion ; iii), nous réussissons à proposer une défense réactive qui n'est pas coûteuse et améliore la robustesse contre les attaques sans dégrader sévèrement la précision du réseau.

CHAPTER 1

INTRODUCTION

Machine Learning (ML) is a branch of *Artificial Intelligence (AI)* that learns from data to identify patterns, make predictions or decisions with minimum human intervention. All the ML techniques take data as input and aim at different *tasks*, *i.e*. *classification, regression, clustering, dimensionality reduction and ranking,* etc..

Benefiting from massive data and high computation resources, ML applications are becoming omnipresent in our daily life. These applications free people from repetitive and complicated work and allow them to acquire useful information easily. For instance, face recognition systems assist humans in identification and authorization. Search engine gathers and organizes information related to a given query by indexing, searching, and matching. Navigation applications recommend the best path to their destination for an autonomous vehicle.

Computer Vision (CV) studies how computers can acquire, process, analyze and understand digital images. Advances in ML facilitate the development of CV, especially *image classification task*. *Deep Learning (DL)* is a kind of powerful ML technique. It allows the design of *Deep Neural Network (DNN)* which can recognize the visual content of images automatically. Learning from thousands of images of animals, places, people, plants, *etc*., DNNs are able to detect with high confidence what an unknown image contains.

1.1 Deep learning and deep neural network

In the last decades, DNNs developed rapidly in the domain of image classification. *Convolutional Neural Network (CNN)* [LBBH98] obtains useful and semantic visual features. A typical deep CNN has numerous layers and complex architectures, like AlexNet [KSH12], Inception [SVI+16], ResNet [HZRS16a], DenseNet [HLVDMW17] and others. These are few well-known examples of DL or DNNs. These networks calculate gradients from the high dimensional representational space of images to find how to separate classes. Recent DNN models achieve tasks of classification, detection and segmentation with a high confidence. The performance of DNN models on *ImageNet* [RDS+15], a challenging and realistic dataset, is close to

that of humans.

The performance of DNNs is often in relation to their depth: the deeper the network is, the better the performance is. However, high complexity of DNNs due to deep stacking of large number of layers to acquire semantic visual features cause difficulty in training. This could be due to vanishing gradients during the back-propagation, named *gradient vanishing problem*. Many variations of the original architecture of DNN exist, trying to circumvent such problems. This includes residual units from ResNet [HZRS16a] and transformers [VSP+17]. Residual units skipping layers during training process effectively simplify networks, speed up the training process and offer to explore larger feature space. However, it also makes ResNet more vulnerable to perturbations. *Transformer* [VSP+17] uses the concept of self-attention which aids the networks to focus on important features. Overall, all these advances in DNNs augment the performance of tasks like classification. DL not only succeeds in processing massive image data but also manages to deal with images containing noise, occlusions, or other visual artifacts.

1.2 Adversarial examples

In 2013, researchers found that with a slight modification of the images lead to classifiers making erroneous prediction [SZS+13]. The big surprise was that these modifications were of a small amplitude and are almost imperceptible to human eyes. This discovery revealed the vulnerability of DNNs.

Adversarial phenomena widely affect ML. This impacts different media, such as image [SZS+13, GSS14, TPG+17], audio [CW18, YS18, YLCS18], and text [RDHC19, ZSAL20, ASE+18]. Furthermore, attackers not only produce adversarial examples saved as digital data, *e.g.* images, in a computer but also create adversarial examples in the physical world, such as adversarial patches [TVRG19]. These are printed pictures and 3D objects [KGB16, SBBR16], captured by visual sensors like cameras, and affect the ML applications using them.

Adversarial perturbation is an invisible perturbation that misleads DNNs to classify perturbed input into an incorrect category. For example, through adversarial perturbations one could make the classifier to classify a cat as a dog as in Figure 1.1 [1]. Furthermore, adversarial phenomena transfer among classifiers. Attacks that take advantage of a certain vulnerability of a DNN may deceive other DNNs whatever architectures or training set they use.

Modifying one visual content to another is a big problem [EEF+18, TVRG19, TRC19a, YLDT18, GSS14]. An attacker whose goal is to delude the classifier to make inappropriate

1. Source: `https://nicholas.carlini.com/slides/2019_rsa_attacking_ml.pdf`

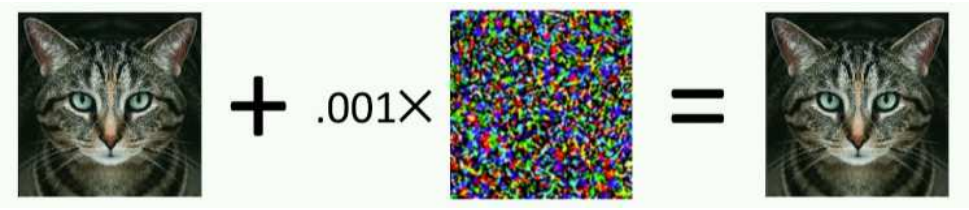


Figure 1.1 – This image comes from the presentation *Attacking Machine Learning: On the Security and Privacy of Neural Networks* from Nicholas Carlini. It shows with adversarial perturbation an image of cat is classified as a dog.

decisions can perpetrate adversarial perturbation conveniently. This is disturbing and dangerous, especially when network decisions put lives at stake. For example, putting small pieces of paper of a particular shape and color on some road signs prevents them from being recognized [BMR+17]. Wearing a medallion decorated cloth with a particular texture makes a person invisible to the algorithm which aims at detecting the presence of pedestrians [XZL+20]. Considering all these potential risks, it is crucial to understand the fundamental problems of adversarial examples to make sure algorithms process contents fairly and correctly. The typical research tasks in adversarial ML include *attacks and defenses*. Researchers study these two tasks in order to i) make practical contributions and ii) understand this phenomenon.

Attacks. Attacks intend to create adversarial perturbations towards a target DNN. They formalize the invisibility and misclassification as an optimization problem. The difficulty of attacks depends on whether attackers know the architecture of the networks or not. The foundational case is that attackers have access to the architecture and parameters of the networks, *i.e. white-box setting*. They benefit from this information to craft adversarial perturbations.

Attacking a network without knowing the architecture and parameters, *i.e. black-box setting*, is a more complex case. *Transferability* implies that adversarial samples generalize very well across different networks and across different machine learning models [GSS14, TPG+17]. This indicates that adversarial samples generated to deceive a local classifier have a certain probability to deceive an unknown classifier too. It provides a tool to attack DNN in black-box setting.

Existing attacks succeed to create adversarial perturbations even if the constraint is strict. These extraordinary adversarial perturbations manifest different properties of adversarial phenomena and the vulnerability of DNNs. Surprisingly, for instance, a one-pixel attack [SVS19] changes the prediction of networks by only modifying one pixel of the input image. Universal perturbation [MFFF17, HD18] reveals that one particular perturbation is enough to lead to misclassification on every image from a given dataset.

Defenses. Defenses aim to improve the robustness of DNN against adversarial attacks. They either add an extra component to assist networks against adversarial attacks or improve the intrinsic robustness of networks.

Defenses introducing an extra component retain networks unchanged. Applying pre-processing to images is a particular defense in this category. They treat adversarial perturbation as a particular kind of noise and attempt to remove it via transformations [MC17, GRCvdM17, STL+19]. Ones who regard adversarial examples as malicious data employ a detector to recognize adversarial examples and reject or correct them [XEQ17, LLS+18]. These defenses are cheap and easily adapt to given networks, however, are usually vulnerable under the white-box settings [ACW18].

Defenses improving the intrinsic robustness attempt to ameliorate the training method [GSS14, MMS+17], augment architecture [PMW+16], or advance loss function [HXSS15, MMS+17, TKP+17]. Adversarial training [GSS14, MMS+17], as a typical defense in this category, ameliorate the training method by including adversarial examples as a part of training data. The assumption behind this defense is that the vulnerability of DNN is due to the insufficiency of training data. These defenses perform decently on both robustness and accuracy, however, are usually expensive because they need to train networks from scratch.

1.3 Overview and contributions

In this thesis, we attempt to understand adversarial phenomena. We explore both how to generate adversarial examples and how to defend them. From the analysis of the multiple facets of adversarial ML, we find that the key elements to investigate include:

Speed. Speed matters for both adversarial attacks and defenses. Although time-consuming processes, like optimization of creating adversarial perturbation and training a DNN model, produce high-quality results, it is not feasible if it takes an extremely long time to generate an adversarial example, verify inputs, or build a robust model.

Invisibility. The magnitude of distortion is widely used to estimate the invisibility of perturbation but it is not equivalent to invisibility. Invisibility indicates the perturbation is imperceptible to humans from a neurological and psychological point of view. It is still an open question to measure invisibility in computer science.

Distortion. As an alternative plan to measure the quality of invisibility, numerous attacks estimate the magnitude of distortion. Humans hardly perceive perturbations when the magnitude is small. The magnitude of distortion also matters for defenses. Normally, defenses against adversarial perturbations with larger distortion are more robust against adversarial effects. It is an important

metric for both adversarial attacks and defenses.

Transferability. Transferability describes the possibility that adversarial examples generated to fool a target network successfully deceive other networks. Transferability is crucial to attacks under *black-box settings*, *i.e*. attackers can only acquire the input-output pairs of networks.

Our works are motivated by the concepts of *speed*, *distortion* and *invisibility*. We test the *transferability* of our adversarial perturbations. To improve the quality of adversarial perturbations, we work in two directions, *i.e*. producing invisible adversarial perturbations and creating adversarial perturbation efficiently with low magnitude. To defend against attacks, we propose a lightweight algorithm that achieves a decent performance on both robustness and accuracy. We emphasize speed as well as performance.

To let readers have a better understanding, we first give an overview of the adversarial context in DL in chapter 2. This includes 1) the minimum knowledge of ML and DNN needed to understand our work, 2) the basic definition of the adversarial problem and 3) the high-level review on existing related works both on generating adversarial perturbations and augmenting robustness against attacks.

Success attack rate and magnitude of distortion are two standard criteria to measure the quality of adversarial perturbation. In chapter 3, we introduce the standard evaluation to adversarial perturbations, including dataset, networks and evaluation metrics. Furthermore, in section 3.3, we propose our evaluation metrics that allow a fair comparison between *targeted distortion* attacks and *targeted success* attacks.

We investigate two algorithms performing attacks in order to gain understanding in *invisibility* (see chapter 4) and creation *speed* (see chapter 5).

Smooth adversarial perturbation. In chapter 4, we study the definition of *invisibility* and formulate it as a constraint function so that it can be added in a straightforward way to existing attacks. We conjecture that adversarial perturbations are invisible when the similarity between the pixels of perturbations and their neighbor pixels is analogous to the similarity graph of their original images. We succeed in generating *smooth adversarial perturbation* and surprisingly with a small magnitude of *distortion*. These smooth adversarial perturbations are invisible to the naked eyes even if the adversarial examples are artificially magnified.

Fast, low-distortion adversarial examples. To accelerate the attacks without degrading the quality of adversarial examples, we improve the optimization algorithm with the specific knowl-

edge of adversarial perturbation. In chapter 5, we propose *Boundary Projection (BP)* attack that changes search directions according to the current solution. When the current solution is not adversarial, BP attack searches long the direction of *gradients* to direct the current solution to cross the *boundary* of networks. When the current solution is adversarial, BP attack searches along the boundary to direct the current solution to decrease the magnitude of *distortion*. Comparing to the state-of-the-art attacks, BP attack avoids wasting calculation on the oscillation caused by only following gradients. That earns *speed* for BP attack. Experiments show BP attack succeeds in creating adversarial perturbation with a very small magnitude but a high success attack rate speedily.

To have a more complete understanding of the adversarial ML problems, we then investigate defense strategies. Chapter 6 presents a patch replacement defense.

Patch replacement. DNNs are more robust to random noise than adversarial perturbations. To understand it, we investigate the transition of the magnitude of *distortion* (random noise/adversarial perturbation) through the DNN. Inspired by the different behavior of both random noise and adversarial perturbations inside DNN, we propose a reactive defense named *patch replacement* in chapter 6. Patch replacement attempts to eliminate adversarial effects at inference by replacing patches of suspicious input (images/features) with their most similar neighbors in legitimate training data. The usage of training data increases the complexity of attacks even if attackers are aware of patch replacement defense. Since we not only consider images but also intermediate *features* of networks, patch replacement is rather robust than other defenses based on input transformation. One disadvantage is that poisoning the dataset at training time causes troubles to the patch replacement strategy. This connects to adversarial backdoors.

Finally, we give the conclusion and we propose a few perspectives in chapter 7. In brief, our contributions in understanding adversarial ML problems are i) defining invisibility in another view and proposing an approach to produce smooth adversarial perturbation under our definition; ii) proposing an algorithm to generate adversarial examples fast with a high success rate and low distortion; iii), we succeed to propose a reactive defense that is not expensive and improves the robustness against attacks without degrading the accuracy of the network severely.

CHAPTER 2

BACKGROUND

This chapter presents an overview of the background material that is needed to understand our contributions as well as positioning them in the field of adversarial attack and defense. We begin this chapter by elucidating the main features of ML and DNN based image classification in section 2.1 and section 2.2 respectively. We also present a survey of the domain of adversarial attacks and defenses, and the vocabulary used in section 2.3.

After defining in section 2.4 what are the adversarial images, those that can deceive classifiers, this chapter draws up in section 2.5 a panorama of attacks intended to deceive a classifier whose technology is based on DNN. In response, many studies have proposed defenses, which are presented in section 2.6. With all this knowledge, we give our point of view and discuss our contributions in section 2.7.

The reader is reminded that the focus of this chapter is to give a holistic overview and the background to the field, we postpone until latter chapters to provide a precise description and details of the existing solutions that are related to our contributions. A few reasons for this kind of treatment is as follows.

First, this background section is written such that it provides the reader a high-level perception of the most important facets of adversarial attacks and defenses, without disrupting that presentation with particular low-level details that are specific to this or that existing technique. Second, we naturally compare our contribution to existing solutions. It is very important to precisely detail these existing solutions, by presenting at a very fine grain their assumptions, their parameters, their algorithmic structure. Once the general overview is understood, it is much easier to present such details. It is for this reason we postpone the discussion of Carlini and Wagner attack (C&W) and Decoupling Direction and Norm (DDN) attack schemes to chapter 4 and chapter 5. We apply the same rationale to the defense schemes as well and postpone the detailed presentation of the existing defense techniques and their comparison with our contributions to chapter 6.

2.1 Machine learning

ML is a sub-branch of AI. ML is an interdisciplinary field of applied mathematics, statistics, and algorithmics. It allows a computer to perform a task learning from a representative data. Normally the set of rules that the machine requires to execute a task is impossible to enumerate "manually" as it is complex (for example machine translation). Even when one can define them it leads to a combinatorial behavior when considering all possible options to arrive at an optimal set (for example chess, go, ...). ML requires on the analysis of colossal amount of data to build a model for performing the task using these set of rules/options. One requires numerous and diverse data to better estimate the model and realize an automated execution of the target task.

Normally ML comprises two-phase process. The learning or training phase where the model first learns from the training data [MRT18]. The second phase is testing phase which tests the well-trained model on unseen data. This process sometimes realized by intertwining learning and testing phase better learn the model.

Based on the available information about the data during the learning phase there are two families of learning. Supervised learning [RN02] uses data-label pairs, where the supervision is manually provided. This supervision in classification is in the form of labels or categorical values that are discrete; in regression the supervision provided through continuous values. In contrast, in unsupervised learning [HS$^+$99] there does not exist manual supervision. This type of learning is useful as it is not always possible to label massive amounts of data, many approaches are proposed with varying degrees of supervision. One such approach is semi-supervised learning (where not all data is labeled) or this approach is also referred to as partially supervised approach.

The applications of the field are extremely varied, as tasks are. This includes classification, recognition, translation, grouping, analysis, prediction, the list is almost infinite. Learning applies to data of very different kinds: symbolic, numeric, continuous, or discrete data, graphs, trees, feature vectors, including images, sounds, texts, time series, *etc*.

In this chapter, we focus on the classification of images. The classification output is labels associated with the categories of images. These categories could be for example airplane, boat, road, table, chair, building, person, dog, cat, balloon, cutlery, daisy, tomato, *etc*. We remain in the scope of a supervised learning environment. Once the model has been learned, it classifies new, unknown, unlabeled images into the correct visual category or categories.

Any ML process is built from a few fundamental notions that we present in detail in this chapter. We begin with the notion of the objective function. An objective function, in general, is designed to measure the model's performance, *i.e*. it represents the capability of the model to

achieve the target task. It is often easier to adopt a mathematical representation of the problem. When gradually optimizing the objective function, we learn a set of parameters of the model. An objective function is often a decreasing function of an error related to the measure of completion of the task. The opposite of a root mean square error or a cross entropy are two classical examples of objective functions. These functions are designed to be continuous with respect to the model parameters. Thus allowing to detect if a variation in the parameters of the model improves performance on the task.

Learning means gradually adjusting the parameters of the model such that the new values of these parameters increase the quality of the model as perceived through the objective function. An optimization process is, therefore, to find the optimum parameters without making the search process too costly. The adjustment of parameters is aimed at, for instance, finding a better position of the hyperplane separating the two classes of data.

At the end of optimization, the model performs better on the training data. However, it is expected by the model to have good generalization capabilities to deal with the unseen or new data. Sometimes due to the nature of optimization, the model may have little or no ability to generalize well. This phenomenon is called *over-fitting* [ES02] which should be avoided. Generalization is improved through techniques such as cross-validation, regularization, random pruning, and others.

2.2 Classification of images by DNN

In this section, we discuss DNNs and image classification task. We also provide an overview of some important notions. We invite the reader to the book by Goodfellow and colleagues for greater detail and precision [GBCB16]. Besides, this section introduces the mathematical notations necessary to later describe attacks and defenses.

In the color space (red, green, blue), an image $\mathbf{I}$ of L rows and C columns is represented by a three-dimensional array existing in the space $[0, 1, \ldots, 255]^{3\times L\times C}$. Pixels are integers between 0 and 255 (if encoded on one byte). The output of the classifier is a class, *i.e.* categorical label. The k possible categories (airplanes, boats, cars, tables, chairs, buildings, people, dogs, cats, *etc.*) are arbitrarily ordered and the output of the classifier is an integer between 1 and k, noted $\hat{\ell}$.

The image classifier built on a DNN is here schematically decomposed into three parts. The first part performs a pre-processing that adapts the input image to the neural network. It often includes a sub-sampling of the image to a given size $c \times c$ (typically 224×224), and above all reduces the pixel dynamics to the range $[0, 1]$ (historical choice but other choices are

possible, such as reducing to $[-1, 1]$). This can be done by dividing the pixel value by 255. More elaborate transfer functions, which sometimes differ from one color channel to another, are also used. The output of this pre-processing is $\mathbf{x} = \mathsf{T}(\mathbf{I})$, classically noted as a column vector with $m = 3 \times c \times c$ components in $[0, 1]^m$.

The second part is the neural network. A neuron is a small automaton that combines the data it receives from other neurons, produces a value that is then transmitted to one or more neurons, each of which will combine it with the values received from other neurons, and so on, which together make a network. The neurons are often organized in layers, connected to each other [Hop82], and we name such kinds of layers as *fully-connected layers*. It is the great multitude of these layers that gives existence to the deep term. For example, there are networks made up of hundreds of interconnected layers, each composed of thousands of artificial neurons.

A neuron is therefore a small automaton that first operates a linear combination of the values received (from other neurons for example) which are weighted by synaptic weights, then summed. The value produced is then passed to an activation function or threshold function. This function introduces a non-linearity in the neuron's behavior, which is essential. Activation functions such as sigmoid activation, hyperbolic tangent activation, or activation based on a *Linear Rectification Unit (ReLU)* [NH10] are often used. These non-linear functions are continuous non-decreasing and almost everywhere differentiable.

Apart from fully-connected layers, there are convolutional layers that are the core component of a *CNN* [LBBH98]. A convolutional layer contains a set of learnable filters (or kernels) that have a small receptive field but convolve across the width and height of the input. In detail, during the forward pass, each filter is convoluted across the width and height of the input, computing the dot product between the filter and the input. With this design, the filters are learned to detect some specific type of features in same spatial positions. The pooling layer is another type of layer which is important to CNNs. It is a form of non-linear down-sampling. There are several non-linear functions to implement pooling, for instance, max pooling. Max pooling partitions the input into a set of patches (or sub-regions) and each patch outputs its maximum value. The pooling layer is designed to reduce the resolution of representation so that it is possible to reduce the computation in the network.

The output of the neural network is a k-component real vector called logit vector: $\mathbf{u} = \mathsf{R}(\mathbf{x}, \boldsymbol{\theta}) \in \mathbb{R}^k$. The larger the value of the j-th logit $u(j)$ is, the more likely the input $\mathbf{x}$ is of class j. The symbol $\boldsymbol{\theta}$ is a parameter representing the set of synaptic weights (of all neurons of all layers).

The third part translates the logits into a vector of probabilities $\mathbf{p} \in [0, 1]^k$ such that

$\sum_{j=1}^{k} p(k) = 1$. The value $p(j)$ is the probability that the input image is of class j. This translation is done with the function *softmax* [GBCB16], $\mathbf{p} = \mathsf{S}(\mathbf{u})$, defined by, $\forall\ 1 \leq j \leq k$:

$$p(j) = \frac{e^{u(j)}}{\sum_{i=1}^{k} e^{u(i)}}. \tag{2.1}$$

It is the gradual adjustment of the synaptic weights $\boldsymbol{\theta}$, parameters of the $\mathsf{R}(\cdot)$ function, which forms the heart of the supervised learning, the first and last parts being non-parametric. These weights are gradually adjusted so that the final value produced at the output of the classifier ultimately corresponds to the label associated with the input data.

The network is used in propagation mode when the input data passes through it layer by layer and the network produces at the end of the chain the probability vector $\mathbf{p}$. The error between the output $\mathbf{p}$ and what should have been produced is measured. For a label ℓ associated with input $\mathbf{x}$, the output is ideally a probability vector $\mathbf{p}^\star_\ell$ where $p^\star_\ell(j) = 1$ and where the other components of this vector are null. The *cross entropy* [GBCB16] $h(\mathbf{p}, \mathbf{p}^\star_\ell) = -\sum_{j=1}^{k} p^\star_\ell(j) \log(p(j))$ is a metric quantifying how $\mathbf{p}$ is different from $\mathbf{p}^\star$. The *loss* for the label ℓ and input $\mathbf{x}$ is measured by $\mathcal{J}(\mathbf{x}, \ell, \boldsymbol{\theta}) = h(\mathsf{S}(\mathsf{R}(\mathbf{x}, \boldsymbol{\theta})), \mathbf{p}^\star_\ell)$.

Backpropagation [GBCB16] consists of tracing back through the networks, from downstream to upstream, the error made by a neuron at its synapses and therefore to the upstream neurons which are connected to them. The gradient of the cross entropy with respect to each weight in the network is calculated. This is greatly simplified by the chain rule because the network is a composition of functions, and layers. Thus the set of these weights $\boldsymbol{\theta}$ is updated iteratively by an algorithm of gradient descent to reduce cross entropy. At iteration i:

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{J}(\mathbf{x}_{j(i)}, \ell_{j(i)}, \boldsymbol{\theta}), \tag{2.2}$$

where $\{\mathbf{x}_{j(i)}, \ell_{j(i)}\}$ is a training data randomly drawn at the i-th iteration of the stochastic gradient descent and $\eta > 0$ is the learning rate. During the training, networks process the data in full/mini batch. By changing the size of the batch, we define the number of samples to work through before updating the internal model parameter. *Batch normalization* [IS15] is proposed to mitigate the problem of *internal covariate shift*, where parameter initialization and the shift of inputs' distribution through layers affect the learning rate of the network.

The weight update between forward propagation (calculation of $\mathsf{S}(R(\mathbf{x}, \boldsymbol{\theta}))$) and backward propagation (calculation of $\nabla_{\boldsymbol{\theta}} h(S(R(\mathbf{x}, \boldsymbol{\theta})), \mathbf{p}^\star_\ell)$) ensures the learning process with respect to weights $\boldsymbol{\theta}$ to converge to a local minimum of cross entropy calculated over the training data.

Once the training is done the trained model is input with new/test data during the test phase; the output of this is a probability vector $\mathbf{p}$ in which the predicted class is the class with the highest probability:

$$\hat{\ell} = f(\mathbf{x}) := \arg \max_{1 \leq i \leq k} p(i). \tag{2.3}$$

2.3 Vulnerabilities in ML: A bit of history and vocabulary

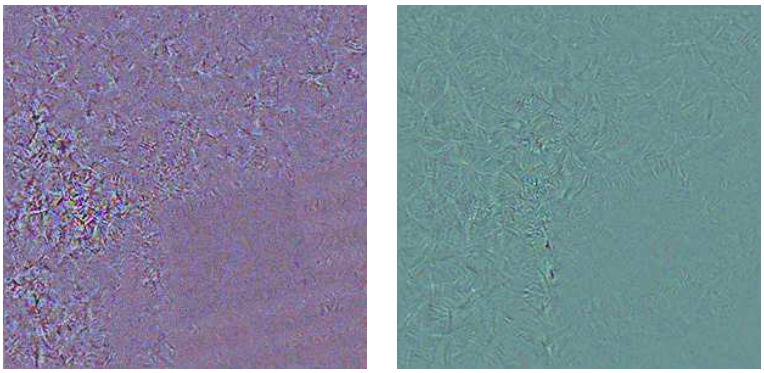
In this section, we discuss the vulnerabilities of DNNs. In fact, *all* ML algorithms have flaws and are vulnerable to intentional attacks. It was while researchers were working on the automatic classification of e-mails to separate *spam* from real messages that the first flaws were revealed. In 2004, Dalvi and his colleagues and also Lowd and Meek showed that it was possible to fool a linear classifier detecting flaws [DDS+04, LM05]. At that time, DNNs were not popular, and the techniques of choice for ML relied in particular on Support Vector Machine (SVM) based classifiers.

Ten years later, the field of *adversarial ML* is gaining the attention it deserves, because at this moment the incredible power of DNNs is revealed making them the most used ML models. Hence it is vital to study their vulnerabilities and fix them. *Adversarial learning*, sometimes also refers to *Generative Adversarial Networks (GAN)* [GPAM+14] where a generator and a discriminator contest with each other in a zero-sum game, but for us, it refers to *adversarial machine learning*, namely a machine learning technique that attempts to deceive models by adversarial inputs. Around 2014, researchers were working on the forgery of images likely to deceive a classifier based on a deep neural network.

To reiterate, adversarial images are the images that have been manipulated so that classifiers recognize them wrongly after analyzing them. For example when the classifier network is presented with an image of a cat and the network responds by categorizing the image as the image of airplane. In these cases the manipulation is such that it is almost invisible and when one looks at it, it looks like an image of a cat. The network, however, asserts with very strong confidence that it is an airplane. Figure 2.1 illustrates this, where the American flag, when intentionally modified, is recognized as a vending machine or, alternatively, as a sandal. One can notice the imperceptibility of an attack.

Historically, one of the first explorations of facets of the vulnerability of ML algorithms was made by Barreno and his colleagues [BNS+06]. In their seminal article, they discuss the vulnerability of ML algorithms during the learning phases (they discuss about poisoning), the test phases (they talk about evasion), they distinguish targeted attacks from untargeted attacks, they

(a) (b) (c)



(d) (e)

Figure 2.1 – Original image and adversarial images; the manipulations are almost imperceptible and the classification is wrong. a) Original image, correctly classified as a 'flag'. b) Adversarial image, created by the C&W method, classified by the network as a 'vending machine'. c) Adversarial image, created by the PGD_2 method, classified by the network as a 'sandal'. d) Distortion (contrast enhanced to make it more visible) existing in image (b) created using C&W method. e) Distortion (contrast enhanced to make it more visible) existing in image (c) created by PGD_2 method.

propose techniques to evaluate the power of these attacks and mention some defense mechanisms. They also differentiate vulnerabilities according to an attacker's knowledge of the system he/she wants to deceive, distinguishing between "white box" and "black box" attacks. We will return to these terms later in this chapter.

A very good historical perspective can be read in Biggio and Roli's article [BR18]. An excellent state-of-the-art exists thanks to Serban and his colleagues [SP18].

2.4 Adversarial images: definition

Let $f : \mathcal{R}^m \rightarrow \{1, \cdots, k\}$ be a classifier from the vector of pixels composing an image to a discrete label among the k possible classes. For an image $\mathbf{x} \in \mathcal{R}^m$ with a target label $\ell \in \{1, \cdots, k\}$, an adversarial perturbation $\mathbf{r}$ is produced by solving the following optimization problem:

$$\begin{aligned} &\min \|\mathbf{r}\|_p \\ &\text{such that } f(\mathbf{x}+\mathbf{r}) = \ell \\ &\qquad (\mathbf{x}+\mathbf{r}) \; is \; an \; image. \end{aligned} \tag{2.4}$$

This equation describes both targeted and untargeted attacks as well as white box and black box attacks.

Targeted attacks aim to make the adversarial image $\mathbf{x} + \mathbf{r}$ classified exactly in the given category ℓ (for example, the attacker wants a dog image to be classified as a cat image). *Untargeted attacks* aim to make the adversarial image $\mathbf{x} + \mathbf{r}$ classified in any class, as long as the predicted class ℓ is different from class $f(\mathbf{x})$ which $\mathbf{x}$ belongs to (for example, the attacker wants a dog image to be classified as anything but a dog).

White box attacks consider the scenario where the attacker knows everything about the classifier network. The attacker can reproduce it, set it up and test an attack, and once successful can deploy it. *Black box attacks*, on the other hand, consider that the attacker does not know the details of the network. The attacker has a copy of the trained classifier, which is not be "exposed" but can be used as an oracle, *i.e.* submit the images and observe their predictions as long as needed. Some articles also consider "grey boxes", *i.e.* a context where the attacker has a partial knowledge of the network. We shall return to all this later, in subsection 2.5.1.

The main part of Equation 2.4 is to minimize a distortion term. The norms normally used are the L_2, L_1, or L_∞ norm that are applied as a criterion to measure the distortion [GSS14, CW17]. When the measurement gives a very low distortion, it is almost invisible to human's

eyes. Nevertheless, these criterion do not account well for the human's perception of images. Distortions of small norms can sometimes be very visible. Therefore, it is relevant to measure distortion according to account for the way our visual system functions from a neurological and psychological point of view [WBSS04, SBR18, FBHD19]. But such metric is unfortunately much more complex to calculate. L_p norms are therefore often preferred in practice.

Equation 2.4 defines adversarial images via an optimization problem. The attack is the process used to find the solution to this problem. The input $\mathbf{x}$ existing in a very large m-dimensional space, finding this solution is difficult because the $f(\cdot)$ function is not explicit. It is very likely that the attack finds an approximate solution, *i.e.* it finds a perturbation $\mathbf{r}$ of greater distortion than the strict minimum given by the solution of Equation 2.4. *A first criterion to evaluate the quality of an attack is therefore the distortion it generates.*

A second criterion is given by the second line of Equation 2.4. An attack reaches its goal (targeted or untargeted) if $f(\mathbf{x} + \mathbf{r}) = \ell$. This problem is so difficult to solve that an attack can sometimes fail to produce an adversarial image $(\mathbf{x} + \mathbf{r})$. *The second criterion for evaluating the quality of an attack is thus the probability of its success.*

These two criteria are intimately linked by a trade-off. It is easy to build an attack that always succeeds: it is the attack that replaces $\mathbf{x}$ with a completely different image $\mathbf{x}'$ whose predicted class is ℓ, but the distortion $\|\mathbf{x}' - \mathbf{x}\|_p$ is huge and clearly visible to the naked eye. It is easy to build a zero distortion attack: it is the attack that substitutes $\mathbf{x}$ for the same $\mathbf{x}$, but the probability of success is zero (unless $\mathbf{x}$ is misclassified by the network). These two extreme attacks are of no interest, but they illustrate this trade-off. The probability of success is an increasing function of the distortion.

The third criterion is the complexity of the algorithm measured by its memory consumption or by the computing time required. The algorithms listed below are often iterative. Counting the number of iterations that are necessary to produce a visually good adversarial image is a valuable indicator. The lower the complexity, the faster the attack, but then often the probability of success is low or the distortion is high.

Two main principles are proposed to tackle Equation 2.4, *i.e. target distortion and target success.* Target distortion attacks (see subsection 2.5.3) focus on maximizing the success attack rate with a fixed distortion. Target success attacks (see subsection 2.5.4) focus on minimizing the distortion and always generate adversarial images.

Before presenting different attacks, let us return to the last term of Equation 2.4. It is said that $(\mathbf{x} + \mathbf{r})$ $is\ an\ image$. Let's see what this means and what it implies.

This definition, which is based on $\mathbf{x} = \mathsf{T}(\mathbf{I})$ and not on the $\mathbf{I}$ image, is historical. It reflects

the fact that the community working on computer vision and the one working on neural networks are not interested in the pre-processing part because there is nothing to learn or train. Thus the condition $(\mathbf{x} + \mathbf{r})$ $is\ an\ image$ simply means that $\mathbf{x} + \mathbf{r} \in [0, 1]^m$, just like $\mathbf{x}$. It would be possible to return to a "real" digital image (with pixel values between 0 and 255) by simply applying the inverse pre-processing $\mathsf{T}^{-1}(\cdot)$.

In reality, from our point of view, things are not that simple. We have described pre-processing as an integral part of the classifier. Thus $\mathbf{x}$ is an internal variable to which the attacker does not have access. However, the attacker's goal is to corrupt the input image $\mathbf{I}$ and not $\mathbf{x}$. Also, the method consisting in firstly finding $\mathbf{x} + \mathbf{r}$, then applying reverse pre-processing to form an image is sometimes impossible because there is no image such that its pre-processing gives $\mathbf{x} + \mathbf{r}$. We will come back to this later in this chapter.

2.5 Attacks

This section presents a quick overview of techniques for making adversarial images. We are not exhaustive, we describe attacks that are somehow exemplary of what the literature proposes, a very vast and rapidly expanding literature. All the techniques presented here are based on the Equation 2.4 and they enrich it in various ways.

2.5.1 White, grey, and black box

The white box model, as it is introduced in section 2.4, considers the case where attackers know exactly the architecture and parameters of the targeted network. In this case, attackers are able to not only access the input-output pairs from the targeted network but also access the gradient of the network.

The black box model is much stricter than the white box model. The attacker knows nothing about the targeted network. Here, it is impossible to calculate gradients and therefore impossible to apply the techniques mentioned so far. Nevertheless, the attacker can use the targeted network as an oracle and observe the way it labels an image.

The grey box and black box models assume that the attacker has much less information from his side. For the grey box model, we assume that the attacker knows some elements of the targeted network. For example, the network uses a pre-trained, off-the-shelf model, but with defense mechanisms that are secret. The attacker can partly reproduce the behavior of the targeted network to implement his attacks.

Two designs of the black box. If "black box" means to observe the inputs/outputs of a system, what are these outputs? Some consider that the output to which the attacker has access is the predicted probability vector, others consider that the output is the predicted class $\hat{\ell} = f(\mathbf{x})$.

The very nature of these outputs makes a big difference, as noted in the article by Ilyas and colleagues [IEAL18]. The predicted class $f(\mathbf{x})$ is a piece-wise constant function. Almost surely, the attacker cannot see if a small amplitude perturbation $\mathbf{r}$ is going in the right direction since it does not necessarily change the output. This is not the case for the predicted probability vector.

The output is the probability vector. In this case, there is not much difference with white box attacks. The attacker calculates an objective function as in the previous section and looks for the perturbation that minimizes it. The only difference is that now the gradient is no longer available. The attacker then uses so-called zero-order numerical methods such as differential evolution algorithms [SP97] sometimes combined with genetic algorithms. These algorithms randomly draw perturbations, calculate their objective functions, select the perturbations that obtained the lowest values and recombine them with random mutations. This selection-recombination-mutation cycle is repeated.

For example, the One-pixel attack [SVS19] is the counterpart of the Jacobian-based Saliency Map Attack (JSMA) where the pixels (or even the pixel) to be modified are found thanks to a genetic algorithm. Similarly, Engstrom's work [ETT+17] is the black box counterpart of Xiao's [XZL+18] geometric attacks in the white box. Also, Zhao and his colleagues [ZDS18] train a generator to produce adversarial perturbation just like [BF17] while the discriminator is a black box classifier.

An alternative to genetic algorithms is to estimate the gradient of the objective function in certain directions:

$$\frac{\partial \mathcal{J}(\mathbf{x})}{\partial x(i)} \approx \frac{\mathcal{J}(\mathbf{x} + \Delta \mathbf{e}_i) - \mathcal{J}(\mathbf{x} - \Delta \mathbf{e}_i)}{2\Delta}. \tag{2.5}$$

To estimate the gradient, it is necessary to calculate the deviation for all pixels $1 \leq i \leq m$, which is expensive. Chen and colleagues show that this can be avoided [CZS+17]. They apply a stochastic gradient descent where the directions $\mathbf{e}_i$ are drawn iteratively and randomly. This attack is called Zeroth Order Optimization (ZOO). Another attack in this category is the one designed by Narodytska and Kasiviswanathan [NK17].

The output is the predicted class. Szegedy and his colleagues were among the first to find that adversarial images created to attack a specific network are also adversarial for another network [SZS+13]. But it was Papernot and his colleagues who first explored the transfer

properties of attacks by studying so-called grey box and black box attacks [PMG16]. The work of Liu and colleagues is also worth mentioning as a remarkable paper on this subject [LCLS16]. A similar phenomenon, traditional in machine learning, is well known: it is possible, to some extent, to transfer what has been learned by one network to another.

Papernot and his colleagues rely on the observation of the proportion of adversarial images fooling the first system that also succeeds in fooling the second. To be more precise, they distinguish transfers between learning systems built on the same fundamental principle from transfers between systems built on different principles (a transfer between a deep network and a SVM-based system for example).

The results of their study [PMG16] show that transfers are possible and easy between neural systems. On the other hand, although still possible, the transfer of attacks is more difficult between systems built on models that cannot be derived, mathematically speaking, such as those built from SVM, K-nearest neighbors, decision trees (this contrasts for example with networks for which it is easy to calculate the gradient).

This observation makes it possible to attack black-box learning systems. By multiplying the requests to the targeted classifier, the attacker can build a model of it: a new classifier is trained to imitate the black box in the sense that the outputs of this classifier under construction must eventually be identical to those of the targeted black box. Then, the attacker uses this new model, but as a white box, to forge adversarial images with the hope that they also fool the black box network thanks to the transferability.

