

Politechnika Warszawska

W Y D Z I A Ł E L E K T R O N I K I
I T E C H N I K I N F O R M A C Y J N Y C H



Instytut Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Systemy Informacyjno-Decyzyjne

Aplikacja do testowania odporności modeli klasyfikacyjnych
na ataki z użyciem złośliwych danych

Jan Ambroziak

Numer albumu 269260

promotor
dr inż. Paweł Zawistowski

WARSZAWA 2020

Aplikacja do testowania odporności modeli klasyfikacyjnych na ataki z użyciem złośliwych danych

Streszczenie. Wraz z rozwojem i upowszechnianiem się modeli klasyfikujących obrazy, pojawiają się nowe zagrożenia związane z bezpieczeństwem tej klasy rozwiązań. Jednym z nich są złośliwe dane, które mogą powodować, że pozornie dobrze działający klasyfikator zaczyna udzielać błędnych odpowiedzi. Może to stanowić realne zagrożenie nawet dla życia i zdrowia ludzkiego. W pracy opisano, utworzoną na jej potrzeby, aplikację implementującą kilka najpopularniejszych ataków tworzących złośliwe dane. Aplikacja została stworzona w celu pomocy przy testowaniu odporności modeli oraz tworzenia zbiorów zawierających wspomniane wcześniej złośliwe przykłady. W pracy zawarte zostały wyniki eksperymentów pokazujące skuteczność ataków na różnych modelach klasyfikacyjnych, wytrenowanych na popularnych zbiorach danych. Omówione również zostały algorytmy wykorzystanych metod, modele testowe oraz metodologia wykorzystana do uzyskania wyników. Przeprowadzono także dyskusje na temat wydajności sposobów tworzenia złośliwych danych oraz decyzji projektowych podjętych przy projektowaniu aplikacji.

Słowa kluczowe: Złośliwe Dane, Modele Klasyfikacyjne, Nauczanie Maszynowe,

Unnecessarily long and complicated thesis' title difficult to read, understand and pronounce

Abstract. With the rapid growth in popularity and development of image classifying models, new threats appear that threaten the security of this class of model. Among these threats are adversarial examples, which may cause seemingly properly functioning classifiers to malfunction and misclassify data. This poses a serious problem, especially when human life and wellbeing are concerned. In this paper an application implementing a few of the most popular attacks is being described. The application was created to aid in the testing of classification models and creation of datasets containing these malicious examples. I am demonstrating experimental results that show the effectiveness of implemented attack methods, when used on different classification models that are trained on popular datasets. Algorithms of the attack methods, test models, and experimental setup are also described. In the paper I also discuss the efficiency of the methods used and design decisions that were made when creating the application.

Keywords: Adversarial Examples, Classifier Models, Machine Learning



.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanego z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp

Choć pierwsze wzmianki o perceptronie pojawiają się w publikacjach jeszcze z lat 50 [1], a o splotowych sieciach neuronowych w latach 80 [2], to ostatnia dekada przyniosła bezprecedensowy skok w zainteresowaniu tematem zastosowania sieci neuronowych w związku z klasyfikacją obrazów. Od momentu gdy głęboka splotowa sieć neuronowa AlexNet [3] wygrała edycję konkursu ILSVRC [4] w 2012 roku, sieci neuronowe stały się dominującą techniką na polu rozpoznawania obrazów. Współczesne modele korzystają z zdecydowanie bardziej zaawansowanych technik oraz są rzędu wielkości większe od wspominanych prekursorów. Ten postęp pozwolił na rozwiązywanie szeregu wcześniej skomplikowanych problemów, takich jak automatyzacja diagnostyki obrazów medycznych, rozwój autonomicznych pojazdów, czy wachlarz technik biometrycznej identyfikacji.

Jako dobry przykład skoku jakościowego można potraktować wyniki jakie modele osiągają w klasyfikacji zbioru ImageNet [4] czy pokrewnego ImageNet-v2 [5]. W 2011 precyza top-1 najlepszych modeli, nie opierających się na sieciach neuronowych, znajdowała się w okolicach 50%, natomiast opisana w kwietniu bieżącego roku, sieć FixEfficientNet-L2 [6] osiąga, dla zbioru ImageNet-v2 precyzę top-1 na poziomie około 88.5%.

Wraz z rozwojem i upowszechnianiem się modeli klasyfikujących obrazy, pojawiają się nowe zagrożenia związane z bezpieczeństwem tej klasy rozwiązań. Jednym z nich są złośliwe dane, które mogą powodować, że pozornie dobrze działający klasyfikator zaczyna udzielać błędnych odpowiedzi. Może to stanowić realne zagrożenie nawet dla życia i zdrowia ludzkiego. Od momentu publikacji artykułu *Intriguing properties of neural networks* [7] w 2014 roku, w którym autorzy opisali pierwszą metodę tworzenia złośliwych danych. Od tego czasu w środowisku zajmującym się tematyką nauczania maszynowego można zaobserwować swojego rodzaju "wyścig zbrojeń" w poszukiwaniu nowych metod obrony i ataku.

1.1. Cel

Celem pracy inżynierskiej jest stworzenie aplikacji która ma wspierać testowanie odporności modeli klasyfikacyjnych na ataki z użyciem złośliwych danych. Dla dostarczonego w narzuconej postaci modelu klasyfikatora możliwe będzie przeprowadzenie ataku przy pomocy jednej z dostępnych metod.

1.2. Istniejące rozwiązania

Aktualnie na rynku znajduje się kilka rozwiązań implementujących podobne funkcjonalności do tej która jest tu opisana. Warto zapoznać się z nimi oraz ich implementacjami. Jednym z najbardziej popularnych narzędzi tego typu jest *CleverHans* [8] wykorzystujący jako swoją bazę framework TensorFlow oraz *Adversarial Robustness Toolbox* [9] opraco-

1. Wstep

wany przez IBM. Ciekawym przypadkiem jest też biblioteka *Foolbox* [10] która dostarcza natywne wsparcie dla wykorzystywanych frameworków.

2. Złośliwe Dane

W związku z dużym rozwojem uczenia maszynowego (z ang. Machine Learning) w ostatnich latach i zastosowaniem go do rozwiązywania problemów w których poprawne funkcjonowanie jest krytyczne. Zagadnienie bezpieczeństwa tego typu rozwiązań stało się niezwykle istotne. Szereg publikacji z ostatnich lat pokazał martwiące luki w działaniu popularnych technik. Jedną z pierwszych tego typu publikacji jest, udostępniona w 2014 roku, *Intriguing properties of neural networks* [7], która pokazała skuteczną metodę ataku, która wykorzystywała małą zmianę w przykładzie wejściowym klasyfikatora aby zmienić wynik klasyfikacji. Przygotowane w ten sposób złośliwe przykłady (z ang. adversarial examples), zawierające specjalnie przygotowaną perturbację, choć dla ludzkiego oka, praktycznie nieodróżnialne od oryginału, są w stanie skutecznie oszukać klasyfikatory co możemy zaobserwować na rysunku 2.1. Warto też dodać że choć popularne modele splotowych sieci neuronowych są stosunkowo odporne na niewielki losowy szum [11], to wykazują niską odporność na omawiane przez nas ataki.

Implikacje dla bezpieczeństwa, na przykład, systemu rozpoznawania obrazów autonomicznego pojazdu są poważne co pokazano w kilku osobnych publikacjach [12–14]. Kolejne prace starały się rozwinąć wachlarz metod ataków, oraz metod obrony [15–17], które pokrótko opisujemy w rozdziale 2.2.

Możemy opisać złośliwy atak jako funkcję:

$$q(\mathbf{x}, f) = \mathbf{x}' \text{ taką że } f(\mathbf{x}') = \tilde{y} \text{ lub } f(\mathbf{x}') \neq y, \quad (1)$$

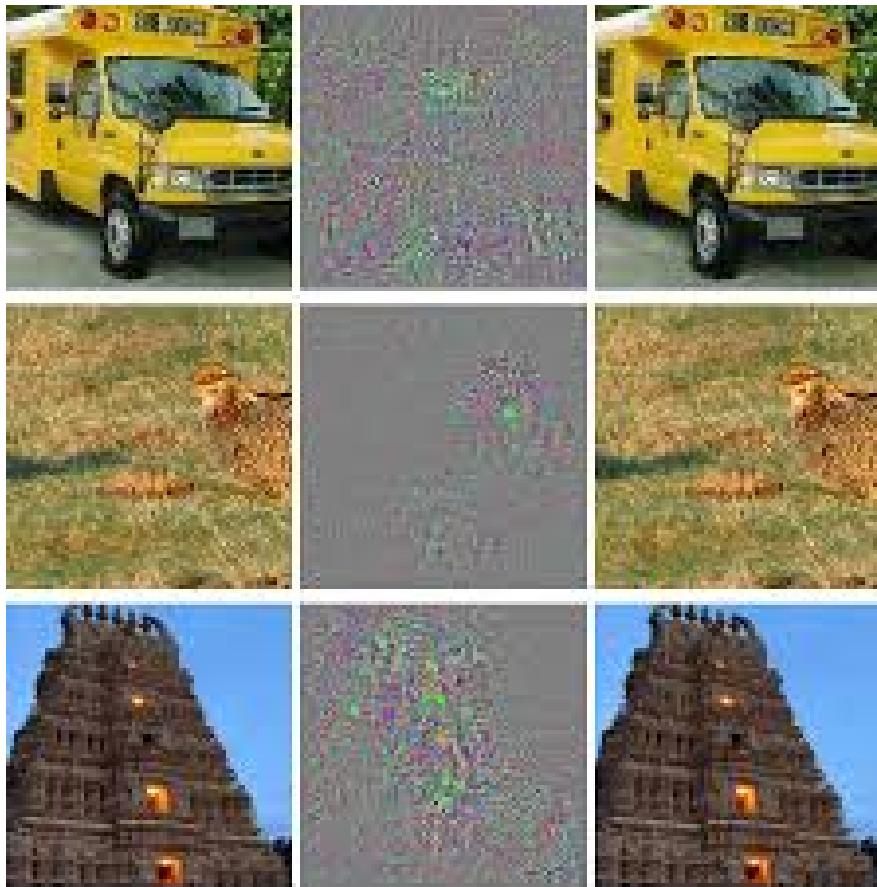
gdzie \mathbf{x} to wektor wejściowy, f to funkcja opisująca nasz model klasyfikacyjny, \mathbf{x}' to wektor reprezentujący wygenerowany złośliwy przykład, y to prawdziwa klasa do której należy przykład wejściowy, a \tilde{y} to zadana klasa z którą chcemy żeby model sklasyfikował nasz złośliwy przykład. Większość metod stara się także minimalizować wprowadzoną do obrazu perturbację \mathbf{r} zdefiniowaną w następujący sposób:

$$\mathbf{x}' = \mathbf{x} + \mathbf{r}. \quad (2)$$

Problem ograniczenia perturbacji zazwyczaj jest rozwiązywany dwójako, albo przez twardy ograniczenie maksymalnej różnicy atrybutów w myśl metryki L_∞ bądź jako próba minimalizacji normy L_2 różnicy wektorów. Podejścia te możemy sformalizować w następujący sposób:

$$\|\mathbf{x} - \mathbf{x}'\|_\infty < \epsilon \text{ lub } \min \|\mathbf{x} - \mathbf{x}'\|_2. \quad (3)$$

Powyższe definicje z uwagi na różnorodność ataków sa opisem dość upraszczającym faktyczną sytuację. Cele naszego ataku mogą być różne, tak samo jak i dostępna dla atakującego wiedza (np. znajomość modelu f czy dostępność przykładu wejściowego \mathbf{x}) opisujemy to szerzej w podrozdziale 2.1.



Rysunek 2.1. Przykład z *Intriguing properties of neural networks* [7]. W lewej kolumnie: poprawnie klasyfikowane oryginalne obrazy, w środkowej: dodana do obrazów perturbacja, w prawej: obrazy z perturbacją klasyfikowane jako struś

Przyczyna skuteczności i istnienia złośliwych danych nie jest znana i wydaje się nie być łatwym problemem do rozwiązania. Natomiast istnieje kilka hipotez starających się wyjaśnić to zjawisko. Autorzy *Explaining and Harnessing Adversarial Examples* [18] sugerują że skuteczność złośliwych danych wynika z zbytniej liniowości modeli klasyfikacyjnych i jest konsekwencją wysoko wymiarowych danych. Natomiast w publikacji *Adversarial examples from computational constraints* [19], zaproponowano że istnienie złośliwych danych jest wynikiem ograniczeń spowodowanych przez wysokie nakłady obliczeniowe współczesnych technik nauczania maszynowego. Autorzy usiłują wykazać, że wytrenowanie modelu który byłby odporny na ataki jest możliwe, lecz bardzo kosztowne obliczeniowo i przez to trudne. Hipoteza zaproponowana w *Intriguing properties of neural networks* [7] zakłada, że skuteczność ataku proponowanego przez autorów jest konsekwencją nieciągłości modelu, wbrew wcześniejszym założeniom dotyczącym metod wykorzystujących filtry splotowe.

2.1. Rodzaje Ataków

Autorzy *The Limitations of Deep Learning in Adversarial Settings* [20] dokonali bardzo pomocnego podziału ataków w dwóch kategoriach - dostępnych danych oraz narzuconego atakowi celu. Tak określony model zagrożenia (z ang. threat model) pozwala nam na klarowny podział ataków i zrozumienie ich przeznaczenia. Warto też dodać że nie wszystkie ataki starają się minimalizować perturbację \mathbf{r} czy też modyfikować przykłady wejściowe. Opublikowane zostały metody ataków wykorzystujących algorytmy ewolucyjne które generują złośliwe dane niezależne od danych wejściowych i nie starające się imitować przykładów ze zbioru danych. Przykładem publikacji w którym opisany został tego typu atak jest *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images* [21].

Narzucone cele ataku możemy uporządkować zgodnie z rosnącą trudnością zadania,

1. Zmniejszenie prawdopodobieństwa z jakim model klasyfikuje przykład jako należący do prawdziwej klasy.
2. Zła klasyfikacja przykładu wejściowego bez zadanej z góry przez atak klasy. Takie ataki nazywamy nieukierunkowanymi(z ang. untargeted), bądź atakami bez zadanej klasy.
3. Wygenerowanie przykładu który zostanie przyporządkowany do zadanej z góry klasy.
4. Zmodyfikowanie istniejącego już przykładu który zostanie przyporządkowany do zadanej z góry klasy. Takie ataki nazywamy ukierunkowanymi(z ang. targeted), bądź atakami z zadaną klasą.

Omawiane przez nas dalej w rozdziale 3.1 ataki są w stanie osiągnąć cel 4 bądź cel 2 z tych opisanych powyżej.

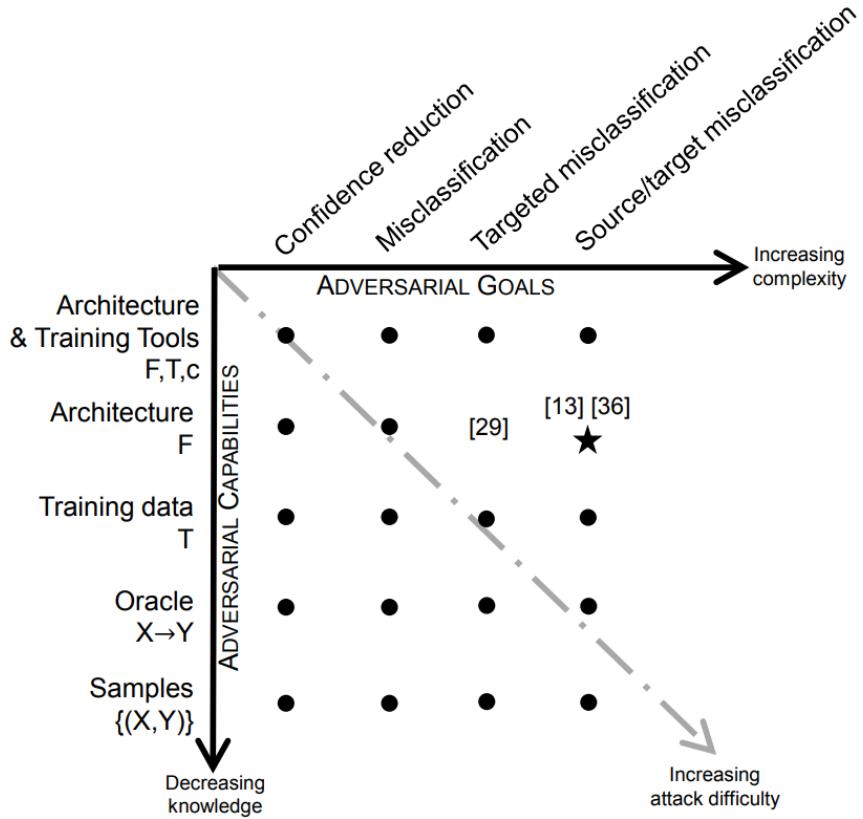
Możemy też rozróżnić ataki z uwagi na to jakie dane są dostępne dla algorytmu atakującego,

1. Dostępna jest pełna wiedza o modelu, który atakujemy. Jego architektura, dane treningowe, sposób trenowania, wartości parametrów dla wszystkich warstw etc.
2. Dostępna jest tylko architektura modelu i wartości parametrów.
3. Dostępny jest podziór zbioru danych używanych do wytrenowania modelu.
4. Dostępny jest model w postaci czarnej skrzynki. Otrzymujemy tylko klasyfikacje na podstawie dostarczonych przez nas przykładów.
5. Dostępne są tylko pary w postaci (*Przykład, Wyjście Modelu*).

W myśl tego rozróżnienia SimpleNet CIFAR-10 4.2, SimpleNet CIFAR-100 4.3, LeNet5 4.1 i Model Splotowy 4.1 w przypadku naszych ataków należą do kategorii 1 jako że są zdefiniowane i wytrenowane przez nas w aplikacji. Natomiast modele MobileNetV2 4.4 i InceptionV3 4.4 do kategorii 2, ponieważ korzystamy z dostarczonych przez bibliotekę

2. Złośliwe Dane

Keras wytrenowanych wag i implementacji. W rozdziale 3.1 opisano kilka popularnych ataków zaimplementowanych w naszej aplikacji.



