

Всеукраїнський конкурс студентських наукових робіт
з природничих, технічних та гуманітарних наук

Спеціальність: Інженерія програмного забезпечення

Конкурсна робота

Шифр роботи «Цикада»

**Програмне забезпечення захисту авторських прав на зображення текстових
матеріалів шляхом керованого шуму**

2020-2021 навчальний рік

Зміст

Анотація	3
Вступ	4
Аналіз стану питання	6
1.1. Аналіз стану в прикладній галузі.	6
1.2 Класифікація за наявністю цілі	10
1.3 Види захисту	10
1.4 Класифікація за доступністю внутрішніх даних	11
1.5 Постановка задачі	12
3. Дослідження	14
3.1 Перша спроба	14
3.2 MNIST	16
3.3 Успішна модель	18
3.4 Результат роботи	19
Висновки	24
Література	25
Додатки	25
Додаток А	25
attack.py	25
genattack_tf2.py	27
img_ocr.py	33
img.py	37
main.py	39
text_item.py	40

Анотація

Штучні нейронні мережі дозволили виконувати задачі, які ще кілька десятків років тому здавались недосяжними. Обчислення невідомих значень на основі мільйонів параметрів, конвертація голосу в текст та навпаки, розпізнавання тексту на зображенні тощо. З розвитком алгоритмів з'являються задачі захисту інформації від них. **Актуальність** роботи полягає в тому, що в наш час, коли більша частина інтернету сканується та індексується щодня, важливо не дати знайти та розпізнати конфіденційну інформацію на фотографіях, перетворивши її на текст. Також часом є потреба захистити окремі ключові слова на зображеннях. Утім, наразі не існує доступних застосунків для захисту тексту на зображеннях від розпізнавання, хоча потреба, очевидно, існує. Звідси витікає **мета роботи**: захистити текстові дані на зображенні від програм-розпізнавачів та **завдання**: створення інструменту для захисту інформації на зображеннях від систем розпізнавання тексту. Для розв'язання проблеми було використано генетичні алгоритми, що дозволило відмовитись від потреби мати доступ до внутрішніх станів розпізнавачів.

Робота обсягом 41 аркуш складається із вступу, 3 розділів, висновку, переліку посилань із 12 джерел. Містить 20 рисунків.

Ключові слова: генетичний алгоритм, змагальні атаки, нейронні мережі, класифікація зображень, атаки на нейронні мережі, алгоритми захисту.

Вступ

При швидкому розвитку методів штучного інтелекту (ШІ) та методів глибокого навчання (ГН) надзвичайно важливо гарантувати безпеку та надійність розгорнутих алгоритмів. Останнім часом широке визнання отримали вразливості безпеки алгоритмів ГН для змінених зразків, що отримало називу "змагальні атаки". Сформовані зразки можуть привести до різної поведінки моделей ГН, одночасно сприймаючись людиною як добрякісні. Успішні реалізації змагальних атак у реальних сценаріях фізичного світу ще більше демонструють їхню практичність. Отже, методи боротьби з атакою та оборону привертали все більшу увагу як з боку машинного навчання, так і з боку служб безпеки та стали актуальною темою дослідження в останні роки.

У цій роботі було використано генетичні алгоритми для видозмінення даних з метою унеможливлення розпізнавання тексту за допомогою нейронних мереж.

Об'єкт досліджень — змагальні атаки

Предмет досліджень — використання змагальних атак для захисту зображень з текстом

Завдання дослідження, які дозволяють досягти поставленої мети

1. Аналіз досліджень та розробок за темою роботи.
2. Порівняння роботи різних методів та вибір найкращого.
3. Реалізація або пошук та впровадження методу, що показав найкращий результат.
4. Створення графічного застосунку для використання методу
5. Тестування застосунку.

При виконанні досліджень та прийнятті рішень використовувались такі **методи та підходи:** генетичний алгоритм, змагальні атаки.

Наукова новизна:

1. Обрано та покращено наявний підхід до змагальних атак на моделі без доступу до внутрішнього стану за допомогою генетичних алгоритмів,

використано для нової задачі (приховування тексту). Підібрано оптимальні гіперпараметри.

Практична цінність роботи: Можливо захистити наявне зображення від конкретних розпізнавачів за допомогою шуму таким чином, щоб людина могла прочитати написане. Створено робочу програму, що дозволяє зручно використати модель.

Стислий опис результатів дослідження:

Результатом цієї роботи є метод, що використовує генетичні алгоритми для змагальних атак на системи з закритим внутрішнім станом, та модель, що здатна приховувати текст на зображеннях від програм-розпізнавачів шляхом додавання шуму, який не перешкоджає сприйняттю його людиною. У межах цієї роботи були проведені дослідження щодо впливу параметрів на результат і швидкодію та використано ті, що дозволяють досягти найкращих результатів. У результаті створена модель дозволяє захистити наявне зображення від конкретних розпізнавачів за допомогою шуму, не приховуючи зміст від людини.

1. Аналіз стану питання

1.1. Аналіз стану в прикладній галузі.

Як багато хто з вас може знати, ГН — це дуже експресивні мережі машинного навчання, які існують вже багато десятиліть. У 2012 році, заручившись обчислювальною потужністю та вдосконаленими інструментами, сімейство цих моделей машинного навчання під назвою ConvNets почало досягати найкращих результатів щодо завдань візуального розпізнавання. До цього моменту алгоритми машинного навчання просто не працювали достатньо добре, щоб хтось здивувався, коли вони не могли зробити належну справу. У 2014 році група дослідників Google та NYU виявила, що обдурити ConvNets надзвичайно просто нечутним, але ретельно побудованим впливом на вхідні дані [1].