2.5.2 White box: target distortion/target success

In white box scenario, the attacker has access to the probability vector $\mathbf{p}$ computed by the model. Since the original image is of class ℓ_g, the vector $\mathbf{p}$ must get away from $\mathbf{p}^\star_{\ell_g}$ (the image is less recognized as being from the ℓ_g class). At the end, the predicted vector $\mathbf{p}$ must be close to $\mathbf{p}^\star_\ell$. As in supervised learning, the attacker also works with an objective function defined via cross entropy:

$$\begin{aligned} \mathcal{J}(\mathbf{x}, \ell) &= h(\mathbf{p}, \mathbf{p}^\star_\ell) - h(\mathbf{p}, \mathbf{p}^\star_{\ell_g}) && (2.6) \\ &= \log(p(\ell_g)) - \log(p(\ell)). && (2.7) \end{aligned}$$

Decreasing the objective function is equivalent to increasing the predicted probability for the class ℓ and decreasing that of the original class ℓ_g. Note that a perturbation makes the image

adversarial if $\mathcal{J}(\mathbf{x}+\mathbf{r},\ell) < 0$.

The definition of the objective function is more delicate for untargeted attacks. Decreasing only $h(\mathbf{p}, \mathbf{p}^{\star}_{\ell_g})$ is not enough. A first trick is to target the most probable class that is different from ℓ_g. Hence an objective function:

$$\mathcal{J}(\mathbf{x}) = \log(p(\ell_g)) - \max_{\ell \neq \ell_g} \log(p(\ell)). \tag{2.8}$$

There are other objective functions in the literature. For example, we will see the DeepFool attack (DeepFool) [MDFF16] in section 2.5.4 which notes that a class with a high predicted probability is not necessarily an easier class to reach.

In the following, we mainly describe cases where the network is perfectly known to the attacker ("white box attack") and the attacks are untargeted ("untargeted attacks").

Two big families. The core of Equation 2.4 is formed, on the one hand, by minimizing distortion and, on the other hand, by successfully fooling the system. Thus, algorithms that produce adversarial images are divided into two families, depending on whether they aim at never exceeding a specified distortion, or whether they aim at producing adversarial images almost surely without limiting the distortion (although a minimal distortion is a target). Let's characterize these two families before listing a few algorithms.

Target Distortion. All algorithms in this family aim to maximize the probability of success while not exceeding a fixed distortion. This distortion is not necessarily an explicit parameter of the attack, but it is governed by some of its parameters. This is generally expressed by:

$$\begin{aligned} &\min \mathcal{J}(\mathbf{x}+\mathbf{r}) \\ &\text{such that } \ \|\mathbf{r}\|_p \leq \epsilon, \end{aligned} \tag{2.9}$$

where $\mathcal{J}$ is the attacker's loss function and ϵ is the maximum allowed distortion.

The performance of this type of attacks is measured by their probability of success $P_{suc} = \mathcal{P}(f(\mathbf{x}+\mathbf{r}) \neq \ell_g)$ which depends, of course, on the value given to ϵ. If the attack does not succeed for a given upper bound ϵ, then this value can be increased and the algorithm starts again with a new higher ϵ. It is worth notify that the ϵ having finally allowed the creation of an adversarial image is not necessarily minimal.

Target Success. In this family, the algorithms aim for success and always produce an adversarial image at the cost of an arbitrary distortion, but minimal. This is expressed by:

$$\begin{aligned} &\min \|\mathbf{r}\|_p \\ &\text{such that } \mathcal{J}(\mathbf{x}+\mathbf{r}) < 0. \end{aligned} \tag{2.10}$$

The performance of these algorithms is characterized by the average distortion over successfully attacked images.

These two families of attacks use as a basic principle of Taylor development to the first order of the objective function:

$$\mathcal{J}(\mathbf{x}+\mathbf{r}) = \mathcal{J}(\mathbf{x}) + \mathbf{r}^\top \nabla_{\mathbf{x}} \mathcal{J}(\mathbf{x}) + o(\|\mathbf{r}\|). \tag{2.11}$$

The perturbations $\mathbf{r}$ that diminish the objective function $\mathcal{J}(\mathbf{x}+\mathbf{r})$ are therefore directed towards the opposite of the gradient, *i.e.* $-\nabla_{\mathbf{x}}\mathcal{J}(\mathbf{x})$. Since this approximation is local, it is only valid for low amplitude perturbations.

Starting from the value of the objective function, a backpropagation process is initiated which goes back to the matrix representing the image while keeping the weights of the neural network unchanged along the way. It is thus this matrix that modifies the original image so that the system is, *in fine*, deluded. The perturbation is thus a function of the gradient. Thanks to self-differentiation [GBCB16], the calculation of the gradient is automatic. On the other hand, the complexity of this process is just the double of the complexity of the forward propagation. Let us now list the main algorithms belonging to these two families.

2.5.3 Target Distortion attacks

FGSM. The first algorithm using the gradient to generate the perturbation and build an adversarial image has been proposed by Goodfellow and colleagues in 2014 [GSS14]. It is called *Fast Gradient sign method (FGSM)*. This method is very simple and is based on a perturbation calculated as follows:

$$\mathbf{y} = \mathbf{x} + \mathbf{r} = \mathbf{x} - \epsilon \operatorname{sign}(\nabla_{\mathbf{x}} \mathcal{J}(\mathbf{x})). \tag{2.12}$$

It is the perturbation that minimizes the first-order objective function for the constraint $\|\mathbf{r}\|_\infty = \epsilon$.

We recognize here the main elements for the process of attacks targeting distortion (see

Equation 2.9). By studying the gradient of the objective function $\mathcal{J}$, and taking its opposite, it is then possible to determine how to modify the input of the network to decrease $\mathcal{J}$ and hopefully lead to misclassification. The value of ϵ controls the maximum permissible distortion. This method is very simple and creates adversarial images very quickly that sometimes mislead the classifier. Nevertheless, it is rather crude, since it determines the perturbation to be applied from a single observation of the gradient.

I-FGSM. It is possible to refine FGSM iteratively. *Iterative Fast Gradient sign method (I-FGSM) [KGB16]* is the iterative version of FGSM. In contrast to Equation 2.12, the perturbation is not directly calculated. Iterative Fast Gradient sign method (I-FGSM) initializes $\mathbf{y}_0 := \mathbf{x}$ and then iterates by progressing in the opposite direction of the gradient with stepsize α. The recurrence is therefore:

$$\mathbf{y}_{i+1} := \mathrm{proj}_{B_\infty[\mathbf{x};\epsilon]}(\mathbf{y}_i - \alpha \operatorname{sign} \nabla_{\mathbf{x}} \mathcal{J}(\mathbf{y}_i)), \tag{2.13}$$

where $\mathrm{proj}_{\mathcal{A}}$ is the projection on the region $\mathcal{A}$ followed by a term-by-term thresholding to remain in the hypercube $[0, 1]$.

Here, the region $\mathcal{A}$ is the ball $B_\infty[\mathbf{x}; \epsilon]$ of L_∞ norm centered in $\mathbf{x}$ and radius $\epsilon > \alpha$. Projection on the ball $B_\infty[\mathbf{x}; \epsilon]$ is defined as

$$\mathrm{proj}_{B_\infty[\mathbf{x};\epsilon]}(\mathbf{y}) := \mathbf{x} + \mathrm{clip}_{[-\epsilon,\epsilon]}(\mathbf{y} - \mathbf{x}). \tag{2.14}$$

Thus, when the current solution remains inside the ball, the projection is not active. While the iterations calculate perturbations that get outside the ball, then the projection brings them back to its surface. This iterative approach is also known as Basic Iterative Method (BIM) [PFC+18].

I-FGSM and BIM perform targeted or untargeted attacks, depending on the definition of the attacker's loss function.

PGD. *Projected Gradient Descent (PGD)* is also an iterative method but it projects the gradient onto an L_2 norm ball [MMS+17]:

$$\mathbf{y}_{i+1} := \mathrm{proj}_{B_2[\mathbf{x};\epsilon]}(\mathbf{y}_i - \alpha \mathsf{n}(\nabla_{\mathbf{x}} \mathcal{J}(\mathbf{y}_i))), \tag{2.15}$$

where $\mathsf{n}(\mathbf{x}) := \mathbf{x}/\left\|\mathbf{x}\right\|_2$, and

$$\mathrm{proj}_{B_2[\mathbf{x};\epsilon]}(\mathbf{y}) := \begin{cases} \mathbf{y} & \|\mathbf{y}-\mathbf{x}\|_2 < \epsilon \\ \mathbf{x} + \epsilon \mathsf{n}(\mathbf{y}-\mathbf{x}) & \text{otherwise.} \end{cases} \tag{2.16}$$

.

The L_2 norm ball used for projection is $B_2[\mathbf{x};\epsilon]$, centered in $\mathbf{x}$ and of radius ϵ. Again, the attack does not end when $\mathbf{y}_i$ hits the $B_2[\mathbf{x};\epsilon]$ ball for the first time. It continues and seeks to minimize the objective function while remaining on the ball.

M-IFGSM. Iterative approaches progress along the gradient at a fixed step α (see Equation 2.13 and Equation 2.15). Setting this value is difficult: too small, the algorithms will stall and not find an adversarial image because the number of iterations is limited; too large, the algorithms will go fast but the gradient cannot be finely followed, which can create oscillations.

Approaches such as *Momentum Iterative FGSM (M-IFGSM) [DLP+18]* incorporate a step-wise adaptation mechanism: in the first iterations, it is advantageous to progress rapidly along the gradient. Later, however, it is better to take small steps to better follow the gradient and reach a local minimum.

2.5.4 Target Success attacks

The techniques in this family are typically more expensive. If the computation is not limited, for sure an adversarial image with the minimum distortion can be found.

L-BFGS. Szegedy and colleagues tackle the problem of creating adversarial images using a Lagrangian formulation [SZS+13]. Distortion is no longer a constraint but is integrated in the objective function:

$$\mathcal{L}(\mathbf{r}) := \mathcal{J}(\mathbf{x}+\mathbf{r}) + \lambda \left\|\mathbf{r}\right\|_2 . \tag{2.17}$$

For a given value of $\lambda > 0$, the unconstrained objective minimization is tackled by the numerical method *Limited-memory Broyden-Fletcher-Goldfarb-Channo (L-BFGS)*. It is an iterative method of gradient descent.

A large value for λ means that the minimum $\mathbf{r}^\star$ of the objective function might not be an adversarial perturbation because too much weight is given to the Euclidean distortion. Conversely, a value that is too small gives a minimum $\mathbf{r}^\star$ making $\mathcal{J}(\mathbf{x}+\mathbf{r})$ very negative but also a large distortion. This is illustrated in Figure 2.2. A linear search finds a suitable Lagrange multiplier.

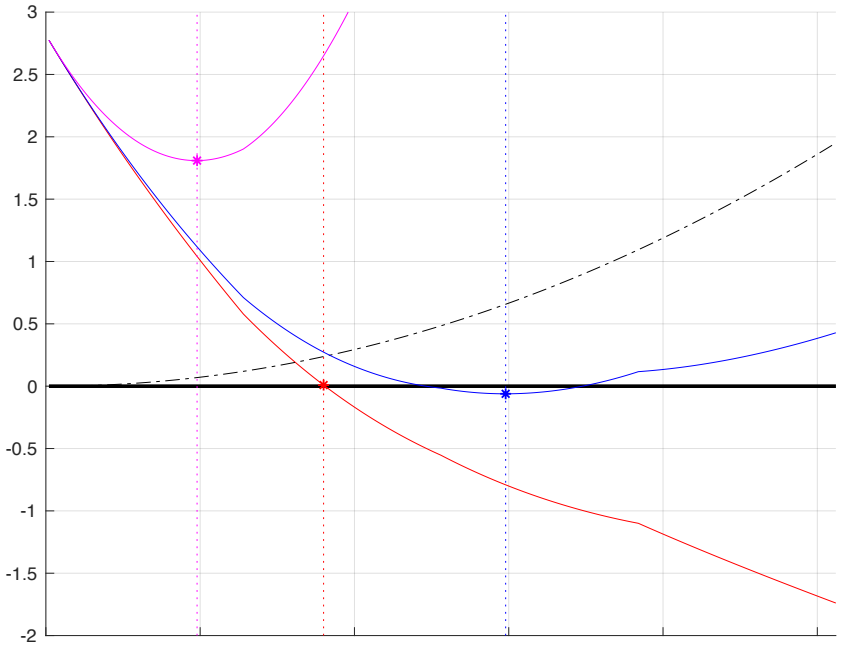


Figure 2.2 – Illustration of L-BFGS in 1D. The perturbation $\mathbf{r}$ is co-linear to the gradient of the objective function. The x-axis is the norm of $\mathbf{r}$. The red curve describes the objective function $\mathcal{J}(\mathbf{x}+\mathbf{r})$. It decreases and becomes negative when $\mathbf{r}$ is strong enough to be adversarial. The dotted black curve describes the distortion $\|\mathbf{r}\|^2$, The blue and magenta curves describe the function $\mathcal{J}(\mathbf{x}+\mathbf{r}) + \lambda * \|\mathbf{r}\|^2$ for two values of λ. The first value is too low (in blue): the minimum is "far after" the red asterisk; the perturbation is adversarial but of great distortion. The second value is too large (in magenta): the minimum is "far before" the red asterisk; the perturbation is not adversarial.

This means that the program performing the attack has two nested loops, which explains its great complexity.

C&W. The well-known *Carlini and Wagner attack (C&W)*, later noted C&W, continues this idea [CW17]. A change of variable eliminates the box constraint that $\mathbf{x}+\mathbf{r}$ must remain in $[0,1]^m$, replacing $(\mathbf{x}+\mathbf{r})$ by $\sigma(\mathbf{z})$, where $\mathbf{z} \in [-\infty,+\infty]^n$ and σ is the element-wise sigmoid function. Here we denote the cost function of C&W as

$$\mathcal{L}(\mathbf{z},\lambda) := \mathcal{J}_\mu(f(\sigma(\mathbf{z})),\ell_g) + \lambda \left\|\sigma(\mathbf{z})-\mathbf{x}\right\|^2, \tag{2.18}$$

where classification loss $\mathcal{J}_\mu$ encourages the logit $u(\ell_g)$ of ground truth to be less than any other logit $u(k)$ for $k \neq \ell_g$ by at least margin $\mu \geq 0$. It is defined as

$$\mathcal{J}_\mu(\mathbf{u}, \ell_g) := [u(\ell_g) - \max_{k \neq \ell_g} u(k) + \mu]_+, \tag{2.19}$$

where $[\cdot]_+$ denotes the positive part, *i.e.* $[x]_+ := x$ if $x > 0$, 0 otherwise. This function is similar to the multi-class SVM loss by Crammer and Singer [CS01], where $\mu = 1$ and, apart from the margin, it is a hard version of negative cross-entropy $\mathcal{J}$ where softmax is producing the classifier probabilities. It does not have the problem of being flat in the region of class ℓ_g.

When the margin is reached, loss $\mathcal{J}_\mu$ vanishes, and the distortion term pulls $\sigma(\mathbf{z})$ back towards $\mathbf{x}$ as shown in Figure 2.3, causing oscillations around the margin.

Parameter λ controls the quality of adversarial examples. If λ is too large, the optimizer will return a solution with small distortion but failing in misclassification. While λ is too small, the optimizer will return an adversarial solution but a large distortion. For a given λ, C&W uses the numerical method Adam [KB15] to find the minimum of the objective function in $\mathbb{R}^m$ and update the best solution with the current optimal solution if it is better. This is repeated for different λ [1] by line search, which is expensive.

DDN. *Decoupling Direction and Norm (DDN) [RHO+19])*, is an iterative attack very similar to PGD_2 seen above. The formulation of DDN is:

$$\mathbf{y}_{i+1} := \text{proj}_{S_2[\mathbf{x};\rho_i]}(\mathbf{y}_i - \alpha \mathsf{n}(\nabla_{\mathbf{x}} \mathcal{J}(\mathbf{y}_i))). \tag{2.20}$$

Here, the projection is performed on the sphere $S_2[\mathbf{x}; \rho_i]$ of radius ρ_i and centered in $\mathbf{x}$ even if $\mathbf{y}_{i+1}$ is inside the ball. The main difference with the formulation of PGD_2 given by Equation 2.15 is that the radius of this sphere changes from one iteration to another. This radius at iteration i is obtained by calculating $\rho_i = (1 - \gamma)\|\mathbf{y}_i - \mathbf{x}\|$ when $\mathbf{y}_i$ is adversarial. When this is not the case, then $\rho_i = (1 + \gamma)\|\mathbf{y}_i - \mathbf{x}\|$, with a hyper-parameter $\gamma \in (0, 1)$.

DeepFool. This is a untargeted white box attack that uses a more elaborate objective function. In Equation 2.8, a untargeted attack uses the objective function in order to modify the adversarial image into the most likely class. Thus the objective function has a positive but low value in $\mathbf{x}$. It seems easier to make it negative.

1. Referred to as c in [CW17].

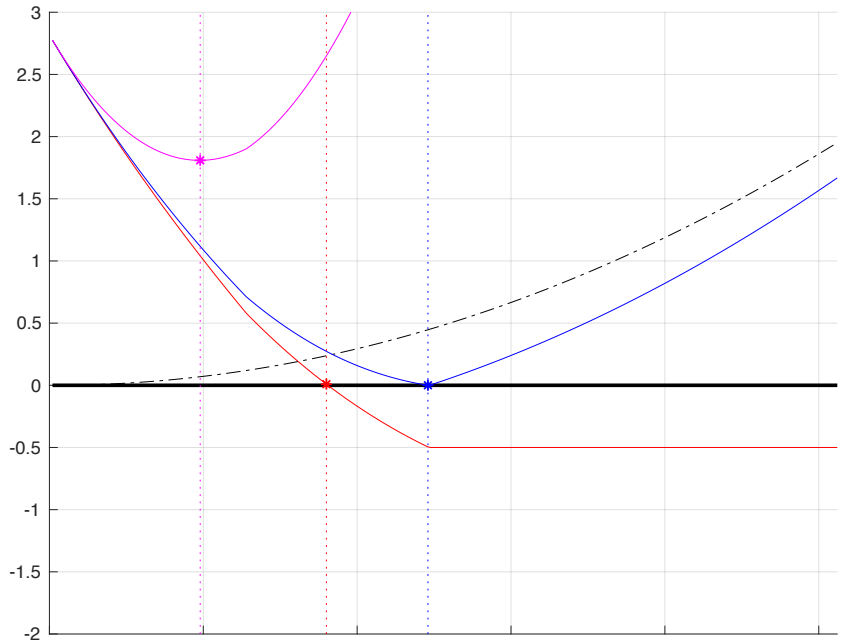


Figure 2.3 – Illustration of C&W in 1D. The perturbation $\mathbf{r}$ is co-linear with the gradient of the objective function. It is the same configuration as in Figure 2.2, except that the margin $\mu = 0.5$ for the threshold of Equation 2.18. Remark its effect: the blue minimum is closer to the red asterisk.

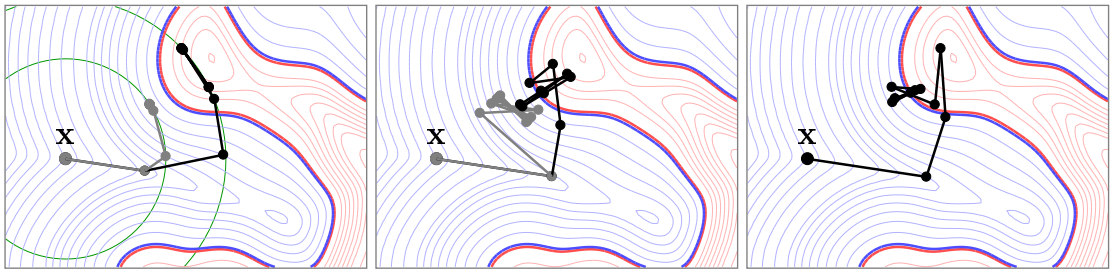


Figure 2.4 – Two-dimensional illustration of adversarial attacks on a binary classifier. From left to right: PGD_2, C&W, DDN. The regions associated with the two classes are in red and blue. The level lines indicate the predicted probabilities. The objective is to find an adverse point in the red region that is as close as possible to the starting point $\mathbf{x}$. In grey (respectively black) the paths taken for a low (respectively high) distortion ϵ for PGD_2 [KGB16] (radius of the green circle) or for a parameter λ for C&W [CW17].

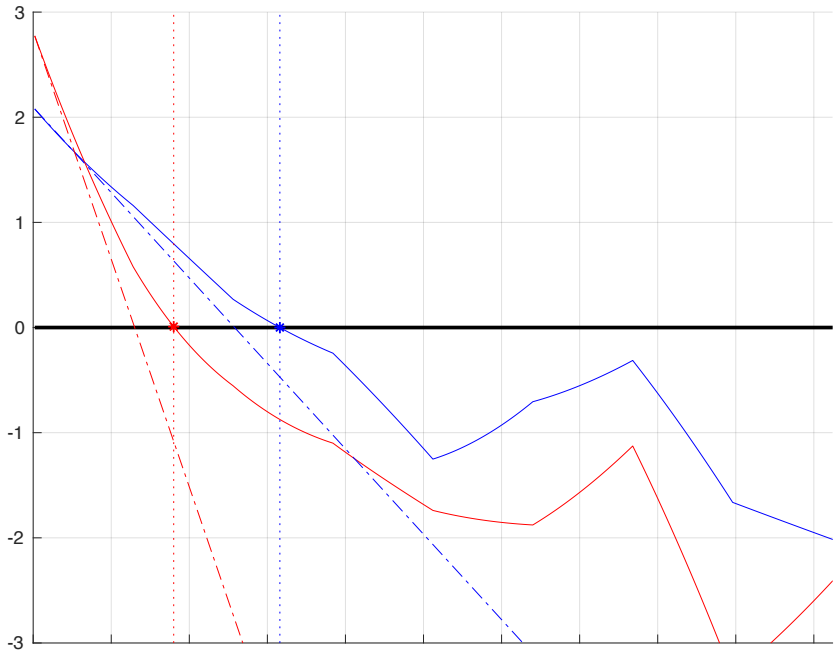


Figure 2.5 – Illustration of DeepFool. The blue and red curves correspond to the objective function $\mathcal{J}(\mathbf{x}, \ell)$ for two different classes ℓ_1 and ℓ_2 when the perturbation $\mathbf{r}$ is co-linear to their gradient. The x-axis corresponds to the norm of $\mathbf{r}$. Note that $p(\ell_g) = 0.8$, $p(\ell_1) = 0.1$, $p(\ell_2) = 0.05$. Since $p(\ell_1) > p(\ell_2)$, it seems interesting to target class ℓ_1: the blue objective function starts from a lower value. This is a mistake because it cancels "further" than the red ℓ_2 class. To find this out, DeepFool calculates the gradient in $\mathbf{x}$, which is equivalent to approaching the objective function by its tangent in $\|\mathbf{r}\| = 0$ (dotted line).

With DeepFool, Moosavi-Dezfooli shows that this reasoning is wrong. How easy it is to make the objective function negative depends not only on its initial value but also on its slope. In the first order for a targeted class, according to Equation 2.11, the minimum distortion necessary in L_2 norm is reached when $\mathbf{r} \propto -\nabla_{\mathbf{x}} \mathcal{J}(\mathbf{x}, \ell)$ with:

$$\|\mathbf{r}\| = \frac{\mathcal{J}(\mathbf{x}, \ell)}{\|\nabla_{\mathbf{x}} \mathcal{J}(\mathbf{x}, \ell)\|}. \tag{2.21}$$

It is best to target the class ℓ that will cause the addition of the smallest distortion. This is illustrated in Figure 2.5. But this formula is only a first-order approximation. Moreover, it must be estimated for all or a subset of classes except of course the class ℓ_g.

ILC. *Iterative Least-likely Class (ILC)* [PFC+18] proposes another choice of objective function. The class assigned to the attacked image can be sometimes semantically close to the original ℓ_g class. An image of a swallow taken for an image of a sparrow seems to us more harmless than if the same image of a swallow is taken for an image of a car. Thus, Iterative Least-likely Class (ILC) prefers to target the least likely class for the original image.

JSMA. Papernot and his colleagues propose a targeted attack for low distortion in L_0 norm. The attack discovers pixels that play an important role in the classification [PMJ+16]. This approach called *Jacobian-based Saliency Map Attack (JSMA)* estimates the Jacobian matrix of the function $\mathbf{x} \rightarrow \mathbf{p}$. This calculation determines which elements of $\mathbf{x}$ have the most influence not only to increase the predicted probability of the target class $p(\ell)$ but also to decrease the predicted probabilities of all other classes.

Few pixels have this property, but modifying them is extremely effective in luring the classifier. However, they must be modified with a large amplitude, which produces a "salt and pepper" noise in the image. This modification is often very visible but can pass for an error in the coding of the photo.

This technique is remarkable because it is quite fascinating that changing the value of a few pixels or even a single pixel is enough to cause misclassification.

2.5.5 Other attacks

Some other attacks are in the same vein but with variations either on the definition of the objective function or on the definition of the distortion. Finally, this panorama of attacks closes with a description of some techniques that take different paths to achieve their goals. They are separate because it is not easy to classify them into one of the two families presented above.

Universal attacks. Moosavi-Dezfooli and colleagues have shown that it is possible to create a single adversarial perturbation that works regardless of the image feed to the network [MFFF17]. To do this, they repeatedly apply the DeepFool algorithm (see section 2.5.4) to all images of the training data until a particular perturbation causes a large portion of the images to be misclassified. More formally, their approach looks for the perturbation $\mathbf{r}$, bounded by ϵ, such as:

$$\begin{aligned} &\min \|\mathbf{r}\|_p \leq \epsilon \\ &\text{such that } \mathcal{P}_{\mathbf{x} \sim P_{data}}(f(\mathbf{x}+\mathbf{r}) \neq l_g) \geq 1-\delta, \end{aligned} \tag{2.22}$$

where, δ indicates the proportion of images of the training data that have become adversarial images and belong to the P_{data} sample of all images.

Overall, the algorithm succeeds in finding unique adversarial perturbation, often visually very different from each other, thus facilitating universal attacks.

Geometric attacks. So far the attacks change the pixel values additively: $\mathbf{y} = \mathbf{x} + \mathbf{r}$. They are sometimes called value-metric attacks. Geometric attacks do not change the pixel value but their position by slight local rotations and translations. An optical flow applies a displacement field to the pixels of the original image: the pixel at position (k, l) is moved to position $(k, l) + \Delta(k, l)$ in the adversarial image.

Xiao and his colleagues seek to optimize this optical flow by observing the variations in classification probabilities [XZL+18]. The objective function integrates $\mathcal{J}(\mathbf{x}, \ell)$ and a part regulating the optical flow so that it generates small continuous displacements. Once again, a numerical method like L-BFGS is used. Adversarial images often look perfect.

Attacks by generative networks. Generative Adversarial Networks (GAN) [GPAM+14] are a class of glsml algorithms that learn to estimate a probability distribution from samples submitted to them. The learned distribution forms a model that the network can then use to generate new samples that are completely synthetic but belong to the same distribution. Training with a very large collection of images of faces of existing people, a generative network can then synthesize new artificial but realistic faces.

GANs are composed of two distinct parts, a generator that learns the distribution and generates a new sample, and a discriminator that estimates whether the sample it is observing comes from the generator or directly from the training set. The discriminator shares its information with the generator. These two parts compete against each other, in that the generator tries to create an artificial sample that the discriminator will not be able to distinguish from a real sample. The discriminator, therefore, forces the generator to improve the quality of the synthesis.

Based on this general idea, it seems natural to use these generative networks to generate adversarial images. The generator creates adversarial images that appear legitimate to us. This is what Baluja and Fischer [BF17], for example, do. They train a generator to create images that deceive a particular network by modifying them via a residual network [HZRS16a].

We are therefore very far from the gradient calculation mechanism of the first attacks. The advantage of this generative approach is to produce an adversarial image in real-time. But it also suffers numerous disadvantages. The training time is very long. This approach only makes sense

if the attacker has a large number of adversarial images to create. Moreover, a learned network only targets a given class and is only valid against a particular classifier.

2.6 Defenses

There are as many defenses as attacks. This section gives an overview of them. Schematically, we can distinguish three families of defenses:

— **Reactive techniques**: these techniques are based on some pre-processing that is carried out upstream of the network. These either block images if adversarial content is detected or filter and clean the images, hoping to remove the adversarial perturbation.

— **Proactive techniques**: These techniques build networks that are inherently more robust to adversarial attacks. Approaches that add many adversarial images during the learning phase are examples in this category.

— **Obfuscation Technique**: These techniques mask or blur important parameters that an attacker needs to generate adversarial images.

Another point of view is to distinguish whether the defense is an added module connected to the network (and thus the classifier works with or without defense), or whether the defense is an integral part of the network resulting in a profound transformation of the classifier.

2.6.1 Reactive defenses

This family includes techniques that detect the adversarial image and/or apply pre-processing to images submitted to the network in order to remove content that makes them adversarial.

Detection techniques introduce an additional class other than the existing classes of the classification. This class is not necessarily labeled "adversarial images", but simply "suspicious class". The fundamental hypothesis is that the attacker's goal is not to lead the classifier to recognize adversarial images as this suspicious class. The attack fails if the adversarial image is predicted as an image from this class. Detection is sometimes justified as follows: i) images are points in a large $[0, 1]^m$ space concentrated along some manifolds, ii) attacks push these images out of their manifold. Detectors learn to distinguish these manifolds by collecting statistics computed either in the image domain or in the hidden layers.

The pre-processing filters the images to remove the adversarial perturbation without altering the visual content. Here, the images are never rejected, they are hopefully clean and therefore harmless. In theory, filtering amounts to projecting an image onto the manifolds of natural images

mentioned above. Once again, some technologies filter images before classification, others filter the representations that go through the networks.

In reality, reactive techniques mix pre-processing and detection. It is possible to build a detector from pre-processing by measuring the amount of filtered noise in the image. We list examples of reactive defenses without any clear distinction.

Learning the manifold of natural images. is the challenge of many defenses. The advantage is that the learning phase consumes only pristine images. Thus, the defense is not biased towards one or more specific attacks. MagNet [MC17] uses auto-encoders to project the image and bring it closer to the manifold of natural images. Variations use sparse representations of image patches such as D3 [MDST18], estimations based on Gaussian mixtures [GLB19], or generative adversarial networks such as PixelDefend [SKN+17] or Defense-GAN [SKC18]. Less mainstream, the authors of [DMY+19] search on the Internet for the images most similar to the query, then decide its class by a majority vote on the predictions of similar images.

Interaction with the classifier. A simple detection method is Feature Squeezing [XEQ17]. Many simple filters are applied to degrade or simplify the image, hence the name "squeezer" (compression, slight blurring filter, ...) before submitting it to the network. Then, deviations at the output of the network are observed with respect to the predicted probability vector given for the original image. Any significant deviation suggests that the tested image is adversarial. The papers [GRCvdM17, LLS+18] are very similar. SafetyNet [LIF17] is based on the analysis of active neurons in the classifier. They encode the typical activation patterns of a deep layer of a network processing clean images and compare this description to the current description when processing an unknown image. This comparison is done via a radial kernel SVM. Bypassing this defense forces the attacker to integrate the response of all the normalized units of the network into the attack, which is difficult in practice. This defense works well even when the scale is large. A more recent version of this defense idea is called *Network Invariance Checking (NIC)* [MLT+19].

2.6.2 Proactive Defenses

Proactive defenses aim to improve the intrinsic robustness of the models. They attempt to propose a better architecture or new training process so that both the accuracy and the robustness of the neural network are improved at the same time [XTG+20, SNG+19].

Reducing the amplitude of the gradients. If the loss function of the network produces strong gradients, then a very small perturbation is enough to greatly alter the output of the network. This explains the vulnerability of the network to attacks. Reducing the amplitude of the gradients makes the network more robust.

One of the earliest approaches is "distillation", which is originally a technique for transferring what was learned from a large network to a smaller network [HVD15]. Very roughly, distillation trains the small network not with the image labels but with the probability vectors predicted by the large network, which are more informative than simple labels. Papernot and his colleagues [PMW$^+$16] rely on distillation but apply this transfer on the same network architecture. Thus, the first version of the network is trained on labels, the second on the knowledge learned from the first. This "auto-transfer" is done at high temperature in the softmax function, which reduces the amplitude of the gradients of the loss function of the network. Nevertheless, Carlini and Wagner have designed attacks that fail defenses by distillation (see the C&W attack, section 2.5.4).

Gu and Rigazio propose to train networks with a new constraint: each layer must be "contracting" in the sense of a Lipschitzian function [GR14, TSS18]. This is integrated during training by a penalty that aims to reduce the variation of its response to perturbations that it receives at the input. Overall, this increases the robustness of the network and requires that the applied distortion be significantly higher for an attack to succeed.

Adversarial training. Training with more data allows a network to better generalize, to refine the boundaries between classes in the representation space. Data augmentation is a classical trick adding near copies of images that have undergone small translations or rotations.

The idea is the same here by augmenting training with adversarial images. The network thus learns that despite the perturbation, such an image is in such a category. The principle is simple, but the implementation is difficult. An attack targets a network, which during training is by definition evolving. Each time the weights of the networks are updated, the adversarial versions of the training images must be recalculated. The attack must therefore be ultra-fast. This is why Goodfellow and his colleagues [GSS14] proceed with the simplest of attacks: FGSM.

This idea leads to the concept of robust optimization through a min-max formulation where the learning objective is:

$$\min_{\boldsymbol{\theta}} \sum_{j} \max_{\mathbf{r}_j \| \|\mathbf{r}_j\| < \epsilon} \mathcal{J}(\mathbf{x}_j + \mathbf{r}_j, \ell_j, \boldsymbol{\theta}), \tag{2.23}$$

where $\{\mathbf{x}_j, \ell_j\}$ is the training dataset.

In a way, the learning process tries to make all the images in the ball with center $\mathbf{x}_j$ and radius ϵ be classified as $\mathbf{x}_j$. We can also cite various works exploring these same ideas [HXSS15, MMS+17, TKP+17]. Let us also mention the approach of Lee and colleagues [LHL17] where a generative network creates adversarial images that feed a classifier performing adversarial training.

There are still a lot of poorly investigated details about adversarial training. The robustness brought in is sometimes contested. The network is more robust against simple attacks, but still vulnerable to more complex attacks. The robustness is obvious on training pictures but it does not generalize well. A price has to be paid: the network is more robust against attacks but less accurate on the original images. The consensus has not yet been established with certainty because adversarial training is tricky to carry out. Many variations exist, gradually increasing the number of adversarial images while increasing the strength of the attacks from a very large number of original images. All this is expensive, the benefits do not always compensate for the extra costs.

2.6.3 Obfuscation technique

The creation of adversarial images in a white box setup is usually based on the exploitation of the gradient of the differentiable loss function. Introducing strong non-linearities makes the network non-differentiable and blocks the calculation of a gradient. This family of techniques has been explored by Goodfellow and colleagues (see [BRRG18]). Athalye and colleagues [ACW18] have explored this subject and show the inefficiency of this approach: in the white box setup, nothing obliges the attacker to use the gradient of the objective function. The network can be modified and any non-linearity is replaced by a smoother function.

The defense becomes more serious when it is based on the insertion of a secret key into the classifier, as in cryptography. This is the only way to block white box attacks. The attacker knows all the details of the network except a high entropy secret parameter. This is difficult to combine with ML and often requires re-training the network from scratch or in part each time a secret key is drawn [SZMA18, TRHV20].

Another possibility is to make the classifier random. For each call to the network, the final prediction depends on a random seed that the attacker cannot know. This may be a slight modification of the input image or modifications inside the network: Dhillon and his colleagues [DAL+18] propose to remove randomly some neurons (those that react weakly) and to increase proportionally the importance of the reaction of the preserved neurons.

2.7 Positioning

After providing a holistic overview and the background to the field, we illustrate our viewpoint towards the existing work and position our contributions inside the field.

2.7.1 Challenges

In this chapter, we mention many attacks and defenses separately, however, researches on both sides do not develop alone. Most defenses measure their robustness based on attacks [MMS+17], while attacks are proposed to crack existing defenses [ACW18]. It is like a game between attacks and defenses. They learn from each other and improve themselves. The other works studying the theories or improving the evaluation of robustness also provide extra information for defenses and attacks to augment their performance. As the game between attacks and defenses continues, we get better attacks and defenses and new knowledge about neural networks.

When it comes to producing adversarial examples, the important question is what kind of adversarial examples are worth to generate. Attackers and defenders have different answers to this question.

For attackers, all adversarial examples matching the definition in Equation 2.4, *i.e*. misleading neural networks with invisible perturbation, are perfect. As we discussed in the previous sections, the definition of the adversarial problem is not certain. There is still space to explore. For instance, how to define and measure invisibility is still an open question. Most of the existing works relate the invisibility to magnitude, and they measure the invisibility by minimizing the L_2 norm of the distortion. Perturbations of small magnitude are indeed more likely imperceptible, however, it is a practical definition of invisibility for machines rather than a definition depicting truthfully invisibility for humans. Some existing works define invisibility as smoothness in the sense of low frequency [GFW18, HZJ18], however, sometimes it is still visible to humans.

Defenders attach more importance to adversarial examples which give them extra information to improve the robustness of neural networks. Adversarial training [GSS14, MMS+17], as we mentioned in section 2.6, includes adversarial examples as a part of training data. That improves the robustness and provides extra information on how neural networks learn more precise boundaries. However, experiments show that adversarial training gains robustness against attack used to generate adversarial examples during the training phase, while being still vulnerable to other unknown attacks. Furthermore, adversarial training also loses accuracy on legitimate examples. In our point of view, it is due to adversarial training using adversarial examples generated by FGSM or PGD. These target distortion attacks cannot certify the success rate by

design. As a result, in practice, a large ϵ is chosen for adversarial training. These adversarial examples with large distortion introduce noises and might cause oscillation during the training. In this perspective, adversarial training needs adversarial examples that are closest to the original data point so that they could correct boundaries more cautiously.

For both attackers and defenders, speed is always important when producing adversarial examples. Attackers do not want to wait days to attack the target classification system. Training for neural networks is not cheap in general, if it takes hours to generate adversarial examples, adversarial training is not feasible for defenders. However, the optimization problem of producing adversarial examples is not easy to solve directly. It includes a box-constrain, *i.e.* $(\mathbf{x} + \mathbf{r})$ $is\ an\ image$ in Equation 2.4 and the complex neural network is part of the optimization. When we directly solve it via an existing optimization algorithm [CW17, SZS$^+$13], it is time-consuming to find optimal solutions. It is true that methods like DeepFool [MDFF16], FGSM [GSS14], PGD [MMS$^+$17], *etc*., speed up the algorithm with some approximation, however, adversarial examples generated by these approaches are not optimal solutions. It is worth investigating how to improve the efficiency of attacks producing optimal adversarial examples.