Rysunek 2.2. Ilustracja powyższego podziału ataków z oryginalnej publikacji [20]

2.2. Metody Obrony

Jak wspomniano wcześniej, istotną częścią zagadnienia ataków na modele klasyfikacyjne z użyciem złośliwych danych są także techniki obrony przed nimi. Warto tu wspomnieć o kilku metodach obrony przed atakami które zostały zaproponowane w publikacjach.

Trenowanie Adwersaryjne Trenowanie Adwersaryjne (z ang. Adversarial Training) [22] to termin opisujący szeroką gamę technik obrony które posiadają zasadniczą cechę wspólną - wykorzystują złośliwe przykłady i metody ich generacji w procesie trenowania. Podstawowa idea polega tego zestawu metod polega na optymalizacji problemu min-max zadanego w następujący sposób:

$$\min_{\theta} \max_{\|(\mathbf{x}, \mathbf{x}')\|_2 < \eta} J(\theta, \mathbf{x}', y), \quad (4)$$

gdzie człon min stara się zminimalizować błąd modelu klasyfikacyjnego poprzez modyfikacje parametrów θ , a człon max stara się znaleźć złośliwe przykłady maksymalizujące funkcję kosztu J .

Różnorodne Trenowanie Adwersaryjne Różnorodne Trenowanie Adwersaryjne (z ang. *Ensemble adversarial training*) czy też EAT. Opisane w *Ensemble Adversarial Training: Attacks and Defenses* [23] to rozwinięcie techniki Trenowania Adwersaryjnego. W tym przypadku zamiast dokonywać kosztownego Trenowania Adwersaryjnego, które musi w każdej treningowej iteracji wygenerować złośliwy przykład oraz przeprowadzić standardową propagację wsteczną w sieci, trenujemy model korzystając ze zbioru wygenerowanych (dla różnych modeli) a priori złośliwych przykładów. To podejście oprócz próby zmniejszenia kosztów obliczeniowych związanych z klasycznym trenowaniem adwersaryjnym, ma na celu zmniejszenie szans powodzenia atak black-box korzystającego z przykładów wygenerowanych dla innego modelu.

Modyfikacja przykładów Głębokie splotowe sieci neuronowe zazwyczaj są dość odporne na losowe zmiany wektora wejściowego. Modyfikacja potencjalnie złośliwych przykładów wejściowych ma na celu “zatrucie” zmian adwersaryjnych w losowej modyfikacji obrazu, które w normalnych warunkach nie mają znacznego wpływu na precyzję klasyfikacji. Publikacje opisujące ten temat podają szereg różnych metod takich jak na przykład zmiana rozmiarów obrazu [24] czy wprowadzenie przed każdą warstwą splotową warstwy zaszumiającej wektor wejściowy warstwy [25].

Maskowanie Gradientu Idea maskowania gradientu opisana oryginalnie w *Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples* [26], stara się utrudnić bądź uniemożliwić obliczenie gradientu obrazu wejściowego względem wektora wyjściowego modelu. Wynika to z tego że duża liczba ataków opiera się na gradiencie.

2. Złośliwe Dane

Przykładem techniki starającej się obfuscować gradient jest destylacja defensywna (z ang. defensive distillation) opisana w *Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks* [15]. Opiera się ona na trenowaniu dwóch modeli. Pierwszy trenujemy z wykorzystaniem oryginalnych etykiet z zbioru wejściowego. Drugi natomiast w procesie trenowania jako etykiety przyjmuje wektor wyjściowy produkowany przez pierwszą sieć dla danego przykładu wejściowego. Natomiast na wyjściu zamiast podawać standardowo, prawdopodobieństwo przynależności przykładowi wejściowemu do każdej z klas, uzyskujemy tylko informacje o klasie z najwyższym wynikiem.

Z maskowaniem gradientu wiązało nadzieję jako z rozwiązaniem problemu ataków wykorzystujących gradient. Niestety pomimo kilku obiecujących publikacji metoda ta okazała się być znacznie mniej skuteczna niż oryginalnie zakładano. Autorzy *Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples* [27], wykazują nieskuteczność tej klasy metod, pokazując dla kilku popularnych wariantów skuteczne ataki ignorujące niedostępność gradientu w modelach. Przykładem takiego ataku jest trenowanie modeli na tych samych, bądź podobnych zbiorach danych oraz tworzenie dla nich złośliwych przykładów, które następnie podajemy na wejściu modelu który atakujemy. Wykorzystuje to fakt że przykłady adwersaryjne dobrze przenoszą się pomiędzy modelami trenowanymi na podobnych zbiorach danych, nawet jeżeli znacznie różnią się architekturą.

Oczywiście nie opisujemy tutaj wszystkich możliwych metod obrony, a jedynie te najbardziej popularne. Każda z wspomnianych powyżej metod posiada co najmniej kilka wariantów opisanych w osobnych publikacjach.

3. Implementacja

Głównym założeniem w trakcie projektowania aplikacji była implementacja algorytmów wykorzystywanych do tworzenia złośliwych przykładów, tak aby mogły być one w prosty sposób wykorzystywane przez użytkowników oraz zapewniały wysoką wydajność przez korzystanie z akceleracji za pomocą GPU. Język skryptowy Python stał się lingua franca w dziedzinie nauczania maszynowego, dlatego rozważania dotyczące biblioteki na której miała bazować nasza aplikacja zostały ograniczone do tych wspierających ten język. Decyzja wyboru TensorFlow była podyktowana głównie łatwością użycia wysokopoziomowego interfejsu i popularnością. Nie mały wpływ na tą decyzję miało też to że zapewnia akceleracje GPU oraz dostarcza API Keras które pozwala na łatwe portowanie modeli między bibliotekami Theano [28], TensorFlow [29] czy CNTK [30]. Aplikacja udostępnia też proste w użyciu funkcjonalności umożliwiające tworzenie zbiorów danych zawierających złośliwe przykłady. Ze strony użytkownika wystarczy dostarczyć model zgodny z API Keras dostępny w TensorFlow, aby móc wykonać wszystkie ataki dostępne w bibliotece.

Szczegóły dotyczące korzystania z dostarczanej przez nas aplikacji, oraz uruchomienia znajdują się w załączonym pliku README.md

Tabela 3.1. Porównanie bibliotek generujących złośliwe przykłady.

X	wsparcie GPU	Liczba Ataków	Wspierane frameworki	Testy自动yczne	L. Modeli
Foolbox	Tak ¹	~33	PyTorch, TensorFlow, JAX, NumPy	Tak	~3
CleverHans	Tak ¹	~12	TensorFlow (JAX i PyTorch wkrótce)	Tak	~5
Adversarial Robustness Toolbox	Tak ¹	~39	TensorFlow, Keras, PyTorch, MXNet, scikit-learn, XGBoost, LightGBM, CatBoost, GPy	Tak	0
Nasza Biblioteka	Tak	6	TensorFlow	Tak	6 + 5 ²

Tabela 3.1 przedstawia porównanie naszej biblioteki z kilkoma popularnymi i ogólnodostępymi rozwiązaniami. Cleverhans jest najstarszą z wymienionych bibliotek natomiast Adversarial Robustness Toolbox najmłodszą. Jeśli oceniać popularność względem zainteresowania użytkowników serwisu github.com to cleverhans zdecydowanie

¹ Zależy od konkretnego frameworku

² 5 modeli nie wykorzystujemy w pracy

3. Implementacja

zajmuje pierwsze miejsce. W naszej bibliotece implementujemy 6 metod ataków oraz 6 pomniejszych wariantów. Łącznie w bibliotece znajduje się 11 modeli, wytrenowanych na wcześniej wspominanych zbiorach, które zostały dostosowane do użycia z naszymi implementacjami ataków. Ataki FGSM, DeepFool, JSMA, L-BFGS-B i Carlini & Wagner wymagają dostępu do gradientu modelu. Wyjątkiem jest metoda GenAttack która, jako że korzysta z algorytmów ewolucyjnych, nie wymaga od nas jakiekolwiek wiedzy o wewnętrznej strukturze modelu. Wydawałoby się że dostęp do wewnętrznej architektury modelu to dość silne ograniczenie aby móc zastosować te ataki w praktyce, aczkolwiek jak pokazano w innych publikacjach złośliwe przykłady z stosunkowo dobrą skutecznością oszukują inne od bazowego modele wytrenowane na tym samym, bądź podobnym zbiorze [31]. Szczegółowy opis metod ataków znajduje się w podrozdziale 3.1.

3.1. Wykorzystane Ataki

W projekcie zaimplementowanych zostało kilka popularnych ataków które zostały opisane w anglojęzycznych publikacjach. Poniżej znajdują się ich opisy, uwagi dotyczące implementacji, przykłady zastosowania oraz wyniki testów zastosowanych do oceny ataków. Szersza dyskusja dotycząca użyteczności oraz porównanie znajdują się w sekcji 5

3.1.1. FGSM

FGSM czyli Metoda szybkiego znaku gradientu (z ang. Fast Gradient Sign Method) opisana w artykule pod tytułem *Explaining and Harnessing Adversarial Examples* [18] to jedna z pierwszych opisanych metod ataku na modele klasyfikacyjne wykorzystujący złośliwe dane. Aby wygenerować złośliwy przykład metoda ta wymaga znajomości gradientu funkcji kosztu obliczanego względem danych wejściowych oznaczonego jako

$$\nabla_x J(\mathbf{x}, y), \quad (5)$$

gdzie, \mathbf{x} to wektor wejściowy, a y to klasa do której należy wektor wejściowy. Idea metody polega na dodaniu małej, arbitralnie ustalonej, wartości ϵ do wektora wejściowego w kierunku znaku gradientu. Pozwala nam to zwiększyć wartość funkcji kosztu, doprowadzając do niepoprawnej klasyfikacji danych wejściowych zmodyfikowanych w niewielkim stopniu.

Możemy zapisać to działanie w następujący sposób:

$$\mathbf{x}' = \mathbf{x} + \epsilon \operatorname{sign}(\nabla_x J(\mathbf{x}, y)), \quad (6)$$

gdzie \mathbf{x}' to spreparowany przykład ze złośliwymi danymi.

Podstawową wadą opisywanej metody jest fakt że nie mamy wpływu na to jaka będzie klasa wyjściowa spreparowanych przez nas danych wejściowych. Koleną jest to że niektóre dane i modele wymagają od nas doboru wysokiej wartości parametru ϵ aby być

w stanie spreparować pożądane dane, co z kolei powoduje dużą różnicę między danymi wejściowymi a tymi utworzonymi w toku stosowania metody. W publikacji pod tytułem *Adversarial examples in the physical world* [32] autorzy opisują kilka metod podobnych do FGSM.

3.1.1.1. I-FGSM czyli Iteracyjna Metoda Szybkiego Znaku Gradientu (z ang. Iterative Fast Gradient Sign Method) to metoda które rozwiązuje problem konieczności doboru wysokiej wartości ϵ poprzez zastosowanie metody FGSM kilkakrotnie, jednocześnie zapewniając pod koniec każdego kroku że przygotowany każdy piksel obrazu nie różni się od oryginału o więcej niż ϵ , z niższą wartością parametru ϵ niż wymagane oryginalnie w metodzie jednokrokowej. Pozwala to potencjalnie ograniczyć różnicę między preparowanymi przez nas danymi, a oryginalnymi danymi wejściowymi oraz przerwanie działania metody kiedy model przestanie poprawnie klasyfikować dane wejściowe.

Algorithm 1 I-FGSM

```

1:  $i \leftarrow 0$ 
2:  $\mathbf{x}' \leftarrow \mathbf{x}$ 
3: while  $i < i_{max}$  do
4:    $\mathbf{x}' \leftarrow \mathbf{x}' + \alpha \text{sign}(\nabla_{\mathbf{x}} J(\mathbf{x}, y))$ 
5:    $i \leftarrow i + 1$ 
6:   Przytnij( $\mathbf{x}', \epsilon$ )
7: end while
```

Funkcja Przytnij() ma na celu ograniczenie różnicy wartości atrybutów przykładu oryginalnego i przykładu złośliwego do co najwyżej ϵ . Robimy to iteracyjnie stosując dla każdego atrybutu wektora reprezentującego złośliwy przykład poniższą metodę:

$$\mathbf{x}'_i = \begin{cases} \mathbf{x}_i + \epsilon & \text{jeśli } \mathbf{x}'_i > \mathbf{x}_i + \epsilon \\ \mathbf{x}_i - \epsilon & \text{jeśli } \mathbf{x}'_i < \mathbf{x}_i - \epsilon \\ \mathbf{x}'_i & \text{jeśli } \mathbf{x}_i - \epsilon < \mathbf{x}'_i < \mathbf{x}_i + \epsilon \end{cases} \quad (7)$$

3.1.1.2. LL-FGSM (z ang. Least Likely FGSM) to metoda pozwalająca nam na spreparowanie danych które będą przydzielane do konkretnej klasy (ozn. \tilde{y}), a nie tylko na niepoprawną klasyfikację. W tym wypadku zamiast starać się zmieniać wartości pikseli obrazu zgodnie z kierunkiem gradientu funkcji kosztu dla prawidłowej klasy staramy się zmieniać wartości pikseli przeciwnie do kierunku gradientu funkcji kosztu dla klasy \tilde{y} . Intuicyjnie można to opisać jako nie oddalanie się od prawdziwej klasy obrazu, a jako przybliżanie się do zadanej przez nas klasy. Autorzy metody zwracają uwagę na przydatność tej metody w przypadku gdy korzystamy z modeli operujących wieloma klasami i gdzie różnice między obiektami z różnych klas mogą być bardzo niewielkie (np. między rasami psów). Metoda ta jest także iteracyjna i zapewne różnice między odpowiednimi pikselami obrazów nie większą niż ϵ więc można uznać ją za rozszerzenie metody I-FGSM.

3. Implementacja

Algorithm 2 LL-FGSM

Input: zadana klasa \tilde{y} z którą chcemy żeby złośliwy przykład był klasyfikowany

```
1:  $i \leftarrow 0$ 
2:  $\mathbf{x}' \leftarrow \mathbf{x}$ 
3: while  $i < i_{max}$  do
4:    $\mathbf{x}' \leftarrow \mathbf{x}' - \alpha \text{sign}(\nabla_x J(\mathbf{x}, \tilde{\mathbf{y}}))$ 
5:    $i \leftarrow i + 1$ 
6:   Przytnij( $\mathbf{x}', \epsilon$ )
7: end while
```

3.1.2. DeepFool

W publikacji *DeepFool: a simple and accurate method to fool deep neural networks* [33] autorzy proponują metodę alternatywną do FGSM mającą minimalizować wprowadzaną do danych wejściowych perturbacje jednocześnie osiągając zmianę klasy do której model przyporządkowuje dane wejściowe.

Wadą DeepFool jest brak możliwości narzucenia klasy do której chcielibyśmy aby przyporządkowywane były nasze zmodyfikowane dane. Metoda ta jest nieco bardziej złożona obliczeniowo jako że pojedynczy krok algorytmu wymaga od nas obliczenia gradientu dla prawdopodobieństwa każdej klasy względem danych wejściowych, co przy zbiorach danych z wieloma klasami takich jak np. ImageNet może być problematyczne. Poniżej znajduje się pseudokod opisujący metodę DeepFool dla modelu wieloklasowego.

Algorithm 3 DeepFool

Input: oryginalny przykład \mathbf{x} , funkcja określająca model $f(\mathbf{x})$

```
1:  $\mathbf{x}^0 \leftarrow \mathbf{x}$ 
2: while  $f(\mathbf{x}^i) = y$  do
3:   for  $k \neq y$  do
4:      $\mathbf{w}'_k \leftarrow \nabla f_k(\mathbf{x}^i) - \nabla f_y(\mathbf{x}^i)$             $\triangleright$  gdzie  $f_{k/y}(\mathbf{x}^i)$  oznacza i/k-ty element wektora
       wyjściowego
5:      $f'_k \leftarrow f_k(\mathbf{x}^i) - f_y(\mathbf{x}^i)$ 
6:   end for
7:    $\hat{l} \leftarrow \arg \min_{k \neq y} \frac{|f'_k|}{\|\mathbf{w}'_k\|_2}$ 
8:    $r_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\mathbf{w}'_{\hat{l}}\|_2^2} \mathbf{w}'_{\hat{l}}$ 
9:    $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \mathbf{r}_i$ 
10:   $i \leftarrow i + 1$ 
11: end while
12:  $\hat{\mathbf{r}} = \sum_i \mathbf{r}_i$ 
13:  $\mathbf{x}' = \mathbf{x}_0 + \hat{\mathbf{r}}$ 
14: return  $\mathbf{x}'$ 
```

3.1.3. L-BFGS-B

Metoda opisana w *Intriguing properties of neural networks* [7] to próba rozwiązania poniższego problemu optymalizacyjnego w celu uzyskania minimalnej perturbacji danych wejściowych która powoduje niepoprawną klasyfikację danych przez model:

$$\min \|\mathbf{r}\|_2, \quad (8)$$

przy warunkach: $\operatorname{argmax} f(\mathbf{x} + \mathbf{r}) = \tilde{\mathbf{y}}$.

Znalezienie dokładnego rozwiązania powyższego problemu jest skomplikowane, dla tego autorzy postanowili szukać aproksymacji rozwiązania poprzez liniowe przeszukiwanie w celu znalezienia najmniejszej wartości parametru $c > 0$ dla którego spełniony zostaje warunek $\operatorname{argmax} f(\mathbf{x} + \mathbf{r}) = \tilde{\mathbf{y}}$ gdzie \mathbf{r} uzyskujemy z zastosowania algorytmu L-BFGS-B dla poniższego problemu:

$$\min c|\mathbf{r}| + J(\mathbf{x} + \mathbf{r}, \tilde{\mathbf{y}}), \quad (9)$$

przy warunkach: $\mathbf{x} + \mathbf{r} \in [0, 1]^m$, gdzie J to funkcja kosztu wykorzystana przy trenowaniu zadanego modelu f .