When it comes to defending adversarial examples, we need to define what is a good defense. We have mentioned many defenses; there are many others, sometimes simple variations, sometimes more original contributions as well. In general, the evaluation of their effectiveness leaves a series of questions. Many of them are evaluated on very small test sets, resist only this or that class of attacks, are too expensive to be usable in practice, *etc*. There is not yet a rigorous protocol to evaluate the quality of a defense technique that makes a network more robust to adversarial attacks. It is very difficult to compare the respective merits of different defensive strategies.

Even though it is difficult to evaluate what is a good defense quantitatively, we can define an ideal defense qualitatively. An ideal defense should not degrade the network's quality and improve the network's robustness. The addition of defense strategies sometimes leads to a degradation of the network's quality: the classification performance on natural (unattacked) images of a defended network is worse than that of a defenseless network. This observation is contested because nothing implies, fundamentally, such a tension (even if some theoretical papers claim the contrary). Recently, there exist works to improve both the quality and the robustness of networks by changing the training scheme [XTG$^+$20, SNG$^+$19]. AdvProp [XTG$^+$20] uses separate auxiliary batch norm for adversarial examples, as they have different underlying distributions than normal examples. The "free" adversarial training [SNG$^+$19] updates the network parameters while generating adversarial examples. For each iteration, the network is updated by the gradient with

respect to the network parameters while the adversarial perturbation is updated by the gradient with respect to the input. It overheads the cost of generating adversarial examples by recycling the gradient information computed when updating model parameters. However, both defenses need to train a network from scratch, which is expensive.

On the other hand, an ideal defense should be robust under different settings and not too costly. In section 2.6, we categorize defense works as reactive defenses and proactive defenses. The reactive defense introduces an external model, normally a transformation module or a detector, to protect the network away from the adversarial attack. Reactive defenses are cheap to apply but most of them are proven vulnerable when attackers are aware of their existence. In the white-box setting, the detector is easily regarded as a part of the classification system and applied backpropagation to acquire gradients for generating undetectable adversarial examples. For transformations that are non-differentiable, attackers consider them as part of the classification system in the forward while replacing them with identity function in the backward [ACW18]. In this way, these defenses are also vulnerable. Proactive defenses improving the intrinsic robustness of the models are normally expensive because they need to train the neural network from scratch when the architecture of the network or the training process is changed. These defenses are either only proposed by the company with plenty of GPU like Google, and FaceBook, either only verified on small datasets like MNIST, and CIFAR.

Some approaches opt for a more formal point of view and try to certify the robustness of the network as long as the distortion remains under a limit whose value must be calculated. Still, in their infancy, we nevertheless cite the studies that have been carried out [WK17, RSL18, KBD$^+$17, HKWW17, RHK18, SND17].

To answer these questions, we make our assumptions, verify them through experiments and then correct them according to our observations.

2.7.2 Our approaches and contributions

We start with our definition of *invisibility*. Adversarial images produced with existing attacks are not invisible. When we enlarge adversarial images, the perturbations are obvious even with small magnitudes. The situation is even worse if we magnify images with a local kernel at each point. By investigating this case, we find that adversarial perturbations are perceptive due to independent perturbations neighboring pixels. Humans are sensitive to high-frequency information in images. In this perspective, we enforce invisibility by making perturbation similar over neighbor pixels. To do this, we build a graph to maintain the pixel similarity of the original image. This succeeds in producing *smooth adversarial examples* that not only are smooth

according to our definition but also achieve better performance concerning success rate and distortion. This brings new knowledge to the type of adversarial examples we can produce.

Besides, when we compare our attack to others, we realize we can improve the *evaluation protocol*. Success rate and L_2 norm of distortion are usually treated as two important criteria. Normally, we compare the value of one of them when the other is fixed. We argue that it is limited. For instance, if we compare the success rate under an extremely large (or small) distortion, two attacks might give very similar results, *i.e.* around 100% (or 0%). Moreover, it is hard to compare the attacks from target distortion attacks to those from target success attacks. As we mentioned in section 2.5, target distortion attacks set the threshold of distortion so that we compare the success rate to measure the capability of these attacks; while target success attacks to aim to attack successfully so that we compare the average distortion. If the parameter ϵ for the target distortion attacks like PGD [MMS$^+$17] is optimized, the target distortion attacks become target success attacks. In this way, we propose a fair evaluation protocol to compare attacks from the two different families.

When we implement the algorithm to generate smooth adversarial perturbation, we build on C&W [CW17], and we suffer from its inefficiency. In practice, the reason why C&W is not efficient enough is due to the redundant optimization algorithm. It applies a linear search to find a decent parameter λ in Equation 2.18. For each parameter, Equation 2.18 gives a different optimization problem and C&W applies the existing optimization algorithm Adam with 10000 maximum iterations to solve the optimization problem. DDN [RHO$^+$19] proposes an attack combining the gradient of the neural network and the distortion. It is efficient compared to lots of targeted success attacks. However, it only uses the gradient as the searching direction that leads to oscillations around the boundary and, as a result, wastes the computation.

Loss and distortion have different importance during the attack process. When the current solution is still in the region of the original class, we attach more importance to the loss. When the current solution crosses the boundary and becomes adversarial, the target becomes to reduce the *distortion* while maintaining misclassification. To achieve that, we propose a *Boundary Projection (BP)* that searches adversarial examples along the boundary. Experiments show that BP attack is very efficient. It provides *speed*, small magnitude of *distortion* and high success rate.

On the other hand, we also notice the *quantization* problem of adversarial images. When adversarial images are generated inside a neural network, they are represented as continuous matrices in the range $[0, 1]$. However, true images are represented in a discrete format in the range $0, 1, \cdots, 255$. Plenty of attacks evaluate their performance without saving images, and this makes a difference in the performance. We should not ignore the difference caused by quantizing

the adversarial images onto true images.

We also work on defense. We investigate how adversarial perturbation performs inside a neural network, and try to eliminate the adversarial effects through a cheap model. We find that in the different layers of neural networks, random noises have different behavior than adversarial perturbation. On images or in the first layer, neural networks can withstand random noises. Different from the defenses based on transformation using clean training data [MDST18, GRCvdM17], *patch replacement* removes adversarial effects by replacing the inputs by untouched training data. It gives us more chance to improve the robustness while maintaining the accuracy on legitimate images. Besides, since there is no need to train the neural network from scratch, the defense is not expensive. The usage of the patches of untouched data in the testing phase increases the complexity of the attack even if the attacker is aware of our defense. This increases the robustness in white/gray-box settings. Experiments show this defense performs better on adversarial attacks with small distortion.

Our works make the field of adversarial attacks and defenses more complete. We give a new definition of the invisibility of adversarial perturbation, propose an efficient optimization algorithm to solve adversarial problems, explore the defense on the features, propose a fair evaluation protocol, and underline the quantization problem of adversarial images. We not only get better results, and improve the performance but also bring new knowledge.

PART I

Attack

CHAPTER 3

EVALUATION

The previous chapter includes an overview of some well-established adversarial attack strategies. They are diverse in their assumptions, in their approaches. However, most of them try to do their best to minimize the magnitude of distortion existing in successful adversarial samples. In other words, all contributions aim at achieving the lowest *distortion* and highest success rate.

We were among the first to consider *invisibility* and *speed*. The contributions about adversarial attacks presented in this manuscript have the same two targets. On the one hand, we propose a new technique to craft adversarial samples where the distortion is hardly visible – we target here *invisibility*. This technique is described in detail inside the chapter 4. On the other hand, we propose another technique that is very efficient to create adversarial samples – we target here *speed*. This technique is described in detail inside the chapter 5.

To compare our contributions to the existing solutions, we need evaluation metrics that refer to the quality of adversarial images, *i.e*. the success of the attacks and the resulting visual distortion, *etc*.

Before we introduce these evaluation metrics, we also need to clarify some publicly available datasets as well as some off-the-shelf network architectures that we use in our experiments. We also would like to elaborate on the quantization problem of adversarial images, *i.e*. the quantization applied to adversarial images when saving them into image files might affect the quality of adversarial images.

Last but not least, we introduce the standard evaluation metrics that mainly focus on the success rate and magnitude of the adversarial perturbation. We propose our evaluation metrics that improve the standard evaluation. We also discuss other evaluation metrics that refer to speed and invisibility.

3.1 Datasets

We basically test on three public and widely used datasets, *i.e*. MNIST [LCB10], CIFAR10 [Kri09] and ImageNet [DDS+09].

MNIST is a dataset consisting of 28×28 grayscale images of handwritten digits, *i.e*. range from 0 to 9. It contains $60,000$ training images and $10,000$ testing images.

Figure 3.1 – Here are ten examples of MNIST from different classes.

CIFAR10 is a collection of low-resolution (32×32) color images in 10 classes, *i.e*. airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. It contains $60,000$ training images, in which $6,000$ images of each class, and $10,000$ testing images uniformly from 10 classes.

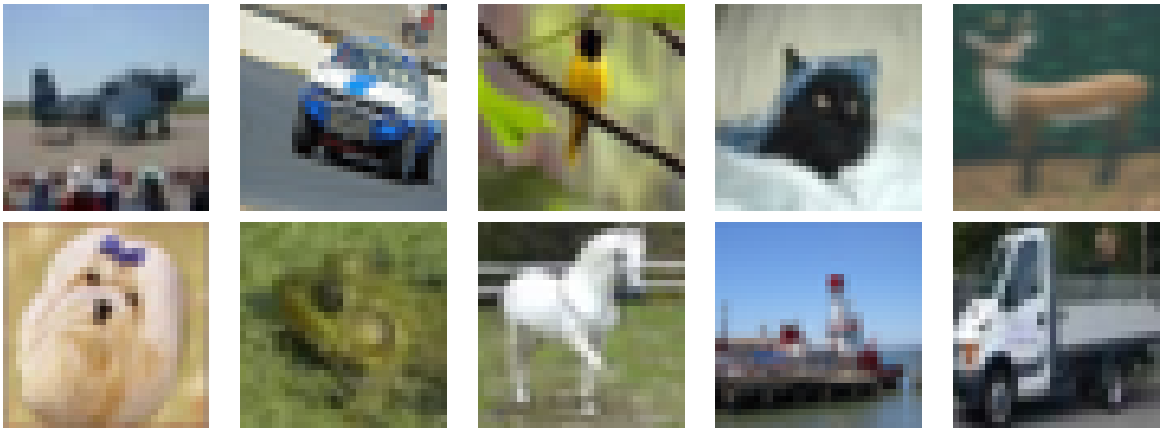


Figure 3.2 – Here are ten examples of CIFAR10 from different classes.

ImageNet indicates the dataset from the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2012)*. There are $1,000$ categories, with a typical category, such as "balloon" or

"strawberry". It contains $10,000,000$ images as the training data, $50,000$ images as validation data, and $100,000$ images as test data. The images vary in resolution, and the average image resolution on ImageNet is 469×387 pixels. Normally, it is necessary to apply a pre-processing that down samples them to the same resolution: $299 \times 299 \times 299 \times 3$.

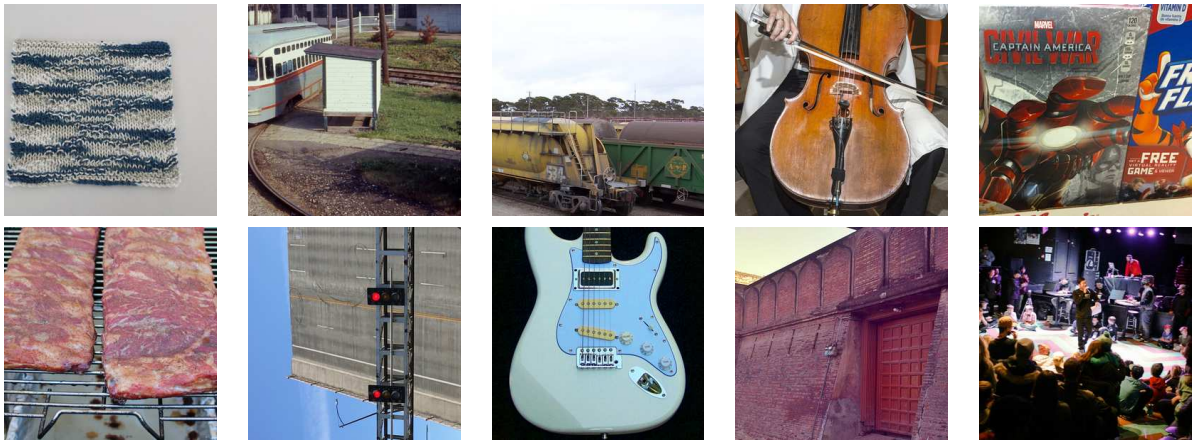


Figure 3.3 – Here are ten examples of ImageNet from different classes.

For our experiments, we follow the setting of adversarial attacks and defense competition [KGB+18a] where $1,000$ images are used as the test data to evaluate the quality of adversarial attacks. These $1,000$ images are well-selected, they are easy to classify to networks. Networks achieve high accuracy on this test set. The exactly same $1,000$ images from the competition are used in chapter 4 and chapter 5. We call it the *test set for attacks*. The attacks are built on TensorFlow [ABC+16], where the pre-trained models are trained on images with size $299 \times 299 \times 3$.

3.2 Networks

To reproduce the experiments, we list here adequate details of the networks we use. We first introduce the off-the-shelf networks used as target network. Then we introduce the robust models against attacks. We use different models on different dataset.

3.2.1 Off-the-shelf network

MNIST [LCB10]. We use a simple network with three convolutional layers and one fully-connected layer. The first convolutional layer has 64 feature kernels (or filters) of size 8 and

strides 2 [1]; the second has 128 feature kernels of size 6 and strides 2; the third has 128 feature kernels of size 5 and stride 1. It uses LeakyRelu activation [MHN13]. The loss function is the cross-entropy.

We train the network with the $60,000$ images of the training set. After a random initialization, training lasts 6 epochs with batch size 128, learning rate 0.001, and the optimizer is Adam. Between epochs, the training data are shuffled. This simple network achieves an accuracy of 0.99. This setting is following an example of Cleverhans [PFC+18] that gives decent results.

CIFAR10 [Kri09]. We use a simple CNN network with nine convolutional layers, two max-pooling layers, two dropout layers, ending in global average pooling, and a fully connected layer. Batch normalization [IS15] is applied after every convolutional layer. It also uses LeakyRelu. The loss function is the cross-entropy. All the kernels of the convolutional layers are initialized with the HeReLuNormal initializer, and their kernel size is 3. For the first three convolutional layers, the number of filters is 128 and the padding mode is 'same' [2]. For the following three convolutional layers, the number of filters is 256 and the padding mode is 'same'. For the last three convolutional layers, the padding mode is 'valid'. The seventh layer has 512 filters, while 256 filters for the eighth layer, and 128 filters for the last layer. The parameter α for LeakyRelu is 0.1, and the rate for dropout is 0.5. The pool size of the max-pooling layer is 2, the shape of the stride is 2 and the padding mode is 'valid'. The pool size of the average pooling layer is 2 and the shape of the stride is 6 and the padding mode is 'valid'.

We trained the model with the $60,000$ images of the training set. After a random initialization, training lasts 200 epochs with batch size 128, learning rate 0.001, and the optimizer is Adam. Between epochs, the training data are shuffled. Its accuracy is 0.927.

ImageNet [DDS+09]. We use InceptionV3 [SVI+16] and ResNet V2-50 [HZRS16b] with the pre-trained model from TensorFlow-Slim image classification library [3], whose accuracy are 0.96 and 0.93 respectively on the test set for attacks; 0.78 and 0.76 on random images.

1. Here, the stride is a parameter that sets the shift of the kernel. For instance, if a stride is set to 1, the kernel will move one pixel, or unit, at a time.

2. The padding is needed when the size of filters does not perfectly fit the input. Two modes of padding are used in practice, *i.e.*'same' and 'valid'. The 'same' mode tries to add extra columns and rows to input with value zero when the input size does not fit the filter size. The 'valid' mode tries to drop the columns and rows of input that can not fit the filter.

3. `https://github.com/tensorflow/models/tree/master/research/slim`

3.2.2 Robust models

We use the same network for MNIST and CIFAR10, but the training differs. We follow the adversarial training methods [MMS+17, GSS14], *i.e.* the models are trained with training data and their adversarial version generated by PGD_2 or FGSM.

MNIST. On MNIST, it is trained from scratch with the same setup but with training data and their adversarial examples. For adversarial training, PGD_2 iterates 40 times with $\epsilon = 0.3$ while $\alpha = 0.01$. The training batch size is 50 over 100 epochs. [4] FGSM defense model uses FGSM with $\epsilon = 0.3$. We also train adversarial training models with DDN and BP. DDN and BP defense model use these attacks with 20 iterations and the same parameters as described in the attack methods.

CIFAR10. On CIFAR10, FGSM and PGD_2 defense models are trained from scratch. FGSM defense model uses FGSM with $\epsilon = 0.3$. PGD_2 iterates 100 times, with $\epsilon = 8/255$ while $\alpha = 2/255$. The training batch size is 128 over 200 epochs. [5] Between epochs, the training data are shuffled.

For DDN and BP defense model, we follow the training suggested by [RHO+19]. The model is first trained on clean examples, then fine-tuned for 30 iterations with adversarial examples. The parameters are initialized randomly. The optimizer is Momentum Optimizer, the initial learning rate is 0.001 and the momentum is 0.9. It is trained with 200 epochs and the batch size 128. Between epochs, the training data are shuffled. DDN and BP defense model use these attacks with 20 iterations and the same parameters as attack methods.

ImageNet. The robust model for ImageNet is ensemble adversarial training [TKP+17], which is directly taken from TensorFlow library [6].

Experiments of attacks run on TensorFlow1.8.0-py2.7 over CUDA 9.0.176; Cleverhans [PFC+18] produces the existing attacks [7].

4. This parameter setting is from MadryLab (`https://github.com/MadryLab/mnist_challenge`) following the paper "Towards Deep Learning Models Resistant to Adversarial Attacks".

5. This parameter setting is from MadryLab (`https://github.com/MadryLab/cifar10_challenge`) following the paper "Towards Deep Learning Models Resistant to Adversarial Attacks".

6. `https://github.com/tensorflow/models/tree/master/research/adv_imagenet_models`

7. Cleverhans v1.0.0 is used for chapter 4. Cleverhans v2.0.0 is used for chapter 5. When we worked on experiments in chapter 4, only Cleverhans v1.0.0 existed and it was a reliable tool to implement adversarial attacks. When we started the work in chapter 5, Cleverhans upgraded to Cleverhans v2.0.0. They improved their codes and the old version was covered. So we shifted to the higher version.

3.3 Evaluation metrics

This section presents the evaluation metrics used to evaluate the quality of adversarial images. We start with the standard evaluation metrics, *i.e.* success rate, and distortion. We discuss the metrics and propose our new evaluation metrics.

3.3.1 Standard evaluation metrics

According to the definition of adversarial images (see Equation 2.4) two criteria to evaluate their quality: distortion and probability of its success.

These two criteria are intimately linked by a trade-off. It is easy to build an attack that always succeeds: it is the attack that replaces $\mathbf{x}$ with a completely different image $\mathbf{x}'$ whose predicted class is ℓ, but the distortion $\|\mathbf{x}' - \mathbf{x}\|_p$ is huge and clearly visible to the naked eye. It is easy to build a zero distortion attack: it is the attack that substitutes $\mathbf{x}$ for the same $\mathbf{x}$, but the probability of success is zero (unless $\mathbf{x}$ is misclassified by the network). These two extreme attacks are of no interest, but they illustrate this trade-off.

It is respectively difficult to compare a target distortion attack (see subsection 2.5.3) to a target success attack (see subsection 2.5.4). Target distortion attacks and target success attacks attach different importance towards success rate and distortion. It is instinctive to fix the success rate and compare the minimum distortion in order to measure the capability of target success attacks. On another hand, fixing the upper bound of distortion (by setting parameter ϵ), the success rate indicates the capability of target distortion attacks. It is hard to control the success rate for target distortion attacks while it is hard to control the distortion for target success attacks. Besides, the selection of the value of reference distortion or success rate might give different perspectives.

3.3.2 Our evaluation metrics

It is difficult to define a fair comparison of attacks targeting distortions and attacks targeting success. For the first family, we run a given attack several times over the test set with different target distortion ϵ as we described in section 2.5. The attack succeeds on image $\mathbf{x}_o \in X$ if it succeeds on any of the runs.

Given a test set of N' images, we only consider its subset X of N images that are classified correctly without attack. The accuracy of the classifier on pristine images is thus N/N'. Let X_{suc} be the subset of X with $N_{\text{suc}} := |X_{\text{suc}}|$ where the attack succeeds and let $D(\mathbf{x}) := \|\mathbf{x} - \mathbf{y}\|$ be

the distortion for image $\mathbf{x} \in X_{\text{suc}}$ where $\mathbf{y}$ is the closest adversarial example the attack succeeded to forge. The global statistics are the *success probability* P_{suc} and *conditional average distortion* $\overline{D}$

$$P_{\text{suc}} := \frac{N_{\text{suc}}}{N}, \quad \overline{D} := \frac{1}{N_{\text{suc}}} \sum_{\mathbf{x} \in X_{\text{suc}}} D(\mathbf{x}). \tag{3.1}$$

Here, $\overline{D}$ is conditioned on success. Indeed, distortion makes no sense for a failure.

We define the *operating characteristic* of a given attack over the set X as the function $\mathsf{P} : [0, D_{\max}] \to [0, 1]$, where $D_{\max} := \max_{\mathbf{x} \in X_{\text{suc}}} D(\mathbf{x})$. Given $D \in [0, D_{\max}]$, $\mathsf{P}(D)$ is the probability of success subject to distortion being upper bounded by D,

$$\mathsf{P}(D) := \frac{1}{N} |\{\mathbf{x} \in X_{\text{suc}} : D(\mathbf{x}) \leq D\}|. \tag{3.2}$$

This function increases from $\mathsf{P}(0) = 0$ to $\mathsf{P}(D_{\max}) = P_{\text{suc}}$. If we choose one intermediate point with upper bounded distortion $D_{\text{upp}} \in (0, D_{\max})$, then the relative success rate is $P_{\text{upp}} := \mathsf{P}(D_{\text{upp}})$. This is equivalent to standard evaluation metrics.

3.3.3 Other evaluation metrics we introduce

MAD. According to the paper [FBHD19], that compares fifteen metrics (including Structural Similarity Index Measure (SSIM), Peak Signal-to-Noise Radio (PSNR), and Weighted Peak Signal-to-Noise Radio (wPSNR)) to the subjective perceptual evaluation of a panel of users, *Most Apparent Distortion (MAD)* [LC10] is the metric best reflecting user assessment about the quality of adversarial images.

MAD attempts to explicitly model two separate measures of perceived distortion, *i.e*. one for near-threshold distortions in high-quality images and another for suprathreshold distortion in low-quality images. MAD uses a weighted geometric mean of the two different distortions to capture the interacting strategies of measurement. A low MAD score means better fidelity. It is used in chapter 4 to evaluate the invisibility of smooth adversarial perturbation.

Complexity. The complexity of an attack measured by its memory consumption or by the computing time required is also an important criterion. The attacks listed in section 2.5 are often iterative. Counting the number of iterations that are necessary to produce a good visual quality adversarial image is a valuable indicator more reliable than memory or runtimes as it is independent of the software implementation and the hardware.

The lower the complexity, the faster the attack, but then often the probability of success is

low or the distortion is high. It is used in chapter 5 to evaluate the efficiency of BP attack.

CHAPTER 4

SMOOTH ADVERSARIAL EXAMPLES

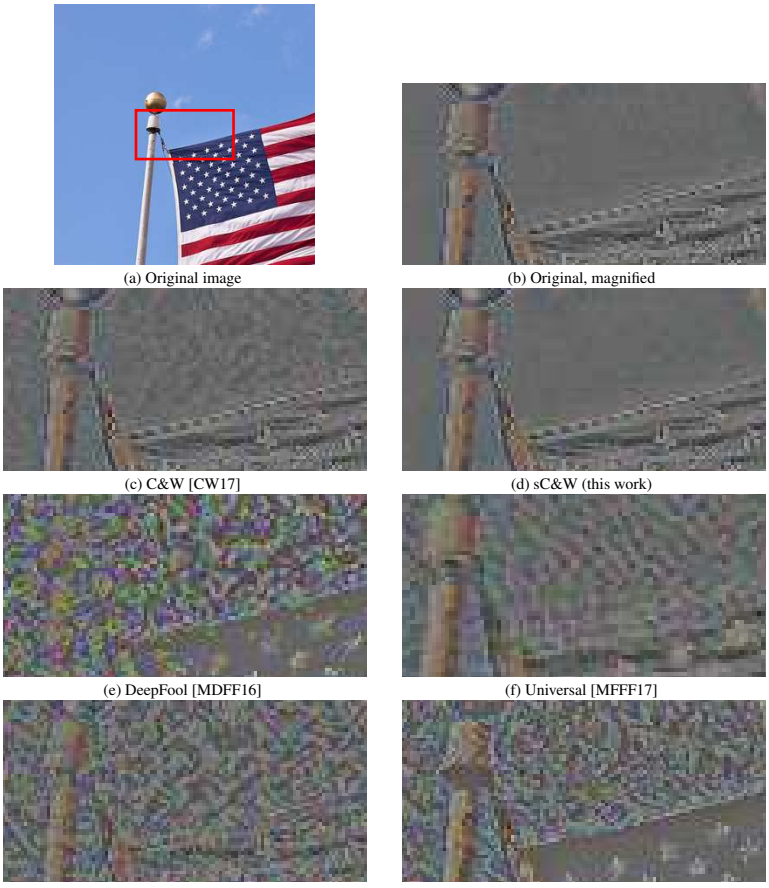
This work investigates the visual quality of the adversarial examples. Recent papers propose to smooth the perturbations to get rid of high frequency artefacts. In this work, smoothing has a different meaning as it perceptually shapes the perturbation according to the visual content of the image to be attacked. The perturbation becomes locally smooth on the flat areas of the input image, but it may be noisy on its textured areas and sharp across its edges.

This operation relies on Laplacian smoothing, well-known in graph signal processing, which we integrate in the attack pipeline. We benchmark several attacks with and without smoothing under a white-box scenario and evaluate their transferability. Despite the additional constraint of smoothness, our attack has the same probability of success at lower distortion. This work [ZAFA20a] is published on *EURASIP Journal on Information Security*.

4.1 Introduction

Adversarial examples where introduced by Szegedy *et al*. [SZS+13] as *imperceptible* perturbations of a test image that can change a neural network's prediction. This has spawned active research on adversarial attacks and defenses with competitions among research teams [KGB+18b]. Despite the theoretical and practical progress in understanding the sensitivity of neural networks to their input, assessing the imperceptibility of adversarial attacks remains elusive: user studies show that L_p norms are largely unsuitable, whereas more sophisticated measures are limited too [SBR18].

Machine assessment of perceptual similarity between two images (the input image and its adversarial example) is arguably as difficult as the original classification task, while human assessment of whether one image is adversarial is hard when the L_p norm of the perturbation is small. Of course, when both images are available and the perturbation is isolated, one can always see it. To make the problem interesting, we ask the following question: *given a single image, can the effect of a perturbation be magnified to the extent that it becomes visible and a human may decide whether this example is benign or adversarial?*



(a) Original image (b) Original, magnified

(c) C&W [CW17] (d) sC&W (this work)

(e) DeepFool [MDFF16] (f) Universal [MFFF17]

(g) FGSM [GSS14] (h) I-FGSM [KGB16]

Figure 4.1 – Given a single input image, our *adversarial magnification* (*cf*. section 4.5) reveals the effect of a potential adversarial perturbation. We show (a) the original image followed by (b) its own magnified version as well as (c)-(h) magnified versions of adversarial examples generated by different attacks. Our *smooth adversarial example* (d) is invisible even when magnified.

Figure 4.1 shows that the answer is positive for a range of popular adversarial attacks. In section 4.5 we propose a simple *adversarial magnification* producing a "magnified" version of a given image, without the knowledge of any other reference image. Assuming that natural images are locally smooth, this can reveal not only the existence of an adversarial perturbation but also its pattern. One can recognize, for instance, the pattern of Figure. 4 of [MFFF17] in our Figure 4.1(f), revealing a universal adversarial perturbation.

Motivated by this example, we argue that popular adversarial attacks have a fundamental limitation in terms of imperceptibility that we attempt to overcome by introducing *smooth adversarial examples*. Our attack assumes local smoothness and generates examples that are consistent with the precise smoothness pattern of the input image. More than just looking "natural" [ZDS18] or being smooth [HZJ18, GFW18], our adversarial examples are photorealistic, low-distortion, and virtually invisible even under magnification. This is evident by comparing our magnified example in Figure 4.1(d) to the magnified original in Figure 4.1(b).

Given that our adversarial examples are more constrained, an interesting question is whether they perform well according to metrics like probability of success and L_p distortion. We show that our attack is not only competitive but outperforms Carlini and Wagner attack (C&W) [CW17], from which our own attack differs basically by a smoothness penalty.

Contributions. As *primary* contributions, we

— investigate the behavior of existing attacks when perturbations become "*smooth like*" the input image; and

— devise one attack that performs well on standard metrics while satisfying the new constraint.

As *secondary* contributions, we

— *magnify* perturbations to facilitate qualitative evaluation of their imperceptibility;

— show that properly integrating the smoothness constraint is not as easy as smoothing the perturbation generated by an attack; and

— define a new, more complete/fair *evaluation protocol*.

The remaining text is organized as follows. section 4.2 explains Laplacian smoothing, on which we build our method. section 4.3 presents our *smooth adversarial attacks*, and section 4.4 provides experimental evaluation. Conclusions are drawn section 4.6. Our *adversarial magnification* used to generate Figure 4.1 is specified in section 4.5.

4.1.1 Related work on imperceptibility

Adversarial perturbations are often invisible only because their amplitude is extremely small. Few papers deal with the need of improving the imperceptibility of the adversarial perturbations. The main idea in this direction is to create low or mid-frequency perturbation patterns.

Zhou *et al*. [ZHC+18] add a regularization term for the sake of transferability, which removes the high frequencies of the perturbation via low-pass spatial filtering. Heng *et al*. [HZJ18] propose a harmonic adversarial attack where perturbations are very smooth gradient-like images. Guo *et al*. [GFW18] design an attack explicitly in the Fourier domain. However, in all cases above, the convolution and the bases of the harmonic functions and of the Fourier transform are independent of the visual content of the input image.

In contrast, the adversarial examples in this work are crafted to be locally compliant with the smoothness of the original image. Our perturbation may be sharp across the edges of $\mathbf{x}_o$ but smooth wherever $\mathbf{x}_o$ is, *e.g*. on background regions. It is not just smooth but photorealistic, because its smoothness pattern is guided by the input image.

An analogy becomes evident with *digital watermarking* [QAR18]. In this application, the watermark signal pushes the input image into the detection region (the set of images deemed as watermarked by the detector), whereas here the adversarial perturbation drives the image outside its class region. The watermark is invisible thanks to the masking property of the input image [CMB+08]. Its textured areas and its contours can hide a lot of watermarking power, but the flat areas can not be modified without producing noticeable artefacts. Perceptually shaping the watermark signal allows a stronger power, which in turn yields more robustness.

Another related problem, with similar solutions mathematically, is photorealistic *style transfer*. Luan *et al*. [LPSB17] transfer style from a reference style image to an input image, while constraining the output to being photorealistic with respect to the input. This work as well as follow-up works [PP18, LLL+18] are based on variants of Laplacian smoothing or regularization much like we do.

It is important to highlight that high frequencies can be powerful for deluding a network, as illustrated by the extreme example of the *one pixel attack* [SVS19]. However this is arguably one of the most visible attacks.

4.2 Background on graph Laplacian smoothing

Popular attacks typically produce noisy patterns that are not found in natural images. They may not be visible at first sight because of their low amplitude, but they are easily detected once magnified (see Fig. 4.1). Our objective is to craft an adversarial perturbation that is locally as smooth as the input image, remaining invisible through magnification. This section gives background on *Laplacian smoothing* [ZBL+03, KLL08], a classical operator in *graph signal processing* [SM13, SNF+13], which we adapt to images here. Section 4.3 uses it generate a smooth perturbation guided by the original input image.

Graph. Laplacian smoothing builds on a weighted undirected graph whose n vertices correspond to the n pixels of the input image $\mathbf{x}_o$. The i-th vertex of the graph is associated with feature $\mathbf{x}_i \in [0,1]^d$ that is the i-th row of $\mathbf{x}_o$, that is, $\mathbf{x}_o = [\mathbf{x}_1, \dots, \mathbf{x}_n]^\top$. Matrix $\mathbf{t} \in \mathbb{R}^{n\times 2}$ denotes the spatial coordinates of the n pixels in the image, and similarly $\mathbf{t} = [\mathbf{t}_1, \dots, \mathbf{t}_n]^\top$. An edge (i,j) of the graph is associated with weight $w_{ij} \geq 0$, giving rise to an $n \times n$ symmetric adjacency matrix $\mathbf{W}$, for instance defined as

$$w_{ij} := \begin{cases} \mathsf{k_f}(\mathbf{x}_i, \mathbf{x}_j)\mathsf{k_s}(\mathbf{t}_i, \mathbf{t}_j), & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases} \tag{4.1}$$

for $i, j \in \{1, \dots, n\}$, where $\mathsf{k_f}$ is a *feature kernel* and $\mathsf{k_s}$ is a *spatial kernel*, both being usually Gaussian or Laplacian. The spatial kernel is typically nonzero only on nearest neighbors, resulting in a sparse matrix $\mathbf{W}$. We further define the $n \times n$ *degree matrix* $\mathbf{D} := \mathrm{diag}(\mathbf{W}\mathbf{1}_n)$ where $\mathbf{1}_n$ is the all-ones n-vector.

Regularization [ZBL+03]. Now, given a new signal $\mathbf{z} \in \mathbb{R}^{n\times d}$ on this graph, the objective of graph smoothing is to find another signal $\mathbf{r}$, which is close to $\mathbf{z}$, while at the same time being smooth according to the neighborhood system represented by the graph. Precisely, given $\mathbf{z}$, we define the *output signal* $\mathsf{s}_\alpha(\mathbf{z}) := \arg\min_{\mathbf{r}\in\mathbb{R}^{n\times d}} \phi_\alpha(\mathbf{r}, \mathbf{z})$, with

$$\phi_\alpha(\mathbf{r}, \mathbf{z}) := \frac{\alpha}{2} \sum_{i,j} w_{ij} \left\| \hat{\mathbf{r}}_i - \hat{\mathbf{r}}_j \right\|^2 + (1-\alpha) \left\| \mathbf{r} - \mathbf{z} \right\|_F^2 \tag{4.2}$$

where $\hat{\mathbf{r}} := \mathbf{D}^{-1/2}\mathbf{r}$ and $\|\cdot\|_F$ is the Frobenius norm. The first summand is the *smoothness term*. It encourages $\hat{\mathbf{r}}_i$ to be close to $\hat{\mathbf{r}}_j$ when w_{ij} is large, *i.e.* when pixels i and j of input $\mathbf{x}_o$ are neighbours and similar. This encourages $\mathbf{r}$ to be smooth wherever $\mathbf{x}_o$ is. The second summand is

the *fitness term* that encourages $\mathbf{r}$ to stay close to $\mathbf{z}$. Parameter $\alpha \in [0, 1)$ controls the trade-off between the two.

Filtering. If we symmetrically normalize matrix $\mathbf{W}$ as $\mathcal{W} := \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ and define the $n \times n$ *regularized Laplacian* matrix $\mathcal{L}_\alpha := (\mathbf{I}_n - \alpha\mathcal{W})/(1-\alpha)$, then the expression (4.2) simplifies to the following quadratic form:

$$\phi_\alpha(\mathbf{r}, \mathbf{z}) = (1-\alpha)\,\mathrm{tr}\left(\mathbf{r}^\top \mathcal{L}_\alpha \mathbf{r} - 2\mathbf{z}^\top \mathbf{r} + \mathbf{z}^\top \mathbf{z}\right). \tag{4.3}$$

This reveals, by letting the derivative $\partial\phi/\partial\mathbf{r}$ vanish independently per column, that the smoothed signal is given in closed form:

$$\mathsf{s}_\alpha(\mathbf{z}) = \mathcal{L}_\alpha^{-1}\mathbf{z}. \tag{4.4}$$

This solution is unique because matrix $\mathcal{L}_\alpha$ is positive-definite. Parameter α controls the bandwidth of the smoothing: function s_α is the all-pass filter for $\alpha = 0$ and becomes a strict 'low-pass' filter when $\alpha \to 1$ [IAT+18].

Variants of the model above have been used for instance for interactive image segmentation [Gra06, KLL08, VC17], transductive semi-supervised classification [ZGL03, ZBL+03], and ranking on manifolds [ZWG+03, ITA+17]. Input $\mathbf{z}$ expresses labels known for some input pixels (for segmentation) or samples (for classification), or identifies queries (for ranking), and is null for the remaining vertices. Smoothing then spreads the labels to these vertices according the weights of the graph.

Normalization. Contrary to applications like interactive segmentation or semi-supervised classification [ZBL+03, KLL08], $\mathbf{z}$ does not represent a binary labeling but rather an arbitrary perturbation in this work. Also contrary to such applications, the output is neither normalized nor taken as the maximum over feature dimensions (channels). If $\mathcal{L}_\alpha^{-1}$ is seen as a spatial filter, we therefore row-wise normalize it to one in order to preserve the dynamic range of $\mathbf{z}$. We therefore define the *normalized smoothing function* as

$$\hat{\mathsf{s}}_\alpha(\mathbf{z}) := \mathrm{diag}(\mathsf{s}_\alpha(\mathbf{1}_n))^{-1}\mathsf{s}_\alpha(\mathbf{z}). \tag{4.5}$$

This function of course depends on $\mathbf{x}_o$. We omit this from notation but we say $\hat{\mathsf{s}}_\alpha$ is *smoothing guided* by $\mathbf{x}_o$ and the output is *smooth like* $\mathbf{x}_o$.

4.3 Integrating smoothness into the attack

The key idea of the paper is that the smoothness of the perturbation is now consistent with the smoothness of the original input image $\mathbf{x}_o$, which is achieved by smoothing operations guided by $\mathbf{x}_o$. This section integrates smoothness into attacks targeting distortion (section 4.3.1) and attacks targeting success (section 4.3.2), but in very different ways.

4.3.1 Simple attacks

We consider here simple attacks targeting distortion or success based on gradient descent of the loss function. There are many variations which normalize or clip the update according to the norm used for measuring the distortion, a learning rate or a fixed step *etc*. These variants are loosely prototyped as the iterative process

$$\mathbf{g} = \nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}_a^{(k)}), \ell_g), \tag{4.6}$$

$$\mathbf{x}_a^{(k+1)} = \mathrm{clip}_{[0,1]} \left(\mathbf{x}_a^{(k)} - \mathsf{n}(\mathbf{g}) \right), \tag{4.7}$$

where $\mathrm{clip}_{[0,1]}$ is a *clipping* function and n a *normalization* function according to the variant. Function $\mathrm{clip}_{[0,1]}$ should at least produce a valid image: $\mathrm{clip}_{[0,1]}(\mathbf{x}) \in \mathcal{X} = [0,1]^{n \times d}$.