3.1.4. JSMA

Idea ataku JSMA (Jacobian Saliency Map Attack) opisanego w *The Limitations of Deep Learning in Adversarial Settings* [20] polega na utworzeniu mapy istotności (z ang. Saliency Map) pikseli obrazu dla każdej z klas. Dzięki temu jesteśmy w stanie określić jak dane piksele w obrazie wpływają na określanie prawdopodobieństwa należenia do danej klasy przez model. Sposób tworzenia mapy istotności możemy sformalizować w następujący sposób:

$$S^+(\mathbf{x}, \tilde{\mathbf{y}})[i] = \begin{cases} 0 \text{ jeśli } \frac{\partial \mathbf{f}_{\tilde{\mathbf{y}}}(\mathbf{x})}{\partial \mathbf{x}_i} < 0 \text{ lub } \sum_{j \neq \tilde{\mathbf{y}}} \frac{\partial \mathbf{f}_j(\mathbf{x})}{\partial \mathbf{x}_i} > 0 \\ \left(\frac{\partial \mathbf{f}_{\tilde{\mathbf{y}}}(\mathbf{x})}{\partial \mathbf{x}_i} \right) \left| \sum_{j \neq \tilde{\mathbf{y}}} \frac{\partial \mathbf{f}_j(\mathbf{x})}{\partial \mathbf{x}_i} \right| \text{ w.p.p} \end{cases}, \quad (10)$$

gdzie $S(\mathbf{x}, \tilde{\mathbf{y}})[i]$ to wartość mapy istotności dla i-tego piksela i klasy $\tilde{\mathbf{y}}$, \mathbf{x}_i to wartość i-tego piksela obrazu wejściowego, a $f_{\tilde{\mathbf{y}}}(\mathbf{x})$ to prawdopodobieństwo przynależności \mathbf{x} do klasy $\tilde{\mathbf{y}}$.

W powyżej zdefiniowanej mapie nieujemne wartości oznaczają poziom wpływu zwiększenia intensywności danego piksela i na przynależność dla danej klasy y . Warunek $\frac{\partial \mathbf{f}_{\tilde{\mathbf{y}}}(\mathbf{x})}{\partial \mathbf{x}_i} < 0$ zapewnia że nie będziemy rozpatrywać pikseli które mają negatywny wpływ na wartość prawdopodobieństwa należenia przykładu do klasy y . Natomiast warunek $\sum_{j \neq \tilde{\mathbf{y}}} \frac{\partial \mathbf{f}_j(\mathbf{x})}{\partial \mathbf{x}_i} > 0$ zapewnia że nie będziemy rozpatrywać pikseli które mają pozytywny wpływ

3. Implementacja

na wartość prawdopodobieństwa należenia przykłady do klas innych niż narzucona przez nas klasa \tilde{y} . Autorzy metody proponują też analogiczną mapę istotności w która zamiast określać wpływ zwiększenia intensywności pikseli na przynależność przykładu do danej klasy, jak ma to miejsce w równaniu (10), określa wpływ zmniejszania intensywności piksela na przynależność do danej klasy. Możemy opisać tą metodę w następujący sposób:

$$S^-(\mathbf{x}, \tilde{y})[i] = \begin{cases} 0 \text{ jeśli } \frac{\partial f_{\tilde{y}}(\mathbf{x})}{\partial \mathbf{x}_i} > 0 \text{ lub } \sum_{j \neq \tilde{y}} \frac{\partial f_j(\mathbf{x})}{\partial \mathbf{x}_i} < 0 \\ \left| \frac{\partial f_{\tilde{y}}(\mathbf{x})}{\partial \mathbf{x}_i} \right| \left(\sum_{j \neq \tilde{y}} \frac{\partial f_j(\mathbf{x})}{\partial \mathbf{x}_i} \right) \text{ w.p.p} \end{cases}. \quad (11)$$

W praktyce jednak większość pikseli nie spełnia warunków określonych w pierwszej linii podanych równań, dlatego autorzy zastosowali metodę wyboru par pikseli p_1 i p_2 zamiast pojedynczego piksela. W tym celu stosowana jest opisana poniżej metoda.

$$\arg \max_{(p_1, p_2)} \left(\sum_{i=p_1, p_2} \frac{\partial f_{\tilde{y}}(\mathbf{x})}{\partial \mathbf{x}_i} \right) \times \left| \sum_{i=p_1, p_2} \sum_{j \neq \tilde{y}} \frac{\partial f_j(\mathbf{x})}{\partial \mathbf{x}_i} \right| \quad (12)$$

O ile rozwiązuje to problem znalezienia pikseli które spełniają warunki o tyle metoda ta ma większą złożoność obliczeniową z uwagi na to że musimy rozpatrzyć wszystkie możliwe pary pikseli zamiast tylko pojedynczych pikseli. Algorytm z listingu 4 opisuje idee działania metody JSMA.

Algorithm 4 JSMA

```

1:  $\mathbf{x}' \leftarrow \mathbf{x}$ 
2: while  $f(\mathbf{x}') \neq \tilde{y}$  &  $i < i_{max}$  do
3:    $p_1, p_2 = S(\mathbf{x}', \tilde{y})$                                  $\triangleright$  przez S rozumiemy (12)
4:   zmodyfikuj  $p_1$  i  $p_2$  o  $\theta$ 
5:   jeśli  $p_1$  lub  $p_2$  wynosi 0 albo 1 usuń je z listy pikseli
6:    $i \leftarrow i + 1$ 
7: end while
8: return  $\mathbf{x}'$ 

```

Istnieje wiele różnych wariantów metody JSMA, których zasadnicze działanie nie różni się bardzo od tego opisanego powyżej, *Maximal Jacobian-based Saliency Map Attack* [34] przeprowadza bardzo zwięźle ich podsumowanie. Warto tutaj przytoczyć kilka różnic między opisanymi w tej publikacji wariantami JSMA.

JSMA+ i JSMA- Autorzy *Maximal Jacobian-based Saliency Map Attack* [34] wprowadzają rozróżnienie pomiędzy JSMA+ a JSMA- w zależności od tego czy intensywność pikseli jest zmniejszana i wykorzystywane są warunki (11) czy też intensywność pikseli jest zwiększana i wykorzystywane są warunki (10)

JSMA-F i JSMA-Z W publikacji *Towards Evaluating the Robustness of Neural Networks* [35] autorzy proponują rozróżnienie pomiędzy JSMA-F, które w metodzie (12) wykorzystuje do obliczania pochodnej cząstkowej wyjście funkcji softmax stosowanej zazwyczaj jako ostatnia warstwa modelu, a JSMA-Z które zamiast tego wykorzystuje wektor wyjściowy przedostatniej warstwy określany zazwyczaj jako logits.

NT-JSMA czyli wersja JSMA bez zadanej klasy którą chcemy osiągnąć (z ang. Non Targeted JSMA). To postulowany przez autorów *Maximal Jacobian-based Saliency Map Attack* [34] wariant metody JSMA który poprzez zdobycie ograniczenia polegającego na tym że spreparowana dane mają być klasyfikowane jako należące do zadanej klasy ma umożliwić zmniejszenie perturbacji wymaganej do zmiany klasyfikacji danych przez model. To usprawnienie niesie ze sobą jednak dodatkowy koszt obliczeniowy jako że w każdej iteracji rozpatrujemy $S^-()$ bądź $S^+()$ nie dla jednej zadanej klasy tylko dla wszystkich.

M-JSMA [34] to wariant metody JSMA który łączy w sobie metodę NT-JSMA oraz pozbywa się dotychczas istniejącego ograniczenia w postaci wyboru tego czy intensywność pikseli jest zmniejszana czy zwiększa. Zamiast tego w każdym kroku ewaluowane są wszystkie pary pikseli w kontekście zwiększenia i zmniejszenia intensywności dla wszystkich możliwych klas. Wybierana jest taka para która najbardziej oddala nas od prawdziwej klasy obrazu.

3.1.5. Carlini & Wagner

W metodach takich jak FGSM, gdzie w celach optymalizacji wykorzystywany jest gradient, twarde ograniczanie wartości atrybutów tak aby spełniały następujące warunki:

$$\mathbf{x} + \mathbf{r} \in [0, 1]^m, \quad (13)$$

może mieć niepożądane konsekwencje. Autorzy *Towards Evaluating the Robustness of Neural Networks* [35] starali się zaadresować ten problem. Zamiast stosowania twardego ograniczenia, jak np. funkcja Przytnij, proponują oni inne podejście. Poprzez podstawienie zmiennej:

$$\mathbf{r} = \frac{1}{2}(\tanh(\mathbf{w}) + 1) - \mathbf{x}, \quad (14)$$

możemy zapewnić spełnienie warunków (13). Następnym krokiem jest rozwiązywanie tak zadanego problemu:

$$\min_{\mathbf{w}} \left\| \frac{1}{2}(\tanh(\mathbf{w}) + 1) - \mathbf{x} \right\|_2^2 + c \cdot g\left(\frac{1}{2}(\tanh(\mathbf{w}) + 1)\right) \quad (15)$$

gdzie g definiujemy jako:

$$g(\mathbf{x}') = \max(\max\{f(\mathbf{x}')_i : i \neq \tilde{y}\} - f(\mathbf{x}')_{\tilde{y}}, -\kappa), \quad (16)$$

za pomocą metod optymalizacji, bazujących na gradiencie, znanych z nauczania maszynowego i znalezieniu w ten sposób wektora perturbacji $\mathbf{r} = \frac{1}{2}(\tanh(\mathbf{w}) + 1) - \mathbf{x}$. Pierwsza składowa sumy (15) odpowiada za minimalizację odstępstwa spreparowanego obrazu od oryginału. Druga składowa odpowiada za zwiększenie prawdopodobieństwa z jakim model klasyfikuje nasz przykład jako należący do zadanej klasy. Parametr κ odpowiada za pewność z jaką chcemy żeby model klasyfikował nasz przykład jako należący do zadanej klasy. Parametr c jest arbitralnie dobraną wartością, możliwe najmniejszą dla której minimalizacja znajduje przykład zdolny do oszukania klasyfikatora, którą znajdujemy za pomocą bisekcji w zadanym przedziale.

3.1.6. GenAttack

Metoda GenAttack opisana w *GenAttack: Practical Black-box Attacks with Gradient-Free Optimization* [36] to metoda opierająca się na algorytmach ewolucyjnych gdzie, strategia ewolucyjna przyjęta przez autorów polega na generacji przykładu adwersaryjnego tylko dla jednej klasy naraz. Autorzy proponują aby oceniać jakość złośliwych przykładów nie tylko na podstawie tego z jakim dużym prawdopodobieństwem model klasyfikuje złośliwy przykład jako należący do $\tilde{\mathbf{y}}$, ale także na podstawie tego jak duża jest jego odległość od bycia sklasyfikowanym z inną klasą. Możemy sformułować tą metodę oceny w następujący sposób:

$$Q = \log f_{\tilde{\mathbf{y}}}(\mathbf{x}) - \log \sum_{j=0, j \neq \tilde{\mathbf{y}}}^{j=k} f_j(\mathbf{x}), \quad (17)$$

gdzie k to liczba klas w zbiorze, a $f(\mathbf{x})_j$ to prawdopodobieństwo należenia przykładu \mathbf{x} do klasy j przypisane przez model f .

W algorytmie 5, $U(a, b)$ oznacza zmienną losową z rozkładu jednostajnego ciągłego, funkcja Skrzyżuj(s_1, s_2) krzyżuje osobniki z prawdopodobieństwem przekazania danego atrybuty dziecku od rodzica s_1 z prawdopodobieństwem $\frac{Q(s_1)}{Q(s_1) + Q(s_2)}$. Funkcja Przytnij() jest identyczna do tej która znajduje się w sekcji 3.1.1. Funkcja ZaktualizujParametry(ρ, α) ma na celu modyfikacje parametrów ρ i α , na wypadek braku poprawy w jakości Q kolejnych generacji, w następujący sposób:

$$\begin{aligned} \rho &= \max(\rho_{\min}, 0.5 \times (0.9)^{\text{iteracje bez poprawy}}) \\ \alpha &= \max(\alpha_{\min}, 0.4 \times (0.9)^{\text{iteracje bez poprawy}}) \end{aligned} \quad (18)$$

Algorithm 5 GenAttack

Input: oryginalny przykład \mathbf{x} , zadana klasa $\tilde{\mathbf{y}}$, maksymalna perturbacja δ_{max} w sensie metryki L_∞ , zakres mutacji α , prawdopodobieństwo mutacji ρ , rozmiar populacji N , temperatura próbkowania τ

```

1: for  $i = 1, \dots, N$  w populacji do
2:    $P_i^0 \leftarrow \mathbf{x} + U(-\delta_{max}, \delta_{max})$                                  $\triangleright$  Tworzymy oryginalną populację
3: end for
4: for  $g = 1, 2, \dots$  generacji do
5:   for  $i = 1, \dots, N$  w populacji do
6:      $F_i^{g-1} = Q(P_i^{g-1})$ 
7:   end for
8:    $\mathbf{x}' = P^{g-1} \underset{\arg \max_j F_j^{g-1}}{} \quad \triangleright$  Znajdź najlepszy złośliwy przykład
9:   if  $\arg \max_c f(\mathbf{x}')_c == \tilde{\mathbf{y}}$  then
10:    return  $\mathbf{x}'$ 
11:   end if
12:    $P_1^g = \{\mathbf{x}'\}$ 
13:    $p = \text{Softmax}(F^{g-1}/\tau) \quad \triangleright$  Oblicz prawdopodobieństwa wyboru osobników
14:   for  $i = 2, \dots, N$  w populacji do
15:     Wybierz  $\mathbf{s}_1$  z  $P^{g-1}$  zgodnie z  $p$ 
16:     Wybierz  $\mathbf{s}_2$  z  $P^{g-1}$  zgodnie z  $p$ 
17:      $\mathbf{c} = \text{Skrzyżuj}(\mathbf{s}_1, \mathbf{s}_2)$ 
18:      $\mathbf{c}_{mut} = \mathbf{c} + \text{Bernouli}(\rho) * U(-\alpha \delta_{max}, \alpha \delta_{max})$ 
19:      $\mathbf{c}_{mut} = \text{Przytnij}(\mathbf{c}_{mut}, \delta_{max}) \quad \triangleright$  Zmutuj i przytnij dziecko
20:      $P_i^g = \{\mathbf{c}_{mut}\} \quad \triangleright$  Dodaj zmutowane dziecko do następnej generacji
21:   end for
22:    $\rho, \alpha = \text{ZaktualizujParametry}(\rho, \alpha) \quad \triangleright$  Zaktualizuj parametry  $\rho$  i  $\alpha$ 
23: end for

```

4. Modele i dane testowe

Aby sprawdzić poprawność implementacji oraz skuteczność ataków stworzono kilka modeli klasyfikacyjnych o różnej złożoności, opartych o popularne zbiory danych wykorzystywanych jako platforma testowa w publikacjach pokrewnych tematyką.

4.1. MNIST

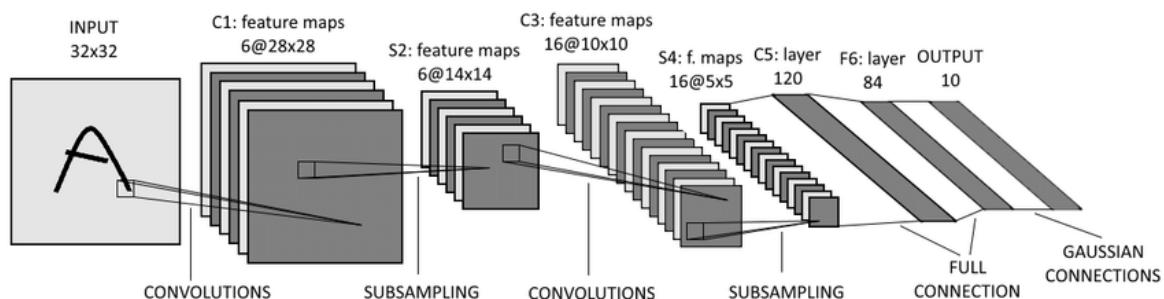
Zbiór danych MNIST [37] to najprawdopodobniej najpopularniejszy zbiór związany z tematyką nauczania maszynowego. Zawiera on 60,000 obrazów o rozmiarach 28 na 28 pikseli, w skali szarości, przedstawiających odręcznie narysowane cyfry. Zbiór dzieli się na 50,000 przykładów używanych do trenowania modelu oraz 10.000 służących do testowania.

Model Splotowy to prosty przykład zastosowania splotowych sieci neuronowych, warstw

Dropout i poolingu który w praktyce pozwala osiągnąć bardzo dobre wyniki dla zbioru MNIST. Tutaj posłużyono się przykładem ze Tensorflow Model Garden [38] który jest stosunkowo prosty. Ten model pozwala nam na uzyskanie 93% precyzji dla zbioru MNIST.

LeNet5 to historyczny istotny model stworzony przez Yana LeCuna stworzony w latach.

Jest jednym z najbardziej znanych zastosowań splotowych sieci neuronowych. Wykorzystuje tylko warstwy splotowe pooling i standardowe perceptrony. Jego struktura jest chyba najprostrszą z wykorzystywanych przez nas modeli.



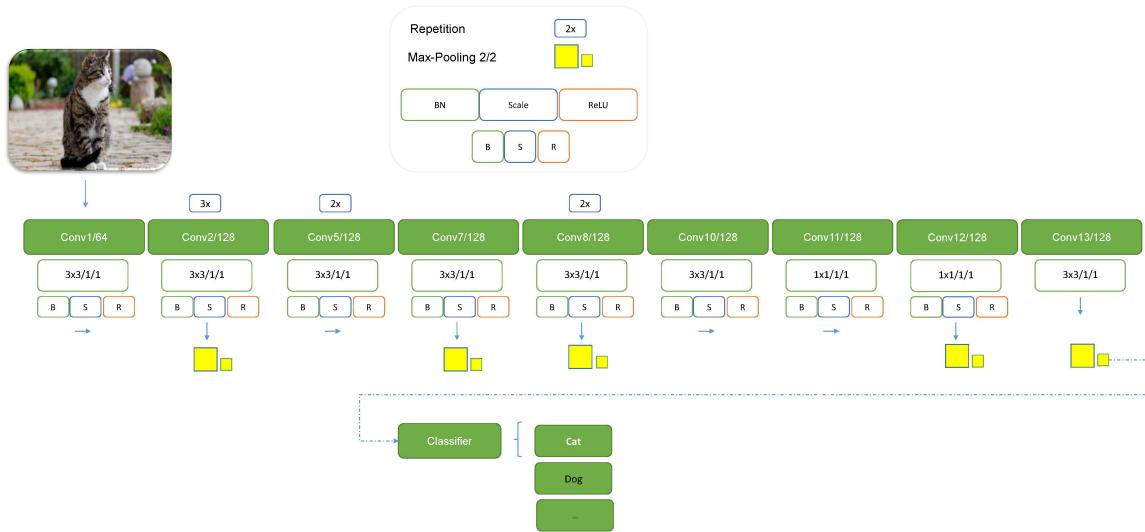
Rysunek 4.1. Struktura Modelu LeNet5 [39]

4.2. CIFAR-10

Zbiór CIFAR-10 opisany w *Learning Multiple Layers of Features from Tiny Images* [40] zawiera kolorowe obrazy o rozdzielczości 32 na 32 piksele, podzielone na 10 klas takich jak np. koń, samolot czy żaba. Podobnie jak zbiór MNIST, CIFAR-10 zawiera 60,000 obrazów podzielonych na 50,000 przykładów używanych do treningu jak i 10,000 do testowania. Aby usunąć problem nadmiernego dopasowania modeli obrazy z zbioru CIFAR10 używane do trenowania są losowo modyfikowane. Każdy obraz ma szansę 0.25 na modyfikację

natężenia, nasycenia, kontrastu oraz odcienia. Dodatkowo każdy obraz może zostać obrócony bądź odbity.

SimpleNet CIFAR-10 opisany w *Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures* [41] jest odpowiedzią na bardziej złożone architektury sieci neuronowych (np. InceptionV3 4.4), które swoją nieznaczenie wyższą precyzję okupywały niewspółmiernie większą złożonością architektury i liczbą parametrów. SimpleNet wykorzystuje klasyczne techniki znane z splotowych sieci neuronowych takie jak batch normalization, warstwy splotowe, pooling, rektyfikująca funkcja aktywacji czy warstwy Dropout. Skutkiem tego jest model wymagający znaczco mniejszy nakładów obliczeniowych. Zarówno przy trenowaniu sieci jak i przy klasyfikacji. Model WRN [42] posiadający 11 milionów parametrów uzyskuje precyzję na poziomie około 95% dla zbioru CIFAR-10. SimpleNet uzyskuje w naszej implementacji precyzję 91.4% dla około 5 milionów parametrów. Autorzy SimpleNet zgłaszał precyzję modelu na poziomie 95.51% natomiast precyzja na poziomie 91.4% jest wystarczająca do naszych potrzeb. Z uwagi na to że nie jest dostępna oficjalna wersja modelu SimpleNet kompatybilna z tensorflow, konieczne było stworzenie własnej. Na Rysunku 4.2 znajduje się poglądowy schemat architektury modelu SimpleNet.



Rysunek 4.2. Schemat modelu SimpleNet [41]

4.3. CIFAR-100

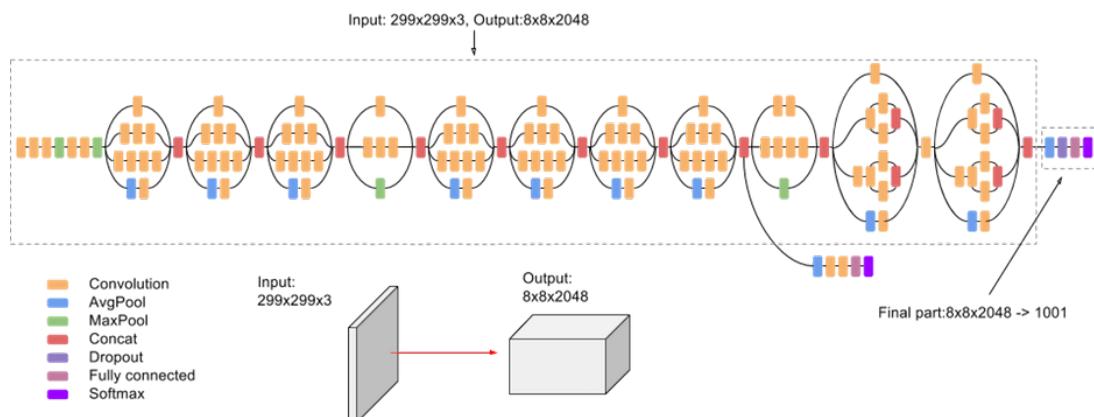
Zbiór CIFAR100 opisany, podobnie jak CIFAR-10 w *Learning Multiple Layers of Features from Tiny Images* [40] jest zbiorem obrazów, o rozdzielczości 32x32 pikseli z trzema kanałami kolorów, podzielonym na 100 klas. Każda klasa posiada w tym zbiorze 600 przykładów, z czego 500 jest przykładami treningowymi, a 100 przykładami testowymi.

SimpleNet CIFAR-100 - Jako klasyfikator dla zbioru CIFAR-100 wybrany został także model SimpleNet. Architektura sieci i sposób trenowania jest identyczny jak w przypadku zbioru CIFAR-10. Jedyną różnicą jest zbiór treningowy którym w tym przypadku jest CIFAR-100. Autorzy modelu [41] zgłoszają dla zbioru CIFAR-100 precyzyję top-1 na poziomie 73.42%. Nasza implementacja uzyskuje precyzyję top-1 na poziomie 65.90% i top-5 na poziomie 89.81%

4.4. ILSVRC2012

Zbiór ILSVRC 2012 popularnie nazywany zbiorem ImageNet to, bazowany na bazie danych obrazów ImageNet, zbiór przygotowany specjalnie pod konkurs *ImageNet Large Scale Visual Recognition Challenge* [4]. Baza danych ImageNet bazuje na bazie WordNet i tworzy hierarchie obrazów uporządkowaną względem synsetów czyli zbiorów synonimów (syn od synonym i set z ang. zbiór). Zbiór ILSVRC 2012 posiada 1000 klas i około 1000 obrazów dla każdej klasy. Obrazy mają rozdzielcość 299x299 pikseli oraz 3 kanały kolorów. Łącznie zbiór posiada 1281167 przykładów treningowych oraz 50000 przykładów testowych.

InceptionV3 to zaproponowany przez autorów *Rethinking the Inception Architecture for Computer Vision* [43] model którego celem było umożliwienie skalowania architektur modeli splotowych przy ograniczeniu wzrostu wymaganej mocy obliczeniowej za pomocą zastosowania warstw "Inception". InceptionV3 uzyskuje precyzyje top-1 na poziomie 78.8% oraz precyzyję top-5 na poziomie 94.4% dla zbioru ILSVRC 4.4 w implementacji dostarczanej przez bibliotekę Keras. Jest to też największy z wykorzystanych modeli. Posiada ponad 23 miliony parametrów oraz 159 warstw.



Rysunek 4.3. Schemat modelu InceptionV3

MobileNetV2 - Autorzy *Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification* [44] stworzyli model którego celem, podobnie jak SimpleNet, było zmniejszenie rozmiarów sieci i potrzebnych nakładów obliczeniowych do jej używania,

przy jednoczesnym zachowaniu wysokiego poziomu precyzyji. W tym celu autorzy wykorzystali zmodyfikowaną wersję warstw rezydualnych (z ang. residual layer) z takich modeli jak np. ResNet [45]. O ile autorzy modelu SimpleNet, starali się uzyskać podobny cel dla zbiorów CIFAR-10 i CIFAR-100, tak tutaj autorzy skupili się przede wszystkim na zbiorze ILSVRC. MobileNetV2 działa z precyją top-1 na poziomie 70.6% i top-5 89.8% w implementacji dostarczanej przez bibliotekę Keras. MobileNetV2 posiada 4.2 miliona parametrów oraz 88 warstw.

Rysunek 4.4. Schemat warstwy *residual bottleneck* [46]

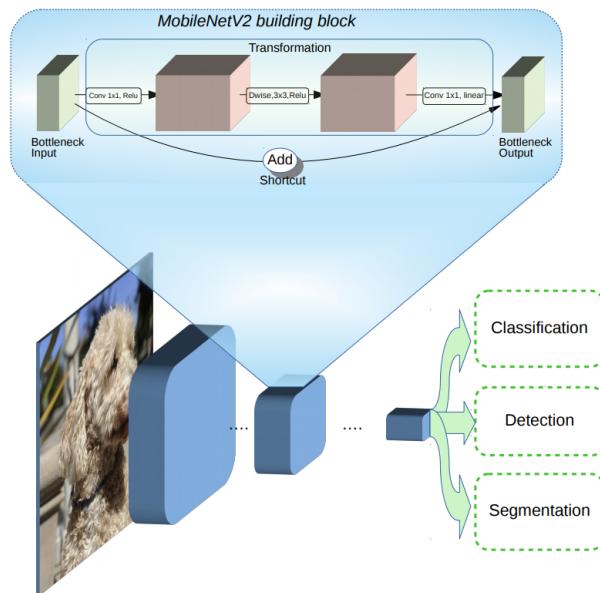


Tabela 4.1. Precyza top-1 i top-5 wykorzystywanych przez nas modeli

Model	Zbiór	Top-1	Top-5
MNIST TF Model	MNIST	93.3%	99.4%
Le Net 5	MNIST	97.7%	100.0%
SimpleNet CIFAR-10	CIFAR-10	91.4%	99.7%
SimpleNet CIFAR-100	CIFAR-100	65.9%	89.8%
InceptionV3	ILSVRC2012	76.8%	93.5%
MobileNetV2	ILSVRC2012	70.6%	89.9%

5. Wyniki Eksperymentów

W celu oceny skuteczności ataków oraz ich porównania zastosujemy miary które nam to umożliwiają. Autorzy *Towards Evaluating the Robustness of Neural Networks* [35] wykorzystują do oceny swojego ataku precyzję z jaką jest on w stanie oszukać model, którą możemy sformułować w następujący sposób:

$$\text{ACC} = \begin{cases} \frac{|\{\arg \max(f(q(\mathbf{x})) \neq y : \mathbf{x} \in X\}|}{|X|} & \text{gdy } \tilde{y} \text{ nie istnieje} \\ \frac{|\{\arg \max(f(q(\mathbf{x})) = \tilde{y} : \mathbf{x} \in X\}|}{|X|} & \text{w p.p.} \end{cases}, \quad (19)$$

gdzie $\arg \max(f(q(\mathbf{x}))$ to klasa z jaką model klasyfikuje złośliwy przykład $q(\mathbf{x})$. W tej samej publikacji posługują się też średnią z metryki L_2 różnicy między obrazem oryginalnym a złośliwym określoną jak poniżej:

$$\overline{L_2} = \frac{\sum_{\mathbf{x} \in X} \|q(\mathbf{x}) - \mathbf{x}\|_2}{|X|}. \quad (20)$$

Autorzy *DeepFool: a simple and accurate method to fool deep neural networks* [33] zastosowali średnią wszechstronność (z ang. *robustness*) jako miarę ataku. Można ją sformułować w następujący sposób:

$$\rho_{adw} = \frac{1}{|X|} \sum_{\mathbf{x} \in X} \frac{\|q(\mathbf{x}) - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} \quad (21)$$

miara ρ_{adw} jest przydatna zwłaszcza w kontekście porównywania ataków dla obrazów należących do różnych zbiorów danych gdzie porównanie za pomocą metryki L_2 nie uwzględnia różnicy w ilości atrybutów między zbiorami danych.

Dodatkowo mierzymy czas T jaki średnio potrzebujemy aby znaleźć złośliwy przykład dla pojedynczego przykładu wejściowego.

5.1. Procedura Eksperimentalna

Procedurę eksperimentalną możemy opisać w następujących krokach:

1. Filtrowanie Danych - wpierw odrzucamy przykłady z zbiorów danych które modele klasyfikują niepoprawnie, a w przypadku ataków zadaną klasą, także takie które już należą do zadanej klasy.
2. Dobór parametrów - dla każdego modelu wykonujemy atak dla różnych zestawów, arbitralnie dobranych, parametrów ataku.
3. Wykonanie ataku - wykonujemy atak na transzy (z ang. batches) przykładów ze zbioru. Jeśli atak jest atakiem zadaną klasą \tilde{y} , wówczas cała transza będzie miała taką samą zadaną klasę \tilde{y} wybraną losowo z prawdopodobieństwem o rozkładzie jednostajnym.

4. Obliczenie miar - po wykonaniu ataku na transzy przykładów obliczamy miary w celu zebrania wyników.
5. Powtarzamy punkty 3 oraz 4 aż do wyczerpania zbioru danych albo narzuconego limitu transz w wypadku gdy atak jest czasochłonny.

Z uwagi na to że wykonanie wszystkich eksperymentów dla całych zbiorów testowych jest w naszym przypadku bardzo czasochłonne, to liczba przykładów wejściowych na podstawie których obliczamy miary ataku jest różna i zależy od wymagań obliczeniowych danego algorytmu.

5.2. Wyniki

W celu porównania ze sobą ataków postanowiliśmy dokonać porównania podobnego do tego które wykorzystali organizatorzy konkursu *Adversarial Vision Challenge* [47] odbywającego się w ramach konferencji *Conference on Neural Information Processing Systems*. Dla każdego zestawu parametrów ataku obliczamy budżet. Budżet rozumiemy jako maksymalną wartość sumy liczby wywołań gradientu i wnioskowań potrzebnych aby uzyskać finalny złośliwy przykład. Następnie dokonujemy porównania dla ataków których budżety zawierają się w tych samych przedziałach, osobno dla każdego ze zbiorów danych. Bardziej szczegółowe wyniki eksperymentów wraz z opisem parametrów ataków znajdują się w Załączniku A. Natomiast przykłady złośliwych obrazów wygenerowanych przez zaimplementowane przez nas ataki znajdują się w Załączniku B.

Budżet	Atak	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
10	FGSM	100.0%	10.66839	10.62969	0.00155	1.18090
	LL-FGSM	33.3%	5.48417	5.45619	0.00171	0.60710
100	FGSM	63.4%	1.86275	1.85770	0.00137	0.20514
	LL-FGSM	4.6%	1.91948	1.89876	0.00200	0.21466
	Carlini & Wagner	0.0%	18.82104	18.82911	0.01011	2.13603
1000	JSMA	100.0%	4.02858	4.02593	0.77459	0.44574
	L-BFGS-B	100.0%	1.70078	1.75999	0.39974	0.18722
	DeepFool	97.0%	1.27814	1.37602	0.38435	0.14677
	FGSM	64.2%	1.85295	1.84908	0.00301	0.20428
	LL-FGSM	39.9%	1.76034	1.73667	0.00289	0.19132
	Carlini & Wagner	0.0%	18.72053	18.72017	0.01603	2.08537
10000	L-BFGS-B	100.0%	1.87018	1.85254	0.39557	0.19865
	FGSM	64.5%	1.85179	1.84821	0.02314	0.20419
	LL-FGSM	40.6%	1.76015	1.75396	0.02267	0.19331
	GenAttack	2.0%	0.93940	0.93640	8.24402	0.10567
	Carlini & Wagner	0.0%	18.01404	18.01498	0.05206	2.01053
100000	Carlini & Wagner	76.1%	8.84674	9.03522	0.29088	1.01435

Tabela 5.1. Wyniki eksperymentów dla zbioru MNIST

5. Wyniki Eksperymentów

Budżet	Atak	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
10	FGSM	91.9%	5.49757	5.43992	0.02374	0.19611
	LL-FGSM	10.7%	0.55418	0.55182	0.01759	0.01953
100	FGSM	100.0%	2.20887	2.22380	0.03305	0.07995
	LL-FGSM	93.0%	2.01795	2.01958	0.02415	0.07151
	Carlini & Wagner	0.0%	19.66507	19.83514	0.03554	0.75201
1000	DeepFool	100.0%	0.99746	0.98337	0.03880	0.03523
	FGSM	100.0%	2.40864	2.38752	0.09468	0.08581
	LL-FGSM	86.6%	2.32934	2.08754	0.08763	0.07547
	L-BFGS-B	59.2%	0.19474	0.16874	5.42402	0.00621
	Carlini & Wagner	0.0%	19.62749	19.81175	0.07000	0.75111
10000	JSMA	100.0%	4.31146	4.32629	38.52092	0.16045
	FGSM	100.0%	2.61824	2.57203	0.75909	0.09246
	LL-FGSM	79.6%	2.35453	1.96145	0.75822	0.07200
	L-BFGS-B	54.9%	0.13877	0.14815	8.25832	0.00546
	Carlini & Wagner	37.0%	18.50197	18.65609	0.40281	0.70730
	GenAttack	33.0%	1.96935	1.94352	10.27956	0.06963
100000	Carlini & Wagner	93.6%	4.26353	4.51120	3.72238	0.17541