Quick and dirty. To keep these simple attacks simple, smoothness is loosely integrated after the gradient computation and before the update normalization:

$$\mathbf{x}_a^{(k+1)} = \mathrm{clip}_{[0,1]} \left(\mathbf{x}_a^{(k)} - \mathsf{n}(\hat{\mathsf{s}}_\alpha(\mathbf{g})) \right). \tag{4.8}$$

This approach can be seen as a projected gradient descent on the manifold of perturbations that are smooth like $\mathbf{x}_o$. When applied to PGD_2, we call this attack qPGD_2 where the 'q' stands for a 'quick and dirty' integration of the smoothness constraint.

4.3.2 Attack targeting optimality

This section integrates smoothness in the attacks targeting optimality like C&W. Our starting point is the unconstrained problem in Equation 2.18 [CW17]. However, instead of representing the perturbation signal $\mathbf{r} := \mathbf{x} - \mathbf{x}_o$ implicitly as a function $\sigma(\mathbf{w}) - \mathbf{x}_o$ of another parameter $\mathbf{w}$, we express the objective explicitly as a function of variable $\mathbf{r}$, as in the original formulation of Equation 2.17 in [SZS+13]. We make this choice because we need to directly process the

perturbation $\mathbf{r}$. On the other hand, we now need the element-wise clipping function $\mathrm{clip}_{[0,1]}(\mathbf{x}) := \min([\mathbf{x}]_+, 1)$ to satisfy the constraint $\mathbf{x} = \mathbf{x}_o + \mathbf{r} \in \mathcal{X}$ Equation 2.17. Our problem is then

$$\min_{\mathbf{r}} \quad \lambda \left\|\mathbf{r}\right\|^2 + \mathcal{L}(f(\mathrm{clip}_{[0,1]}(\mathbf{x}_o + \mathbf{r})), \ell_g), \tag{4.9}$$

where $\mathbf{r}$ is unconstrained in $\mathbb{R}^{n \times d}$.

Smoothness penalty. At this point, optimizing Equation 4.9 results in 'independent' updates at each pixel. We would rather like to take the smoothness structure of the input $\mathbf{x}_o$ into account and impose a similar structure on $\mathbf{r}$. Representing the pairwise relations by a graph as discussed in section 4.2, a straightforward choice is to introduce a pairwise loss term

$$\mu \sum_{i,j} w_{ij} \left\|\hat{\mathbf{r}}_i - \hat{\mathbf{r}}_j\right\|^2 \tag{4.10}$$

into Equation 4.9, where we recall that w_{ij} are the elements of the adjacency matrix $\mathbf{W}$ of $\mathbf{x}_o$, $\hat{\mathbf{r}} := \mathbf{D}^{-1/2}\mathbf{r}$ and $\mathbf{D} := \mathrm{diag}(\mathbf{W}\mathbf{1}_n)$. A problem is that the spatial kernel is typically narrow to capture smoothness only locally. Even if parameter μ is large, it would take a lot of iterations for the information to propagate globally, each iteration needing a forward and backward pass through the network.

Smoothness constraint. What we advocate instead is to apply a global smoothing process at each iteration: we introduce a *latent variable* $\mathbf{z} \in \mathbb{R}^{n \times d}$ and seek for a joint solution with respect to $\mathbf{r}$ and $\mathbf{z}$ of the following

$$\min_{\mathbf{r},\mathbf{z}} \quad \mu\phi_\alpha(\mathbf{r}, \mathbf{z}) + \lambda \left\|\mathbf{r}\right\|^2 + \mathcal{L}(f(\mathrm{clip}_{[0,1]}(\mathbf{x}_o + \mathbf{r})), \ell_g), \tag{4.11}$$

where ϕ is defined by Equation 4.2. In words, $\mathbf{z}$ represents an unconstrained perturbation, while $\mathbf{r}$ should be close to $\mathbf{z}$, smooth like $\mathbf{x}_o$, small, and such that the perturbed input $\mathbf{x}_o + \mathbf{r}$ satisfies the classification objective. Then, by letting $\mu \to \infty$, the first term becomes a hard constraint imposing a globally smooth solution at each iteration:

$$\min_{\mathbf{r},\mathbf{z}} \quad \lambda \left\|\mathbf{r}\right\|^2 + \mathcal{L}(f(\mathrm{clip}_{[0,1]}(\mathbf{x}_o + \mathbf{r})), \ell_g) \tag{4.12}$$

$$\text{subject to} \quad \mathbf{r} = \hat{\mathsf{s}}_\alpha(\mathbf{z}), \tag{4.13}$$

where $\hat{\mathsf{s}}_\alpha$ is defined by Equation 4.5. During optimization, every iterate of this perturbation $\mathbf{r}$ is smooth like $\mathbf{x}_o$.

Optimization. With this definition in place, we solve for $\mathbf{z}$ the following unconstrained problem over $\mathbb{R}^{n\times d}$:

$$\min_{\mathbf{z}} \quad \lambda \left\|\hat{\mathsf{s}}_\alpha(\mathbf{z})\right\|^2 + \mathcal{L}(f(\mathrm{clip}_{[0,1]}(\mathbf{x}_o + \hat{\mathsf{s}}_\alpha(\mathbf{z}))), \ell_g). \tag{4.14}$$

Observe that this problem has the same form as Equation 4.9, where $\mathbf{r}$ has been replaced by $\hat{\mathsf{s}}_\alpha(\mathbf{z})$. This implies that we can use the same optimization method as the C&W attack. The only difference is that the variable is $\mathbf{z}$, which we initialize by $\mathbf{z} = \mathbf{0}_{n\times d}$, and we apply function $\hat{\mathsf{s}}_\alpha$ at each iteration.

Gradients are easy to compute because our smoothing is a linear operator. We denote the loss on this new variable by $L(\mathbf{z}) := \mathcal{L}(f(\mathrm{clip}_{[0,1]}(\mathbf{x}_o + \hat{\mathsf{s}}_\alpha(\mathbf{z}))), \ell_g)$. Its gradient is

$$\nabla_{\mathbf{z}} L(\mathbf{z}) = \mathbf{J}_{\hat{\mathsf{s}}_\alpha}(\mathbf{z})^\top \cdot \nabla_{\mathbf{x}} \mathcal{L}(f(\mathrm{clip}_{[0,1]}(\mathbf{x}_o + \hat{\mathsf{s}}_\alpha(\mathbf{z}))), \ell_g), \tag{4.15}$$

where $\mathbf{J}_{\hat{\mathsf{s}}_\alpha}(\mathbf{z})$ is the $n \times n$ Jacobian matrix of the smoothing operator at $\mathbf{z}$. Since our smoothing operator as defined by Equation 4.4 and Equation 4.5 is linear, $\mathbf{J}_{\hat{\mathsf{s}}_\alpha}(\mathbf{z}) = \mathrm{diag}(\mathsf{s}_\alpha(\mathbf{1}_n))^{-1}\mathcal{L}_\alpha^{-1}$ is a matrix constant in $\mathbf{z}$, and multiplication by this matrix is equivalent to smoothing. The same holds for the distortion penalty $\left\|\hat{\mathsf{s}}_\alpha(\mathbf{z})\right\|^2$. This means that in the backward pass, the gradient of the objective Equation 4.14 w.r.t. $\mathbf{z}$ is obtained from the gradient w.r.t. $\mathbf{r}$ (or $\mathbf{x}$) by smoothing, much like how $\mathbf{r}$ is obtained from $\mathbf{z}$ in the forward pass Equation 4.13.

Matrix $\mathcal{L}_\alpha$ is fixed during optimization, depending only on input $\mathbf{x}_o$. For small images like in the MNIST dataset [LBBH98], it can be inverted: function $\hat{\mathsf{s}}_\alpha$ is really a matrix multiplication. For larger images, we use the *Conjugate Gradient (CG)* method [NW06] to solve the set of linear systems $\mathcal{L}_\alpha\mathbf{r} = \mathbf{z}$ for $\mathbf{r}$ given $\mathbf{z}$. Again, this is possible because matrix $\mathcal{L}_\alpha$ is positive-definite, and indeed it is the most common solution in similar problems [Gra06, CK16, ITA+17]. At each iteration, one computes a product of the form $\mathbf{v} \mapsto \mathcal{L}_\alpha\mathbf{v}$, which is efficient because $\mathcal{L}_\alpha$ is sparse. In the backward pass, one can either use CG on the gradient, or *Auto-Differentiate (AD)* through the forward CG iterations. We choose the latter because it is the simplest implementation-wise. The two options have the same complexity and should have the same run-time in theory. In practice, Tensorflow AD takes 0.43s on average for 50 CG iterations on ImageNet and InceptionV3, while CG forward takes 0.33s.

Discussion. The *clipping function* $\mathrm{clip}_{[0,1]}$ that we use is just the identity over the interval $[0, 1]$ but outside this interval its derivative is zero. Carlini & Wagner [CW17] therefore argue that the numerical solver of problem Equation 4.9 suffers from getting stuck in flat spots: when a pixel of the perturbed input $\mathbf{x}_o + \mathbf{r}$ falls outside $[0, 1]$, it keeps having zero derivative after that and with no chance of returning to $[0, 1]$ even if this is beneficial. This limitation does not apply to our case thanks to the L_2 distortion penalty in Equation 4.9 and to the updates in its neighborhood: such a value may return to $[0, 1]$ thanks to the smoothing operation.

4.4 Experiments

Our experiments focus on the *white-box* setting, where the defender first exhibits a network, and then the attacker mounts an attack specific to this network; but we also investigate a *transferability* scenario. All attacks are *untargetted*, as defined by loss function Equation 2.19.

4.4.1 Attacks and parameters

Attacks. The following six attacks are benchmarked:

— L_∞ distortion: FGSM [GSS14] and I-FGSM [KGB16].

— L_2 distortion: an L_2 version of I-FGSM [PFC+18], denoted as PGD_2 (projected gradient descent).

— Optimality: The L_2 version of C&W [CW17].

— Smooth: our smooth versions qPGD_2 of PGD_2 (subsection 4.3.1) and sC&W of C&W (subsection 4.3.2). Note that the smoothness constraint integration differs a lot between qPGD_2 and sC&W.

Parameters. On MNIST, we use $\epsilon = 0.3$ for FGSM; $\epsilon = 0.3, \alpha = 0.08$ for I-FGSM; $\epsilon = 5, \alpha = 3$ for PGD_2; confidence margin $m = 1$, learning rate $\eta = 0.1$, and initial constant $c = 15$ (the inverse of λ in (2.18)) for C&W. For smoothing, we use Laplacian feature kernel, set $\alpha = 0.95$, and pre-compute $\mathcal{L}_\alpha^{-1}$. On ImageNet, we use $\epsilon = 0.1255$ for FGSM; $\epsilon = 0.1255, \alpha = 0.08$ for I-FGSM; $\epsilon = 5, \alpha = 3$ for PGD_2; $m = 0$, $\eta = 0.1$, and $c = 100$ for C&W. For smoothing, we use Laplacian feature kernel, set $\alpha = 0.997$, and use 50 iterations of CG. These settings are used in subsection 4.4.4.

4.4.2 White box scenario

Qualitative results and perceptual evaluation. Figure 4.2 and Figure 4.3 show MNIST and ImageNet examples respectively, focusing on worst cases. Both sC&W and qPGD$_2$ produce smooth perturbations that look more natural. However, smoothing of qPGD$_2$ is more aggressive especially on MNIST, as these images contain flat black or white areas.

This is due to the 'quick and dirty' integration of the smoothness constraint: On some images, the perturbation update $\hat{\mathrm{s}}_\alpha(\mathbf{g})$ is weakly correlated with gradient $\mathbf{g}$, which is does not help in lowering the classification loss. Consequently, the perturbation becomes stronger in order to succeed. For the same reason, qPGD$_2$ completely fails on natural images like Figure 4.3(a) and (c). It consumes way more distortion than PGD$_2$, and although this perturbation is smoother, it is becomes visible.

By contrast, the proper integration of the smoothness constraint in sC&W produces totally invisible perturbation. For images like Figure 4.3(a) or (c), sC&W consumes more distortion than C&W, but the perturbation remains less visible according to the MAD score. The reason is the 'phantom' of the original that is revealed when the perturbation is isolated.

The superior perceptual quality of our smooth adversarial examples is also confirmed quantitatively: On ImageNet, 93% of the images produced by sC&W have lower MAD score than the ones by C&W. Figure 4.4 shows that when the MAD score of sC&W is greater than the one of C&W, it usually happens for very small score values (below 0.1), meaning that both are almost equally imperceptible.

Table 4.1 – Success probability P_{suc} and average L_2 distortion $\overline{D}$.

	MNIST		ImageNet			
	C4		InceptionV3		ResNetV2	
	P_{suc}	$\overline{D}$	P_{suc}	$\overline{D}$	P_{suc}	$\overline{D}$
FGSM	0.89	5.02	0.97	5.92	0.92	8.20
I-FGSM	1.00	2.69	1.00	5.54	0.99	7.58
PGD$_2$	1.00	1.71	1.00	1.80	1.00	3.63
C&W	1.00	2.49	0.99	4.91	0.99	9.84
qPGD$_2$	0.97	3.36	0.96	2.10	0.93	4.80
sC&W	1.00	1.97	0.99	3.00	0.98	5.99

Quantitative results on success and distortion. The global statistics $P_{\text{suc}}, \overline{D}$ are shown in Table 4.1. Operating characteristics over MNIST and ImageNet are shown in Figure 4.5 and Figure 4.6 respectively.

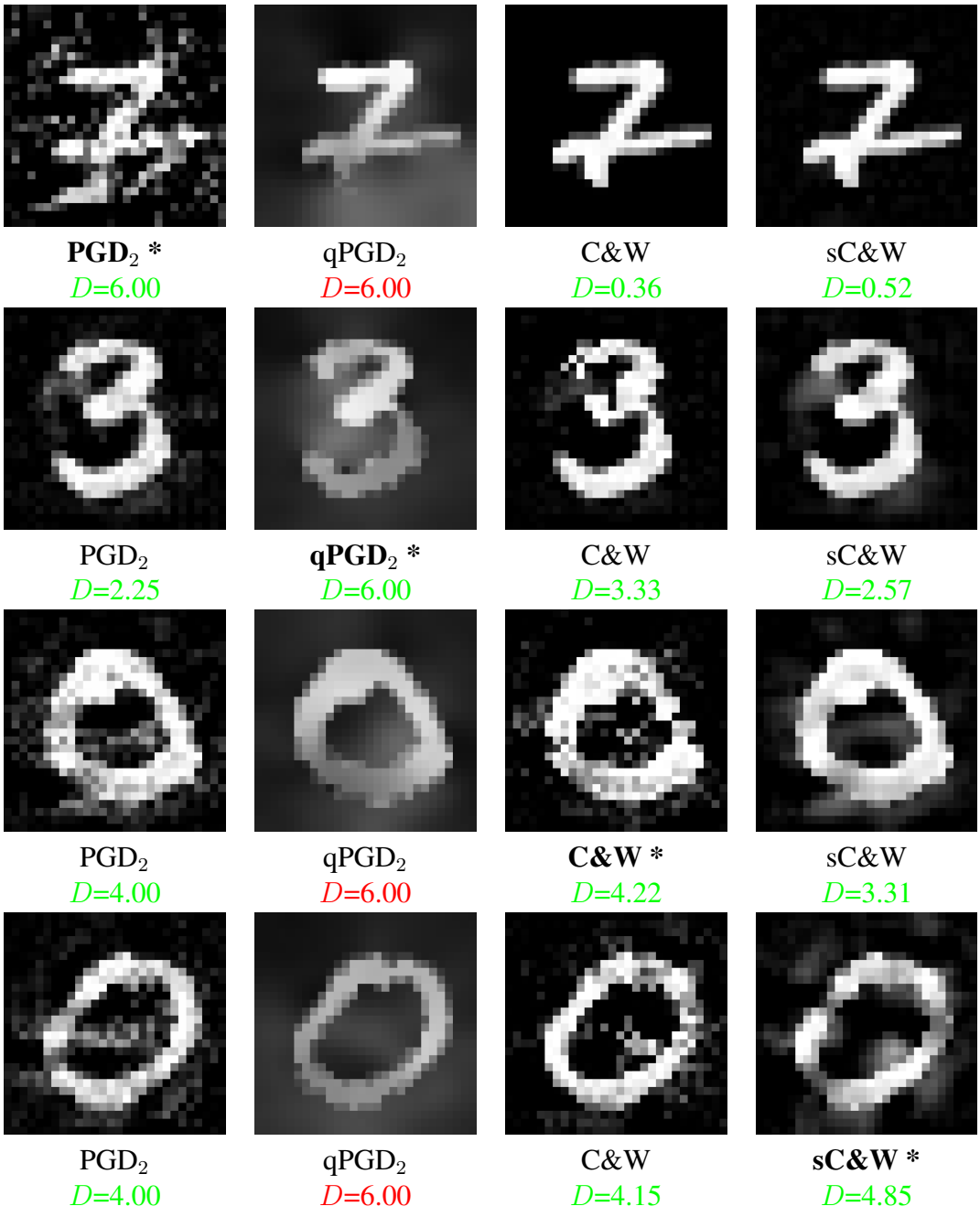


Figure 4.2 – For a given attack (denoted by * and bold typeface), the adversarial image with the strongest distortion D over MNIST. In green, the attack succeeds; in red, it fails.

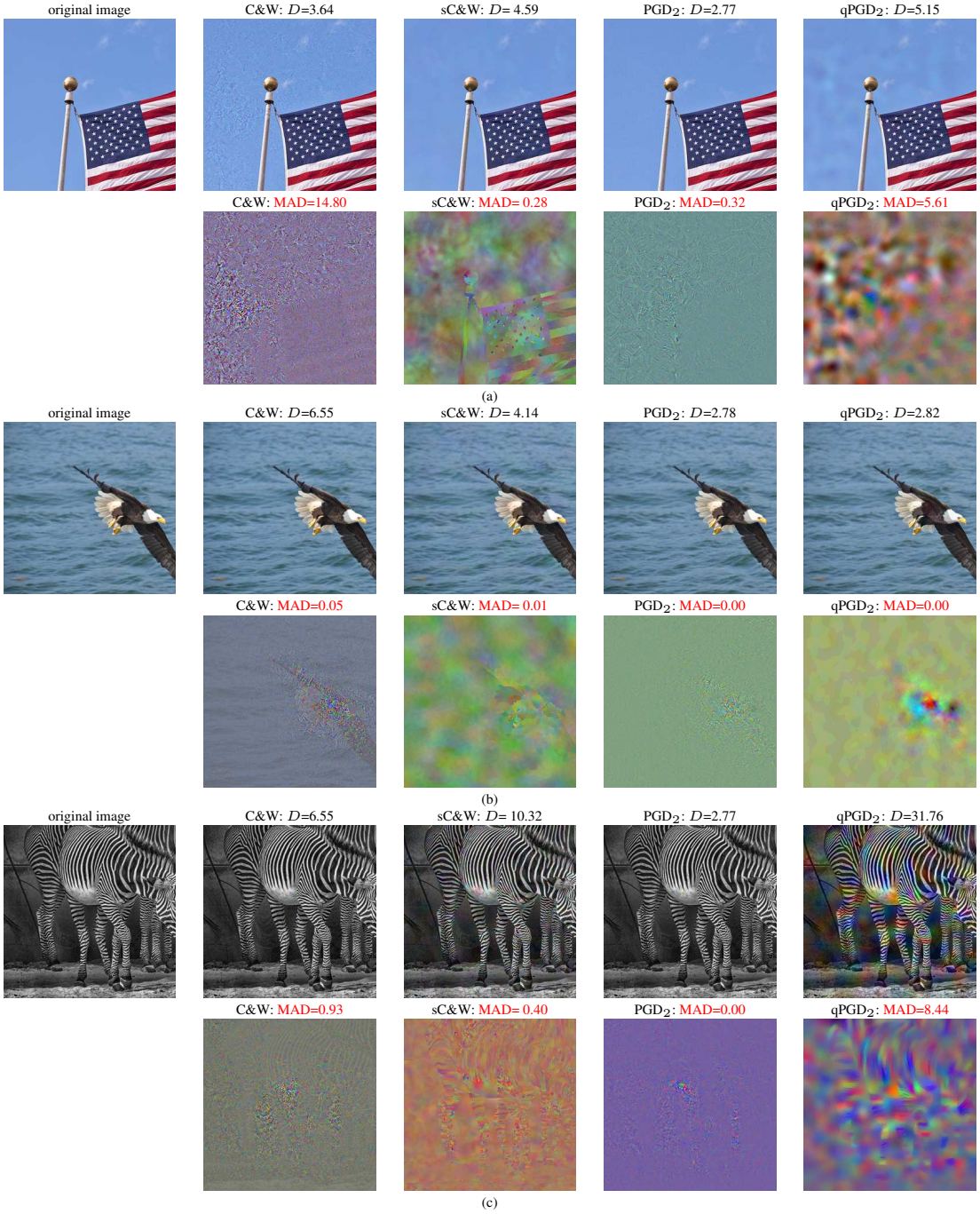


Figure 4.3 – Original image $\mathbf{x}_o$ (left), adversarial image $\mathbf{x}_a = \mathbf{x}_o + \mathbf{r}$ (above) and scaled perturbation $\mathbf{r}$ (below; distortion $D = \|\mathbf{r}\|$ and MAD scores) against InceptionV3 on ImageNet. Scaling maps each perturbation and each color channel independently to $[0, 1]$. The perturbation $\mathbf{r}$ is indeed smooth like $\mathbf{x}_o$ for sC&W. (a) Despite the higher distortion compared to C&W, the perturbation of sC&W is totally invisible, even when magnified (*cf*. Figure 4.1). (b) One of the failing examples of [HZJ18] that look unnatural to human vision. (c) One of the examples with the strongest distortion over ImageNet for sC&W: $\mathbf{x}_o$ is flat along stripes, reducing the dimensionality of the 'smooth like $\mathbf{x}_o$' manifold.

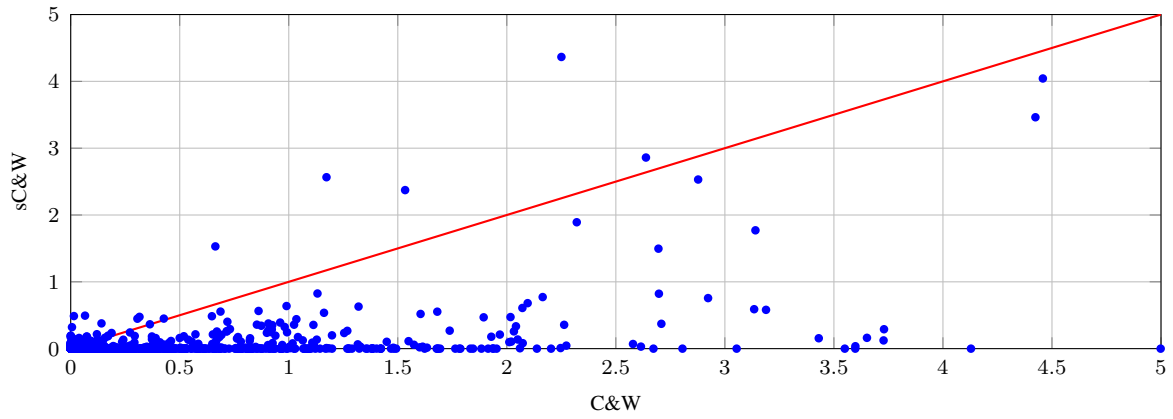


Figure 4.4 – MAD scores [LC10] of sC&W *vs*. C&W for all images of ImageNet. For 93% of the images below the diagonal, sC&W is less perceptible than C&W according to MAD score.

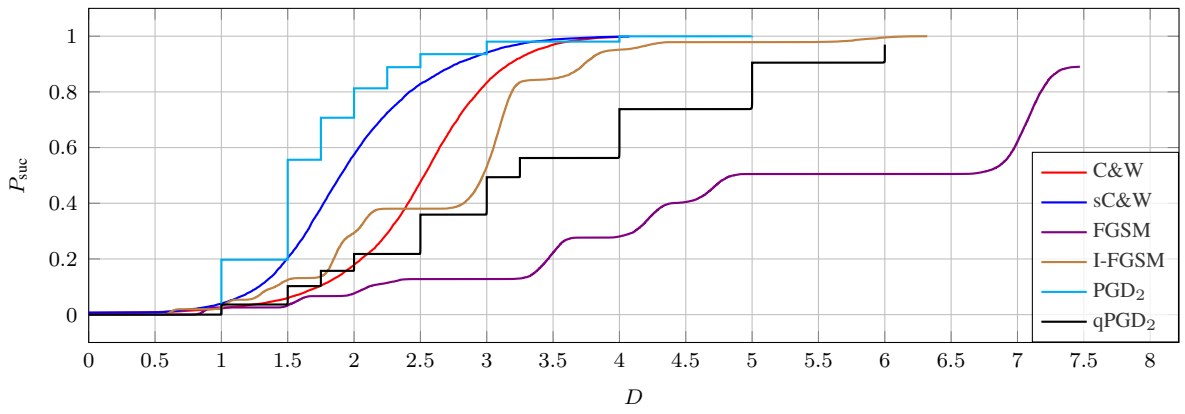


Figure 4.5 – Operating characteristics of the attacks over MNIST. Attacks PGD_2 and qPGD_2 are tested with target distortion $D \in [1, 6]$.

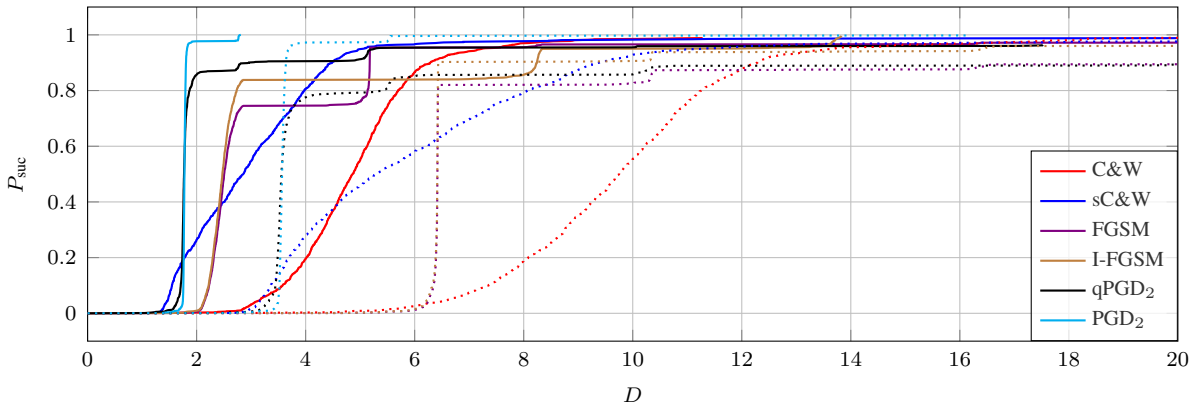


Figure 4.6 – Operating characteristics over ImageNet attacking InceptionV3 (solid lines) and ResNetV2-50 (dotted lines).

We observe that our sC&W, with the proper integration via a latent variable (4.14), improves a lot the original C&W in terms of distortion, while keeping the probability of success roughly the same. This result, consistent in all experiments, is surprising. We would expect a price to be paid for a better invisibility as the smoothing is adding an extra constraint on the perturbation. This price can be rather high in the literature: In order to preserve the success rate, Table 1 of [GFW18] reports an increase of distortion by a factor of 3 when integrating smoothness in the attack. An explanation may be that the smoothing operation of [GFW18] is independent of the input image; while in our case, smoothing is guided by the input.

On the contrary, the 'quick and dirty' integration (4.8) dramatically spoils qPGD$_2$ with big distortion especially on MNIST. This reveals the utmost importance of properly integrating the smoothness constraint. It cannot be just a post-processing filtering of the perturbation.

The price to pay for smoothing is the run-time: using Tensorflow and 50 CG iterations on ImageNet and InceptionV3, sC&W takes 205s per image on average, while C&W takes 47s. This is with our own implementation of CG without particular optimization effort. Runtime was not within our objectives.

We further observe that PGD$_2$ outperforms by a vast margin the C&W attack, which is supposed to be close to optimality. This may be due in part to how the Adam optimizer treats L_2 norm penalties as studied in [LH17]. This interesting finding is a result of our new evaluation protocol: C&W internally optimizes its parameter $c = 1/\lambda$ independently per image, while for PGD$_2$ we externally try a small set of target distortions D on the entire dataset. This is visible in Figure 4.5, where the operating characteristic is piecewise constant. Our comparison is fair,

given that C&W is more expensive.

As already observed in the literature, ResnetV2 is more robust to attacks than InceptionV3: The operating characteristic curves are shifted to the right and increase at a slower rate.

Table 4.2 – Success probability and average L_2 distortion $\overline{D}$ when attacking networks adversarially trained against FGSM.

	MNIST - C4		ImageNet - InceptionV3	
	P_{suc}	$\overline{D}$	P_{suc}	$\overline{D}$
FGSM	0.15	4.53	0.06	6.40
I-FGSM	1.00	3.48	0.97	29.94
PGD_2	1.00	2.52	1.00	3.89
C&W	0.93	3.03	0.95	6.43
qPGD_2	0.99	2.94	0.69	7.86
sC&W	0.99	2.39	0.75	6.22

4.4.3 Adversarial training

The defender now uses adversarial training [GSS14] to gain robustness against attacks. Yet, the white-box scenario still holds: this network is public. The training set comprises images attacked with "step l.l" model [KGB16] [1]. The accuracy of C4 on MNIST (resp. InceptionV3 on ImageNet) is now 0.99 (resp. 0.94).

Table 4.2 shows interesting results. As expected, FGSM is defeated in all cases, while average distortion of all attacks is increased in general. What is unexpected is that on MNIST, sC&W remains successful while the probability of C&W drops. On ImageNet however, it is the probability of the smooth versions qPGD_2 and sC&W that drops. I-FGSM is also defeated in this case, in the sense that average distortion increases too much.

4.4.4 Transferability

This section investigates the transferability of the attacks under the following scenario: the attacker has now a partial knowledge about the network. For instance, he/she knows that the defender chose a variant of InceptionV3, but this variant is not public so he/she attacks InceptionV3 instead. Also, this time he/she is not allowed to test different distortion targets. The results are shown in Table 4.3.

1. Model taken from: `https://github.com/tensorflow/models/tree/master/research/adv_imagenet_models`

Table 4.3 – Success probability and average L_2 distortion $\overline{D}$ of attacks on variants of InceptionV3 under transferability.

	Bilateral filter		Adv. training	
	P_{suc}	$\overline{D}$	P_{suc}	$\overline{D}$
FGSM	0.77	5.13	0.04	10.20
I-FGSM	0.82	5.12	0.02	10.10
PGD_2	1.00	5.14	0.12	10.26
C&W	0.82	4.75	0.02	10.21
qPGD_2	0.95	5.13	0.01	10.17
sC&W	0.68	2.91	0.01	4.63

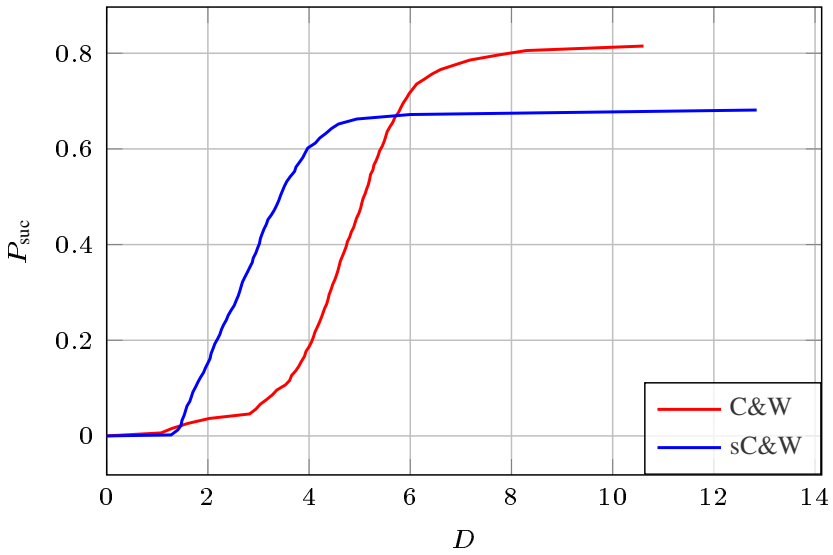


Figure 4.7 – Operating characteristics of C&W and sC&W on ImageNet with InceptionV3 under bilateral filter transferability, corresponding to Table 3.

The first variant uses a bilateral filter (with standard deviation 0.5 and 0.2 in the domain and range kernel respectively; *cf*. section 4.5) before feeding the network. This does not really prevent the attacks. PGD_2 remains a very powerful attack if the distortion is large enough. Smoothing makes the attack less effective but the perturbations are less visible. The second variant uses the adversarially trained InceptionV3, which is, on the contrary, a very efficient counter-measure under this scenario.

Figure 4.7 shows the operating characteristics of C&W and sC&W corresponding to the bilateral filter results of Table 4.3. We see that within a distortion budget of 5, sC&W succeeds with 67% probability, whereas C&W with 50%. Yet, at larger distortion budgets, C&W keeps on forging more adversarial images whereas sC&W stops making progress. This is understandable: C&W creates strong artefacts clearly visible when the distortion is larger or equal to 5, as shown in Figure 4.3. The upfront defense filters out some of these strong perturbations, but the rest remain successful. These images are adversarial by definition, yet not useful in practice because they are too much distorted.

4.5 Adversarial magnification

Given a single-channel image $\mathbf{x} : \Omega \to \mathbb{R}$ as input, its *adversarial magnification* $\mathsf{mag}(\mathbf{x}) : \Omega \to \mathbb{R}$ is defined as the following *local normalization* operation

$$\mathsf{mag}(\mathbf{x}) := \frac{\mathbf{x} - \mu_{\mathbf{x}}(\mathbf{x})}{\beta\sigma_{\mathbf{x}}(\mathbf{x}) + (1-\beta)\sigma_{\Omega}(\mathbf{x})}, \tag{4.16}$$

where $\mu_{\mathbf{x}}(\mathbf{x})$ and $\sigma_{\mathbf{x}}(\mathbf{x})$ are the local mean and standard deviation of $\mathbf{x}$ respectively, and $\sigma_{\Omega}(\mathbf{x}) \in \mathbb{R}^+$ is the global standard deviation of $\mathbf{x}$ over Ω. Parameter $\beta \in [0, 1]$ determines how much local variation is magnified in $\mathbf{x}$.

In our implementation, $\mu_{\mathbf{x}}(\mathbf{x}) = \mathsf{b}(\mathbf{x})$, the *bilateral filtering* of $\mathbf{x}$ [TM98]. It applies a local kernel at each point $p \in \Omega$ that is the product of a domain and a range Gaussian kernel. The *domain kernel* measures the geometric proximity of every point $q \in \Omega$ to p as a function of $\|p - q\|$ and the *range kernel* measures the photometric similarity of every point $q \in \Omega$ to p as a function $|\mathbf{x}(p) - \mathbf{x}(q)|$. On the other hand, $\sigma_{\mathbf{x}}(\mathbf{x}) = \mathsf{b}_{\mathbf{x}}((\mathbf{x} - \mu_{\mathbf{x}}(\mathbf{x}))^2)^{-1/2}$. Here, $\mathsf{b}_{\mathbf{x}}$ is a *guided* version of the bilateral filter, where it is the reference image $\mathbf{x}$ rather than $(\mathbf{x} - \mu_{\mathbf{x}}(\mathbf{x}))^2$ that is used in the range kernel.

When $\mathbf{x} : \Omega \to \mathbb{R}^d$ is a d-channel image, we apply all the filters independently per channel, but photometric similarity is just one scalar per point as a function of the Euclidean distance

$\|\mathbf{x}(p) - \mathbf{x}(q)\|$ measured over all d channels.

In Figure 4.1, $\beta = 0.8$. The standard deviation of both the domain and range Gaussian kernels is 5.

4.6 Conclusion

Smoothing helps masking the adversarial perturbation by shaping it 'like' the input image. However, this rule holds only when smoothness is properly integrated in the attack. Filtering the perturbation by post-processing is not a sound idea, even if it is done in accordance with the original image, even if it is done at each attack iteration. A sounder integration is to inject smoothness as a constraint inside the loss function.

It is impressive how sC&W improves upon C&W in terms of distortion and imperceptibility at the same time while maintaining the same success rate. To our knowledge and as far as a white box scenario is considered, this is the first time smoothness comes for free from this viewpoint. Yet, a price to be paid is the larger complexity.

Smoothing allows the attacker to delude more robust networks thanks to larger distortions while still being invisible. However, its impact on transferability is mitigated. The question raised in the introduction is still open: Figure 4.1 shows that a human does not make the difference between the input image and its adversarial example even with magnification. This does not prove that an algorithm will not detect some statistical evidence.

CHAPTER 5

BOUNDARY PROJECTION ATTACK

Adversarial examples of DNNs are receiving ever increasing attention because they help in understanding and reducing the sensitivity to their input. This is natural given the increasing applications of DNNs in our everyday lives. When white-box attacks are almost always successful, it is typically only the distortion of the perturbations that matters in their evaluation.

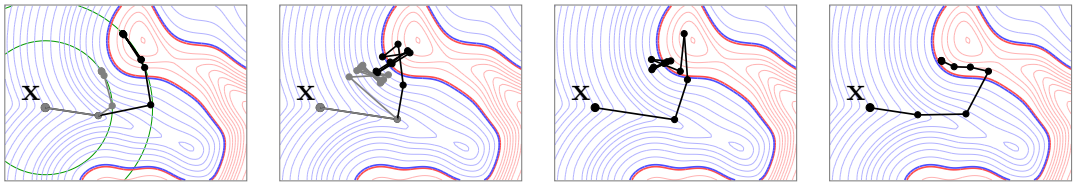
In this work, we argue that speed is important as well, especially when considering that fast attacks are required by adversarial training. Given more time, iterative methods can always find better solutions. We investigate this *speed-distortion trade-off* in some depth and introduce a new attack called *Boundary Projection (BP)* that improves upon existing methods by a large margin. Our key idea is that the classification boundary is a manifold in the image space: we therefore quickly reach the boundary and then optimize distortion on this manifold. This work [ZAFA20b] is published on *IEEE Transactions on Information Forensics and Security*.