Tabela 5.2. Wyniki eksperymentów dla zbioru CIFAR-10

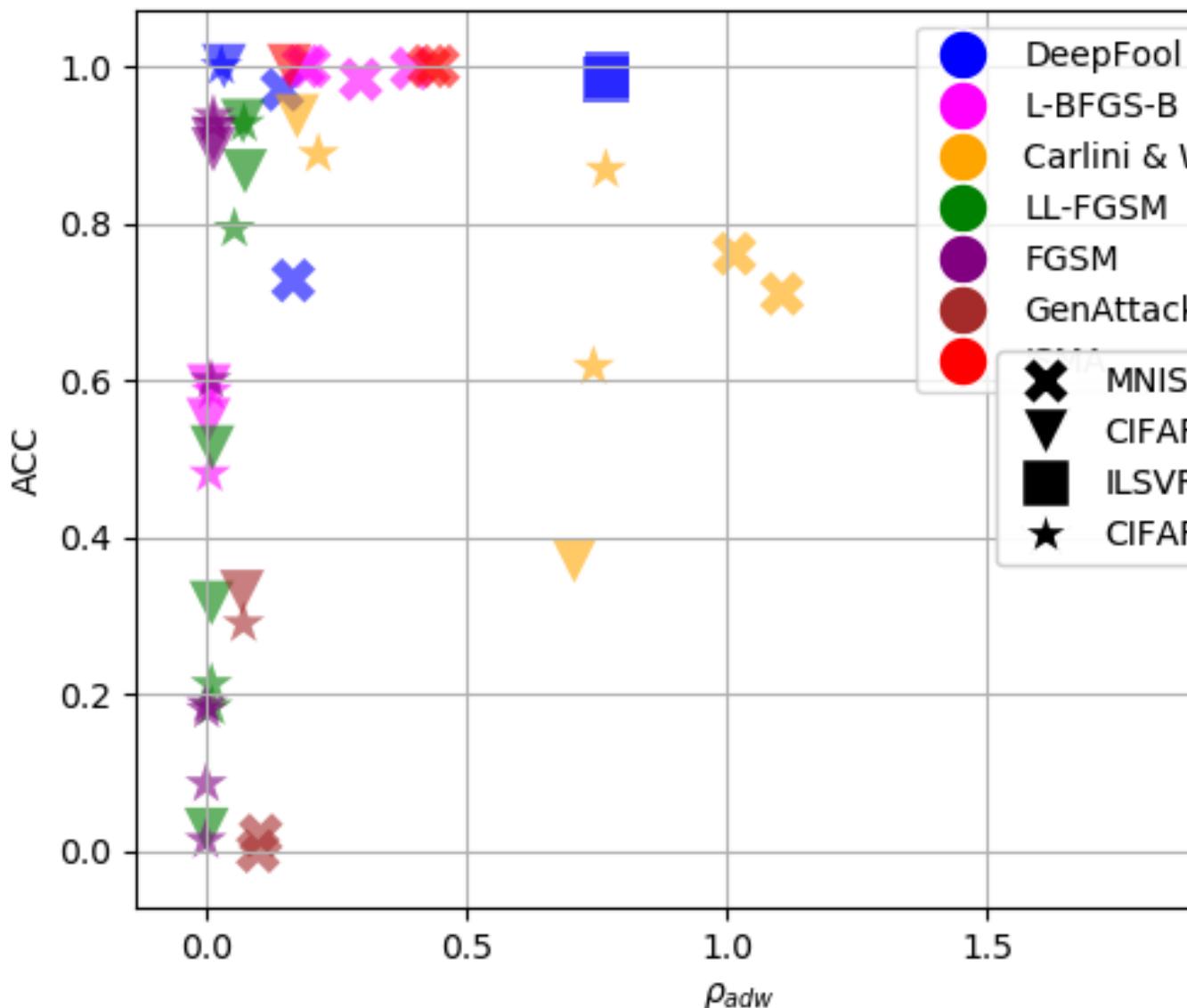
Budżet	Atak	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
10	FGSM	98.4%	23.04987	23.03577	0.03483	0.84004
	LL-FGSM	5.3%	0.55409	0.54851	0.02105	0.02009
100	FGSM	100.0%	1.99148	2.01564	0.04200	0.07334
	LL-FGSM	75.9%	1.84168	1.86557	0.02488	0.06819
	Carlini & Wagner	0.0%	19.94945	20.22396	0.03701	0.78153
1000	FGSM	100.0%	1.99628	2.00247	0.10326	0.07272
	Carlini & Wagner	86.8%	19.41824	19.80680	0.06234	0.76777
	LL-FGSM	79.3%	1.62712	1.49084	0.09101	0.05508
	L-BFGS-B	58.6%	0.29539	0.28461	5.38064	0.01091
10000	DeepFool	100.0%	0.94367	0.84174	0.10817	0.03006
	FGSM	100.0%	2.09150	2.08650	0.75895	0.07588
	LL-FGSM	92.7%	2.10414	1.99924	0.75182	0.07369
	Carlini & Wagner	61.7%	18.93897	19.23968	0.37505	0.74416
	L-BFGS-B	48.1%	0.00000	0.21332	10.14203	0.00797
	GenAttack	29.0%	1.97565	1.94919	10.52978	0.07251
100000	Carlini & Wagner	88.8%	4.72538	5.37915	3.49961	0.21601

Tabela 5.3. Wyniki eksperymentów dla zbioru CIFAR-100

Budżet	Atak	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
10	FGSM	98.9%	203.48324	209.98837	0.07811	0.98907
	LL-FGSM	13.5%	177.04699	181.14480	0.06357	0.83962
100	FGSM	100.0%	225.42040	231.13559	0.39487	0.79270
	LL-FGSM	98.9%	171.07919	173.13286	0.12599	0.79567
1000	LL-FGSM	100.0%	221.21964	226.77946	2.40135	0.77707
	FGSM	100.0%	221.05807	226.61763	2.44989	0.77650
10000	LL-FGSM	100.0%	220.24872	225.73414	23.98917	0.77385
	FGSM	100.0%	219.98650	225.56411	24.25147	0.77277
100000	DeepFool	98.9%	166.62105	167.95214	22.73689	0.77061

Tabela 5.4. Wyniki eksperymentów dla zbioru ILSVRC2012

Obserwując wyniki w tabelach 5.1 do 5.4 warto zwrócić uwagę na wyniki metody Carlini & Wagner. Niskiej jakości precyzaja tego ataku w niższych przedziałach budżetu jest wynikiem tego że do skutecznego działania metoda ta wymaga wyższych nakładów obliczeniowych. Z uwagi na ograniczone moce obliczeniowe część eksperymentów została wykonana przy mniejszej liczbie iteracji w procesie optymalizacji równania (15). W tabeli A.5 możemy zaobserwować że im wyższa jest liczba iteracji optymalizacji i tym lepsze są wyniki jeśli chodzi o precyzaję i wszechstronność ρ_{adw} . Dokładniejsze wyniki eksperymentów dla metody Carlini & Wagner, oraz innych, można zobaczyć w Załączniku A. Należy też zwrócić uwagę na to że tabela 5.4 nie zawiera wyników dla wszystkich dostępnych ataków. Jest to konsekwencja tego że złożoność obliczeniowa większości metod rośnie razem z wzrostem liczby atrybutów danych wejściowych. Dlatego też dla obrazów ze zbioru ILSVRC2012 wykonanie eksperymentów dla bardziej kosztownych metod byłoby zbyt czasochłonne w dostępnych warunkach. Ciekawy jest przypadek metody JSMA, która pomimo że z uwagi na swoją bardzo szybko rosnącą złożoność obliczeniową jest nieco niepraktyczna dla większych zbiorów, uzyskuje dla mniejszych obrazów bardzo dobre wyniki. Atak FGSM wykazuje swoją skuteczność w wszystkich zakresach budżetu jako że w żadnym jego precyzaja nie jest niższa niż 60%. LL-FGSM radzi sobie nieco gorzej w niższych budżetach - co jednak jest zrozumiałe z uwagi na większą trudność zadania jakim jest atak nakierunkowany. Wyjątkowy jest atak DeepFool który praktycznie zawsze osiąga około 100% precyzaji niezależnie od zbioru czy budżetu.



Rysunek 5.1. Porównanie skuteczności ataków dla kilku najlepszych wyników uzyskanych ze wszystkich używanych zbiorów danych.

Na rysunku 5.1 znajduje się porównanie kilku najlepszych wyników eksperymentów dla każdego z ataków zebranych z wyników otrzymanych dla wszystkich zbiorów danych. Osi poziomej odpowiada miara wszechstronności ρ_{adw} , natomiast osi pionowej precyza naszych ataków. Możemy zaobserwować że DeepFool balansuje obie te miary uzyskując przyzwoite wyniki. JSMA uzyskuje dobrą precyzję kosztem gorszej wszechstronności. Carlini & Wagner ma wyniki całej skali spektrum od bardzo dobrych w obu kryteriach do bardzo złych. L-BFGS-B i FGSM z wariantami bardzo dobrze radzi sobie z zachowa-

niem dobrej wszechstronności natomiast przy gorszych wynikach ogólnych jeśli chodzi o precyzję. GenAttack uzyskuje przeciętną precyzję przy nadal dość dobrej wszechstronności, możliwe że jest to konsekwencją tego że dokonywane dla tego ataku eksperymenty musiały zostać ograniczone z uwagi na nakłady obliczeniowe.

6. Podsumowanie

W toku tej pracy udało się skutecznie zaimplementować i opisać kilka popularnych metod tworzenia złośliwych danych dla modeli klasyfikacyjnych. Udało się także stworzyć prostą bibliotekę która pozwala na łatwe wykorzystanie wspomnianych ataków w celu testowania odporności modeli oraz tworzenia zbiorów danych zawierających przykłady adwersaryjne. Pokazano że udało się skutecznie i z dobrymi wynikami zastosować opisane ataki zarówno na prostych jak i na złożonych klasyfikatorach, które osiągają wysokie, bądź bardzo wysokie precyzyje w swoich zbiorach danych. Jednym z największych problemów biblioteki są problemy z jej wydajnością w przypadku próby utworzenia złośliwych przykładów dla całego zbioru danych. Wynika to głównie z nakładów obliczeniowych jakich wymagają niektóre z zaimplementowanych metod. Z pewnością można poprawić wydajność niektórych implementacji poprzez optymalizację kodu. Potencjalnymi kierunkami rozwoju są przede wszystkim rozszerzenie wachlarza dostępnych metod i wspieranych framework'ów, możliwość automatycznego generowania raportów oraz optymalizacja istniejącego kodu. Wartościowym dodatkiem byłoby także dodanie metod pozwalających na atakowanie modeli zajmujących się zadaniami innymi niż tylko klasyfikacja.

Literatura

- [1] F. Rosenblatt, “The perceptron, a perceiving and recognizing automaton : *project para, cornell aeronautical laboratory report*,” 1957.
- [2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.
- [4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [5] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, “Do imagenet classifiers generalize to imagenet?,” *CoRR*, vol. abs/1902.10811, 2019.
- [6] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, “Fixing the train-test resolution discrepancy: Fixefficientnet,” *ArXiv*, vol. abs/2003.08237, 2020.
- [7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
- [8] I. J. Goodfellow, N. Papernot, and P. D. McDaniel, “cleverhans v0.1: an adversarial machine learning library,” *CoRR*, vol. abs/1610.00768, 2016.
- [9] M. Nicolae, M. Sinn, T. N. Minh, A. Rawat, M. Wistuba, V. Zantedeschi, I. M. Molloy, and B. Edwards, “Adversarial robustness toolbox v0.2.2,” *CoRR*, vol. abs/1807.01069, 2018.
- [10] J. Rauber, W. Brendel, and M. Bethge, “ Foolbox: A python toolbox to benchmark the robustness of machine learning models,” in *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017.
- [11] P. Roy, S. Ghosh, S. Bhattacharya, and U. Pal, “Effects of degradations on deep neural network architectures,” *CoRR*, vol. abs/1807.10108, 2018.
- [12] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, “Robust physical-world attacks on machine learning models,” *CoRR*, vol. abs/1707.08945, 2017.
- [13] N. Morgulis, A. Kreines, S. Mendelowitz, and Y. Weisglass, “Fooling a real car with adversarial traffic signs,” *CoRR*, vol. abs/1907.00374, 2019.
- [14] C. Sitawarin, A. N. Bhagoji, A. Mosenia, P. Mittal, and M. Chiang, “Rogue signs: Deceiving traffic sign recognition with malicious ads and logos,” *CoRR*, vol. abs/1801.02780, 2018.
- [15] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” *CoRR*, vol. abs/1511.04508, 2015.
- [16] A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. A. Storer, “Deflecting adversarial attacks with pixel deflection,” *CoRR*, vol. abs/1801.08926, 2018.
- [17] A. Mustafa, S. H. Khan, M. Hayat, R. Goecke, J. Shen, and L. Shao, “Adversarial defense by restricting the hidden space of deep neural networks,” *CoRR*, vol. abs/1904.00887, 2019.

6. Literatura

- [18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *ArXiv e-prints*, Dec. 2014. <https://arxiv.org/abs/1412.6572>.
- [19] S. Bubeck, E. Price, and I. P. Razenshteyn, "Adversarial examples from computational constraints," *ArXiv*, vol. abs/1805.10204, 2019.
- [20] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," *CoRR*, vol. abs/1511.07528, 2015.
- [21] A. M. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *CoRR*, vol. abs/1412.1897, 2014.
- [22] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *CoRR*, vol. abs/1712.07107, 2017.
- [23] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. D. McDaniel, "Ensemble adversarial training: Attacks and defenses," *ArXiv*, vol. abs/1705.07204, 2018.
- [24] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. L. Yuille, "Mitigating adversarial effects through randomization," *CoRR*, vol. abs/1711.01991, 2017.
- [25] X. Liu, M. Cheng, H. Zhang, and C. Hsieh, "Towards robust neural networks via random self-ensemble," *CoRR*, vol. abs/1712.00673, 2017.
- [26] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *CoRR*, vol. abs/1602.02697, 2016.
- [27] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *CoRR*, vol. abs/1802.00420, 2018.
- [28] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.
- [29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016.
- [30] "Cntk website." <https://docs.microsoft.com/en-us/cognitive-toolkit/>. (Accessed: 2020-08-06).
- [31] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *CoRR*, vol. abs/1605.07277, 2016.
- [32] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *CoRR*, vol. abs/1607.02533, 2016.
- [33] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," *CoRR*, vol. abs/1511.04599, 2015.
- [34] R. Wiyatno and A. Xu, "Maximal jacobian-based saliency map attack," *CoRR*, vol. abs/1808.07945, 2018.
- [35] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," *CoRR*, vol. abs/1608.04644, 2016.

- [36] M. Alzantot, Y. Sharma, S. Chakraborty, and M. B. Srivastava, “Genattack: Practical black-box attacks with gradient-free optimization,” *CoRR*, vol. abs/1805.11090, 2018.
- [37] “Mnist website.” <http://yann.lecun.com/exdb/mnist/>. (Accessed: 2018-06-21).
- [38] “Tensorflow model garden.” Link (accessed: 20.06.2020).
- [39] V. Golovko, M. Egor, A. Brich, and A. Sachenko, “A shallow convolutional neural network for accurate handwritten digits classification,” vol. 673, pp. 77–85, 02 2017.
- [40] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.
- [41] S. H. HasanPour, M. Rouhani, M. Fayyaz, and M. Sabokrou, “Lets keep it simple, using simple architectures to outperform deeper and more complex architectures,” *CoRR*, vol. abs/1608.06037, 2016.
- [42] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *CoRR*, vol. abs/1605.07146, 2016.
- [43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *CoRR*, vol. abs/1512.00567, 2015.
- [44] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation,” *CoRR*, vol. abs/1801.04381, 2018.
- [45] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *CoRR*, vol. abs/1603.05027, 2016.
- [46] “Google ai blog post about mobilenetv2.” Link (Accessed: 2020-06-22).
- [47] W. Brendel, J. Rauber, A. Kurakin, N. Papernot, B. Veliqi, M. Salathé, S. P. Mohanty, and M. Bethge, “Adversarial vision challenge,” *CoRR*, vol. abs/1808.01976, 2018.

Wykaz symboli i skrótów

\mathbf{x} – wektor

\mathbf{x}' – spreparowany przez atak złośliwy przykład

\mathbf{r} – perturbacja wprowadzona do oryginalnego przykładu w celu uzyskania złośliwego przykładu $\mathbf{x}' = \mathbf{x} + \mathbf{r}$

\mathbf{x}_i – i -ty element wektora \mathbf{x}

\mathbf{x}^i – i -ty wektor

$f(\mathbf{x})$ – funkcja opisująca model klasyfikacyjny

$f_i(\mathbf{x})$ – i -ty element wektora, będącego wektorem wyjściowym modelu, opisującego prawdopodobieństwo należenia do danej klasy i

$q(\mathbf{x}, f)$ – atak q zastosowany na przykładzie \mathbf{x} i modelu opisanego funkcją f

X – zbiór danych dla którego $\mathbf{x} \in X$

y – prawdziwa klasa do której należy przykład \mathbf{x}

\tilde{y} – narzucona przez atakującego klasa do której ma należeć spreparowany przykład \mathbf{x}'

\hat{y} – klasa przypisana przykładowi wejściowemu przez model klasyfikacyjny taka że $\hat{y} = \operatorname{argmax} f(\mathbf{x})$

Spis rysunków

2.1	Przykład z <i>Intriguing properties of neural networks</i> [7]. W lewej kolumnie: poprawnie klasyfikowane oryginalne obrazy, w środkowej: dodana do obrazów perturbacja, w prawej: obrazy z perturbacją klasyfikowane jako struś	12
2.2	Ilustracja powyższego podziału ataków z oryginalnej publikacji [20]	14
4.1	Struktura Modelu LeNet5 [39]	26
4.2	Schemat modelu SimpleNet [41]	27
4.3	Schemat modelu InceptionV3	28
4.4	Schemat warstwy <i>residual bottleneck</i> [46]	29
5.1	Porównanie skuteczności ataków dla kilku najlepszych wyników uzyskanych ze wszystkich używanych zbiorów danych.	34
B.5	Przykłady wygenerowanych złośliwych przykładów z zadaną klasą za pomocą metody GenAttack	47
B.6	Przykłady wygenerowanych złośliwych za pomocą metody DeepFool	48
B.7	Przykłady wygenerowanych złośliwych przykładów z zadaną klasą za pomocą metody Carlini & Wagner dla parametrów $i = 1000$, $i_b = 10$, $\kappa = 0.0$	49
B.8	Przykłady wygenerowanych złośliwych przykładów z zadaną klasą za pomocą metody JSMA-F+ dla parametrów $\delta_{max} = 0.1$ i $\theta = 1.0$	50
B.1	Przykłady złośliwych przykładów wybranych na podstawie obrazów z różnych zbiorów za pomocą metody FGSM	51

B.2	Przykłady złośliwych przykładów wybranych na podstawie obrazów z różnych zbiorów za pomocą metody I-FGSM	52
B.3	Przykładowe złośliwe obrazy z zbioru ILSVRC2012 4.4 uzyskane za pomocą metody LL-FGSM z użyciem parametrów $\epsilon = 0.0005$ i $i = 1000$	53
B.4	Przykłady wygenerowanych złośliwych przykładów zadaną klasą za pomocą metody L-BFGS-B dla zbioru CIFAR-100	54
B.9	Przykładowe złośliwe obrazy z zbioru MNIST uzyskane za pomocą metody LL-FGSM	55
B.10	Przykładowe złośliwe obrazy z zbioru CIFAR-100 uzyskane za pomocą metody LL-FGSM	56
B.11	Przykładowe złośliwe obrazy z zbioru MNIST uzyskane za pomocą metody L-BFGS-B	57
B.12	Przykładowe złośliwe obrazy z zbioru CIFAR-10 uzyskane za pomocą metody L-BFGS-B	58

Spis tabel

3.1	Porównanie bibliotek generujących złośliwe przykłady.	17
4.1	Precyza top-1 i top-5 wykorzystywanych przez nas modeli	29
5.1	Wyniki eksperymentów dla zbioru MNIST	31
5.2	Wyniki eksperymentów dla zbioru CIFAR-10	32
5.3	Wyniki eksperymentów dla zbioru CIFAR-100	32
5.4	Wyniki eksperymentów dla zbioru ILSVRC2012	33
A.1	porównanie miar ataku FGSM względem różnych wartości parametru ϵ	42
A.2	porównanie miar ataku I-FGSM dla kilku różnych wartości i i ϵ	43
A.3	tabela z charakterystykami dla ataku LL-FGSM dla kilku różnych wartości i i ϵ	44
A.4	Wyniki ataku L-BFGS-B	45
A.5	porównanie miar ataku Carlini & Wagner dla różnych wartości parametrów .	45
A.6	porównanie miar ataku GenAttack dla różnych modeli	45
A.7	porównanie miar ataku DeepFool dla różnych modeli	46
A.8	Miary ataku JSMA+ dla różnych modeli	46

Spis załączników

1.	Szczegółowe Wyniki Ataków	42
2.	Przykłady złośliwych danych	47

A. Szczegółowe Wyniki Ataków

W dalszych podrozdziałach tego załącznika znajdują się tabele zawierające szczegółowe wyniki eksperymentów wraz z informacją o parametrach ataku które zostały użyte oraz ich krótkim omówieniem.

A.1. FGSM

Jedynym parametrem w klasycznym, jednokrokowym, wariantie metody FGSM z Tabeli A.2 jest ϵ który określa rozmiar zmiany wprowadzanej dla każdego atrybutu. W przypadku obrazów jest to zmiana wartości natężenia każdego kanału dla każdego piksela.

Model Name	$\epsilon = 0.0001 i = 1$					$\epsilon = 0.01 i = 1$				
	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	0.1%	0.00225	0.00224	0.00154	0.00025	0.2%	0.02247	0.02240	0.00154	0.00248
Le Net 5	0.0%	0.00217	0.00216	0.00120	0.00024	0.2%	0.02166	0.02159	0.00111	0.00239
SimpleNet CIFAR-10	0.7%	0.00554	0.00553	0.02605	0.00020	8.8%	0.05542	0.05528	0.02581	0.00199
SimpleNet CIFAR-100	1.4%	0.00554	0.00551	0.03488	0.00020	18.0%	0.05542	0.05509	0.03800	0.00201
InceptionV3	27.9%	218.26441	224.04303	0.23568	0.76732	31.1%	218.26463	224.04338	0.24685	0.76732
MobileNetV2	46.0%	166.60342	167.91239	0.07075	0.77042	50.5%	166.60364	167.91266	0.08051	0.77042
$\epsilon = 0.01 i = 1$						$\epsilon = 0.1 i = 1$				
Model Name	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	1.8%	0.22321	0.22225	0.00155	0.02464	41.4%	2.18788	2.18065	0.00159	0.24209
Le Net 5	1.5%	0.21388	0.21345	0.00119	0.02367	53.4%	2.08073	2.07861	0.00120	0.23100
SimpleNet CIFAR-10	66.7%	0.55418	0.55185	0.02498	0.01989	91.9%	5.49757	5.43992	0.02374	0.19611
SimpleNet CIFAR-100	78.3%	0.55409	0.54856	0.03854	0.02006	95.4%	5.46529	5.37468	0.03535	0.19639
InceptionV3	47.7%	218.28519	224.07878	0.24354	0.76746	64.9%	220.58579	227.33248	0.25220	0.78020
MobileNetV2	69.8%	166.62507	167.93928	0.07671	0.77056	89.3%	168.34473	170.37908	0.07770	0.78338
$\epsilon = 0.25 i = 1$						$\epsilon = 0.5 i = 1$				
Model Name	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	97.2%	5.41054	5.39365	0.00154	0.59896	100.0%	10.66839	10.62969	0.00155	1.18090
Le Net 5	89.3%	5.13290	5.12913	0.00105	0.57022	97.3%	10.08902	10.08713	0.00112	1.12182
SimpleNet CIFAR-10	90.2%	13.18794	13.08371	0.02600	0.47150	90.5%	23.36630	23.34816	0.02584	0.84121
SimpleNet CIFAR-100	98.2%	13.02637	12.86764	0.03395	0.46951	98.4%	23.04987	23.03577	0.03483	0.84004
InceptionV3	76.9%	234.10222	241.77968	0.25685	0.83663	86.2%	270.17740	280.55890	0.24818	0.98710
MobileNetV2	97.1%	177.03642	181.14445	0.07809	0.83962	98.9%	203.48324	209.98837	0.07811	0.98907

Tabela A.1. porównanie miar ataku FGSM względem różnych wartości parametru ϵ

A.2. I-FGSM

W metodzie I-FGSM 1 z Tabeli A.2 oprócz parametru ϵ kontrolujemy także liczbą iteracji i w których modyfikujemy oryginalny obraz.

Model Name	$\epsilon = 0.0001 i = 10$					$\epsilon = 0.001 i = 10$				
	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	0.2%	0.02137	0.02129	0.00181	0.00236	1.8%	0.20952	0.20867	0.00180	0.02310
Le Net 5	0.2%	0.02120	0.02113	0.00142	0.00234	1.5%	0.20735	0.20715	0.00122	0.02297
SimpleNet CIFAR-10	9.5%	0.05246	0.05210	0.03123	0.00188	89.4%	0.41481	0.41181	0.03284	0.01483
SimpleNet CIFAR-100	18.7%	0.05178	0.05148	0.04069	0.00188	92.7%	0.40623	0.40291	0.04189	0.01463
InceptionV3	64.2%	218.38348	224.15330	0.37447	0.76771	98.1%	219.06059	224.81292	0.38499	0.77007
MobileNetV2	74.7%	166.70163	168.01269	0.13831	0.77090	99.3%	167.25426	168.58729	0.13913	0.77364
$\epsilon = 0.01 i = 10$					$\epsilon = 0.0001 i = 100$					
Model Name	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	48.8%	1.99013	1.98319	0.00180	0.22011	1.8%	0.21044	0.20945	0.00404	0.02319
Le Net 5	63.4%	1.86275	1.85770	0.00137	0.20514	1.5%	0.20653	0.20638	0.00309	0.02288
SimpleNet CIFAR-10	100.0%	2.20887	2.22380	0.03305	0.07995	90.7%	0.41636	0.41458	0.09455	0.01492
SimpleNet CIFAR-100	100.0%	1.99148	2.01564	0.04200	0.07334	93.7%	0.40602	0.40380	0.10437	0.01467
InceptionV3	100.0%	225.42040	231.13559	0.39487	0.79270	99.1%	219.01709	224.71439	2.40506	0.76973
MobileNetV2	100.0%	171.32952	173.19573	0.13591	0.79570	99.3%	167.33197	168.64840	0.90278	0.77393
$\epsilon = 0.001 i = 100$					$\epsilon = 0.0001 i = 1000$					
Model Name	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	49.3%	1.99075	1.98086	0.00407	0.21984	49.3%	1.99250	1.98335	0.03095	0.22010
Le Net 5	64.2%	1.85295	1.84908	0.00301	0.20428	64.5%	1.85179	1.84821	0.02314	0.20419
SimpleNet CIFAR-10	100.0%	2.40864	2.38752	0.09468	0.08581	100.0%	2.61824	2.57203	0.75909	0.09246
SimpleNet CIFAR-100	100.0%	1.99628	2.00247	0.10326	0.07272	100.0%	2.09150	2.08650	0.75895	0.07588
InceptionV3	100.0%	221.05807	226.61763	2.44989	0.77650	100.0%	219.98650	225.56411	24.25147	0.77277
MobileNetV2	100.0%	168.59136	170.10984	0.90471	0.78088	100.0%	168.01525	169.54835	8.88230	0.77819

Tabela A.2. porównanie miar ataku I-FGSM dla kilku różnych wartości i i ϵ

A.4. L-BFGS-B

Jedynym parametrem metody L-BFGS-B z Tabeli A.4 jest liczba iteracji i optymalizacji równania (9) za pomocą algorytmu L-BFGS-B.

Model Name	$i = 100$					$i = 1000$				
	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	99.7%	3.53984	3.49476	0.81560	0.39524	98.3%	2.66997	2.70581	1.02502	0.29735
Le Net 5	100.0%	1.70078	1.75999	0.39974	0.18722	100.0%	1.87018	1.85254	0.39557	0.19865
SimpleNet CIFAR-10	59.2%	0.19474	0.16874	5.42402	0.00621	54.9%	0.13877	0.14815	8.25832	0.00546
SimpleNet CIFAR-100	58.6%	0.29539	0.28461	5.38064	0.01091	48.1%	0.00000	0.21332	10.14203	0.00797

Tabela A.4. Wyniki ataku L-BFGS-B

A.5. Carlini & Wagner

W Tabeli A.5 i oznacza liczbę iteracji optymalizacji równania (15), natomiast i_b to liczba kroków wykonana wyszukiwania binarnego w celu znalezienia parametru c z równania (15).

Model Name	$i = 1 i_b = 10 \kappa = 0.0$					$i = 10 i_b = 10 \kappa = 0.0$				
	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	0.0%	18.82104	18.82911	0.01011	2.13603	0.0%	18.72053	18.72017	0.01603	2.08537
Le Net 5	0.0%	18.79607	18.78643	0.00904	2.10931	0.0%	18.67157	18.64628	0.01506	2.09228
SimpleNet CIFAR-10	0.0%	19.66507	19.83514	0.03554	0.75201	0.0%	19.62749	19.81175	0.07000	0.75111
SimpleNet CIFAR-100	0.0%	19.94945	20.22396	0.03701	0.78153	86.8%	19.41824	19.80680	0.06234	0.76777
Model Name	$i = 100 i_b = 10 \kappa = 0.0$					$i = 1000 i_b = 10 \kappa = 0.0$				
	ACC	median L_2	\bar{L}_2	T	ρ_{adw}	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	0.0%	18.01404	18.01498	0.05206	2.01053	71.0%	9.99485	9.93737	0.40858	1.10580
Le Net 5	0.0%	18.20825	18.18300	0.03873	2.04379	76.1%	8.84674	9.03522	0.29088	1.01435
SimpleNet CIFAR-10	37.0%	18.50197	18.65609	0.40281	0.70730	93.6%	4.26353	4.51120	3.72238	0.17541
SimpleNet CIFAR-100	61.7%	18.93897	19.23968	0.37505	0.74416	88.8%	4.72538	5.37915	3.49961	0.21601

Tabela A.5. porównanie miar ataku Carlini & Wagner dla różnych wartości parametrów

A.6. GenAttack

Atak GenAttack z Tabeli A.6 posiada 4 parametry które kontrolują jego zachowanie. Jako że atak wykorzystuje techniki znane z algorytmów genetycznych jesteśmy w stanie kontrolować rozmiar populacji N , liczbę generacji i , szanse na mutacje danego osobnika α oraz siłę mutacji δ .

Model Name	$i = 1000 N = 5 \delta = 0.05 \alpha = 0.05$				
	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	0.0%	0.93798	0.93652	8.34605	0.10023
Le Net 5	2.0%	0.93940	0.93640	8.24402	0.10567
SimpleNet CIFAR-10	33.0%	1.96935	1.94352	10.27956	0.06963
SimpleNet CIFAR-100	29.0%	1.97565	1.94919	10.52978	0.07251

Tabela A.6. porównanie miar ataku GenAttack dla różnych modeli

A.7. DeepFool

W ataku DeepFool z Tabeli A.7 jedynym tak naprawdę kontrolowanym przez nas parametrem jest maksymalna liczba iteracji jaką wykona atak jeśli nie uda mu się utworzyć złośliwego przykładu obrazu.

	$i = 100$				
Model Name	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	72.7%	1.54310	1.55501	0.42224	0.16790
Le Net 5	97.0%	1.27814	1.37602	0.38435	0.14677
SimpleNet CIFAR-10	100.0%	0.99746	0.98337	0.03880	0.03523
SimpleNet CIFAR-100	100.0%	0.94367	0.84174	0.10817	0.03006
InceptionV3	98.3%	218.32363	224.14897	88.07846	0.76770
MobileNetV2	98.9%	166.62105	167.95214	22.73689	0.77061

Tabela A.7. porównanie miar ataku DeepFool dla różnych modeli

A.8. JSMA

W metodzie JSMA z Tabeli A.8 parametrami które kontrolujemy jest maksymalna perturbacja δ_{max} , która tak naprawdę określa liczbę iteracji jako że zmiana wartości jest stała w każdej iteracji, oraz zmiana w intensywności pikseli jaką jest dokonywana w każdym kroku θ .

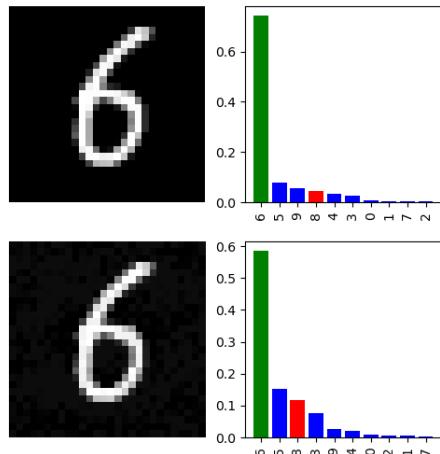
	$\epsilon_{max} = 0.1 \theta = 1$				
Model Name	ACC	median L_2	\bar{L}_2	T	ρ_{adw}
MNIST TF Model	100.0%	4.02858	4.02593	0.77459	0.44574
Le Net 5	100.0%	3.88321	3.92543	0.72626	0.42820
SimpleNet CIFAR-10	100.0%	4.31146	4.32629	38.52092	0.16045

Tabela A.8. Miary ataku JSMA+ dla różnych modeli

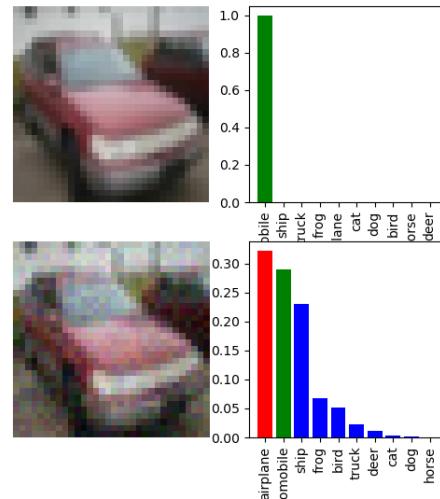
B. Przykłady złośliwych danych

W tym załączniku znajdują się przykładowe złośliwe dane wygenerowane przez różne ataki dostępne w opisanej wcześniej aplikacji.

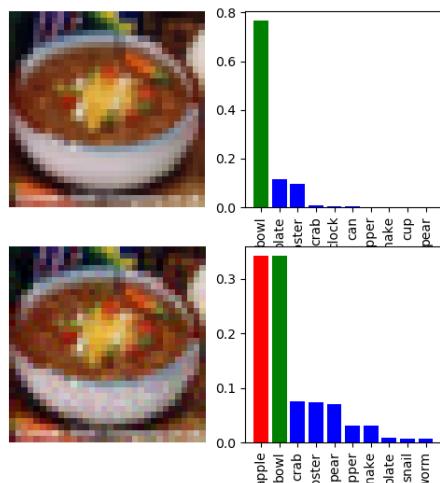
Rysunek B.5. Przykłady wygenerowanych złośliwych przykładów z zadaną klasą za pomocą metody GenAttack



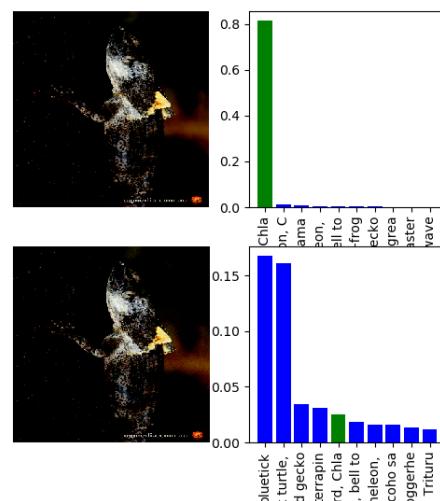
(a) MNIST TF



(b) SimpleNet CIFAR-10

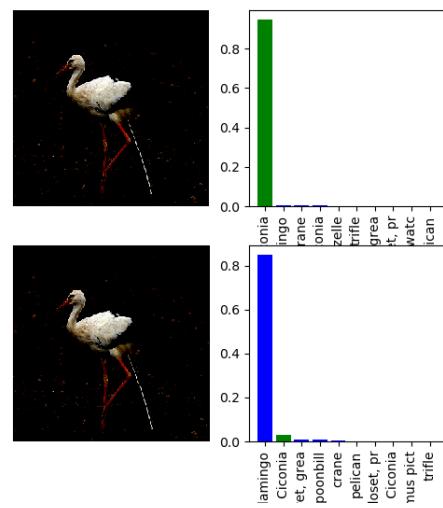
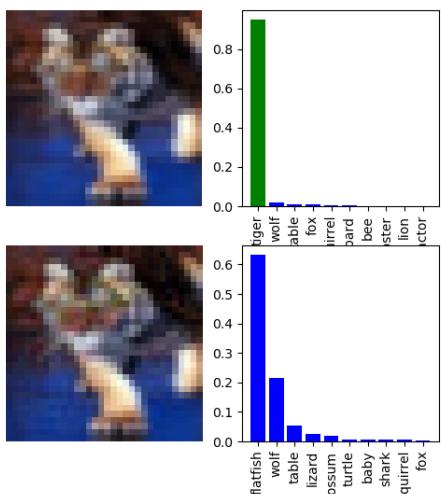
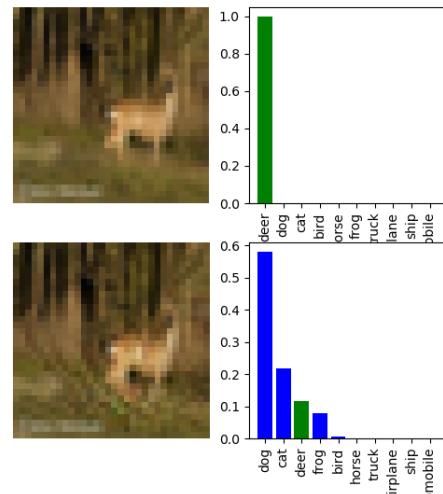
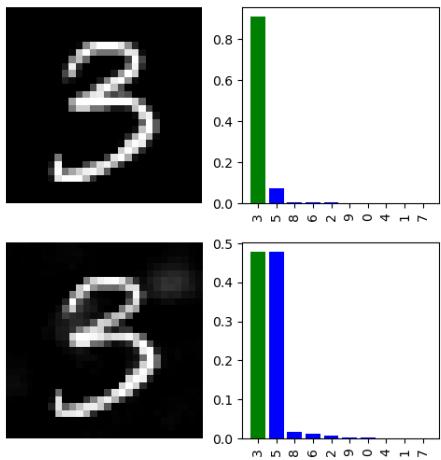