5.1 Introduction

Adversarial examples [SZS+13] are small, usually imperceptible perturbations of images or other data [CW18] that can arbitrarily modify a classifier's prediction. They have been extended to other tasks like object detection or semantic segmentation [XWZ+17b], and image retrieval [LJL+19, TRC19b]. They are typically generated in a *white-box* setting, where the attacker has full access to the classifier model and uses gradient signals through the model to optimize for the perturbation. They are becoming increasingly important because they reveal the *sensitivity* of neural networks to their input [SGOS+18, FMDF16, ABB+17] including trivial cases [AW18, ETT+17] and they easily *transfer* between different models [MFFF17, TPG+17].

5.1.1 Graphical abstract illustrating the attacks.

To better understand how our attack works, Figure 5.1 illustrates qualitatively a number of existing attacks technically described in section 2.5. On this toy 2d classification problem, the



(a) PGD_2 (b) C&W (c) DDN (d) BP (this work)

Figure 5.1 – Adversarial attacks on a binary classifier in two dimensions. The two class regions are shown in red and blue. Contours indicate class probabilities. The objective is to find a point in the red (*adversarial*) region that is at the minimal distance to input $\mathbf{x}$. Gray (black) paths correspond to low (high) distortion budget ϵ for PGD_2 [KGB16] (a, in green) or parameter λ for C&W [CW17] (b). The simulation is only meant to illustrate basic properties of the methods. In particular, it does not include Adam optimizer [KB15] for C&W.

class boundary and the path followed by the optimizer starting at input $\mathbf{x}$ can be easily visualized.

PGD_2, an ℓ_2 version of I-FGSM [KGB16], a.k.a. PGD [MMS+17], is controlled by a distortion budget ϵ and eventually follows a path on a ball of radius ϵ centered at $\mathbf{x}$ (*cf*. Figure 5.1(a)). section 5.3 shows that testing different ϵ values is an expensive strategy for finding the optimal distortion budget per image.

C&W [CW17] depends on a parameter λ that controls the balance between distortion and classification loss. A low value leads to failure. A higher value indeed reaches the optimal perturbation, but with oscillations across the class boundary (Figure 5.1(b)). Therefore, an expensive line search over λ is performed internally.

DDN [RHO+19] increases or decreases distortion on the fly depending on success and while pointing towards the gradient direction (Figure 5.1(c)). It arrives quickly near the optimal perturbation but still suffers from oscillations across the boundary.

On the contrary, *Boundary Projection (BP)*, introduced in this work (*cf*. Figure 5.1(d)), cares more about quickly reaching the boundary, not necessarily near the optimal solution, and then walks *along the boundary*, staying mostly in the *adversarial* (red) region. It therefore makes steady progress towards the solution rather than going back and forth.

5.1.2 Related work

Optimization on manifolds. In the context of deep learning, stochastic gradient descent on Riemanian manifolds has been studied, *e.g*. RSGD [Bon13] and RSVRG [ZRS16]. It is usually applied to manifolds whose geometry is known in analytic form, for instance Grassmann [Bon13] or Stiefel manifolds [HF16]. In most cases, the motivation is to optimize a very complex function

like a classification loss on a well-studied manifold, *e.g*. matrix manifold [AMS09]. On the contrary, we optimize a simple quadratic function (the distortion) on a complex manifold not known in analytic form, *i.e*. a level set of the classification loss.

DDN. Instead of using the existing optimizer to tackle the optimization for generating adversarial examples, *Decoupling direction and norm* (DDN) [RHO+19] searches the optimal solution based on the gradient of network. The method, as it is introduced in section 2.5, is quite simple and efficient. It is iterating similarly to PGD, but the radius is adapted according to the current distortion. Another difference is that each iteration is concluded by a projection onto $\mathcal{X}$ (rather than $\hat{\mathcal{X}}$) by element-wise clipping to $[0, 1]$ and *rounding*. This hints they consider the quantization problem of adversarial images.

DDN attempts to optimize the adversarial problem based on the gradient of neural network and distortion. It is more efficient and problem-related. However, only search long the gradient might lead to the oscillations around the boundary. To improve this, we propose to optimize around the class boundary when the current solution is adversarial.

Optimizing around the class boundary is not a new idea. All of the above attacks do so in order to minimize distortion; implicitly, even distortion targeted attacks like PGD do so, if the minimum parameter ϵ is sought. Even *black-box* attacks do so [BRB18], without having access to the gradient function.

Contributions. To our knowledge, we are the first to

— Study optimization on the *manifold* of the classification boundary for an adversarial attack, providing an *analysis* under the constraints of staying on the tangent space of the manifold and of reaching a distortion budget.

— Investigate theoretically and experimentally the quantization impact on the perturbation.

— Achieve at the same speed as I-FGSM [KGB16] (20 iterations) and under the constraint of a quantization, less distortion than state-of-the-art attacks including DDN, which needs 100 iterations on ImageNet.

The remaining text is organized as follows. Section 5.2 presents our *Boundary Projection (BP)*. Section 5.3 provides primary experimental evaluation. Section 5.4 explains the details of predicting distortion after quantization. Section 5.5 shows the defense evaluation with adversarial training and section 5.6 shows the adversarial images. Conclusions are drawn section 5.7.

5.2 Method

Our attack is an *iterative* process with a fixed number K of iterations. Stage 1 aims at quickly producing an adversarial image, whereas Stage 2 is a refinement phase decreasing distortion. The key property of our method is that while in the adversarial region during refinement, it tries to walk along the classification boundary by projecting the distortion gradient onto the tangent hyperplane of the boundary. Hence we call it *boundary projection* (BP).

5.2.1 Stage 1

This stage begins at $\mathbf{y}_0 = \mathbf{x}$ and *iteratively* updates in the direction of the gradient of the loss function as summarized in Algorithm 1. The gradient is 2-normalized (line 3) and then scaled by two parameters (line 4): a fixed parameter $\alpha > 0$ and a parameter γ_i that is increasing linearly with iteration $i \leq K$ as follows

$$\gamma_i := \gamma_{\min} + \frac{i}{K+1}(1 - \gamma_{\min}) < 1, \tag{5.1}$$

where $\gamma_{\min} \in (0, \gamma_{\max})$. This makes the updates slow at the beginning to keep distortion low, then faster until the attack succeeds. Parameter α is set empirically to a value large enough so that Stage 1 returns an adversarial image in less than K iterations with high probability. This schedule of γ_i is meant to adjust the level of distortion to each original image, since a single value would be hard to fit all cases. After the update, clipping is element-wise (line 4).

Algorithm 1 Stage 1

Input: $\mathbf{x}$: original image to be attacked
Input: ℓ_g: true label (untargeted)
Output: $\mathbf{y}$ with $f(\mathbf{y}) \neq \ell_g$ or failure, iteration i

1: Initialize $\mathbf{y}_0 \leftarrow \mathbf{x},\ i \leftarrow 0$
2: **while** $(f(\mathbf{y}_i) = \ell_g) \wedge (i < K)$ **do**
3: $\quad \hat{\mathbf{g}} \leftarrow \mathsf{n}(\nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{y}_i), \ell_g))$ ▷ n: 2-normalization
4: $\quad \mathbf{y}_{i+1} \leftarrow \mathrm{clip}_{[0,1]}(\mathbf{y}_i - \alpha\gamma_i \hat{\mathbf{g}})$
5: $\quad i \leftarrow i + 1$
6: **end while**

Algorithm 2 Stage 2

Input: ℓ_g: true label (untargeted), i current iteration number
Input: $\mathbf{y}_i$: current adversarial image, ϵ: distortion budget
Output: $\mathbf{y}_K$

1: **while** $i < K$ **do**
2: $\quad \mathbf{r}_i \leftarrow \mathbf{y}_i - \mathbf{x}$ ▷ perturbation
3: $\quad \hat{\mathbf{g}} \leftarrow \mathsf{n}(\nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{y}_i), \ell_g))$ ▷ direction
4: $\quad \rho \leftarrow \langle \mathbf{r}_i, \hat{\mathbf{g}} \rangle$
5: $\quad$ **if** $f(\mathbf{y}_i) \neq \ell_g$ **then** ▷ OUT
6: $\quad\quad \epsilon \leftarrow \gamma_i \|\mathbf{r}_i\|$ ▷ distortion control
7: $\quad\quad \mathbf{v}^\star \leftarrow \mathbf{x} + \rho \hat{\mathbf{g}}$
8: $\quad\quad \mathbf{z} \leftarrow \mathbf{v}^\star + \mathsf{n}(\mathbf{y}_i - \mathbf{v}^\star) \sqrt{[\epsilon^2 - \rho^2]_+}$
9: $\quad\quad \mathbf{y}_{i+1} \leftarrow Q_{\text{OUT}}(\mathbf{z}, \mathbf{y}_i)$ ▷ quantization (5.7)
10: $\quad$ **else** ▷ IN
11: $\quad\quad \epsilon \leftarrow \|\mathbf{r}_i\| / \gamma_i$ ▷ distortion control
12: $\quad\quad \mathbf{z} \leftarrow \mathbf{y}_i - \left(\rho + \sqrt{\epsilon^2 - \|\mathbf{r}_i\|^2 + \rho^2} \right) \hat{\mathbf{g}}$
13: $\quad\quad \mathbf{y}_{i+1} \leftarrow Q_{\text{IN}}(\mathbf{z}, \mathbf{y}_i)$ ▷ quantization (5.10)
14: $\quad$ **end if**
15: $\quad i \leftarrow i + 1$
16: **end while**

5.2.2 Stage 2

Once Stage 1 has succeeded, Stage 2 continues by *iteratively* considering two cases: if $\mathbf{y}_i$ is adversarial, case OUT aims at minimizing distortion while staying in the adversarial region. Otherwise, case IN aims at decreasing the loss while controlling the distortion. Both work with a first order approximation of the loss around $\mathbf{y}_i$:

$$\mathcal{L}(f(\mathbf{y}_i + \Delta), \ell_g) \approx \mathcal{L}(f(\mathbf{y}_i), \ell_g) + \Delta^\top \mathbf{g}, \tag{5.2}$$

where $\mathbf{g} = \nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{y}_i), \ell_g)$. The perturbation at iteration i is $\mathbf{r}_i := \mathbf{y}_i - \mathbf{x}$. Stage 2 is summarized in Algorithm 2. Cases OUT and IN illustrated in Figure 5.2 are explained below.

Case OUT. takes as input $\mathbf{y}_i$ *outside* class ℓ_g region, *i.e.* $f(\mathbf{y}_i) \neq \ell_g$ (line 5). It outputs $\mathbf{y}_{i+1}$ which is a quantized version of vector $\mathbf{z}$ (line 9). The construction of $\mathbf{z}$ stems from two constraints. First, we control the distortion of the next perturbation imposing the constraint $\|\mathbf{z} - \mathbf{x}\| = \epsilon$, so that $\mathbf{z}$ will lie on the hypersphere $S[\mathbf{x}; \epsilon]$. This radius uses again the scheduling Equation 5.1, $\epsilon = \gamma_i \|\mathbf{r}_i\| < \|\mathbf{r}_i\|$, such that updates decelerate to convergence once the attack has already

succeeded. We also impose that $\mathbf{z}$ lies on

$$P := \{\mathbf{v} \in \mathbb{R}^n : \langle \mathbf{v} - \mathbf{y}_i, \hat{\mathbf{g}} \rangle = 0\}, \tag{5.3}$$

the tangent hyperplane of the level set of the loss at $\mathbf{y}_i$, normal to $\hat{\mathbf{g}}$. This second constraint aims at maintaining the value of the loss, up to the first order.

Consider the projection $\mathbf{v}^\star := \mathbf{x} + \rho\hat{\mathbf{g}}$ of $\mathbf{x}$ onto hyperplane P, where $\rho := \langle \mathbf{r}_i, \hat{\mathbf{g}} \rangle$. If $\rho < \epsilon$, the hypersphere intersects the hyperplane as shown in Figure 5.2(a), and both constraints are met. From the infinity of points in this intersection, we pick the one closest to $\mathbf{y}_i$:

$$\mathbf{z} = \mathbf{v}^\star + \mathsf{n}(\mathbf{y}_i - \mathbf{v}^\star)\sqrt{\epsilon^2 - \rho^2}. \tag{5.4}$$

If $\rho \geq \epsilon$, then $S[\mathbf{x}; \epsilon] \cap P = \emptyset$. We prefer to relax the constraint on the distortion, and choose $\mathbf{z} = \mathbf{v}^\star$, which is by definition the closest point of P to $\mathbf{x}$. Note that $\mathbf{v}^\star = \mathbf{y}_i$ if $\mathbf{r}_i, \hat{\mathbf{g}}$ are collinear.

This is formally summarized as follows:

$$\mathbf{z} := \arg\min_{\mathbf{v} \in V} \|\mathbf{v} - \mathbf{y}_i\| \tag{5.5}$$

$$V := \arg\min_{\mathbf{v} \in P} \left| \|\mathbf{v} - \mathbf{x}\| - \epsilon \right|. \tag{5.6}$$

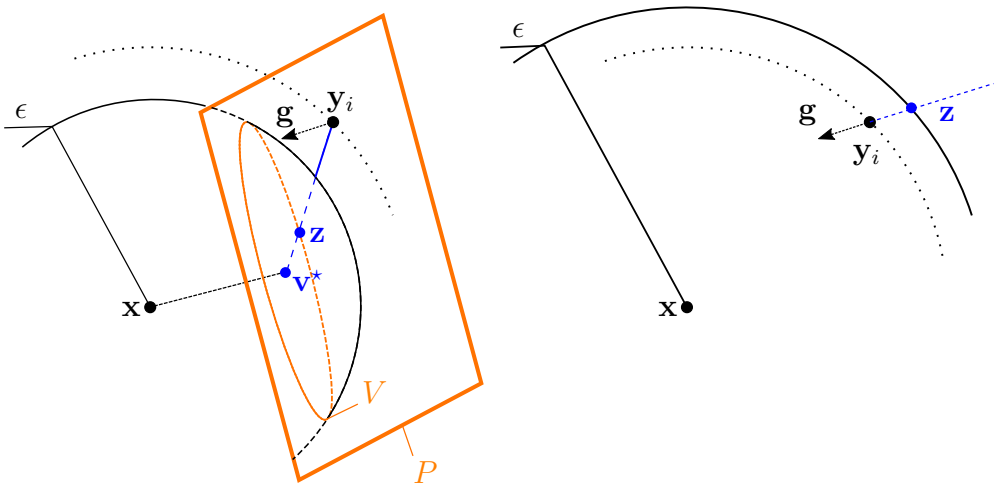
Here, V is the set of points on P having distortion close to ϵ. If $\rho < \epsilon$, $V = S[\mathbf{x}; \epsilon] \cap P \neq \emptyset$ as illustrated in Figure 5.2(a), otherwise $V = \{\mathbf{v}^\star\}$. The solution $\mathbf{z}$ has a unique closed form, implemented in line 8.

Directly quantizing vector $\mathbf{z}$ onto $\mathcal{X}$ by $Q(\cdot)$, the component-wise rounding, modifies its norm (see section 5.4). This pulls down our effort to control the distortion. Instead, the process $Q_{\text{OUT}}(\mathbf{z}, \mathbf{y}_i)$ in line 9 looks for the scale β_{OUT} of the perturbation to be applied such that $\|Q(\mathbf{y}_i + \beta(\mathbf{z} - \mathbf{y}_i))\| = \|\mathbf{z}\|$. This is done with a simple line search over β. Then,

$$Q_{\text{OUT}}(\mathbf{z}, \mathbf{y}_i) := Q(\mathbf{y}_i + \beta_{\text{OUT}}(\mathbf{z} - \mathbf{y}_i)). \tag{5.7}$$

Case IN. takes as input $\mathbf{y}_i$ *inside* class ℓ_g region, *i.e.* $f(\mathbf{y}_i) = \ell_g$ (line 10). We control distortion $\epsilon = \|\mathbf{r}_i\| / \gamma_i > \|\mathbf{r}_i\|$ Equation 5.1 such that updates decelerate as in Case OUT. We then solve the problem:

$$\mathbf{z} := \arg\min_{\mathbf{v} \in S[\mathbf{x}; \epsilon]} \langle \mathbf{v}, \hat{\mathbf{g}} \rangle, \tag{5.8}$$



(a) (b)

Figure 5.2 – Refinement stage of BP. Case OUT when $|V| > 1$ (a); case IN (b).

i.e., find the point $\mathbf{z}$ at the intersection of sphere $S[\mathbf{x}; \epsilon]$ and the ray through $\mathbf{y}_i$ in the direction opposite of $\mathbf{g}$ as shown in Figure 5.2(b). The solution is simple:

$$\mathbf{z} = \mathbf{y}_i - \left(\rho + \sqrt{\epsilon^2 - \|\mathbf{r}_i\|^2 + \rho^2}\right) \hat{\mathbf{g}}, \tag{5.9}$$

Vector $\mathbf{z}$ moves away from $\mathbf{y}_i$ along direction $-\hat{\mathbf{g}}$ by a step size so to reach $S[\mathbf{x}, \epsilon]$. Case IN is not guaranteed to succeed, but invoking it means that Stage 1 has succeeded.

Again a direct rounding jeopardizes the norm of the update $\mathbf{z} - \mathbf{y}_i$. Especially, quantization likely results in $Q(\mathbf{z}) = Q(\mathbf{y}_i)$ if $\|\mathbf{z} - \mathbf{y}_i\| < \beta_{\min} = 0.1$ (see section 5.4). Instead of a line search as in method OUT, line 13 just makes sure that this event will not happen:

$$Q_{\text{IN}}(\mathbf{z}, \mathbf{y}_i) = Q(\mathbf{y}_i + \beta_{\text{IN}}(\mathbf{z} - \mathbf{y}_i)), \tag{5.10}$$

with $\beta_{\text{IN}} = \max(1, \beta_{\min}/\|\mathbf{z} - \mathbf{y}_i\|)$.

5.2.3 Discussion

The heuristic scheduling Equation 5.1 builds on a simpler idea of DDN [RHO+19], where parameter γ is constant across iterations. This scheduling controls the distortion: In stage 1, updates are small at the beginning to keep distortion low, then larger until the attack succeeds. In stage 2, updates are decreasing as γ_i tends to 1. It increases the distortion when the current image

is correctly classified (IN) and decreases the distortion when the current image is adversarial (OUT).

The fact that $(\gamma_i)_i$ is strictly increasing shows that, in Stage 2, an IN iteration (distortion grows by $1/\gamma_i$) followed by an OUT iteration (distortion decays by $\gamma_{i+1} < 1$) is indeed equivalent to a milder IN in the sense that the distortion grows $\gamma_{i+1}/\gamma_i > 1$ smaller than $1/\gamma_i$. Similarly, OUT followed by IN is equivalent to a mild OUT in the sense that distortion decays by $\gamma_i/\gamma_{i+1} < 1$. Both cases lead towards the class boundary by a factor that tends to 1. If the algorithm keeps alternating between OUT and IN and we only look at the OUT iterates (remember, all attacks output the successful iterate of least distortion), this is equivalent to strictly decreasing distortion. This behavior is more stable than having a constant parameter as in DDN.

From all the possible increasing sequences that go to 1 as i goes to the maximum number of iterations, we pick the simplest one: a linear sequence. All this behaviour is controlled by a single parameter, which simplifies the algorithm. That is the only heuristic.

5.3 Experiments

In this section we compare our method *boundary projection* (BP) to the attacks presented in section 2.5, namely: FGSM [GSS14], I-FGSM [KGB16], PGD_2 Equation 2.15, C&W [CW17], and DDN [RHO+19]. This benchmark is carried out on three well-known datasets, with a different neural network for each.

We evaluate an attack by its runtime, two global statistics P_{suc} and $\overline{D}$, and by an operating characteristic curve $D \to \mathsf{P}(D)$ measuring distortion *vs*. probability of success as described below.

Since we focus on the *speed-distortion trade-off*, we measure the required time for all attacks. For the iterative attacks, the complexity of one iteration is largely dominated by the computation of the gradient, which requires one forward and one backward pass through the network. It is thus fair to gauge their complexity by this number, referred to as *iterations* or '# Grads'. Indeed, the actual timings of 100 iterations of I-FGSM, PGD_2, C&W, DDN and BP are 1.08, 1.36, 1.53, 1.46 and 1.17 s/image on average respectively on ImageNet, using Tensorflow, Cleverhans implementation for I-FGSM and C&W, and authors' implementation for DDN.

We measure distortion when *the adversarial images are quantized* by rounding each element to the nearest element in $\mathcal{X}$. This makes sense since adversarial images are meant to be stored or communicated as images rather than real-valued matrices. DDN and BP adversarial images are already quantized. For reference, we report distortion without quantization in section 5.4.

5.3.1 Parameters of the attacks

Below we specify the attacks parameters for each dataset. chapter 3 details the networks and training parameters.

For the distortion constrained attacks *i.e*. FGSM, I-FGSM and PGD_2, we test a set of ϵ and calculate P_{suc} and $\overline{D}$ according to our evaluation protocol (*cf*. section 3.3). For C&W, we test several parameter settings and pick up the optimum setting as specified below. For DDN, the parameter settings are the default [RHO+19], *i.e*. $\epsilon_0 = 1.0$ and $\gamma = 0.05$.

MNIST [LCB10]. $\alpha = 0.08$ for I-FGSM, $\alpha = \epsilon/2$ for PGD_2. For C&W: for 5×20 iterations [1], learning rate $\eta = 0.5$ and initial constant $\lambda = 1.0$; for 1×100 iterations, $\eta = 0.1$ and $\lambda = 10.0$.

CIFAR10 [Kri09]. $\alpha = 0.08$ for I-FGSM, $\alpha = \epsilon/2$ for PGD_2. For C&W: for 5×20 iterations, learning rate $\eta = 0.1$ and initial constant $\lambda = 0.1$; for 1×100 iterations, $\eta = 0.01$, and $\lambda = 1.0$.

ImageNet [KGB+18a] $\alpha = 0.08$ for I-FGSM, $\alpha = 3$ for PGD_2. For C&W: for 5×20 iterations, learning rate $\eta = 0.01$ and initial constant $\lambda = 20$; for 1×100 iterations, $\eta = 0.01$ and $\lambda = 1.0$.

Table 5.1 – Success probability P_{suc} and average distortion $\overline{D}$ of our method BP on ImageNet with different *quantization strategies*.

	# Grads	P_{suc}	$\overline{D}$
Rounding in the end	20	1.00	1.44
	100	1.00	1.43
Rounding at each iteration	20	1.00	0.41
	100	1.00	0.32
Rounding with Q_{IN}, Q_{OUT}	20	1.00	0.35
	100	1.00	0.28

5.3.2 Experimental investigations

Before addressing the benchmark, this section investigates on the role of quantization and of the parameters in BP.

Quantization. Table 5.1 shows the critical role of quantization in our method BP. Since this attack is iterative and works with continuous vectors, one may quantize only at the end of the process, or at the end of each iteration. Another option is to anticipate the detrimental action of quantizing by adapting the length of each step accordingly, as done by $Q_{\text{IN}}(\cdot)$ and $Q_{\text{OUT}}(\cdot)$ in

1. C&W performs line search on λ: "5×20" means 5 values of λ, 20 iterations for each.

Algorithm. 2. The experimental results show that the key is to quantize often so to let the next iterations compensate. Anticipating and adapting gives a substantial extra improvement.

Parameter Study. There are two parameters in BP: α and γ_{min}. Both determine the step size of stage 1, while γ_{min} also determines the step size of stage 2. We consider 4 values for α, *i.e.* $1, 2, 3, 4$ and 9 values for γ_{min}, *i.e.* $0.1, 0.2, ..., 0.9$. For each pair of values, we evaluate BP with 20 iterations on a validation set, which we define as a random subset sampled of the training set: 10000 images for MNIST and CIFAR10, and 1000 images for ImageNet. As shown in Figure 5.3, success probability is close to one in all cases, while average distortion is in general stable up to $\gamma_{\min} = 0.8$. We choose $\alpha = 2$ and $\gamma_{\min} = 0.7$ for all experiments.

5.3.3 Benchmark

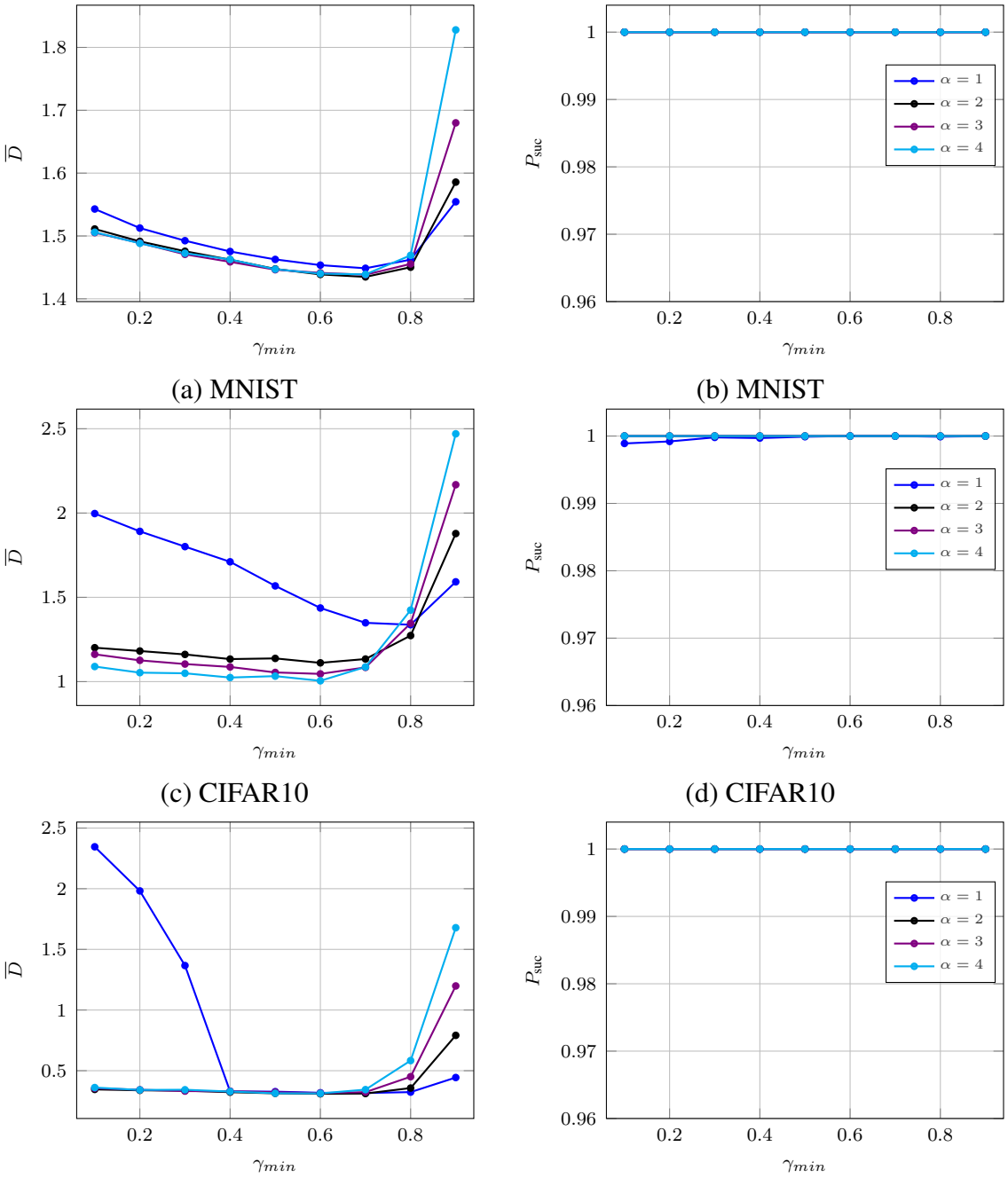
This section compares different attacks mentioned in this paper, with or without quantization, on various classifiers.

Table 5.2 – Success probability P_{suc} and average distortion $\overline{D}$ with quantization. P_{upp} is the success rate under distortion budget $D_{\text{upp}} = 2$ for MNIST, 0.7 for CIFAR10, and 1 for ImageNet.

		MNIST			CIFAR10			ImageNet		
Attack	# Grads	P_{suc}	$\overline{D}$	P_{upp}	P_{suc}	$\overline{D}$	P_{upp}	P_{suc}	$\overline{D}$	P_{upp}
FGSM	1	0.99	5.80	0.00	0.95	5.65	0.00	0.88	9.18	0.00
I-FGSM	20	1.00	3.29	0.17	1.00	3.54	0.00	1.00	4.90	0.00
	100	1.00	3.23	0.18	1.00	3.53	0.00	1.00	4.90	0.00
PGD_2	20	1.00	1.80	0.63	1.00	0.66	0.76	0.63	3.63	0.00
	100	1.00	1.74	0.66	1.00	0.60	0.84	1.00	1.85	0.00
C&W	5×20	1.00	1.94	0.56	0.99	0.56	0.81	1.00	1.70	0.00
	1×100	0.98	1.90	0.57	0.87	0.38	0.76	0.97	2.57	0.00
DDN	20	0.82	1.40	0.70	1.00	0.63	0.74	0.99	1.18	0.05
	100	1.00	1.41	0.87	1.00	0.21	0.98	1.00	0.43	0.97
BP (this work)	20	1.00	1.45	0.86	0.97	0.49	0.87	1.00	0.35	0.96
	100	1.00	1.37	0.91	0.97	0.30	0.97	1.00	0.28	1.00

Attack evaluation with quantization. Table 5.2 summarizes the global statistics of the benchmark. Figure 5.4 offers a more detailed view per dataset with operating characteristic.

In terms of average distortion, all iterative attacks perform much better than the single-step FGSM. The performances of C&W are on par with those of I-FGSM, which is unexpected for

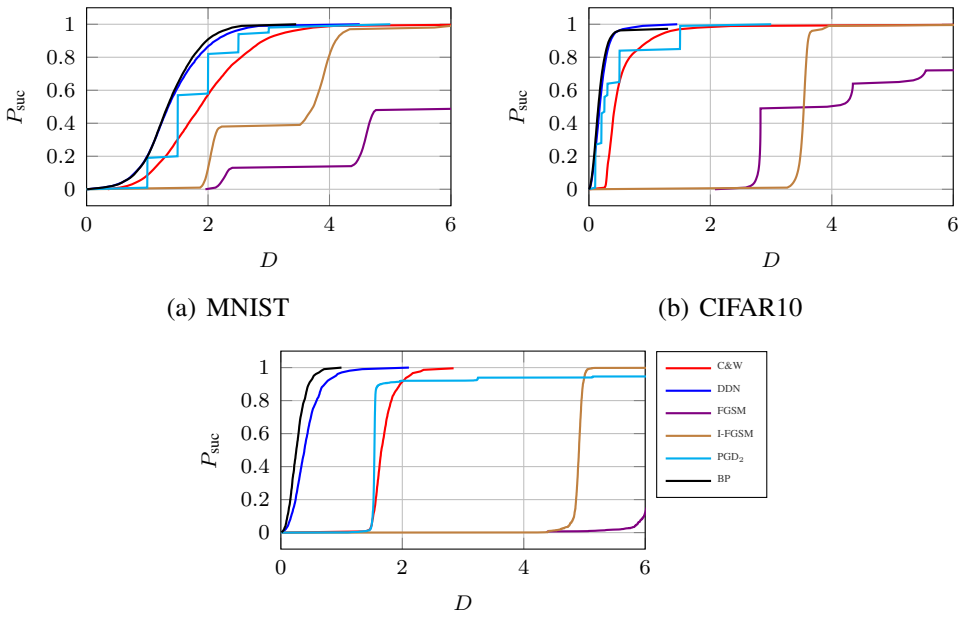


(a) MNIST (b) MNIST

(c) CIFAR10 (d) CIFAR10

(e) ImageNet (f) ImageNet

Figure 5.3 – Success probability P_{suc} and average distortion $\overline{D}$ for different values of parameters α and γ_{min} of BP with 20 iterations.



(a) MNIST (b) CIFAR10

(c) ImageNet

Figure 5.4 – Operating characteristics on MNIST, CIFAR10 and ImageNet. The number of iterations is 5×20 for C&W and 100 for I-FGSM, PGD$_2$, DDN and our BP.

this more elaborated attack design. The reason is that C&W is put under stress in our benchmark. It usually requires a bigger number of iterations to deliver high quality images. Note that it is possible to avoid the line search on parameter λ as shown in row 1×100. However, it requires a fine tuning so that this single value works over all the images of the dataset. This is not possible for ImageNet.

DDN and our method BP are clearly ahead of the benchmark. DDN yields lower distortion on MNIST at fewer iterations, but its probability of success is not satisfying. DDN is indeed better than BP only on CIFAR10 at 100 iterations. Figure 5.4 reveals that the two attacks have similar operating characteristic on all datasets but this is because it refers to 100 iterations.

In terms of success rate, FGSM fails on MNIST; on CIFAR10, I-FGSM and PGD$_2$ fail as well; finally on ImageNet, C&W fails too. DDN also fails on ImageNet at 20 iterations.

Increasing the number of iterations helps but not at the same rate for all the attacks. For instance, going from 20 to 100 iterations is waste of time for I-FGSM while it is essential for decreasing the distortion of DDN or making PGD$_2$ efficient on ImageNet. Most importantly, our attack BP brings a dramatic improvement in the speed *vs*. distortion trade-off. Just within 20 iterations, the distortion achieved on ImageNet is very low compared to the others. section 5.4 shows the speed *vs*. distortion trade-off in more detail.

Statistics of BP stages are as follows: On CIFAR-10 and MNIST, Stage 1 takes 7 iterations on average. On ImageNet, Stage 1 takes on average 3 iterations out of 20, or 8 iterations out of 100. section 5.6 shows examples of images along with corresponding adversarial examples and perturbations for different methods.

5.4 Predicting distortion after quantization

This section aims at predicting when the quantization cancels the perturbation, assuming that they are independent of each other. Iteration i starts with a quantized image $\mathbf{y}_i \in \mathcal{X}$, adds update $\delta \in \mathbb{R}^n$, and then quantizes s.t. $\mathbf{y}_{i+1} = Q(\mathbf{y}_i + \delta)$. Quantization [Ume12] is done by rounding with step $\Delta := 1/(l-1)$ [2]):

$$y_{i+1,j} = y_{i,j} + e_j \tag{5.11}$$

if $u_j \in (e_j - \Delta/2, e_j + \Delta/2]$ with $e_j \in \Delta\mathbb{Z}$. Border effects where $y_{i,j} + e_j \notin \mathcal{X}$ are neglected.

We now take a statistical point of view where the update is modeled by a random vector $\mathbf{U}$ uniformly distributed over the hypersphere of radius ρ, the norm of the perturbation before quantization. The quantization noise is now random, denoted by $E_j \in \Delta\mathbb{Z}$ for pixel j, introducing a distortion

$$D^2 = \sum_{j=1}^{n} (y_{i+1,j} - y_{i,j})^2 = \sum_{j=1}^{n} E_j^2. \tag{5.12}$$

The expectation of a sum is always the sum of the expectations, whatever the dependence between the summands: $\mathbb{E}(D^2) = \sum_{j=1}^{n} \mathbb{E}(E_j^2) = n\mathbb{E}(E_j^2)$. This expectation is not null if $\mathbb{P}(|E_j| \geq \Delta) > 0$ since $\mathbb{E}(D^2) \geq n\Delta^2\mathbb{P}(|E_j| \geq \Delta)$.

This r.v. E_j takes a value depending on the scalar product $S_j := \mathbf{U}^\top \mathbf{c}_j$, where $\mathbf{c}_j$ is the j-th canonical vector. This scalar product lies in $[-\rho, \rho]$, so that $\mathbb{P}(E_j \geq \Delta) = 0$ if $\Delta/2 > \rho$. Otherwise, $|E_j| \geq \Delta$ when $|S_j| \geq \Delta/2$, which happens when $\mathbf{U}$ lies inside the dual hypercone of axis $\mathbf{c}_j$ and semi-angle $\theta = \arccos(c)$ with $c := \Delta/2\rho$. The probability of this event is equal to the ratio of the solid angles of this dual hypercone and the full space $\mathbb{R}^n$. This quantity can be expressed via the incomplete regularized beta function I:

$$\mathbb{P}(|E_j| \geq \Delta) = \begin{cases} 1 - I_{c^2}(1/2, (n-1)/2), & \text{if } c \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

For large n, this probability approximately equals $2\Phi(-\sqrt{n}c)$.

2. l is the quantization level, for instance, here is 256.

In the end, the lower bound of $\mathbb{E}(D^2)$ after quantization depends on Δ, n, and ρ the norm of the perturbation before quantization. When $n = 3 * 299^2$ (*i.e.* ImageNet), this lower bound equals Δ^2 (*i.e.* the smallest distortion if not null) for $\rho = 0.1$. When the update has a smaller norm, quantization is likely to eliminate it, $\mathbf{y}_{i+1} = \mathbf{y}_i$, and the attack wastes one iteration.

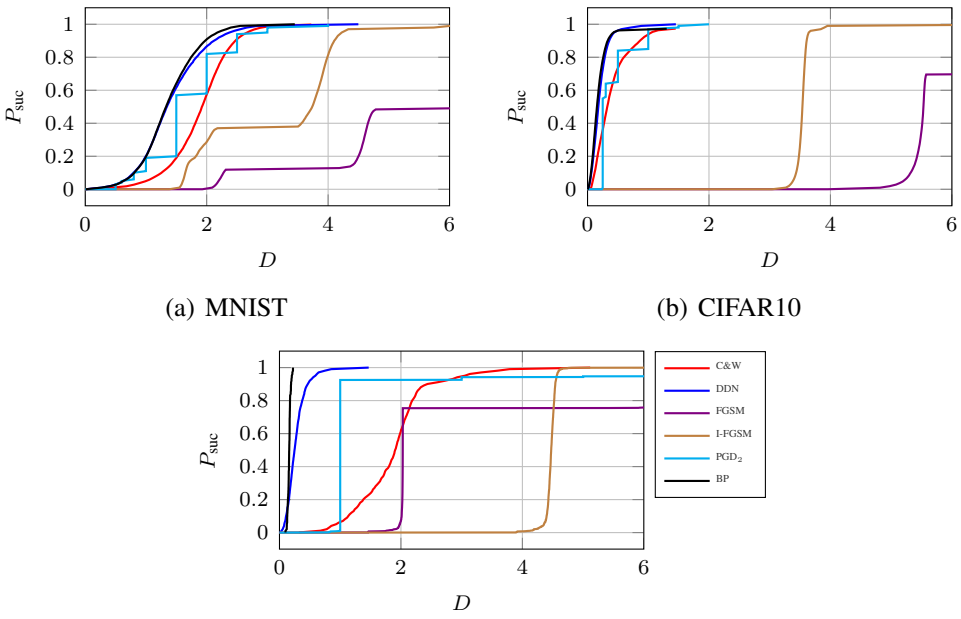
Attack evaluation without quantization. Table 5.3 is the equivalent of Table 5.2 but without the integral constraint: the attack is free to output any real matrix provided that the pixel values all belong to $[0, 1]$. When the distortion is large, there is almost no difference.

When an attack delivers low distortion on average with real matrices, the quantization may lower the probability of success. This is especially true with the iterative attacks finding adversarial examples just nearby the border between the two classes. Quantization jeopardizes this point and sometimes brings it back in the true class region. More importantly, the impact of the quantization on the distortion is no longer negligible. This is clearly visible when comparing Table 5.3 and Table 5.2 for DDN and BP over ImageNet.

Similarly, Figure 5.5 is the equivalent of Figure 5.4 without the integral constraint. By comparing the two figures, it can be seen that PGD_2 and C&W, but also DDN and BP, are improving on ImageNet by having significantly lower distortion. This agrees with measurements of success rate in Table 5.3, where PGD_2 and C&W are not failing as they do in Table 5.2 with quantization. Our BP is still the strongest attack over all datasets.

Table 5.3 – Success probability P_{suc} and average distortion $\overline{D}$ *without quantization*. P_{upp} measured at $D_{\text{upp}} = 2$ for MNIST, 0.7 for CIFAR10, and 1 for ImageNet.

		MNIST			CIFAR10			ImageNet		
Attack	# Grads	P_{suc}	$\overline{D}$	P_{upp}	P_{suc}	$\overline{D}$	P_{upp}	P_{suc}	$\overline{D}$	P_{upp}
FGSM	1	0.99	5.81	0.00	0.97	4.78	0.00	0.85	3.02	0.00
I-FGSM	20	1.00	3.22	0.27	1.00	3.54	0.00	1.00	4.47	0.00
	100	1.00	3.16	0.29	1.00	3.53	0.00	1.00	4.47	0.00
PGD_2	20	1.00	1.76	0.63	1.00	0.51	0.77	0.64	3.94	0.36
	100	1.00	1.70	0.66	1.00	0.43	0.85	0.95	1.11	0.61
C&W	5×20	1.00	1.93	0.56	1.00	0.56	0.81	1.00	1.37	0.23
	1×100	1.00	1.89	0.57	0.97	0.38	0.84	1.00	1.87	0.06
DDN	20	0.82	1.39	0.70	1.00	0.62	0.74	1.00	0.76	0.95
	100	1.00	1.41	0.87	1.00	0.20	0.98	1.00	0.28	0.99
BP (this work)	20	1.00	1.41	0.86	0.97	0.33	0.87	1.00	0.20	1.00
	100	1.00	1.35	0.91	0.97	0.18	0.97	1.00	0.16	1.00



(a) MNIST (b) CIFAR10

(c) ImageNet

Figure 5.5 – Operating characteristics on MNIST, CIFAR10 and ImageNet *without quantization*. The number of iterations is 5×20 for C&W and 100 for I-FGSM, PGD$_2$, DDN and our BP.

Attack evaluation on robust models. Table 5.4 is similar to Table 5.2 but is evaluating attacks on robust models. In particular, on MNIST and CIFAR10, we use the same models as described in subsection 5.3.1, which we adversarially train according to [MMS+17]. On ImageNet, we use off-the shelf [3] InceptionV3 obtained by *ensemble adversarial training* on four models [TKP+17].

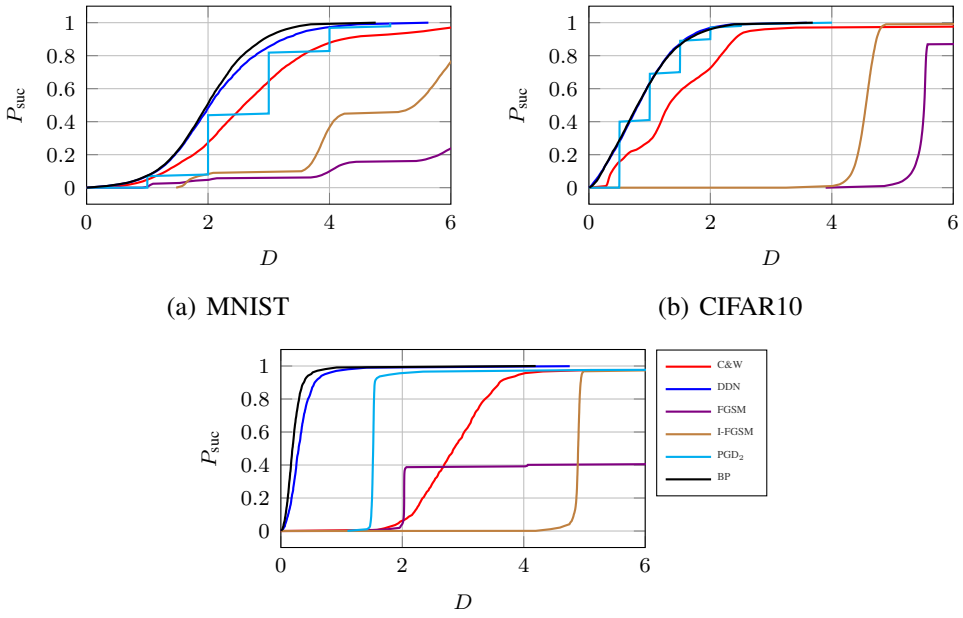
In general,DDN and BP outperform all other attacks in terms of either average distortion $\overline{D}$ or success rate P_{upp}. On ImageNet in particular, all other attacks have significantly higher distortion and fail in terms of success rate. DDN has significantly greater distortion than BP and fails in terms of success rate at 20 iterations, while at 100 iterations BP still has lower distortion. DDN and BP have similar performance on CIFAR10. On MNIST, DDN fails in terms of probability of success at 20 iterations, while at 100 iterations BP is superior.

Figure 5.6 shows a more detailed view of operating characteristics, similarly to Figure 5.4 for models trained on natural images. We can see that BP is still ahead of the competition. It is close to DDN, but this is because Figure 5.6 refers to 100 iterations. The two attacks outperform all others by a large margin.

3. `https://github.com/tensorflow/models/tree/master/research/adv_imagenet_models`

Table 5.4 – Success probability P_{suc}, average distortion $\overline{D}$, and success rate P_{upp} under *adversarial training* with PGD as the reference attack, following [MMS+17] for MNIST and CIFAR10; and *ensemble adversarial training* [TKP+17] for ImageNet. P_{upp} measured at $D_{\text{upp}} = 2$ for MNIST, 0.7 for CIFAR10, and 1 for ImageNet.

		MNIST [MMS+17]			CIFAR10 [MMS+17]			ImageNet [TKP+17]		
Attack	# Grads	P_{suc}	$\overline{D}$	P_{upp}	P_{suc}	$\overline{D}$	P_{upp}	P_{suc}	$\overline{D}$	P_{upp}
FGSM	1	0.48	5.69	0.05	0.98	6.21	0.00	0.44	2.98	0.00
I-FGSM	20	1.00	4.99	0.08	1.00	4.53	0.00	1.00	4.92	0.00
	100	1.00	4.99	0.08	1.00	4.56	0.00	1.00	4.93	0.00
PGD_2	20	0.99	2.76	0.19	1.00	1.03	0.41	0.76	2.14	0.00
	100	1.00	2.68	0.20	1.00	1.02	0.41	0.98	1.59	0.00
C&W	5×20	0.99	2.75	0.27	0.98	1.41	0.22	0.98	2.85	0.00
	1×100	0.94	2.22	0.34	0.60	0.77	0.27	0.97	2.41	0.00
DDN	20	0.43	1.61	0.32	0.97	0.92	0.41	0.99	1.10	0.23
	100	1.00	2.12	0.48	1.00	0.87	0.42	1.00	0.34	0.98
BP (this work)	20	1.00	2.17	0.46	1.00	0.94	0.41	1.00	0.35	0.94
	100	1.00	2.00	0.51	1.00	0.88	0.43	1.00	0.23	0.99

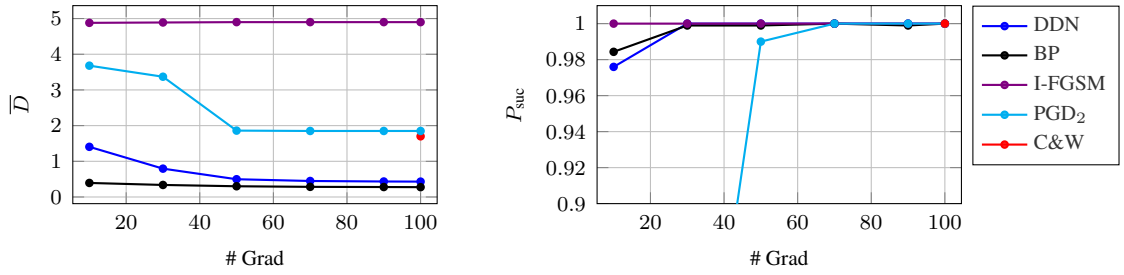


(a) MNIST (b) CIFAR10

(c) ImageNet ens4

Figure 5.6 – Operating characteristics of attacks against *robust models*: *adversarial training* with PGD as the reference attack [MMS+17] for MNIST and CIFAR10, and *ensemble* adversarial training [TKP+17] for ImageNet. The number of iterations is 5×20 for C&W and 100 for I-FGSM, PGD_2, DDN and our BP.

Speed *vs*. distortion trade-off. Figure 5.7(a) is a graphical view of some results reported in Table 5.2 with more choices of number of iterations between 20 and 100, and only for ImageNet where our performance gain is the most significant. Just within 20 iterations, its distortion $\overline{D}$ is already so much lower than that of other attacks, that its decrease (-20% at 100 iterations) is not visible in Figure 5.7. On the contrary, more iterations are useless for I-FGSM, and PGD_2 achieves low distortion only with more than 50 iterations. Figure 5.7(b) confirms that the probability of success is close to 1 for both DDN and BP for the numbers of iterations considered.



(a) (b)

Figure 5.7 – (a) Average distortion *vs*. number of iterations on ImageNet. I-FGSM is not improving with iterations because it is constrained by ϵ. (b) Corresponding probability of success.

5.5 Defense evaluation with adversarial training

We also test under adversarial training [GSS14]. The network is re-trained with a dataset composed of the original training set and the corresponding adversarial images. This training is special: at the end of each epoch, the network is updated and fixed, then the adversarial images for this new update are forged by some reference attack, and the next epoch starts with this new set. This is tractable only if the reference attack is fast. We use it with FGSM as the reference attack.

It is more interesting to study DDN and BP as alternatives to FGSM: at 20 iterations, they are fast enough to play the role of the reference attack in adversarial training. In this case, we follow the training process suggested by [RHO+19]: the model is first trained on clean examples, then fine-tuned for 30 iterations with adversarial examples. As shown in Table 5.5, DDN and BP perform equally better than FGSM on CIFAR10, in terms of either average distortion or success rate. Among the reliable attacks (*i.e*. whose P_{suc} is close to 1), the worst attack now requires a distortion three times larger than the distortion of the worst attack without defense. In the same

way, on MNIST, the distortion of the worst case attack doubles going from 1.37 (baseline) to 2.73 (BP defense). In most cases, BP is a better defense than DDN, forcing the attacker to have 20% more distortion. Note that for a given defense, the strongest attack is almost always BP.

Table 5.5 – Success probability P_{suc}, average distortion $\overline{D}$, and success rate P_{upp} under *adversarial training* defense with I-FGSM, DDN, or BP as the reference attack. P_{upp} measured at distortion $D_{\text{upp}} = 2$ for MNIST, and 0.7 for CIFAR10.

		MNIST						CIFAR10					
Attack $\rightarrow$		PGD_2		DDN		BP		PGD_2		DDN		BP	
$\downarrow$ Defense		20	100	20	100	20	100	20	100	20	100	20	100
baseline	P_{suc}	1.00	1.00	0.82	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.97
	$\overline{D}$	1.80	1.74	1.40	1.41	1.45	1.37	0.66	0.59	0.63	0.21	0.49	0.30
	P_{upp}	0.63	0.66	0.70	0.87	0.86	0.91	0.76	0.84	0.74	0.98	0.87	0.97
FGSM	P_{suc}	1.00	1.00	0.51	1.00	0.89	1.00	1.00	1.00	1.00	1.00	0.99	1.00
	$\overline{D}$	1.92	1.85	1.28	1.60	1.92	1.58	0.68	0.62	0.59	0.24	0.67	0.24
	P_{upp}	0.48	0.53	0.44	0.72	0.53	0.73	0.72	0.79	0.80	0.98	0.72	0.99
DDN	P_{suc}	0.99	1.00	0.29	1.00	0.99	1.00	1.00	1.00	0.98	1.00	1.00	1.00
	$\overline{D}$	3.03	2.89	1.68	2.38	2.69	2.27	0.95	0.94	0.77	0.71	0.75	0.68
	P_{upp}	0.12	0.14	0.20	0.32	0.28	0.34	0.52	0.52	0.54	0.55	0.56	0.58
BP	P_{suc}	0.94	0.96	0.36	1.00	0.95	1.00	1.00	1.00	0.97	1.00	1.00	1.00
	$\overline{D}$	3.14	3.12	1.65	2.81	2.98	2.73	0.96	0.94	0.75	0.70	0.76	0.69
	P_{upp}	0.15	0.15	0.24	0.27	0.25	0.26	0.55	0.55	0.57	0.59	0.56	0.59

5.6 Adversarial image examples

Figure 5.8 shows the worst-case ImageNet examples for BP along with the images generated by all methods and the corresponding normalized perturbations. FGSM has the highest distortion over all methods and BP the lowest. DDN has the highest ∞-norm distortion. Observe that for no method is the perturbation visible, although this is a worst-case example.

5.7 Conclusion

The main idea of BP is to travel on the manifold defined by the class boundary while seeking to minimize distortion. This travel is operated by the refinement stage, which alternates on both sides of the boundary, but attempts to stay mostly in the adversarial region. Referring to section 2.5, BP is in effect doing for the *success constrained* problem what PGD_2 is doing for

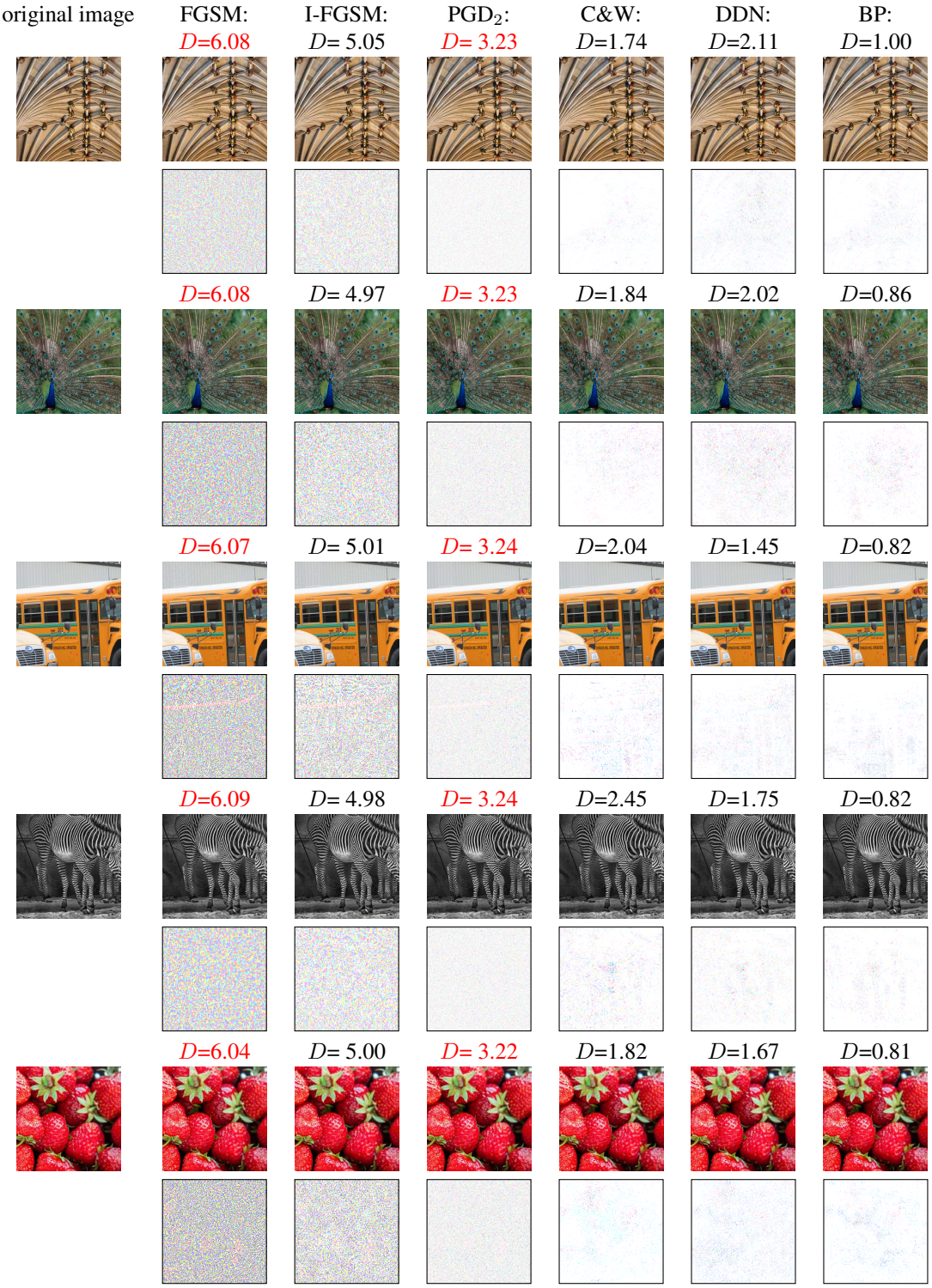


Figure 5.8 – Original (left), adversarial (top row) and scaled perturbation (below) images against InceptionV3 on ImageNet. The five images are the worst 5 images for BP requiring the strongest distortions, yet these are smaller than the distortions necessary with all other methods (The red color means that the forged image is not adversarial). Perturbations are inverted (low is white; high is colored, per channel) and scaled in the same way for a fair comparison.

the *distortion constrained* problem: BP minimizes distortion on the class boundary manifold (a level set of the classification loss), while PGD_2 minimizes the classification loss on a sphere (a level set of the distortion).

BP also takes into account the detrimental effect of *quantization*. By doing so, the amplitude of the perturbation is controlled from one iteration to another. The main advantage of our attack is the small number of iterations required to achieve both reliability (probability of success close to one) and high quality (low average distortion).

PART II

Defense

CHAPTER 6

PATCH REPLACEMENT

The architecture of recent DNN proved to be robust, that is, images are correctly classified even when they contain a fair amount of random noise. The very same architectures, however, completely fail to classify images that contain imperceptible *adversarial* perturbations.

This work investigates the difference between random noise and adversarial perturbation. It highlights some of the respective properties of random noise and adversarial perturbation by observing how they propagate through a DNN and how and when they get amplified. Based on this information, we propose a new adversarial defense mechanism called *patch replacement* that transforms the features of the input image at various levels to get rid as much as possible of adversarial artifacts. Patch replacement improves robustness in white/gray-box scenarios using the training data. Without training DNN models, patch replacement is not expensive.

6.1 Introduction

Improving the robustness of neural networks against random noise is easier than improving the robustness against adversarial perturbation. Popular DNN architectures, such as AlexNet [KSH12] or Inception [SVI+16], are much more robust against random noise or transformations such as random cropping, reshaping, rotations than against adversarial perturbations. Although the magnitude of typical adversarial perturbation is much smaller than that of typical random noise, the impact of adversarial perturbation is much greater on the output of the network.

So a fundamental question can be raised: *What is so different between random noise and adversarial perturbation since their impact on classification is so contrasting?*

6.1.1 Random noise vs. adversarial perturbation

To investigate this question, we design a simple experiment to check what happens inside the neural network when we inject random noise or adversarial perturbation into the input. We measure the magnitude of distortions between intermediate activation from legitimate images

and their (random/adversarial) noisy versions. Let us use x_o to denote the original image and r to denote the perturbation that is either random or adversarial in this experiment. In order to make the magnitude of distortions comparable, we calculate the relative distortion by normalizing it by $\|f_i(x_o) - f_i(x_o + r)\|_2 / \|f_i(x_o)\|_2$, where f_i refer to the activation at the i^{th} layer.

We run an experiment with ResNet-50 which is a DNN model with 50 layers. We observe the distortion at various levels, *i.e.* $0^{th}, 1^{st}, 4^{th}, 7^{th}, 10^{th}, 22^{th}, 40^{th}, 49^{th}$ *and* 50^{th} layer. We select these layers according to the basic architecture component of ResNet-50, *i.e.* convolutional layer and ResNet block. ResNet block consists of convolutional layers. At the beginning and the end, we measure the difference of distortion before and after a convolutional layer because the resolution changes greatly. In the middle, we measure the difference before and after ResNet blocks because the difference is rather small. It is worth mentioning that layer zero is the image, layer one is the output of the first convolution layer and the 50^{th} layer gives the logit vector before prediction.

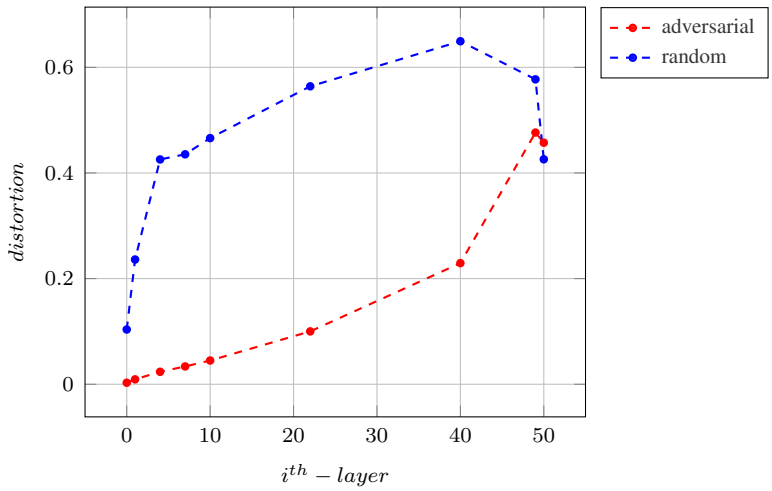


Figure 6.1 – *Relative average distortion over* 1000 *ImageNet images of intermediate activations of a neural network for noisy input images.* The red line presents the adversarial perturbation generated by the DDN attack [RHO+19]. The blue line corresponds to random noise following a normal distribution with mean 0 and variance 0.05. The neural network is ResNet-50, whose accuracy on 1000 original legitimate images is 75.7%, on images with random noise is 68.0% and on adversarial images is 0.1%.

Figure 6.1 shows that the magnitude of adversarial perturbations increases slowly at the beginning and increases quickly at the end, while the magnitude of random noise increases quickly at the beginning, stabilizes in the middle, then decreases at the end. It is not surprising.

Adversarial perturbation is optimized to behave in this way, *i.e*. small at the beginning and large at the end. Random noise increases significantly at the early layer because the max-pooling that picks the largest values locally amplifies the magnitude of distortion. It decreases just before the last layer because the average pooling takes the average value of the local window and it weakens the effect of that noise.

However, even at the level of the image itself, the magnitude of the random noise is way larger than that of adversarial perturbations. This is true anywhere in the network except in the last layers where the adversarial distortion increases drastically. Interestingly, even if the random and adversarial distortions are roughly identical at the last layer, the prediction is different. This phenomenon may be because, in the last layer, *i.e*. the logit layer, adversarial perturbations mainly accumulate on a particular class so that the logit of this class is greater than the logit of the ground truth. By contrast, random noise distributes among different classes. As a result, the logit of the ground truth class remains the highest.

To confirm this, we measure the entropy of the predicted class probability distribution and take the average of over 1000 images from ImageNet. We get 0.76 on legitimate images, 0.69 on adversarial images, and 0.93 on images with random noise. Considering there are 1000 classes, even though the difference of these numbers is not great, it may be sufficient to conclude that random noise results in more evenly distributed perturbation on the predicted probabilities, while adversarial perturbations affect fewer classes but by a larger amount.

Motivation. According to this simple experiment, we conjecture that networks are robust to perturbations if those do not accumulate in a specific class. Adversarial perturbations succeed to accumulate and affect key features so that they fool networks. These key features, grasped by networks from the training data, contain adequate semantic information [BU05, LCW+20] that encourages networks to make correct predictions. To protect neural networks from adversarial perturbations, we propose not only to transform inputs but also their features and make their behavior close to images with random noise instead of adversarial perturbation.

The idea of patch replacement is to replace the features extracted from potentially adversarial input images with similar features that have been extracted from legitimate images used at training time. In detail, we divide the features extracted from the images into a lot of small patches. At training time, a dictionary is computed based on patches of legitimate images and features, while at testing time, each patch is replaced by a similar one from the dictionary. Since Figure 6.1 indicates that perturbations at the early layers affect less than perturbations at the last layers, such transformation should be applied to the early layers. In this way, the input is

reconstructed by training data, which is unknown to the attacker.

This strategy might degrade the classification performance on legitimate images because it replaces the inputs with their nearest neighbors according to the L_2 norm distance. The similarity on L_2 norm distance is not the same as the semantic similarity. As a result, part of the semantic information of inputs is lost or altered. To reduce the severity of this loss, we divide the features into *small* patches, conjecturing that the substitution will be less harmful. The smaller the patches are, the more similar the reconstructed images/features are compared to the inputs. As a result, the model gains robustness to adversarial images while not losing too much on legitimate images.

We expect this patch replacement strategy to be inexpensive since there is no need to train the neural network from scratch. For the same reason, this defense is also easy to adapt to any neural network if needed. The usage of the dictionary in the testing phase increases the complexity of adversarial attacks even if the attacker is aware of the patch replacement defense. It is much more complicated to circumvent a defense process if the attacker must know the entire set of images used at training time and if the attack has to alter a very large (but unknown) number of visual elements inside each image.

Contributions. The contributions of this work are listed as follows:

— We propose a defense method based on a transformation not only onto input images but also onto features, which is easily adapted to any neural network.

— The training data we use to learn the defense is only natural images, not focusing on any attack.

— We investigate the impact of patch replacement in different layers.

— We perform the defense on a combination of layers to improve the trade-off between accuracy on natural and attacked images.

The rest of the chapter is organized as follows. We briefly recap usual methods defending DNN against adversarial perturbations in section 6.2. Overall, we observe that most methods are very costly. We detail a few state-of-the-art input transformation methods that can however be used in practice due to their simplicity and effectiveness. We then describe the adversarial defense mechanism, *i.e.* patch replacement, in section 6.3. That method as well as its few variants are evaluated in section 6.4. Conclusions are drawn in section 6.7.

6.2 Adversarial defense: related work

All defense methods against adversarial perturbations are either expensive or vulnerable in the white-box/grey-box settings. As we discussed in section 2.6, the ones that aim to gain robustness require training DNN and are expensive. Many state-of-the-art DNNs are trained with ImageNet to produce high-quality results. Performing adversarial training in this context [GSS14, TKP+17, MMS+17] is extremely expensive because a large number of adversarial samples must be generated to then efficiently defend the network. Other approaches such as the ones adding regularizers in loss functions to improve robustness [LHL15, CLC+19], building a more robust model [HVD15, PMW+16], or trainable Gaussian noise injection at each layer on either activation or weights [HRF19], achieve decent performance on both robustness and accuracy. They are, however, extremely expensive and hard to adapt to other networks.

In contrast, transformation methods [GRCvdM17, STL+19, XWZ+17a] are cheap and easy to adapt to existing networks. Such methods apply a transformation to input images so that legitimate images and their adversarial version have the same classifier prediction. However, these methods are easily attacked when attackers are aware of their existence [ACW18].

6.2.1 Basic transformation

A defense [GRCvdM17] is proposed to remove the effect of the adversarial perturbation by using input transformation. They apply five different image transformations, *i.e.* image cropping and rescaling, bit-depth reduction, JPEG compression, total variance minimization, and image quilting.

Feature squeezing [XEQ17] also includes several transformation approaches. The principle of feature squeezing is to detect adversarial images by comparing the predicted score of the input image to the predicted score of its transformed version [XEQ17]. Here, we focus on the transform method, *i.e.* spatial smoothing, and list the transformation approaches of the two above works as follows.

Image cropping-rescaling. *Image cropping-rescaling* alters the spatial positioning of the adversarial perturbation. At testing time, the defense averages predictions over random image crops.

Bit-depth reduction. *Bit-depth reduction* performs a simple type of quantization in pixel values. In [GRCvdM17], Guo and his colleagues reduce images to 3 bits in their experiments.

JPEG compression. *JPEG compression* reduces the quality level of the images. They perform compression at a quality level of 75 out of 100.

Total variance minimization. *Total variance minimization* combines pixel dropout with total variation minimization. This method reconstructs the "simplest" image by using a small set of pixels randomly selected from the original image.

In details, for each pixel location, they assign it a boolean value according to a Bernoulli random distribution $\mathcal{B}$, when the value is $True$, *i.e.* 1, they maintain the pixel value. Then they get a pixel maintain map $\mathcal{M}$. Next, they use total variation minimization to construction an image $\mathbf{z}$ that is similar to the input image $\mathbf{x}$ by solving:

$$\min_{\mathbf{z}} \|(1 - \mathcal{M}) \times (\mathbf{z} - \mathbf{x})\|_2 + \lambda TV_p(\mathbf{z}), \tag{6.1}$$

where $\times$ denotes element-wise multiplication, and $TV_p(\mathbf{z})$ denotes the L_p-total variation of $\mathbf{z}$:

$$TV_p(\mathbf{z}) = \sum_{k=1}(\sum_{i=2} \|\mathbf{z}(i,:,k) - \mathbf{z}(i-1,:,k)\|_p + \sum_{j=2} \|\mathbf{z}(:,j,k) - \mathbf{z}(:,j-1,k)\|_p), \tag{6.2}$$

in which $\mathbf{z}(i,j,k)$ denotes the pixel value in location (i,j,k). And they use $p = 2$ in their experiments.

Image Quilting. *Image Quilting* is a non-parametric technique that synthesizes images by assembling small patches that are taken from a database of image patches. The algorithm first stores appropriate patches in the database for a predefined set of grid points and then computes minimum graph cuts in all overlapping boundary regions to remove edge artifacts. In the testing phase, the patches used to create the synthesized image are selected by finding K nearest neighbor in pixel space of the corresponding patch from the input image in the patch database and picking one of these neighbors uniformly at random.

Spatial smoothing. *Spatial smoothing* is known as blurring. Xu and his colleagues describe two types of spatial smoothing methods, *i.e. local smoothing* and *non-local smoothing*.

Local smoothing methods make use of the nearby pixels to smooth each pixel. By selecting different mechanisms in weighting the neighboring pixels, a local smoothing method is described as Gaussian smoothing, mean smoothing, or the median smoothing method.

Different from local smoothing, non-local smoothing smooths over similar pixels in a much larger area instead of just nearby pixels. For a given image patch, it first finds several similar

patches then replaces the center patch with the average of those similar patches. In this way, the noise will cancel out while preserving the edges of an object, under the condition that the mean of the noise is zero. Similar to local smoothing, there are several possible ways to weigh the similar patches in the averaging operation, such as Gaussian, mean, and median. The parameters of a non-local smoothing method typically include the search window size (a large area for searching similar patches), the patch size, and the filter strength (bandwidth of the Gaussian kernel).

6.2.2 Pixel Deflection

Pixel deflection [PMG+18] benefits from the observation that CNNs are robust to the noise produced by randomly replacing some pixels of images with pixels randomly selected from a small neighborhood. By redistributing pixel values, a subsequent wavelet-based denoising operation is applied to soften this noise as well as the adversarial perturbation.

Let $\mathcal{R}_p^r$ be a square neighborhood with apothem r centered at a pixel p. Let $\mathcal{U}(\mathcal{R})$ be the uniform distribution over $\mathcal{R}$. Let $I[p]$ indicate the value of pixel p in image I. Prakash and his colleagues first sample a pixel p from the image I, $p \sim \mathcal{U}(I)$, then sample a noise n from the square neighborhood, $n \sim \mathcal{U}(R_p^r \bigcap I)$, finally let $I'[p] = I[n]$, where I' is transformed image.

In this work, the hypothesis is that not all regions of images are equally important to a classifier and attacks attempt to add adversarial perturbation to the high activation regions. To make it more powerful, they use *Class Activation Maps (CAM)* [ZKL+16] as a robust activation map to determine whether we deflect the pixel.

Both pixel deflection and adversarial perturbation introduce noise into the input image, which is beneficial to reduce the effect. They first convert the image to YC_bC_r space to decorrelate the channels and then project the image into the wavelet domain using the discrete wavelet transformation. They model the threshold of each wavelet coefficient as a Generalized Gaussian Distribution with BayesShrink and compute the inverse wavelet transform on the shrunken wavelet coefficients. Finally, the image is converted back to the RGB format.

6.2.3 D3

Divide, Denoise and Defend (D3) [MDST18] is developed along the idea of image quilting. Moosavi-Dezfooli and his colleagues divide the input image into multiple patches and denoise each patch independently with sparse reconstruction using a dictionary of patches. They use either a novel patch selection algorithm that is optimized to improve the robustness of the classifier, or an efficient greedy algorithm, which is a variant of matching pursuit. It is as expensive as image

quilting.

The main difference between patch replacement that we describe in this chapter and D3 (as well as image quilting) is that we consider not only image space but also feature spaces. If we only apply patch replacement on images, the way patch replacement built the dictionary is much simpler and cheaper than that of D3 (as well as image quilting). It is interesting to compare our method to the two defense approaches. Unfortunately, there is no code of D3 released, while the author of the defense with image quilting gave up maintaining their code and the official code of image quilting does not work in the current environment. Patch replacement is built on Pytorch. The official image quilting is implemented in C++. It is very time-consuming to reimplement these methods according to the papers.

6.2.4 Feature denoising

He and his colleagues [XWM+19] observe that adversarial perturbations on images lead to noise in the features constructed by these networks. Motivated by this observation, they propose a new architecture design, *i.e. denoising block*, to improve the robustness towards adversarial perturbation by performing feature denoising. Although it is expensive since it needs to retrain the network from scratch, the experimental results show it performs well against adversarial attacks. The success of feature denoising indicates that the transformation of features also is helpful to increase the robustness.

Inspired by the transformation works and feature denoising, we aim at proposing an efficient method, namely patch replacement, that weakens the adversarial effect from both images and features. Applying transformation both on images and features provides more chances to improve the robustness against attacks. With the dictionary of patches, patch replacement manages to augment the complexity even if the attacker realizes the defense. More details are given in the following section.

6.3 Our method: patch replacement

Patch replacement is a defense method transforming the data flowing inside the deep neural network. The transformation attempts to remove as much as possible the potential adversarial noise in input images and in their features that are calculated by the network during its classification process. The transformation is also such that legitimate images get correctly classified into their appropriate class. Overall, this method seeks to maintain the accuracy of classification

while strongly reducing the success rate of adversarial attacks.

In general, an image to classify is turned into a feature map that forward-feeds the various layers of a network, undergoing convolutions, computations, until the final probabilities are established and the class identified. The feature map that exists at the start of the network contains the R, G, and B pixels of the image to classify. The next feature map contains the results of some convolutions applied to the pixels of the image, and so on. Across the network, the height, width, and depth of the data in successive feature maps might change. Overall, however, the feature map feeding one particular layer is a 3D matrix, and it can be divided into a multitude of small *data patches*.

The patch replacement approach is not only applied to the input images directly but also applied to features, which are the intermediate results of convolutions, pooling, and fully connected layers of a network. It is based on the observation that the adversarial perturbation transfers as noise in the features and gets amplified (see Figure 6.1) mostly across the last layers. This implies that the transformation aimed to reduce that adversarial noise should be performed at the level of the early layers of the networks where the noise is still of small amplitude. Overall, early adversarial noise elimination should improve the robustness of the neural network to attacks.

The patch replacement approach attempts to remove that noise by replacing the suspicious data patches of a specific feature with very similar pre-determined data patches that originate, however, from the off-line analysis of a very large number of legitimate images. Two comments are in order. First, two similar patches of features (or images) likely hold quite similar (visual) information. At inference time, replacing the data patches of features (or images) of the input image with similar predetermined data patches hence preserves, to some extent, the correctness of its classification. Second, predetermined patches computed over a set of legitimate images are untouched by any adversarial attack. By design, there is therefore no adversarial perturbation in the features (or images) synthesized from these predetermined patches.

Overall, the patch replacement approach removes the adversarial effects by replacing suspicious data patches with legitimate patches and preserves the quality of the classification process by carefully replacing these suspicious patches with very similar legitimate patches.

6.3.1 Features, slices and patches

To understand the patch replacement approach, we first present the concepts of *features, slices, and patches*.

In general, an image is turned into a series of features in the middle layers of a DNN. These features contain the information of the input image after some processing that includes

convolutions. As the next features contain the results of convolutions applied to the previous features (or image for the first convolutional layer), the height H and width W of the data in successive features might change according to the kernel size of the convolutions and the depth D according to the number of filter channels. We denote the feature from layer L as $\mathcal{F} := \mathsf{R}_{0L}(\mathbf{x}) \in \mathbb{R}^{W \times H \times D}$ where R_{0L} denotes the part of network from the beginning to layer L and $\mathcal{F}$ is a tensor. It is represented by the gray cuboid in Figure 6.2(a).

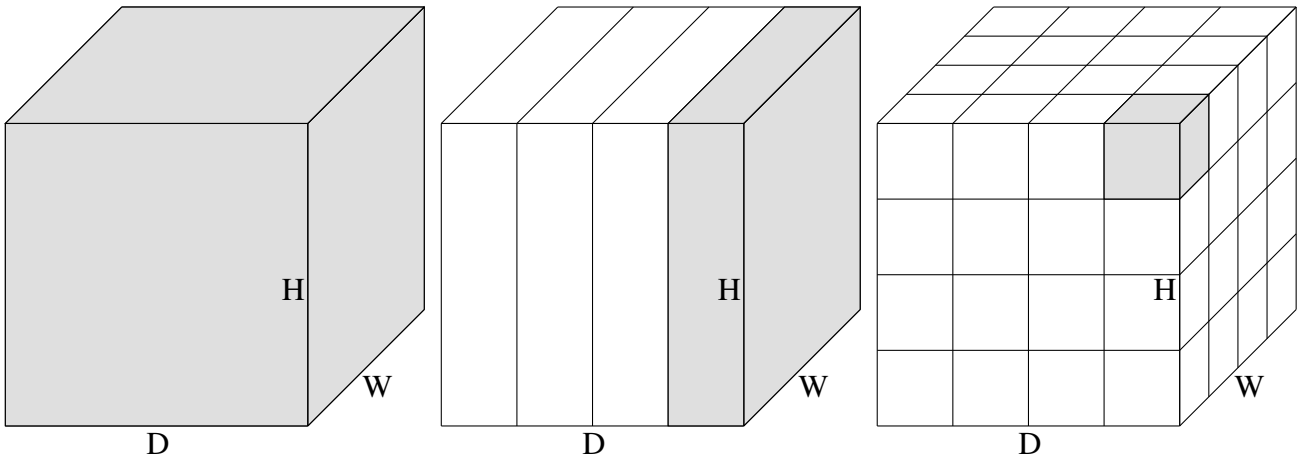
Feature can also be perceived as a concatenation of slices over filter channels, *i.e.* $\mathcal{F} = [\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_m]$ where $\mathcal{F}_k$ denotes a slice. A slice is a collection of features in contiguous filter channels. Slice $\mathcal{F}_k := \mathcal{F}^{(k-1)d+1:kd} \in \mathbb{R}^{W \times H \times d}$ is from channel $(k-1)d+1$ to kd, in which d is the depth of the slice and $d \times m = D$. It is represented by the gray rectangular cuboid in Figure 6.2(b).