(c) SimpleNet CIFAR-100

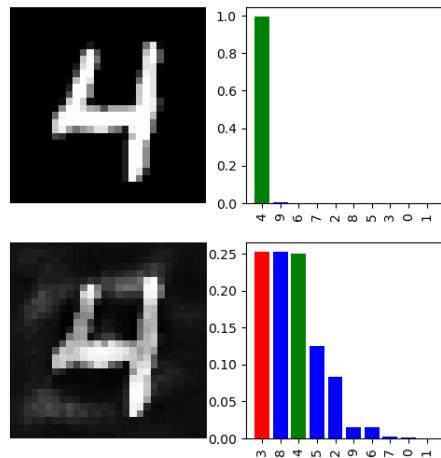


(d) InceptionV3

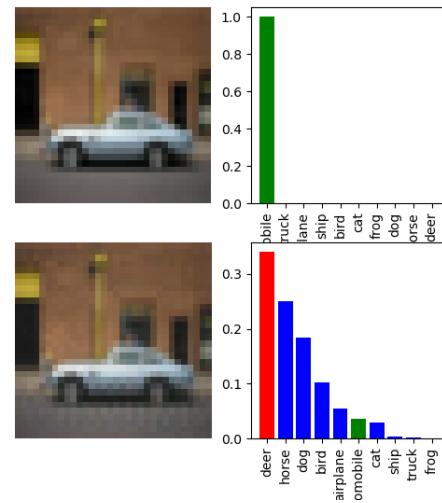
Rysunek B.6. Przykłady wygenerowanych złośliwych za pomocą metody DeepFool



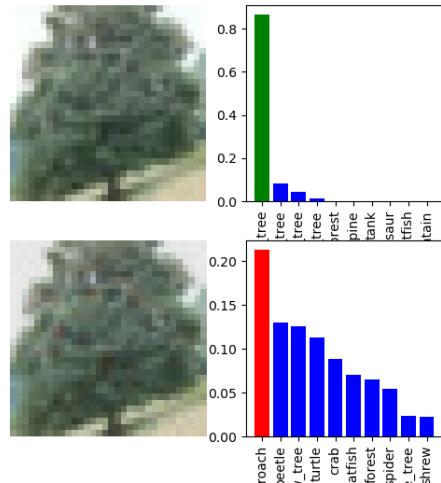
Rysunek B.7. Przykłady wygenerowanych złośliwych przykładów z zadaną klasą za pomocą metody Carlini & Wagner dla parametrów $i = 1000$, $i_b = 10$, $\kappa = 0.0$



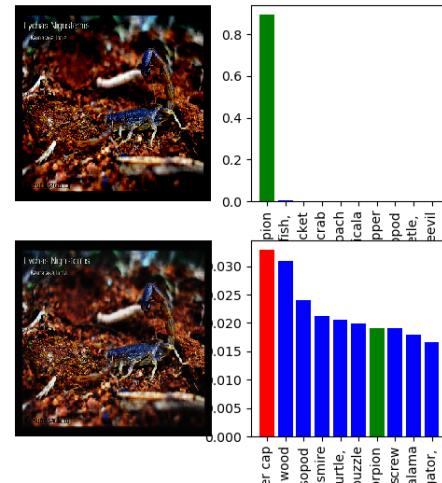
(a) MNIST TF



(b) SimpleNet CIFAR-10

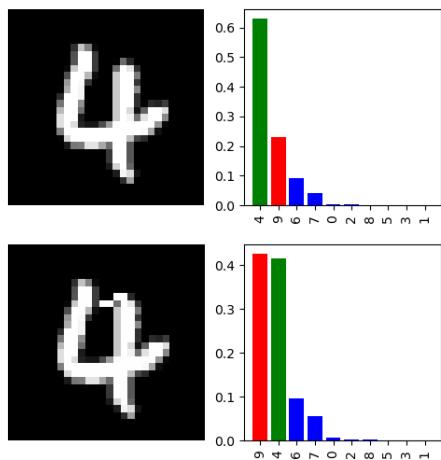


(c) SimpleNet CIFAR-100

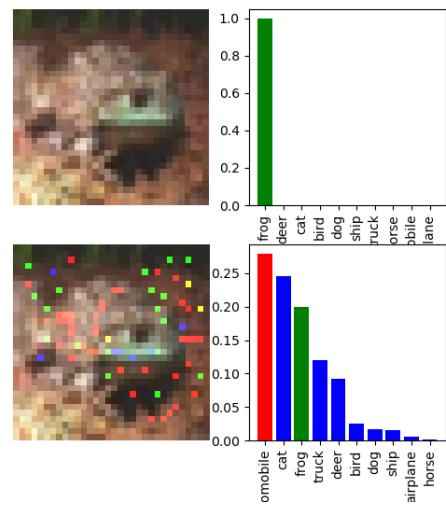


(d) InceptionV3

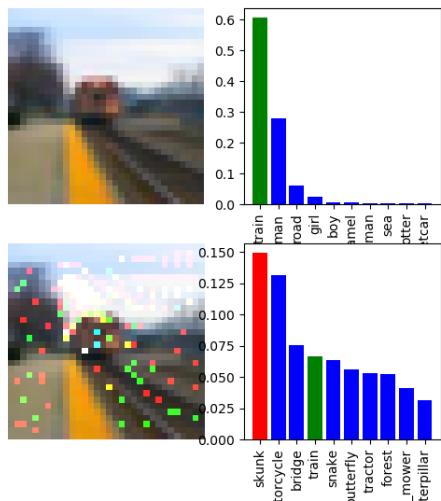
Rysunek B.8. Przykłady wygenerowanych złośliwych przykładów zadaną klasą za pomocą metody JSMA-F+ dla parametrów $\delta_{max} = 0.1$ i $\theta = 1.0$



(a) MNIST TF

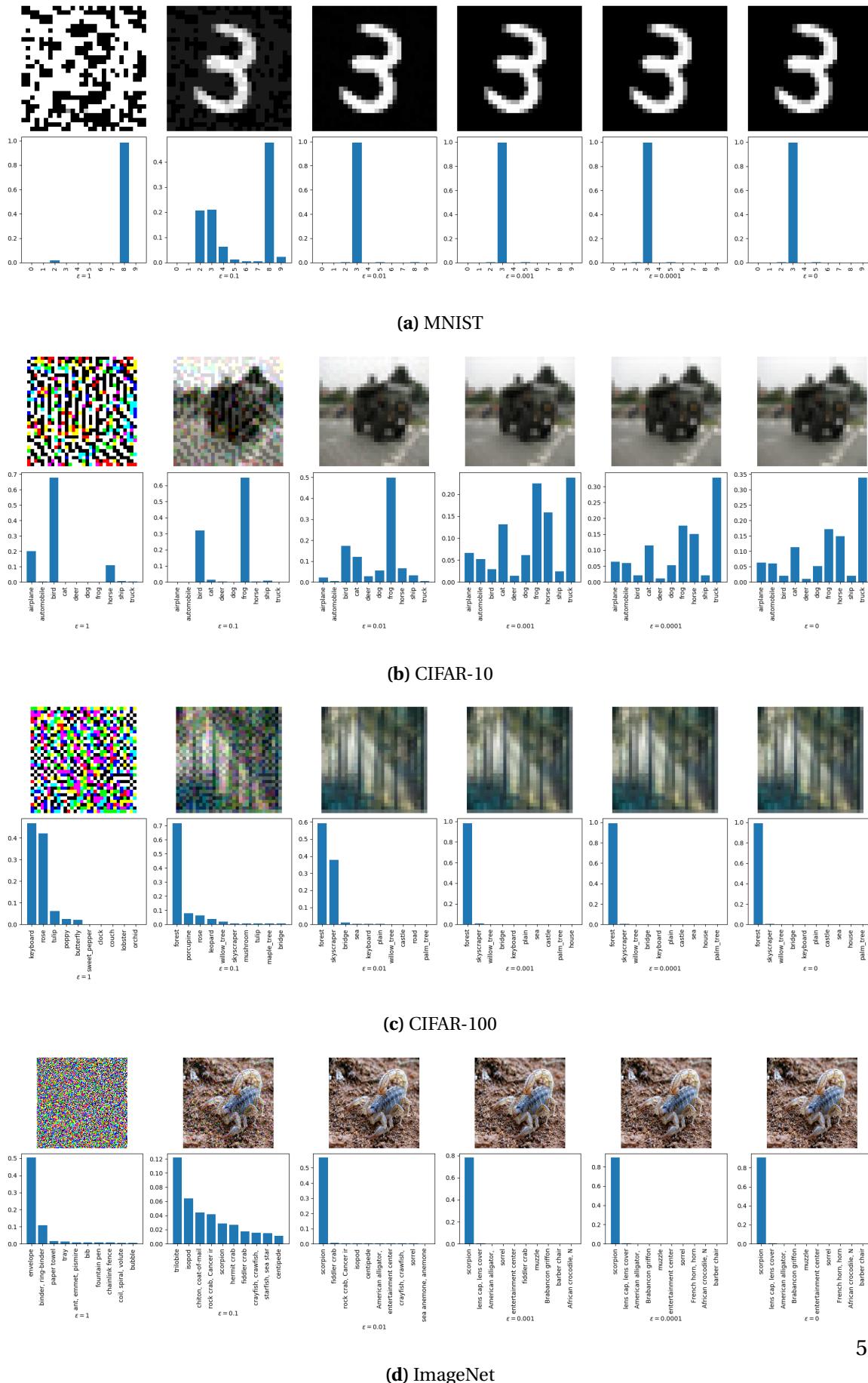


(b) SimpleNet CIFAR-10

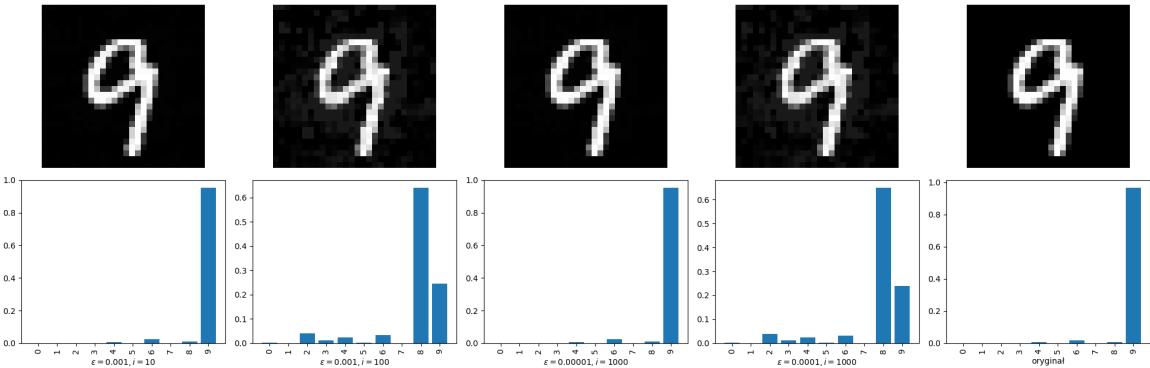


(c) SimpleNet CIFAR-100

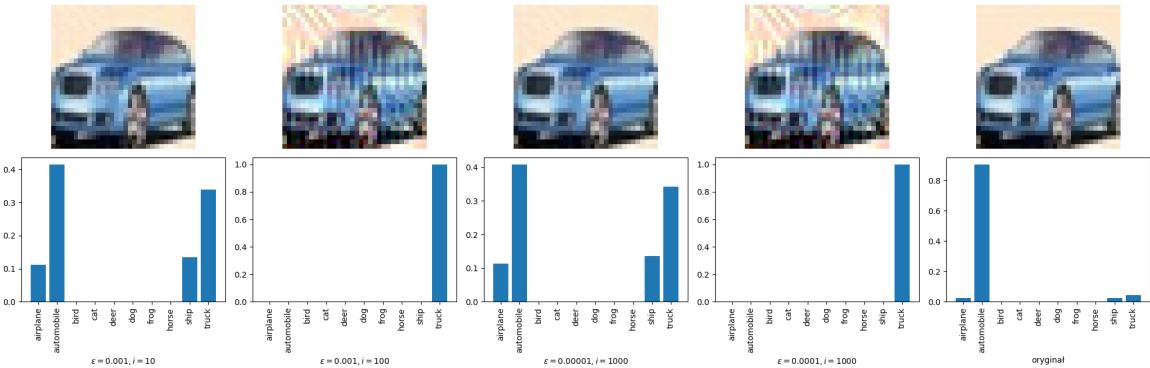
Rysunek B.1. Przykłady złośliwych przykładów wybranych na podstawie obrazów z różnych zbiorów za pomocą metody FGSM



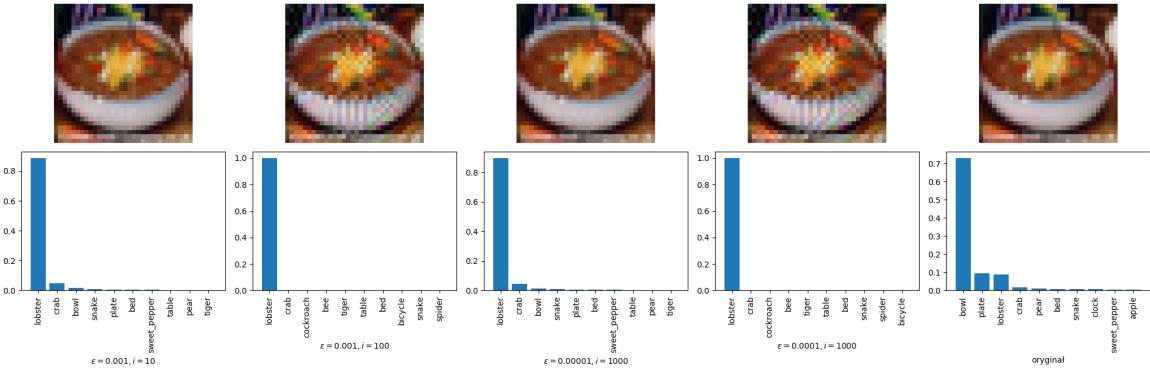
Rysunek B.2. Przykłady złośliwych przykładów wybranych na podstawie obrazów z różnych zbiorów za pomocą metody I-FGSM