Slice $\mathcal{F}_k$ can be decomposed into sub-tensors with same depth but smaller width and height over spacial location. This sub-tensor is called a patch $\mathcal{P}_{ijk}$ where i, j denotes the width and height position and k indicates the patch is from $\mathcal{F}_k$. More precisely, patch $\mathcal{P}_{ijk} := \mathcal{F}_k(i-a : i+a, j-a : j+a)$ is a sub-tensor of size $(2a+1) \times (2a+1) \times d$ at spatial location (i, j) from slice $\mathcal{F}_k$. We obtain different patches by moving the center and the location of the center can be dense or sparse. In the sparse case, the center moves with step size c, like moving on a grid with length c. When $c = (2a+1)$, there is no overlapping among the patches; while for $c < (2a+1)$, there is overlapping. The no overlapping case is represented as the gray cube in Figure 6.2(c).

Discussion: Why do we need slices? Patches are needed to combat the curse of dimensionality when identifying similar elements from the database when performing the replacement. if $\mathcal{F}$ was the replacement granule, then the nearest neighbor searching would be involving vectors with dimensionality $W \times H \times D$, which is very high. In contrast, slices are of dimensionality $W \times H \times d$, where $d = D/m$. Patches are of an even smaller dimensionality since they exist in a $(2a+1)^2 \times d$ space, and $(2a+1)^2 << W \times H$. That dimensionality reduction improves the quality of the neural network searches.

6.3.2 Codebook

The codebook is an essential component in the patch replacement approach to organize patches. It is built during the training phase and used to transform the input during the testing phase. Patch replacement does not need to train the network but we need to train the codebook. We forward an image from the training set a single time through the network and collect patches as data to learn the codebook.



(a) Feature (b) A slice (c) A patch

Figure 6.2 – In this figure, we illustrate the relationship between a feature and its slices, and patches. When an image is fed into the neural network, it results in a feature with height H, width W, and depth D as in (a). Then along with the depth, we can decompose the feature into several slices, as in (b), the gray rectangular cuboid is a slice. For each slice, we decompose it into patches with smaller width and height over spacial location, as in (c), the gray cube is a patch.

In the training phase, we forward image from the training set through the network and collect patches. For each slice $\mathcal{F}_k$, we learn the quantizer q_k whose objective is to quantize the patches that are computed from this slice. Each quantizer q_k has its own codebook $\mathcal{C}_k$ with K codewords. In the testing phase, we decompose the feature of a test image into patches and use the corresponding quantizer to find their nearest neighbors in the training data and replace them.

We investigate two different options to build a codebook, *i.e.* product quantization with a K-means and the use of an $E8$ lattice. No matter which codebook we use, the quantizer q_k returns the quantized patches $q_k(\mathcal{P})$ in the same size, so the replacement and reconstruction are always the same.

Product Quantization (PQ). After decomposing the feature into patches, it is intuitive to train a K-means model as codebook $\mathcal{C}_k$ and then we treat the set of $\mathcal{C}_k$ as a codebook $\mathcal{C}$. As a whole, searching for the nearest neighbor of each patch with the codebook $\mathcal{C}$ shares the same principle with *product quantization* [JDS10]. Comparing to use only one K-means model as a codebook for all the patches, product quantization gives a huge vocabulary.

Of course, we can train one *universal K-means codebook* for all slices. Considering that the data distributions in different slices are not the same, to achieve a similar performance

as the product quantization setting, we need much more centroids. This increases the cost of searching for the nearest neighbors in the testing phase. As a trade-off between searching cost and performance, product quantization is better than a universal one.

$E8$ lattice. $E8$ lattice is a second option for a codebook. It is a special lattice in $\mathbb{R}^8$. This method maps data points to the nearest nodes on the lattice. The number of the nodes in $E8$ lattices is the number of codewords in the codebook $\mathcal{C}$.

For the codebook trained on K-means, the patch is regarded as a whole codeword, the shape of centroids in the codebook can be changed as the shape of patches. Due to its definition, an $E8$ lattice only deals with 8-dimensional vectors. If we treat a patch as a codeword, in order to find its nearest node on $E8$ lattice, the patch needs to be first aggregated into 8-dimensional vector, and then $E8$ lattice is applied on this 8-dimensional vector. However, it is impossible to directly convert 8-dimensional vectors to legitimate patches. We do not want to build another model to mapping each $E8$ node to legitimate patches because cheap is an important advantage of the $E8$ lattice. Instead of building such kind of mapping, we propose to choose $d = 8$ for the slices so that patches consist of a series of 8-dimensional vectors and $E8$ lattice can be directly applied to them separately. As a result, we replace every 8-dimensional vector in patches by its nearest nodes on the $E8$ lattice and obtain the quantized patches $q_k(\mathcal{P})$.

On another hand, there is no training phase for the $E8$ codebook, the $E8$ lattice is cheaper than the K-means codebook.

Discussion. The quality of the codebook influences the quality of the recovered input. To maintain the accuracy of legitimate images, we need the codebook to contain rich enough details; to remove the effects of adversarial perturbation, we need the codebook to be not too fine to preserve the adversarial perturbation. It is a trade-off between the two situations.

The parameter K and the depth d of slices control the quality of the codebook $\mathcal{C}$. The more codewords are in the codebook $\mathcal{C}$, the better codebook describes the distribution of the data. If we increase K or decrease d, we succeed to increase the number of codewords in the codebook $\mathcal{C}$.

Similar to K-means, there is also a parameter scale s to control the density of codewords inside the codebook $\mathcal{C}$. When the scale is greater, the codewords are less numerous.

Adjusting the parameters, *i.e.* K and d for product quantization; c for $E8$ lattice, is one way to achieve this trade-off. Since it is expensive for tuning the parameters for product quantization, we introduce replacement strategies to aid patch replacement to achieve the trade-off.

6.3.3 Replacement Strategies

With the codebook $\mathcal{C}_k$ and quantizer q_k, we can replace $\mathcal{P}$ directly with its nearest patch $q_k(\mathcal{P})$ according to the codebook $\mathcal{C}_k$ with replacement strategies. (To simplify the notation, we denote patches as $\mathcal{P}$ in this part.)

Plain strategy. The simplest strategy is to directly replace the original patches with their nearest codewords, *i.e.* $\mathcal{P}' = q_k(\mathcal{P})$. This strategy is depicted in Figure 6.3(a).

Other than the plain replacement strategy, we proposed three different strategies with parameter ϵ.

L_2 ***strategy.*** If the Euclidean distance between $\mathcal{P}$ and $q_k(\mathcal{P})$ is less than ϵ, we keep $q_k(\mathcal{P})$. Otherwise, we project $q_k(\mathcal{P})$ on the L_2 ball of radius ϵ centered at $\mathcal{P}$. This strategy is formulated as

$$\mathcal{P}' = q_k^{L_2}(\mathcal{P}) := \begin{cases} q_k(\mathcal{P}) & \|q_k(\mathcal{P}) - \mathcal{P}\| < \epsilon \\ \mathcal{P} + \mathsf{n}(q_k(\mathcal{P}) - \mathcal{P})\epsilon & \text{otherwise,} \end{cases} \tag{6.3}$$

where $\mathsf{n}(q_k(\mathcal{P}) - \mathcal{P}) := \frac{q_k(\mathcal{P}) - \mathcal{P}}{\|q_k(\mathcal{P}) - \mathcal{P}\|}$. This strategy is shown in Figure 6.3(b).

L_∞ ***strategy.*** If the Chebyshev distance between $\mathcal{P}$ and $q_k(\mathcal{P})$ is less than ϵ, we keep $q_k(\mathcal{P})$. Otherwise, we project $q_k(\mathcal{P})$ on the L_∞ square of radius ϵ centered at $\mathcal{P}$. This strategy is formulated as

$$\mathcal{P}' = q_k^{L_\infty}(\mathcal{P}) := \mathcal{P} + \mathrm{clip}_{[-\epsilon,\epsilon]}(q_k(\mathcal{P}) - \mathcal{P}). \tag{6.4}$$

This strategy is shown in Figure 6.3(c).

Linear strategy In this case, we use a linear interpolation between the original patches $\mathcal{P}$ and their nearest codeword $q_k(\mathcal{P})$, as

$$\mathcal{P}' = q_k^{\mathrm{Linear}}(\mathcal{P}) := \mathcal{P} + \epsilon(q_k(\mathcal{P}) - \mathcal{P}). \tag{6.5}$$

This strategy is depicted in Figure 6.3(d).

Discussion. Changing the parameter of the codebook $\mathcal{C}$, for instance, K of K-means codebook, we can control the extent of how fine or coarse the vocabulary is. However, it is expensive to get

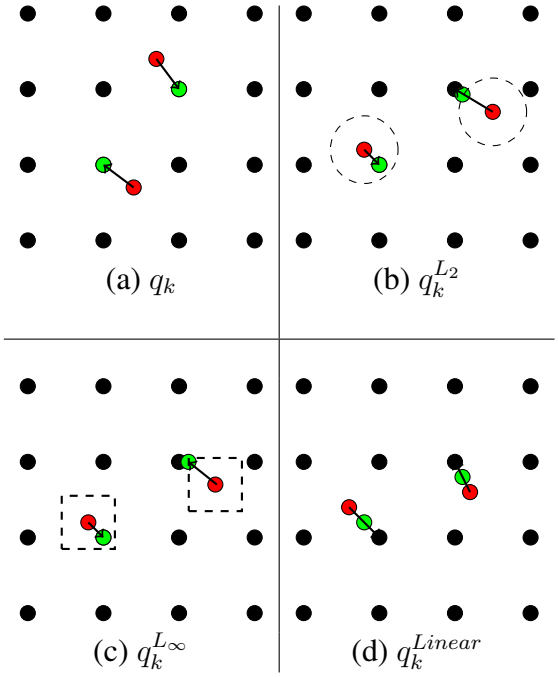


Figure 6.3 – The visualization of the four replacement strategies, *i.e.* plain replacement q_k, $q_k^{L_2}$, $q_k^{L_\infty}$, and q_k^{Linear}. The black points: codewords of the codebook $\mathcal{C}_k$. The red points: original patches $\mathcal{P}_{ijk}$. The green points: replacing patches $\mathcal{P}'_{ijk}$. The dashed lines in (b) and (c) depict the L_2 and L_∞ boundary with radius ϵ. In (b), the dashed circles show the L_2 circle with radius ϵ. If the nearest codeword is inside this L_2 norm circle, we replace $\mathcal{P}$ with $q_k(\mathcal{P})$, otherwise, we project $q_k(\mathcal{P})$ to the L_2 norm circle. In (c), the dashed boxes show the L_∞ norm square with threshold ϵ. If the nearest codeword is inside this L_∞ norm square, we replace $\mathcal{P}$ with $q_k(\mathcal{P})$, otherwise, we clip $q_k(\mathcal{P})$ to the L_∞ norm square. In (a), we always replace $\mathcal{P}$ with $q_k(\mathcal{P})$ and in (d), the replacing patch $\mathcal{P}'_{ijk}$ is calculated by a linear interpolation between $\mathcal{P}$ and $q_k(\mathcal{P})$.

the best quantization by adjusting the codebook. Aside from the codebook $\mathcal{C}$, the replacement strategy is a better way to control quantization. For each patch, if $\|\mathcal{P}_{ijk} - q_k(\mathcal{P}_{ijk})\|$ is too large, the quantized patch $q_k(\mathcal{P}_{ijk})$ is far away from original patch $\mathcal{P}_{ijk}$. Due to the limited number of training data, when the codebook is more coarse than the data manifold, a patch $\mathcal{P}_{ijk}$ might not be similar to any codeword in the codebook $\mathcal{C}_k$, which decreases the accuracy on both original and adversarial images.

To avoid this, we introduce the replacement strategy with parameter ϵ. This helps the algorithm not to distort the patch $\mathcal{P}_{ijk}$ too much.

L_2 and L_∞ strategies share the similar idea and they limit the distance between $q_k(\mathcal{P})$ and $\mathcal{P}$. The main difference is L_2 strategy regards the patch as a whole, but the L_∞ strategy regards the patch as a set of values. L_2 strategy limits the L_2 norm distance of the patch under ϵ, while L_∞ strategy limits each value of the patch not changed beyond ϵ.

Instead of limiting the difference, the linear strategy always chooses an intermediate solution between $q_k(\mathcal{P})$ and $\mathcal{P}$. It never keeps the nearest codeword $q_k(\mathcal{P})$ but interpolates between it and its original patch $\mathcal{P}$.

Using $\epsilon = 0$, for all replacement strategies, is equivalent to the original neural network without any quantization. When ϵ is large, L_2 strategy and L_∞ strategy are equal to plain strategy. Finally, when $\epsilon = 1$, the Linear strategy is equal to the plain strategy.

6.3.4 Reconstruction

After replacing all the patches $\mathcal{P}_{ijk}$ with their quantized version $\mathcal{P}'_{ijk}$, we reconstruct slice $\mathcal{F}'_k$ by merging all the $\mathcal{P}'_{ijk}$ together. When patches are overlapping, we use linear interpolation. Finally, we reconstruct the new tensor $\mathcal{F}'$ by concatenating all the slices $\mathcal{F}' = [\mathcal{F}'_1, \mathcal{F}'_2, \cdots, \mathcal{F}'_m]$ over filter channels. As a whole, we denote the patch replacement as $\mathcal{F}' = Q(\mathcal{F})$.

The reconstructed feature $\mathcal{F}'$ continues its way through the network, untouched from now on, and until it gets to the last layer. Then the network gives its prediction on the feature $\mathcal{F}'$ as the prediction towards the original input. Since the reconstructed feature $\mathcal{F}'$ consists of the nearest patches of the input's patches, $\mathcal{F}'$ perseveres the semantic information of original input while reducing the adversarial effect.

Discussion: Which layer do we apply the replacement patch on? Since patch replacement can be applied in any layer, then it is an important question. According to Figure 6.1, we observe that in the early layers, the distance between the adversarial feature and the original feature is much smaller than the later layers. We assume that the adversarial patch and original patch map

onto the same codeword. So it is more efficient to apply the patch replacement to earlier layers. We verify this hypothesis in our experiments.

6.3.5 Multi-layers

We introduced patch replacement in a single layer and discussed the choice of the layer. With a coarse codebook, the robustness of the network against adversarial attacks increases dramatically but the accuracy on legitimate images decreases. The quality of the codebook and the strategies determine the performance of the patch replacement. Turning these parameters is a way to achieve the trade-off between accuracy and robustness, and applying patch replacement on multi-layers is another way to achieve the trade-off.

By apply patch replacement with a fine codebook on different layers, it is possible to maintain the accuracy of legitimate images while improving the robustness against adversarial images little by little. When we apply patch replacement on more than one layer, we have more degree of freedom to find the trade-off between removing the adversarial perturbation and preserving the semantic information of inputs. For instance, we can optimize the codebook and replacement strategy on the images, and then we optimize the codebook and replacement strategy in the first layer.

Besides, by applying patch replacement on multi-layers, we increase the complexity of patch replacement, and as a result, the whole system is more difficult to attack.

6.4 Experiments

In this section, our experiments focus on studying the codebook, strategies, and which layers to apply patch replacement. Before investigating these performances, we first introduce the experimental settings on datasets, networks, and attacks.

6.4.1 Dataset, networks, and attacks

Dataset. When it comes to patch replacement, we need a training set to build a codebook as well as testing set to verify its performance. We randomly sample $50,000$ from the training dataset of ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2012) (50 images for each class) as training data while randomly sampling $1,000$ from the validation dataset (one image per class) as the test dataset. Instead of using the test dataset of adversarial attacks and defense competition [KGB+18a] as in chapter 4 and chapter 5, we construct our test dataset by

randomly sampling but follow its basic setting, namely selecting one image per class, because the test dataset of the competition consists of $1,000$ correctly classified biased images. It is not fair to measure the accuracy of a network with defenses, which is an important metric to evaluate the performance of defenses, based on this dataset.

Since this defense is built on PyTorch [PGC+17], where the pre-trained models are trained on images with size $224 \times 224 \times 3$, we sample the ImageNet images to $224 \times 224 \times 3$.

Networks. We use ResNet-50 [HZRS16b] with the pre-trained model from PyTorch-Torchvision models [1], whose accuracy is 0.76 on the testing dataset. We measured the performance of different attacks on the network defended by patch replacement.

To also test the performance of defenses (including patch replacement) against a robust network, we take the pre-trained adversarial training model [2] ResNet-50 with adversarial examples generated by PGD [MMS+17] attack, in which L_∞ is used to measure the upper bound of distortion and the parameter ϵ is 8.

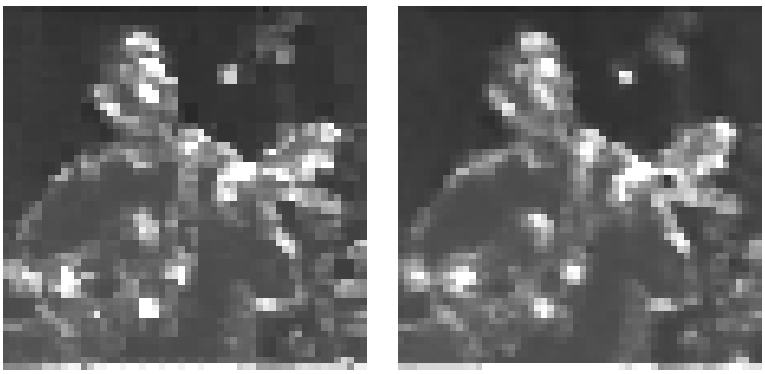
Attacks. We use the DDN [RHO+19] [3] as the main attack method with 20 iterations by default, which achieves 0.999 success rate on the test dataset against ResNet-50. Adversarial images are generated under the white-box setting. We also test with FGSM, PGD, BIM, with a set of ϵ [4], and PGD and BIM run 20 iterations. The implementation of FGSM, PGD, and BIM is from foolbox [5].

All the experiments in this work run on PyTorch1.4.0-py3.7 over CUDA 10.0.130; Foolbox produces most of the attacks, except DDN [RHO+19] [6]. The experiments are supported by Grand Equipment National de Calcul Intensif (GENCI) [7].

6.4.2 Optimization of the codebook for single layers

To obtain a decent codebook, we consider two kinds of models to train codebooks, *i.e.* product quantization, and $E8$ lattice. One major difference is about reconstruction. Since the product

1. `https://github.com/Cadene/pretrained-models.pytorch`
2. `https://github.com/MadryLab/robustness`
3. The implementation of DDN attack comes from `https://github.com/jeromerony/fast_adversarial`
4. $\epsilon \in [0.0005, 0.001, 0.003, 0.005, 0.01, 0.03, 0.05, 0.08, 0.1]$
5. `https://github.com/bethgelab/foolbox`
6. The implementation of DDN attack comes from `https://github.com/jeromerony/fast_adversarial`
7. We use the calculation resources from GENCI:`http://www.genci.fr/?lang=en`



(a) without overlap (b) with overlap

Figure 6.4 – A feature with overlap and without overlap.

quantization regards a patch as a basic unit, when we reconstruct features, the stride c determines to overlap between patches or not. The results in Figure 6.4 show that when $c < (2a + 1)$, the quantized features $\mathcal{F}'$ are smoother than the quantized features with $c = (2a + 1)$. We prefer the smoother feature since it provides slightly better performance. However, $E8$ lattice only applies to 8-dimension vectors inside patches so the quantized features are always without overlapping.

As we discussed previously, product quantization is more precise and data-based while $E8$ lattice without training for codebook is cheaper to apply. To verify their performance, we test on five different layers of ResNet-50, and for each layer, we produce a set of codebooks via changing the parameters.

Product Quantization (PQ). To train codebook for slices with K-means can be regarded as applying product quantization to the whole features/images. To optimize the performance of product quantization, there are two parameters to control the quality of the codebook, *i.e*. the number of clusters K for the K-means and the dimension of the patch d. A larger K or a smaller d results in richer codewords, and that allows restoring more information from inputs. We use Euclidean distance to calculate similarity in the K-means. All codebooks of each slice are trained independently. The different codewords we tried for product quantization in different layers are listed in section 6.4.2.

$E8$ **lattice.** $E8$ lattice does not depend on the data. The parameter controlling the scale, *i.e*. s, determines the magnitude of the lattice. Smaller scale provides denser codewords. We select $s \in [0.1, 0.2, \cdots, 0.9]$.

We measure the accuracy and robustness of different codebooks by comparing the accuracy

layer	K	d (D)
0^{th}-layer	392, 785, 3927, 6284, 7855, 11783, 15711, 19639, 26185, 31422, 39278, 58917, 78557, 117835, 157114, 235671	1 (3)
1^{st}-layer	39278, 78557, 117836, 157114, 235671, 314228, 785568	4 (64)
	39278, 78557, 117836, 157114, 261856, 392784, 785568	8 (64)
4^{th}-layer	78557	2 (256)
	78557, 785568	4 (256)
	78557	8 (256)
7^{th}-layer	78557	2 (256)
	78557	4 (256)
	78557, 785568	8 (256)
10^{th}-layer	78557, 785568	2 (256)
	78557, 785568	4 (256)
	78557, 785568	8 (256)

Table 6.1 – The number of clusters K and the dimension of the patch d for different layers.

between $1,000$ legitimate images and their adversarial version against the ResNet-50 network and under the protection of patch replacement in different layers. We study the performance of different codebooks with different value of parameters, *i.e.* K and d for product quantization (exact value see section 6.4.2) and $s \in [0.1, 0.2, \cdots, 0.9]$ for $E8$ lattice, are plotted in the same curve for a certain layer. There are 14 points in the curve of the first layer on Figure 6.5(a), but there are several points that are too close to tell the difference. Since there are two parameters K and d to control the performance of product quantization, it is hard to order the points according to the increasing of the parameter as $E8$. So we rank the points according to the accuracy of legitimate images for product quantization.

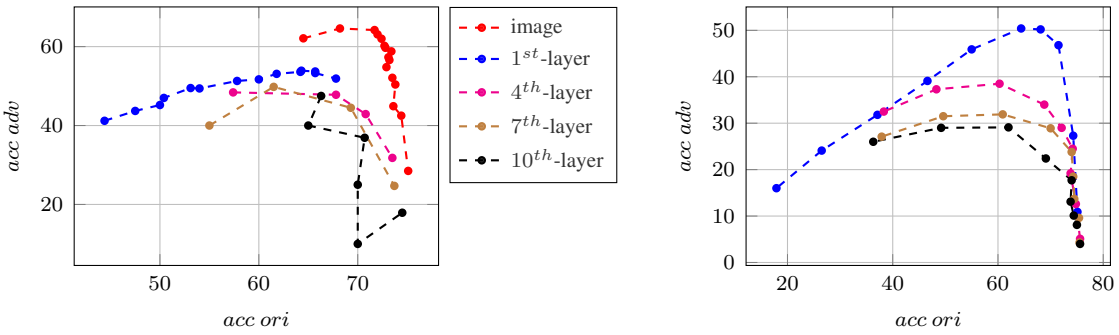
The results are depicted in Figure 6.5. It shows that the finer codewords maintain high accuracy on legitimate images because the finer codewords lose fewer details of inputs. When codewords are well-supplied so that all the information of inputs is represented, then the accuracy on legitimate images is exactly equal to the accuracy of the network without patch replacement as the right bottom point in Figure 6.5(b). We cannot find the same point in Figure 6.5(a) because it is impossible to train such a product quantization codebook. In this sense, patch replacement cannot improve the performance of legitimate images. However, when we focus on the accuracy of adversarial images, we observe the relationship between the richness of codewords and accuracy is not monotonous. When the codewords become coarser, the network gains more robustness against adversarial images which reduces slowly after reaching the peak.

Another tendency that happens in both plots in Figure 6.5 is that the earlier layer the patch

replacement is applied on, the better the performance of patch replacement on adversarial images is. Note, however, that the performance of patch replacement on legitimate images is very stable in different layers. It shows that the noise introduced by the patch replacement has similar effects to random noise on legitimate images. However, in different layers, that effect on adversarial images changes. It is obvious in Figure 6.5(a) that patch replacement can achieve better performance in the former layer than the later layer. As shown in Figure 6.1, adversarial perturbations are rather small in the early stages, it is easy to remove. When adversarial perturbations are amplified in the later layers, it is harder to eliminate adversarial perturbations by replacing them with the most similar feature patch.

In Figure 6.5(a), we also observe that patch replacement on images performs extremely well. Only considering W and H, input images have a larger size than features, for example, four times larger than the features of the first layer. With the same size of the patches concerning height and width, patch replacement preserves much more information on images than on features.

Figure 6.5 shows that $E8$ lattice improves the robustness less than product quantization when tracking a similar amount of accuracy on legitimate images. However, both dictionaries share the similar property of patch replacement in different layers. Since the $E8$ lattice does not need training, it is used to explore the properties of the patch replacement quickly.



(a) PQ (b) $E8$

Figure 6.5 – We apply patch replacement on different layers independently with two different kinds of codebooks. We replace all the patches with their most similar patches in the codebooks. The accuracy of $1,000$ legitimate images, noted as '$acc\ ori$', and their adversarial images, noted as '$acc\ adv$', with different value of parameters, *i.e.* K and d for product quantization (exact value see section 6.4.2) and $s \in [0.1, 0.2, \cdots, 0.9]$ for $E8$ lattice, are plotted in the same curve for a certain layer. Actually, there are 14 points in the curve of the first layer on (a), but there are several points that are too close to tell the difference. Since there are two parameters K and d to control the performance of product quantization, it is hard to order the points according to the increasing of the parameter as $E8$. So we rank the points according to the accuracy of legitimate images for product quantization.

According to Figure 6.5, the best codebook setting for a single layer is product quantization with $K = 3927$ on images (so patch replacement operates at layer 0). It gives 71.7% accuracy on legitimate images while 64.2% on their adversarial images. We lose 4% accuracy on legitimate images and gain 64.1% accuracy on adversarial images.

Before improving further the performance of patch replacement, we study the effect on the level of noise through the network of patch replacement with the best setting for a single layer, *i.e.* using product quantization codebook inside layer 0 with $K = 3927, d = 1$. The effect is depicted in Figure 6.6. It shows that our patch replacement changes the behavior of the adversarial perturbation through the neural network. We observed that the green line (noise introduced by patch replacement) has a similar pattern as the blue line (random noise). We lose 4% accuracy in the presence of these noises. The difference between quantized legitimate inputs $Q(x)$ and their quantized adversarial versions $Q(x + r)$ depicted as the brown line is small compared to others. This perturbation leads to a 7.3% drop in accuracy. The black line, which shows the difference between legitimate images x and their quantized adversarial versions $Q(x + r)$, is almost overlapping the green line. This implies that quantized legitimate inputs $Q(x)$ and their quantized adversarial versions $Q(x + r)$ introduce similar amounts of distortion but in different directions.

6.4.3 Strategies

We chose the best setting for the codebook and we lose accuracy on the legitimate images because patch replacement loses part of the information of inputs and introduces noise to gain robustness towards adversarial images. If we can reduce the amount of noise and preserve more information, the accuracy of the legitimate images improves. This target can be achieved not only by improving the codebook but also by applying the strategies introduced in subsection 6.3.3

We choose two settings of product quantization, *i.e.* $K = 3927$ in the image layer and $K = 235671, d = 4$ in the first layer to investigate the performance of the strategies. These codebooks are chosen by finding the right top corner points in Figure 6.5(a). The codebook, with $K = 235671, d = 4$ in the first layer, achieves the accuracy of 65.7% on legitimate images and 53.7% on their adversarial images. Then, we change the parameter ϵ to control the strength of patch replacement.

In details, on the images, $\epsilon \in [0.0, 0.1, 0.2, \cdots, 0.9, 1.0]$ for L_2 and Linear strategies, and $\epsilon \in [0.0, 0.01, 0.02, \cdots, 0.09, 0.1]$ for L_∞ strategy. For the first layer, $\epsilon \in [0.0, 0.1, 0.2, \cdots, 0.9, 1.0]$ for L_2 and Linear strategies, and $\epsilon \in [0.0, 0.01, 0.02, \cdots, 0.19, 0.2]$ for L_∞ strategy.

As we discussed in subsection 6.3.3, when $\epsilon = 0$, patch replacement with any replacement

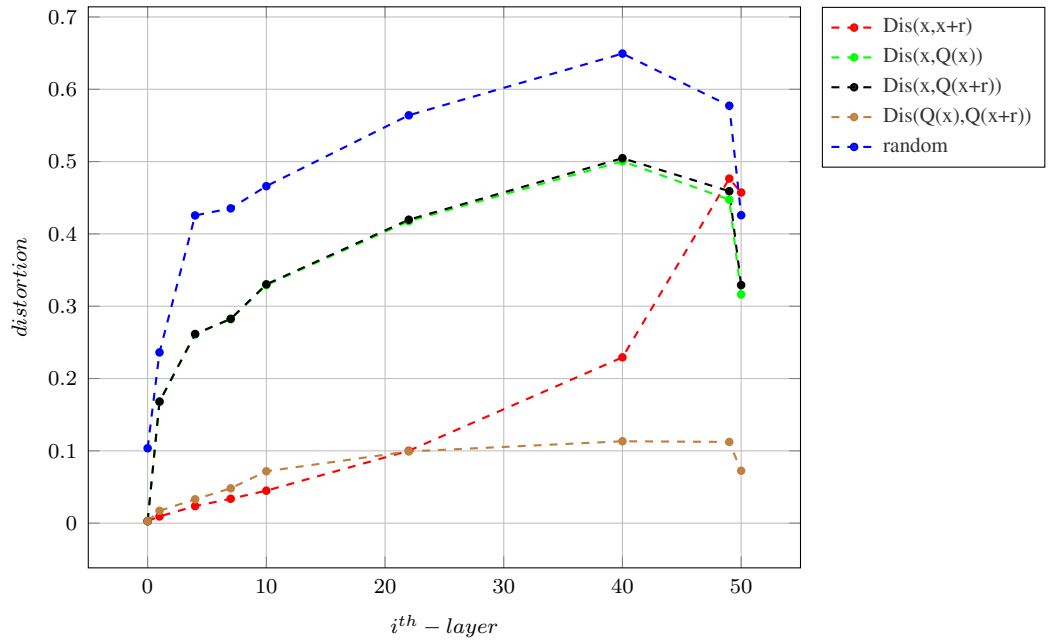
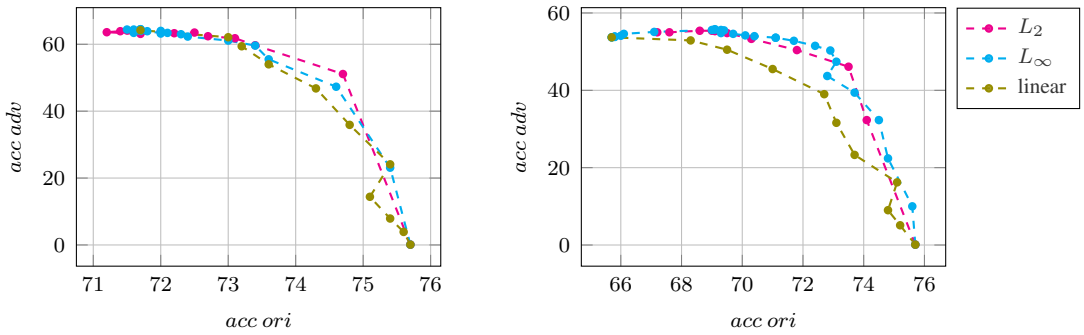


Figure 6.6 – The blue line corresponds to random noise following a normal distribution with mean 0 and variance 0.05. The red line corresponds to the adversarial perturbation generated by DDN [RHO+19] attack. The green line shows the distortion introduced by our patch replacement on the legitimate images. The black line shows the difference between adversarial images modified by patch replacement and their original version. And the brown line describes the difference between legitimate images modified by patch replacement and their adversarial images also modified by patch replacement.

strategies (other than plain strategy) is equivalent to the original network. When ϵ is extremely large, L_2 strategy and L_∞ strategy are equal to the plain strategy. For $Linear$ strategy, when $\epsilon = 1$, linear strategy is equal to plain strategy. Figure 6.7 is consistent with that. The points in the bottom right corner in both Figure 6.7(a) and Figure 6.7(b) correspond to $\epsilon = 0$, while the points in the top left corner correspond to cases equivalent to plain strategy. We notice that, on the image layer, when ϵ of L_2 strategy is large but not large enough to be equivalent to plain strategy, *i.e.* the flat part of the magenta line in the Figure 6.7(a), the accuracy on legitimate images decreases and then increases back to the point of plain strategy. That is the reason why the magenta line has more points on the left. These phenomena might be a result of the non-linearity of the network.

Comparing Figure 6.7(a) and Figure 6.7(b), no strategy dominates the others. That implies for different codebooks different strategies provide the best performance. It is better to optimize the replacement strategy when we train a new codebook.



(a) image-layer (b) 1-layer

Figure 6.7 – We choose the two best codewords of product quantization, *i.e.* $K = 3927$ in the image layer and $K = 235671, d = 4$ in the first layer. Then apply three strategies with different parameters ϵ to control the force of patch replacement. The magenta lines correspond to the L_2 strategy, the cyan lines correspond to the L_∞ strategy, and olive lines correspond to the $Linear$ strategy. We plot the accuracy of legitimate images and their adversarial images.

We are interested in the points in the top right corner of Figure 6.7(a) and Figure 6.7(b). That indicates the strategies improving the accuracy of legitimate images (comparing to the plain strategy) and keeping the robustness on adversarial images the same or improving it slightly. With replacement strategies, the performance of patch replacement improves more obviously in the first layer, not on images. When we use the L_∞ strategy with $\epsilon = 0.1$, we have 72.0% as the accuracy on legitimate images and 64.0% for adversarial images, *i.e.* we gain 0.3% accuracy on legitimate images and lose 0.2% accuracy on adversarial images. If we select the L_∞ strategy with $\epsilon = 0.17$ in the image layer, it gives us 71.7% on legitimate images and 64.5% on adversarial images. We lose nothing on legitimate images but gain 0.3% on adversarial images. Nevertheless, when we use the L_∞ strategy with $\epsilon = 0.17$, we have 69.3% as the accuracy on legitimate images and 55.6% for adversarial images, *i.e.* we gain 3.6% accuracy on legitimate images and gain 1.9% accuracy on adversarial images.

The reason why replacement strategies improve the performance in the first layer and almost do not improve it on images is that the codebook on images is rich enough to recover the main information, while the codebook in the first layer is rather coarse. Since replacement strategies are designed to preserve more information of input, it helps the case in the first layer.

6.4.4 Multi-layer patch replacement

From the previous experiments on a single layer, we know that: 1) applying patch replacement on images gives us the best performance; 2) strategies help patch replacement in the first layer.

To benefit from both lessons, we carry out experiments to apply patch replacement in two layers. We take the best version of patch replacement on images. With it, we lose 4% accuracy on legitimate images but gain 64.4% accuracy on adversarial images. On top of it, we apply patch replacement in the first layer. Then we find the best codebook by changing parameter K in the first layer [8]. The results are shown in Figure 6.8. When increasing K, the accuracy does not change monotonously, this might also be a result of the non-linearity.

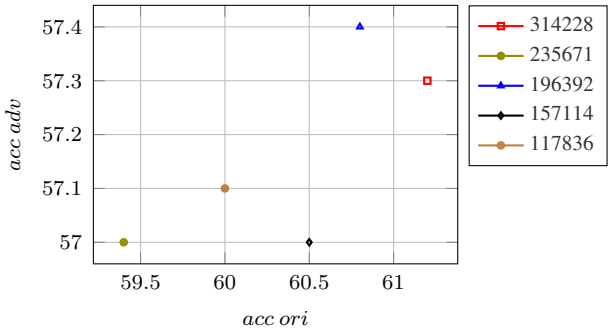


Figure 6.8 – Based on the best codebook on image, *i.e.* $K = 3927$, we apply patch replacement in the first layer and find the best codebook for this combination. We test different K and plot the accuracy on legitimate images and their adversarial images.

When we combine the image layer and the first convolutional layer, the codebooks are trained independently as obtained in the previous experiments. We did not train the codebook of the first convolutional layer based on the neural network with the patch replacement in the image layer since it is costly to try all the combinations. On another hand, training the codebook independently for patch replacement in different layers captures the features of the training data.

Fixing the patch replacement on images as well as the codebook, *i.e.* $K = 314228$, of the first layer, we optimize the replace strategies in the first layer. The result is shown in Figure 6.9. We select the best strategy, *i.e.* L_∞ strategy with parameter 0.01. It gives 71.8% accuracy on legitimate images, 66.0% on adversarial images. We improve the performance greatly comparing to the plain strategy shown in Figure 6.8. By applying patch replacement on both the image layer and the first layer, we improve the performance of patch replacement.

It is possible to include more layers and continue such kind of optimization. However, it is also expensive, while a performance improvement is small concerning the cost. Besides, combining more layers inside the defense increases the complexity of the forward. Considering all these situations, we stop with two layers, and take them as our final results, and test their performance on different attacks.

8. $K \in [117836, 157114, 196392, 235671, 314228]$

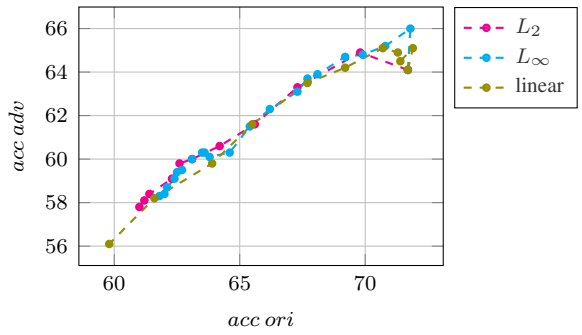


Figure 6.9 – Based on the best codebook and strategies on the layer 0, *i.e.* image layer, we add the layer with the best codebook and find the best strategy for this combination. The best strategy is L_∞ with parameter 0.01.

	Without Defense		With Defense	
	P_{suc}	$\overline{D}$	P_{suc}	$\overline{D}$
DDN	1.00	0.53	0.08	0.57
FGSM	0.95	5.13	0.92	7.98
BIM	1.00	2.50	0.99	6.10
PGD	1.00	3.12	0.83	7.21

Table 6.2 – Test the performance of our patch replacement by apply different attacks, *i.e.* DDN, FGSM, BIM, PGD_2, towards the network without knowing there is a defense. The success rate P_{suc} and average distortion $\overline{D}$, which only accounts success adversarial images, are calculated according our evaluation metrics in section 3.3.

Table 6.2 shows that our defense performs better on the adversarial image with small distortion. The distortion for eventually forging a successful adversarial image increases a lot due to this defense because those adversarial examples with smaller distortion are quantized back to their original images.

Figure 6.10 provides an overview of the performance of patch replacement against different attacks. The dotted lines of the original network are above the solid lines of the network with patch replacement. When fixing a distortion level, *e.g.* $D = 6$, BIM achieves 0.8 success on the original network while the success rate goes down to around 0.4 on the network with patch replacement. Conversely, if we fix the success rate, *e.g.* $P_{\text{suc}} = 0.5$, all attacks only need a distortion around 2 on the original network, while on the network with patch replacement, attacks need more than 6 as distortion.

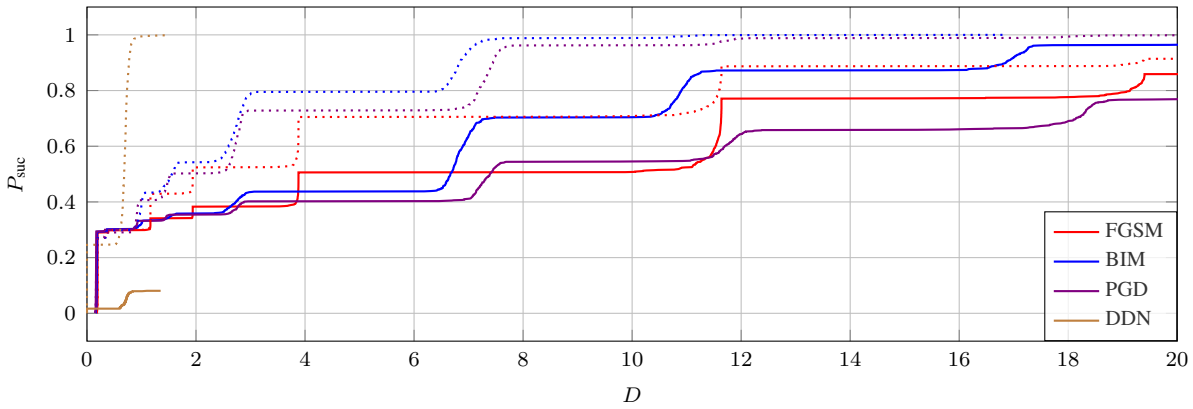


Figure 6.10 – Operating characteristics of the attacks ResNet-50 over the test dataset. The dotted lines represent the results on the original network while the solid lines represent the results on the network with patch replacement. Here P_{suc} is success rate and D is respective L_2 norm distortion.

Cost of patch replacement. The main computation cost of patch replacement is to train the product quantization codebook. We train the codebook with faiss [9]. The actual training time depends on the parameters, *i.e.* K and d, and the size of features. The codebook also takes GPU memories. However, comparing to retrain the neural network, the cost is of patch replacement is cheap.

6.5 Comparison with other defense methods

To investigate further the performance of patch replacement, we compare patch replacement to other defense methods.

Adversarial Training. We test on ResNet-50 of adversarial training with PGD [MMS+17] [10].

Transformation methods. We test bit-depth reduction of reducing images to 3 bits and 5 bits, denoted as bit3 and bit5 [GRCvdM17]; median smoothing filter with a kernel size of two and three, denoted as ms2 and ms3 [XEQ17]; pixel deflection [11] [PMG+18] with CAM as a robust activation map.

Table 6.3 shows that adversarial training with PGD has very bad performance on legitimate images (45.9%), it defends badly against DDN but pretty well against PGD. It is reasonable

9. `https://ai.facebook.com/tools/faiss/`
10. We use L_∞ version and the parameter ϵ is 8. Download pre-trained model from `https://github.com/MadryLab/robustness`.
11. `https://github.com/iamaaditya/pixel-deflection`

	Accuracy (%)		
	clean	DDN	PGD
patch replacement	$71.8_{[-3.1]}$	$66.0_{[+7.4]}$	$46.4_{[+2.1]}$
adversarial training [MMS+17] (PGD)	45.9	19.0	**44.3**
bit3 [GRCvdM17]	64.7	55.1	32.9
bit5 [GRCvdM17]	**74.9**	18.9	6.5
ms2 [XEQ17]	74.2	47.9	26.5
ms3 [XEQ17]	71.8	55.6	34.2
pixel deflection [PMG+18]	73.2	**58.6**	31.0

Table 6.3 – We compare patch replacement to other defenses. We measure the accuracy on legitimate images with the defense method, and on adversarial images generated by DDN and PGD attacks under the white box setting. DDN runs for 20 iterations, while the parameter ϵ for PGD is 0.03. We bold the best accuracy achieved by defenses except patch replacement, and put the difference between the accuracy of patch replacement to the best accuracy. For instance, patch replacement gains 7.4% accuracy with DDN than the best competitor pixel deflection.

since the network is trained with PGD. Simple attacks, like bit5, ms2, achieve better accuracy on legitimate images than patch replacement but less accuracy on adversarial images. Comparing to bit5 and ms2, patch replacement gains more than 20% on adversarial images while loses around 3% on legitimate images. Patch replacement is better than bit3 and ms3. When it comes to pixel deflection, patch replacement gains around 10% on adversarial images and loses around 1.5% on legitimate images.

6.6 Defense against smart attack

Previously, we test the performance of the defenses under the black-box setting, *i.e*. attackers do not know there is a defense. To estimate the performance under the white-box setting, *i.e*. attackers are aware of the defenses, we propose to use a kind of smart attack, called *Backward Pass Differentiable Approximation (BPDA)* [ACW18] [12]. This smart attack generates adversarial examples by including the defense during the forward pass and ignores the defense module during the backward pass if it is impossible to calculate gradients through it. In detail, the defense module is replaced by an identity function for the backward.

We apply BPDA on patch replacement, as well as the other defense methods, and we compare the statistics P_{suc} and $\overline{D}$ according to our evaluation metrics.

12. `https://github.com/Annonymous-repos/attacks-in-pytorch`

	BPDA	
	P_{suc}	$\overline{D}$
patch replacement	0.89	12.89
bit3 [GRCvdM17]	1.00	1.00
bit5 [GRCvdM17]	1.00	1.00
ms2 [XEQ17]	1.00	1.76
ms3 [XEQ17]	1.00	1.25
pixel deflection [PMG+18]	1.00	0.89

Table 6.4 – Compare the performance of our patch replacement to other defenses by applying smart attack BPDA. We measure the success rate P_{suc} and the average distortion $\overline{D}$ according to our evaluation metrics in section 3.3. BPDA runs for 20 iterations.

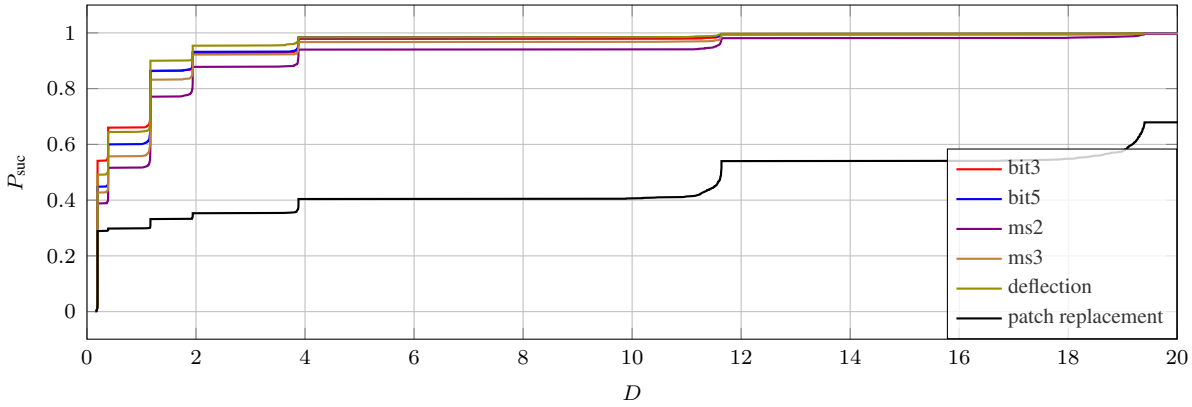


Figure 6.11 – Operating smart attack BPDA to ResNet-50 with different defenses over ImageNet.

Table 6.4 shows that BPDA performs worst in the network with patch replacement. BPDA success attacks the network with other defense with average distortion less than 2 but BPDA only achieves 0.89 success rate with average distortion more than 12. BPDA have to increase the distortion six times larger to fool the network with patch replacement defense. This conclusion is more obvious in Figure 6.11. It indicates that in the white-box setting, the network defended with patch replacement is harder to attack. Even when the distortion is very large and BPDA succeeds to attack networks defended with other methods, BPDA only achieve around 70% success rate.

In Figure 6.11, some examples are changed into adversarial examples with extremely small distortion, even though the network is defended. We check some of these examples and find them easier to be attacked. The predicted score of the ground truth is very close to the second one. For instance, the difference for these images is around 0.05, while for others the difference is around 0.75.

6.7 Conclusion

In this work, we propose a transformation method that succeeds to defend against adversarial attacks. We succeed to include the training data as a part of the defense so that it is more difficult for attackers even in gray-box settings or white-box settings.

Also, we investigate the impact of noise in different layers. There is no surprise that adversarial noises are very small in the image space also in the early layers, however, their values of L_2 norm become great when coming to the end of the neural network. Random noises behave almost in an opposite way, they increase greatly at the beginning and become stable at the later stage.

According to this observation, we apply patch transformation at the beginning of the neural network, combining two layers to improve the trade-off between accuracy and robustness. Our experiments show that it works well, especially for the attack methods like DDN.

CHAPTER 7

CONCLUSION AND PERSPECTIVES

In this last chapter, we review our contributions made within this manuscript and describe possible research questions that remain unanswered but interesting to explore in future works.

Conclusion

After investigating the field of adversarial attacks and defenses, we find there are four important concepts: *speed*, *distortion*, *invisibility*, and *transferability*.

We attach a lot of importance to *speed*, whether it be at attack or defense time. In the manuscript, we propose BP attack which is an efficient algorithm to search for the adversarial examples. On the other hand, we propose the patch replacement defense method which is a relatively cheap –so a rather fast– defense mechanism. It does not need to train the network from scratch and is robust in a white-box setting.

We also improve the *invisibility* (refer to chapter 4) and the magnitude of *distortion* (refer to chapter 5) of adversarial perturbations. We investigate the definition of the adversarial problem and propose our definition of *invisibility* for adversarial perturbation. Moreover, we propose a magnification approach to amplify the adversarial perturbation to make it easier to evaluate *invisibility* with the naked eyes. We propose an evaluation protocol to compare the success rate and magnitude of the *distortion* of adversarial examples generated by attacks from a different families target success or distortion. Although we are not targeting the improvement of *transferability* of adversarial examples, we measure it when we evaluate the quality of adversarial images.

In detail, we succeed to generate smooth adversarial perturbation where *invisibility* is defined as the smoothness according to the similarity graph of input images. The function to acquire this *invisibility* can be integrated into existing adversarial attacks. It provides smooth adversarial perturbation when it is properly integrated into the attack, namely injecting smoothness as a constraint inside the loss function. In our experiments, we find that simply integrating that smoothness function into attacks such as PGD does not perform well even if it is done at each

attack iteration. However, when it is integrated into C&W, sC&W not only produces adversarial perturbation with *invisibility* but also improves the magnitude of *distortion* when the success rate is the same.

Our definition of *invisibility* allows the attacker to produce adversarial perturbations with larger *distortion* but that remain invisible. However, it mitigates the *transferability*. The question of how to define *invisibility* is still open. Although a human does not make the difference between the original image and its smooth adversarial version even with magnification, it does not prove that an algorithm will not detect some statistical evidence.

We find that using the existing optimization algorithm to solve the adversarial problem sometimes leads to oscillation on the boundaries of classes and results in the wasting of computations. To improve the efficiency of the algorithm, we propose a boundary projection (BP) attack that separates the search process into two cases, *i.e*. the IN case and the OUT case. In the IN case, the current solution is not yet adversarial and the main target is misclassifications. So BP searches along the gradient direction. While in the OUT case, the current solution is adversarial and the main target is minimizing distortion. So BP searches on the manifold defined by a class boundary. As a result, BP attempts to stay mostly in the adversarial region when it searches around the boundary. BP also considers the *quantization* problem of adversarial example. The amplitude of the perturbation is controlled at each iteration.

The experiments show that BP succeeds in minimizing distortion on the class boundary manifold. BP attack generates adversarial examples with a high success rate (close to one) and low magnitude of *distortion* with a small number of iterations. This work [ZAFA20b] is published on *IEEE Transactions on Information Forensics and Security*.

We propose a relatively cheap defense named patch replacement. It decreases the effects of adversarial perturbation by decomposing input into patches and replacing them with the most similar ones from training data. Moreover, we not only consider applying patch replacement on images but also in the intermediate features.

We investigate how adversarial perturbations and random noises affect the network through layers. We observe that the magnitude of adversarial perturbations is amplified dramatically in the later layers, while the magnitude of random noise is in contrast much amplified in the early layers and stays flat in the later layers. So we apply patch replacement on images and the features of the first layer. It improves both accuracy of classification and robustness against adversarial attack comparing to patch replacement only on images or in a single layer. The experiments show that patch replacement improves the robustness of the network on both black-box settings and white-box settings.

In summary, our work succeeds to make the research of adversarial attacks and defense more complete. We find that it is easier to build an attack than a defense. Once we know the architecture and the parameters of the target neural networks, it is always possible to generate adversarial examples with gradient descent. As long as we benefit from the end-to-end magic of neural networks, we suffer from adversarial phenomena because we use the same logic to train a neural network and generate adversarial examples. We minimize the loss function of neural networks and use the backpropagation to update parameters of neural networks, while we minimize the optimization function of producing adversarial examples and use the backpropagation to generate the adversarial perturbation. To our point of view, it is due to the fact that the neural network does not recognize objects as humans do. By minimizing the difference between the prediction of neural networks and the ground truth, neural networks learn features themselves. However, these features include some meaningless and specific information of the dataset, which are helpful in the particular task but not general to objects.

When we make a defense, we always suffer from a trade-off between the quality and the robustness of the neural network. If we obfuscate the gradient for the attacker, a part of the information is lost during training and, consequently, the accuracy of the classification process drops. Reactive defenses and proactive defenses trade accuracy for robustness. However, works like AdvProp [XTG+20] succeed in augmenting the quality and robustness of the network at the same time. Then the question is how much free lunch is there.

Perspectives

We propose three works in this field, they can be improved by some little extensions. For instance, it is possible to integrate smooth constraint into BP attack. When we simply integrate smooth constraint with PGD, it does not works well. It indicates that the algorithm should be modified so that BP attack optimizes the adversarial optimization integrating with smooth constraint.

It is also interesting to defend with patch replacement against smooth adversarial perturbation. We conjecture the patch replacement works well on smooth adversarial perturbation because the perturbation is so smooth that it preserves the semantic information of the images. Patch replacement replaces the patches of input images by the most similar legitimate patches and eliminate adversarial effects.

Besides, *transferability* as an important concept in this field, we did not explore it further. We measure the *transferability* for smooth adversarial perturbation and find that when the magnitude

of distortion is small, smooth adversarial perturbations sC&W perform better under bilateral filter transferability than C&W. It is interesting to study how to increase the transferability of adversarial examples and the relationship between their transferability of adversarial examples and the robustness of networks.

It is crucial to explore how much we can augment the robustness against adversarial attacks without degrading the performance on legitimate images. Interpretable neural network could improve the ability to learn the intrinsic knowledge from data, and the development of adversarial attacks and defenses will also promote the researches of interpretation on neural networks.

The game between attacks and defenses will never end. Currently, it seems that attackers dominate the defenders since once there is a defense proposed, the attackers could attack it days later. However, the distortion needed to be successful increases. Leaving the game between attackers and defenders continuing, the distortion needed for attackers will be extremely large, then the defender can claim the classifier is robust.

BIBLIOGRAPHY

[ABB+17] Laurent Amsaleg, James E. Bailey, Dominique Barbe, Sarah Erfani, Michael E Houle, Vinh Nguyen, and Miloš Radovanovic. The Vulnerability of Learning to Adversarial Perturbation Increases with Intrinsic Dimensionality. In *Proceedings of IEEE International Workshop on Information Forensics and Security (WIFS)*, Rennes, France, December 2017.

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th $USENIX$ symposium on operating systems design and implementation ($OSDI$ 16)*, pages 265–283, 2016.

[ACW18] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[AMS09] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[ASE+18] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

[AW18] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *arXiv preprint arXiv:1805.12177*, 2018.

[BF17] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[BMR+17] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[BNS+06] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the ACM ASIA Conference on Computer and Communications Security (AsiaCCS)*, pages 16–25. ACM, 2006.

[Bon13] Silvere Bonnabel. Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, 2013.

[BR18] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[BRB18] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[BRRG18] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[BU05] Evgeniy Bart and Shimon Ullman. Cross-generalization: Learning novel classes from a single example by feature replacement. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, volume 1, pages 672–679. IEEE, 2005.

[CK16] Siddhartha Chandra and Iasonas Kokkinos. Fast, exact and multi-scale inference for semantic image segmentation with deep Gaussian CRFs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[CLC+19] Hao-Yun Chen, Jhao-Hong Liang, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. Improving adversarial robustness via guided complement entropy. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4881–4889, 2019.

[CMB+08] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital Watermarking*. Morgan Kaufmann Publisher, second edition, 2008.

[CS01] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec):265–292, 2001.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, 2017.

[CW18] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *Proceedings of the IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018.

[CZS+17] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, page 15–26, New York, NY, USA, 2017. ACM.

[DAL+18] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

[DDS+04] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the international conference on knowledge discovery and data mining (ACM SIGKDD)*. ACM, 2004.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 248–255. Ieee, 2009.

[DLP+18] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 9185–9193, 2018.

[DMY+19] Abhimanyu Dubey, Laurens van der Maaten, Zeki Yalniz, Yixuan Li, and Dhruv Mahajan. Defense against adversarial images using web-scale nearest-neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 8767–8776, 2019.

[EEF+18] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1625–1634, 2018.

[ES02] Brian Everitt and Anders Skrondal. *The Cambridge dictionary of statistics*, volume 106. Cambridge University Press Cambridge, 2002.

[ETT+17] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 2017.

[FBHD19] S. A. Fezza, Y. Bakhti, W. Hamidouche, and O. Déforges. Perceptual evaluation of adversarial attacks for cnn-based image classification. In *Proceedings of the International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, June 2019.

[FMDF16] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Robustness of classifiers: from adversarial to random noise. *arXiv preprint arXiv:1608.08967*, 2016.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GFW18] Chuan Guo, Jared S. Frank, and Kilian Q. Weinberger. Low frequency adversarial perturbation. *arXiv preprint arXiv:1809.08758*, 2018.

[GLB19] Partha Ghosh, Arpan Losalka, and Michael J Black. Resisting adversarial attacks using gaussian mixture variational autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 541–548, 2019.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, 2014.

[GR14] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

[Gra06] Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE Trans. PAMI)*, 28(11), 2006.

[GRCvdM17] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[HD18] Jamie Hayes and George Danezis. Learning universal adversarial perturbations with generative models. In *Proceedings of the IEEE Security and Privacy Workshops (SPW)*, pages 43–49. IEEE, 2018.

[HF16] Mehrtash Harandi and Basura Fernando. Generalized backpropagation,\'{E} tude de cas: Orthogonality. *arXiv preprint arXiv:1611.05927*, 2016.

[HKWW17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, pages 3–29. Springer, 2017.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 4700–4708, 2017.

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. volume 79, pages 2554–2558. National Acad Sciences, 1982.

[HRF19] Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 588–597, 2019.

[HS+99] Geoffrey E Hinton, Terrence Joseph Sejnowski, et al. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[HXSS15] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*, 2015.

[HZJ18] Wen Heng, Shuchang Zhou, and Tingting Jiang. Harmonic adversarial attack method. *arXiv preprint arXiv:1807.10590*, 2018.

[HZRS16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[HZRS16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 630–645. Springer, 2016.

[IAT+18] Ahmet Iscen, Yannis Avrithis, Giorgos Tolias, Teddy Furon, and Ondrej Chum. Fast spectral ranking for similarity search. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2018.

[IEAL18] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2137–2146. PMLR, 2018.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[ITA+17] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, Teddy Furon, and Ondrej Chum. Efficient diffusion on region manifolds: Recovering small objects with compact cnn representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[JDS10] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE Trans. PAMI)*, 33(1):117–128, 2010.

[KB15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.

[KBD+17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, pages 97–117. Springer, 2017.

[KGB16] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[KGB+18a] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, et al. Adversarial attacks and defences competition. *arXiv preprint arXiv:1804.00097*, 2018.

[KGB+18b] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, Alan Yuille, Sangxia Huang, Yao Zhao, Yuzhe Zhao, Zhonglin Han, Junjiajia Long, Yerkebulan Berdibekov, Takuya Akiba, Seiya Tokui, and Motoki Abe. Adversarial attacks and defences competition. *arXiv preprint arXiv:1804.00097*, 2018.

[KLL08] Tae Hoon Kim, Kyoung Mu Lee, and Sang Uk Lee. Generative image segmentation using random walks with restart. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2008.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LC10] Eric Cooper Larson and Damon Michael Chandler. Most apparent distortion: full-reference image quality assessment and the role of strategy. *Journal of electronic imaging*, 19(1):011006, 2010.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18, 2010.

[LCLS16] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[LCW+20] Canyu Le, Zhonggui Chen, Xihan Wei, Biao Wang, and Lei Zhang. Continual local replacement for few-shot learning. *arXiv e-prints*, pages arXiv–2001, 2020.

[LH17] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[LHL15] Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang. A unified gradient regularization family for adversarial examples. In *Proceedings of the 2015 IEEE International Conference on Data Mining*, pages 301–309. IEEE, 2015.

[LHL17] Hyeungill Lee, Sungyeob Han, and Jungwoo Lee. Generative adversarial trainer: Defense to adversarial perturbations with gan. *arXiv preprint arXiv:1705.03387*, 2017.

[LIF17] Jiajun Lu, Theerasit Issaranon, and David Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 446–454, 2017.

[LJL+19] Jie Li, Rongrong Ji, Hong Liu, Xiaopeng Hong, Yue Gao, and Qi Tian. Universal perturbation attack against image retrieval. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 4899–4908, 2019.

[LLL+18] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. *arXiv preprint arXiv:1802.06474*, 2018.

[LLS+18] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[LM05] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the international conference on knowledge discovery and data mining (ACM SIGKDD)*, pages 641–647. ACM, 2005.

[LPSB17] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[MC17] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the Conference on Computer and Communications Security (SIGSAC)*, pages 135–147. ACM, 2017.

[MDFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[MDST18] Seyed-Mohsen Moosavi-Dezfooli, Ashish Shrivastava, and Oncel Tuzel. Divide, denoise, and defend against adversarial attacks. *arXiv preprint arXiv:1802.06806*, 2018.

[MFFF17] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 86–94, 2017.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 30, 2013.

[MLT+19] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. NIC: detecting adversarial samples with neural network invariant checking. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, 2019.

[MMS+17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[NK17] Nina Narodytska and Shiva Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1310–1318. IEEE, 2017.

[NW06] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer, 2006.

[PFC+18] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer.

Automatic differentiation in pytorch. In *Proceedings of the International conference on Neural Information Processing Systems Workshop Autodiff (NeurIPSW) 2017*, 2017.

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[PMG+18] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. Deflecting adversarial attacks with pixel deflection. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 8571–8580, 2018.

[PMJ+16] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016.

[PMW+16] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016.

[PP18] Gilles Puy and Patrick Pérez. A flexible convolutional solver with application to photorealistic style transfer. *arXiv preprint arXiv:1806.05285*, 2018.

[QAR18] Erwin Quiring, Daniel Arp, and Konrad Rieck. Forgotten siblings: Unifying attacks on machine learning and digital watermarking. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 488–502, April 2018.

[RDHC19] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1085–1097, 2019.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al.

Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RHK18] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*, 2018.

[RHO+19] Jérôme Rony, Luiz G Hafemann, Luiz S Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 4322–4330, 2019.

[RN02] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2002.

[RSL18] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[SBBR16] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the Conference on Computer and Communications Security (SIGSAC)*, pages 1528–1540. ACM, 2016.

[SBR18] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. On the suitability of l_p-norms for creating and preventing adversarial examples. *arXiv preprint arXiv:1802.09653*, 2018.

[SGOS+18] Carl-Johann Simon-Gabriel, Yann Ollivier, Bernhard Schölkopf, Léon Bottou, and David Lopez-Paz. Adversarial vulnerability of neural networks increases with input dimension. arXiv preprint arXiv:1802.01421, 2018.

[SKC18] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

[SKN+17] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

[SM13] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656, 2013.

[SND17] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. *arXiv preprint arXiv:1710.10571*, 2017.

[SNF+13] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3), 2013.

[SNG+19] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.

[SP97] Rainer Storn and Kenneth V. Price. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4), 1997.

[SP18] Alexandru Constantin Serban and Erik Poll. Adversarial examples - A complete characterisation of the phenomenon. *CoRR*, abs/1810.01185, 2018.

[STL+19] Bo Sun, Nian-Hsuan Tsai, Fangchen Liu, Ronald Yu, and Hao Su. Adversarial defense by stratified convolutional sparse coding. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, June 2019.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2818–2826, 2016.

[SVS19] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[SZMA18] Ilia Shumailov, Yiren Zhao, Robert D. Mullins, and Ross Anderson. The taboo trap: Behavioural detection of adversarial samples. *CoRR*, abs/1811.07375, 2018.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[TKP+17] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[TM98] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 1998.

[TPG+17] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[TRC19a] Giorgos Tolias, Filip Radenovic, and Ondrej Chum. Targeted mismatch adversarial attack: Query with a flower to retrieve the tower. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 5037–5046, 2019.

[TRC19b] Giorgos Tolias, Filip Radenovic, and Ondrej Chum. Targeted mismatch adversarial attack: Query with a flower to retrieve the tower. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[TRHV20] Olga Taran, Shideh Rezaeifar, Taras Holotyak, and Slava Voloshynovskiy. Machine learning through cryptographic glasses: combating adversarial attacks by key based diversified aggregation. *EURASIP Journal on Information Security*, January 2020.

[TSS18] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 6541–6550, 2018.

[TVRG19] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection. In *Proceedings*

of the IEEE conference on computer vision and pattern recognition worshops (CVPRW), pages 49–55, 2019.

[Ume12] P Umesh. Image processing in python. *CSI Communications*, 23, 2012.

[VC17] Paul Vernaza and Manmohan Chandraker. Learning random-walk label propagation for weakly-supervised semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Advances in neural information processing systems*, pages 5998–6008, 2017.

[WBSS04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[WK17] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[XEQ17] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[XTG+20] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 819–828, 2020.

[XWM+19] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 501–509, 2019.

[XWZ+17a] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[XWZ+17b] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, pages 1369–1378, 2017.

[XZL+18] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

[XZL+20] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 665–681. Springer, 2020.

[YLCS18] Zhuolin Yang, Bo Li, Pin-Yu Chen, and Dawn Song. Characterizing audio adversarial examples using temporal dependency. *arXiv preprint arXiv:1809.10875*, 2018.

[YLDT18] Erkun Yang, Tongliang Liu, Cheng Deng, and Dacheng Tao. Adversarial examples for hamming space search. *IEEE transactions on cybernetics*, 2018.

[YS18] Hiromu Yakura and Jun Sakuma. Robust audio adversarial example for a physical attack. *arXiv preprint arXiv:1810.11793*, 2018.

[ZAFA20a] Hanwei Zhang, Yannis Avrithis, Teddy Furon, and Laurent Amsaleg. Smooth adversarial examples. *EURASIP Journal on Information Security*, 2020(1):1–12, 2020.

[ZAFA20b] Hanwei Zhang, Yannis Avrithis, Teddy Furon, and Laurent Amsaleg. Walking on the edge: Fast, low-distortion adversarial examples. *IEEE Transactions on Information Forensics and Security*, 16:701–713, 2020.

[ZBL+03] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Proceedings of the International conference on Neural Information Processing Systems (NeurIPS)*, 2003.

[ZDS18] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2003.

[ZHC+18] Wen Zhou, Xin Hou, Yongjun Chen, Mengyun Tang, Xiangqi Huang, Xiang Gan, and Yong Yang. Transferable adversarial perturbations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[ZKL+16] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2921–2929, 2016.

[ZRS16] Hongyi Zhang, Sashank J Reddi, and Suvrit Sra. Riemannian svrg: Fast stochastic optimization on riemannian manifolds. In *Proceedings of the International conference on Neural Information Processing Systems (NeurIPS)*, pages 4592–4600, 2016.

[ZSAL20] Wei Emma Zhang, Quan Z Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(3):1–41, 2020.

[ZWG+03] Denny Zhou, Jason Weston, Arthur Gretton, Olivier Bousquet, and Bernhard Schölkopf. Ranking on data manifolds. In *Proceedings of the International conference on Neural Information Processing Systems (NeurIPS)*, 2003.

ABBREVIATIONS

AD Auto-Differentiate. 71

AI Artificial Intelligence. 9, 17, 24

BIM Basic Iterative Method. 37, 121, 129, 130, 165

BP Boundary Projection. 14, 15, 22, 52, 59, 62, 83–86, 89–92, 94–102, 135–137, 163, 165, 168

BPDA Backward Pass Differentiable Approximation. 131, 132, 164, 166

C&W Carlini and Wagner attack. 23, 29, 39–41, 47, 52, 64, 65, 69, 71–81, 84, 90–92, 94, 96–99, 101, 136, 138, 162

CAM Class Activation Maps. 111

CG Conjugate Gradient. 71, 72

CNN Convolutional Neural Network. 9, 17, 26, 111

CV Computer Vision. 9, 17

D3 Divide, Denoise and Defend. 7, 111, 112

DDN Decoupling Direction and Norm. 23, 40, 41, 52, 59, 84, 85, 89–92, 94, 96–101, 106, 121, 129–131, 133, 165

DeepFool DeepFool attack. 35, 40, 42, 43, 50, 162

DL Deep Learning. 10, 17, 18, 21

DNN Deep Neural Network. 5, 9, 10, 13, 17–23, 25, 28, 83, 105, 106, 108, 109, 113

FGSM Fast Gradient sign method. 36, 37, 47, 49, 50, 59, 64, 72, 73, 76–79, 90–92, 94, 96–101, 121, 129, 130, 165

GAN Generative Adversarial Networks. 28, 44, 46

GENCI Grand Equipment National de Calcul Intensif. 121

I-FGSM Iterative Fast Gradient sign method. 37, 64, 72, 73, 76–79, 84, 85, 90–92, 94, 96–101, 165

ILC Iterative Least-likely Class. 43

ILSVRC 2012 ImageNet Large Scale Visual Recognition Challenge. 56, 120

JPEG Joint Photographic Experts Group. 109, 110

JSMA Jacobian-based Saliency Map Attack. 33, 43

L-BFGS Limited-memory Broyden-Fletcher-Goldfarb-Channo. 38, 39, 44, 162

M-IFGSM Momentum Iterative FGSM. 38

MAD Most Apparent Distortion. 61, 73, 75, 76, 162

ML Machine Learning. 9, 17–24, 28, 48

NIC Network Invariance Checking. 46

PGD Projected Gradient Descent. 29, 37, 40, 41, 49, 50, 52, 59, 69, 72–80, 84, 85, 90–92, 94, 96–102, 121, 129–131, 135, 137, 162, 165

PSNR Peak Signal-to-Noise Radio. 61

ReLU Linear Rectification Unit. 26

ResNet Deep Residual Network. 10, 17, 18, 106, 121–123, 130, 132, 164

SSIM Structural Similarity Index Measure. 61

SVM Support Vector Machine. 28, 34, 40, 46

wPSNR Weighted Peak Signal-to-Noise Radio. 61

ZOO Zeroth Order Optimization. 33

LIST OF SYMBOLS

$\mathbf{I}$	raw images in the space $\{0, 1, \ldots, 255\}^{3\times L\times C}$
$\mathsf{T}(\cdot)$	pre-processing which change raw images to inputs
$\mathbf{x}$	inputs images in $[0, 1]^m$
ℓ	label
ℓ_g	ground truth
$\boldsymbol{\theta}$	parameters of neural network
$\mathsf{S}(\cdot)$	softmax
$\mathsf{R}(\cdot)$	function of neural network which before the softmax
$f(\cdot)$	function of neural network give the predicted class
$h(\cdot)$	cross entropy
$\mathcal{L}(\cdot)$	loss function
$\mathcal{J}(\cdot)$	cost function of neural network
Δ	small interval
$\mathbf{u}$	logit
$\mathbf{p}(\cdot)$	probability
$\mathbf{r}$	perturbation
ρ_i	media radius during the search
ϵ	parameter to control the distortion
$\mathrm{proj}_{\mathcal{A}}$	projection on the region $\mathcal{A}$
$B_\infty[\mathbf{x}; \epsilon]$	standard L_∞ centered in $\mathbf{x}$ and radius $\epsilon > \alpha$
$B_2[\mathbf{x}; \epsilon]$	The standard L_2 ball centered in $\mathbf{x}$ and of radius ϵ
$\overline{D}$	distortion
P_{suc}	success rate of an attack
$\mathsf{P}(D)$	probability of success subject to distortion being upper bounded by D
P_{upp}	success rate within a distortion upper bounded by D_{upp}
D_{upp}	distortion upper bounded
$\sigma(\cdot)$	element-wise sigmoid function
Smooth paper	
$\mathbf{t}$	spatial coordinates of the n pixels in the image
$\mathbf{W}$	symmetric adjacency matrix

$\mathsf{k_f}$	feature kernel
$\mathsf{k_s}$	spatial kernel
$\mathbf{D}$	degree matrix
$\mathbf{z}$	new signal
$\mathcal{L}_\alpha$	regularized Laplacian
$\phi_\alpha(\mathbf{r}, \mathbf{z})$	laplacian smooth
$\mathbf{I}_n$	identical function
$\hat{\mathsf{s}}_\alpha$	smoothing guided
$\mathbf{g}$	gradient
n	L2 normalization
c	clipping
$\mathsf{mag}(\mathbf{x})$	magfinication
$\mu_{\mathbf{x}}(\mathbf{x})$	local mean of $\mathbf{x}$
$\sigma_{\mathbf{x}}(\mathbf{x})$	standard deviation of $\mathbf{x}$
$\sigma_\Omega(\mathbf{x})$	global standard deviation of $\mathbf{x}$ over Ω
BP paper	
K	maximum number for iterations
$\mathrm{clip}_{[0,1]}$	clipping
Q_{OUT}	quantization for the out case
Q_{IN}	quantization for the in case
$\mathbf{v}^\star$	direction normal to the gradient
patch denoising	
$\mathcal{F}$	feature tensor
$\mathcal{F}_k$	slice tensor
$\mathcal{P}_{ijk}$	patch tensor
$\mathcal{C}_k$	code book

LIST OF FIGURES

LIST OF TABLES

LIST OF PUBLICATIONS

The list of publications in decreasing chronological order:

— Hanwei ZHANG, Yannis Avrithis, Teddy Furon, Laurent Amasleg. Smooth adversarial examples. In *EURASIP Journal on Information Security*, 2020(1):1-12.

— Hanwei ZHANG, Yannis Avrithis, Teddy Furon, Laurent Amasleg. Walking on the edge: Fast, low-distortion adversarial examples. In *IEEE Transactions on Information Forensics and Security*, 2020, 16:701-713.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

école normale supérieure rennes

Titre : Apprentissage profond dans un contexte adversaire

Mot clés : Attaque adversaire, apprentissage profond, classification, robustesse, sécurité de l'apprentissage automatique.

Résumé : Cette thèse porte sur les attaques adverses et les défenses en apprentissage profond. Nous proposons d'améliorer les performances des attaques adversariales en termes de *vitesse*, de magnitude de *distorsion* et de *invisibilité*. Nous contribuons en définissant *invisibilité* avec lissage et en l'intégrant dans l'optimisation de la production d'exemples adverses. Nous parvenons à créer des perturbations contradictoires lisses avec une amplitude de distorsion moindre. Pour améliorer l'efficacité de la production d'exemples contradictoires, nous proposons un algorithme d'optimisation, *i.e.* BP attaque, basé sur la connaissance du problème contradictoire. BP attaque recherche contre le gradient du réseau pour conduire à une mauvaise classification lorsque la solution actuelle n'est pas contradictoire. Elle cherche le long de la frontière pour minimiser la distorsion lorsque la solution actuelle est contradictoire. BP réussissons à générer des exemples contradictoires avec une faible distorsion de manière efficace. De plus, nous étudions également les défenses. Nous appliquons le remplacement de patchs à la fois sur les images et les caractéristiques. Il supprime les effets adverses en remplaçant les patchs d'entrée par les patchs les plus similaires des données d'entraînement. Les expériences montrent que le remplacement de patch est bon marché et robuste contre les attaques adverses.

Title: Deep Learning in adversarial context

Keywords: Adversarial attack, Deep learning, Classification, Robustness, Machine Learning Security

Abstract: This thesis is about the adversarial attacks and defenses in deep learning. We propose to improve the performance of adversarial attack in the aspect of *speed*, magnitude of *distortion* and *invisibility*. We contribute by defining *invisibility* with smoothness and integrating it into the optimization of producing adversarial examples. We succeed in creating smooth adversarial perturbations with less magnitude of distortion. To improve the efficiency of producing adversarial examples, we propose an optimization algorithm, *i.e.* BP attack, based on the knowledge of the adversarial problem. BP attack searches against the gradient of the network to lead to misclassification when the current solution is not adversarial. It searches along the boundary to minimize the distortion when the current solution is adversarial. BP succeeds to generate adversarial examples with low distortion efficiently. Moreover, we also study the defenses. We apply patch replacement on both images and features. It removes the adversarial effects by replacing the input patches with the most similar patches of training data. Experiments show patch replacement is cheap and robust against adversarial attacks.