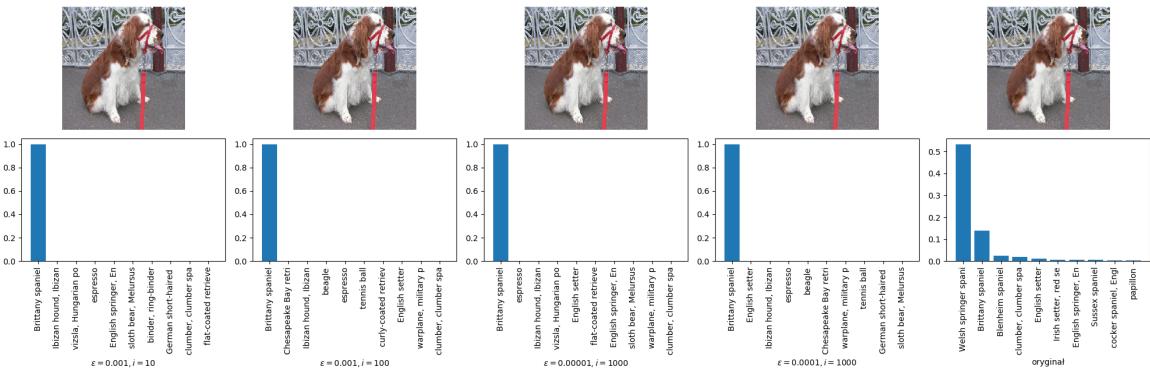
(a) MNIST



(b) CIFAR-10



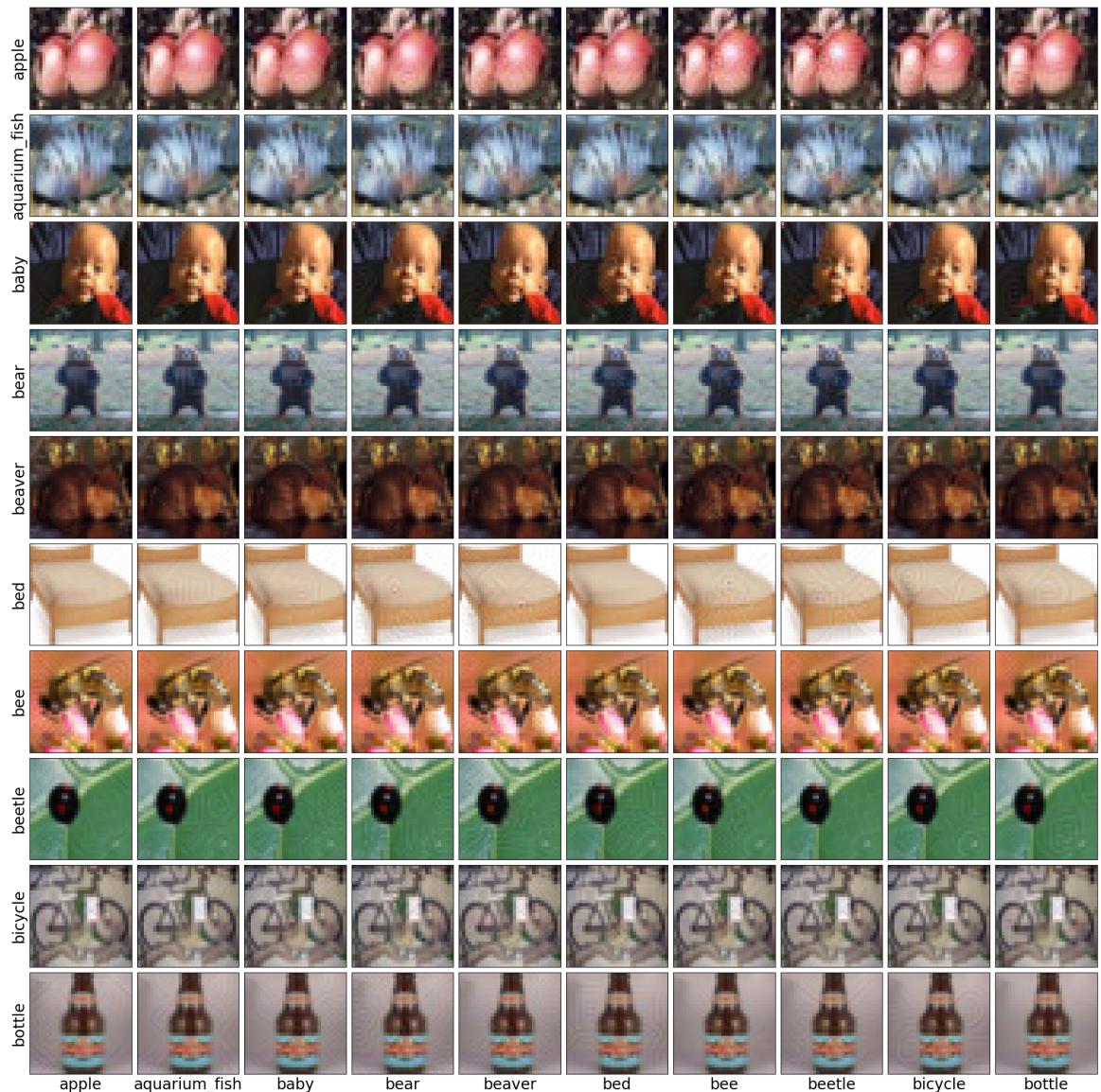
(c) CIFAR-100



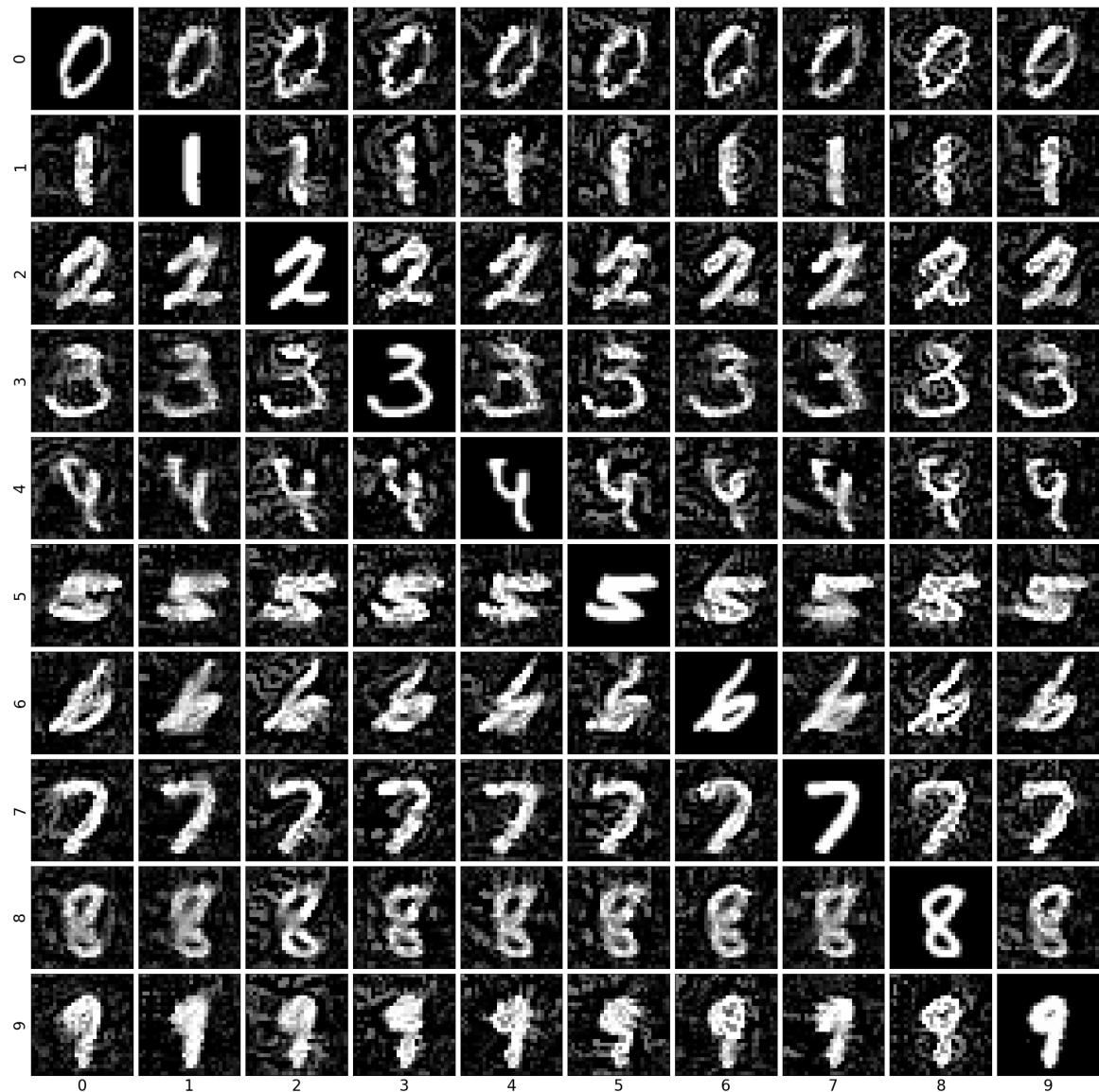
(d) ImageNet



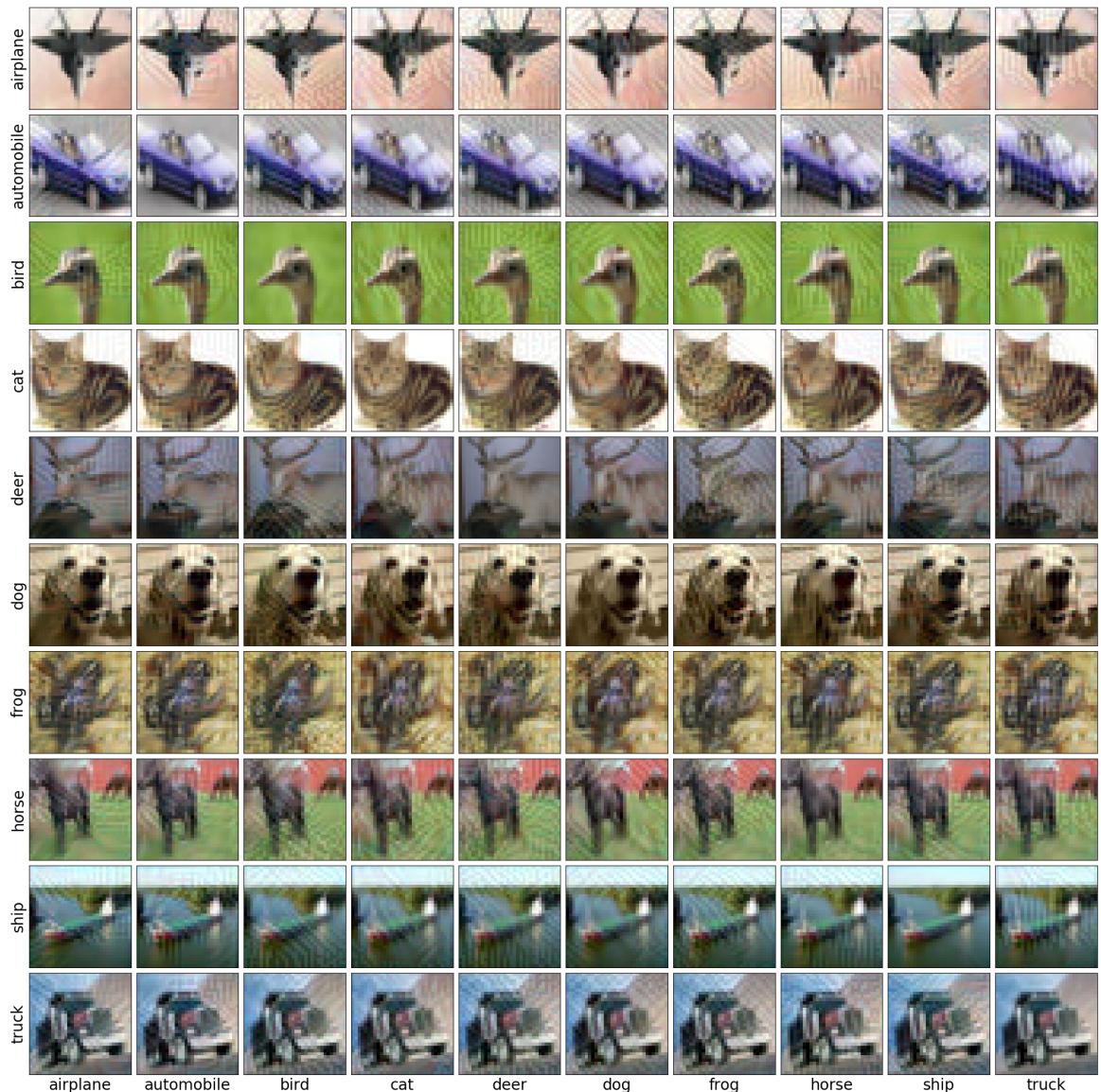
Rysunek B.3. Przykładowe złośliwe obrazy z zbioru ILSVRC2012 4.4 uzyskane za pomocą metody LL-FGSM z użyciem parametrów $\epsilon = 0.0005$ i $i = 1000$



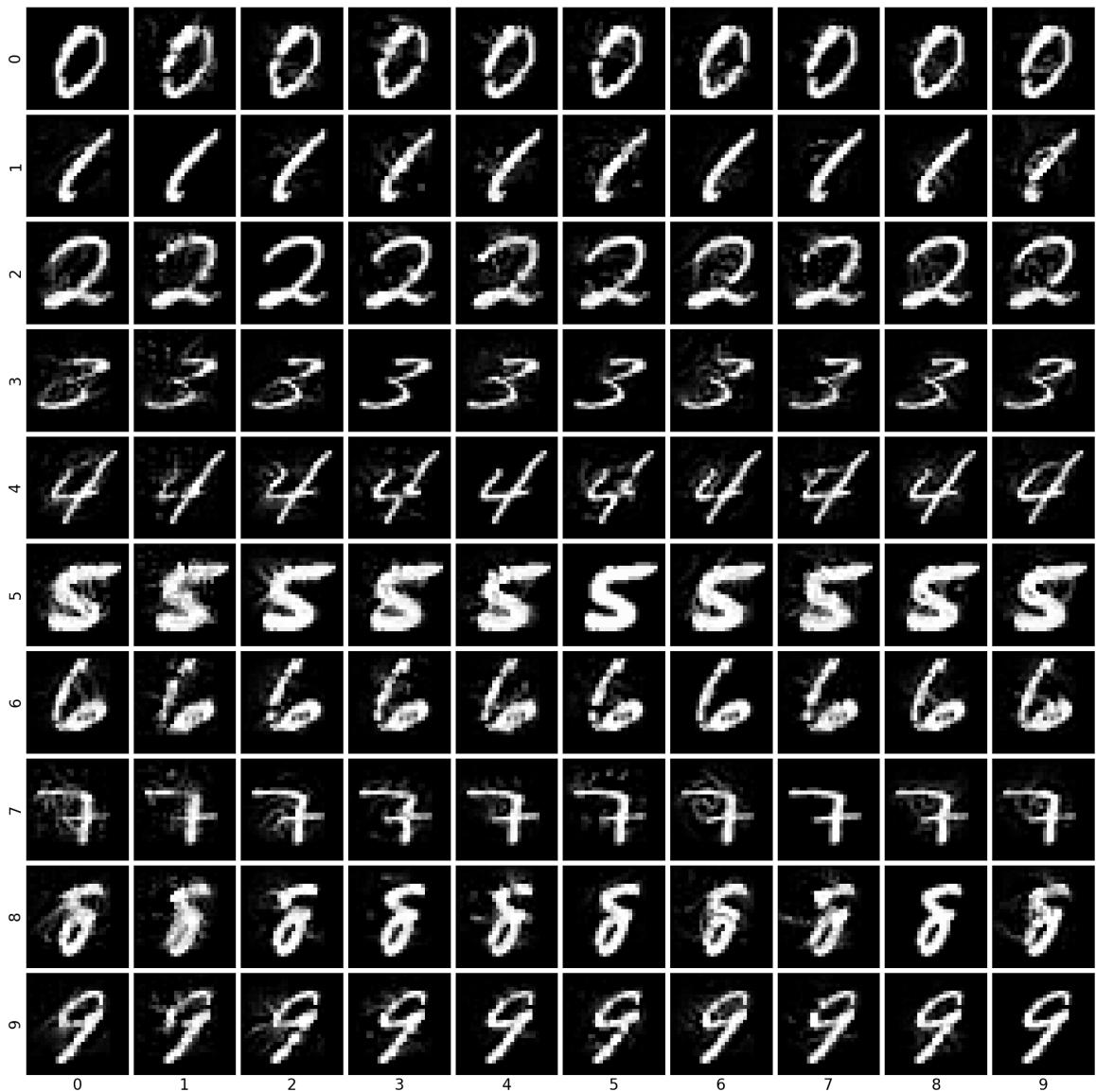
Rysunek B.4. Przykłady wygenerowanych złośliwych przykładów zadaną klasą za pomocą metody L-BFGS-B dla zbioru CIFAR-100



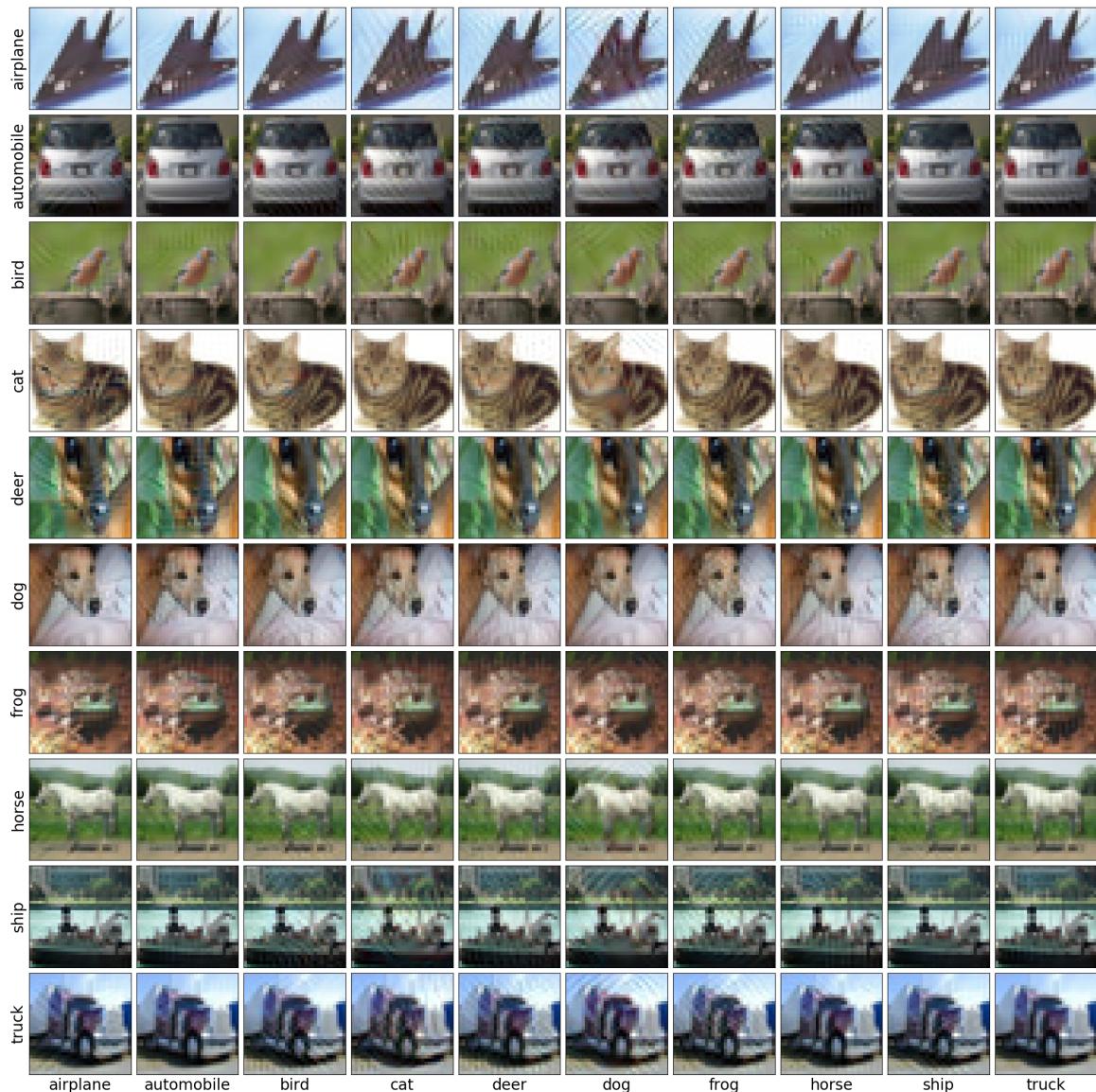
Rysunek B.9. Przykładowe złośliwe obrazy z zbioru MNIST uzyskane za pomocą metody LL-FGSM



Rysunek B.10. Przykładowe złośliwe obrazy z zbioru CIFAR-100 uzyskane za pomocą metody LL-FGSM



Rysunek B.11. Przykładowe złośliwe obrazy z zbioru MNIST uzyskane za pomocą metody L-BFGS-B



Rysunek B.12. Przykładowe złośliwe obrazy z zbioru CIFAR-10 uzyskane za pomocą metody L-BFGS-B