Подивімося на приклад (Рис. 1). Ми починаємо з зображення панди, яку наша нейронна мережа правильно розпізнає як "панду" з упевненістю 57,7%. Додайте трохи ретельно сконструйованого шуму — і та ж нейронна мережа тепер думає, що це зображення гібону з 99,3% упевненості! Це, очевидно, оптична ілюзія — але для нейронної мережі.

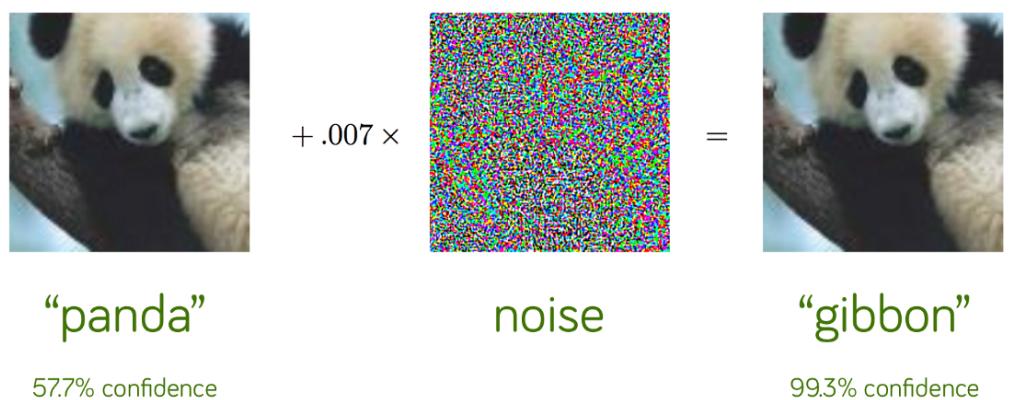


Рис. 1 — приклад змагальної атаки

Приклад гібона для порівняння (Рис. 2).



Рис. 2 — гібон

Ми з вами можемо чітко сказати, що обидва зображення мають вигляд панди; насправді, ми навіть не можемо сказати, що в початкове зображення було додано якийсь шум для побудови штучного прикладу праворуч!

У 2017 році інша група продемонструвала, що ці штучні приклади можна узагальнити до реального світу, показавши, що, надруковані зображення, створені для обману нейронної мережі, будуть продовжувати дурити нейронні мережі під різним освітленням та орієнтацією (Рис 3) [2].



Рис. 3 — приклад змагальної атаки

Ще одна цікава робота під назвою "Приєднатись до злочину: реальні та неприховані напади на сучасне розпізнавання обличчя" показала, що можна обдурити програмне забезпечення для розпізнавання обличчя, побудувавши окуляри шляхом цілковитої відсутності розпізнавання обличчя (Рис. 4) [3].

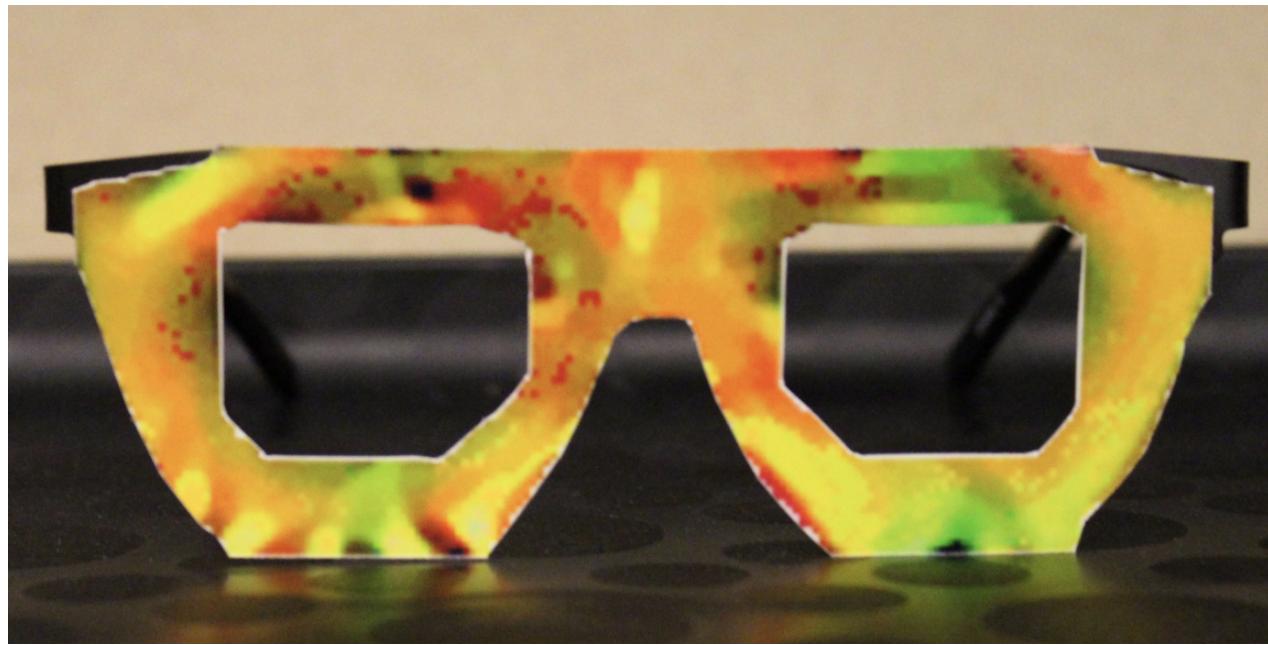


Рис. 4 — окуляри, що запобігають розпізнаванню обличчя

Ці окуляри також можуть дозволити видати себе за когось іншого.

Незабаром інша дослідницька група продемонструвала різні методи обдурення моделей, що розпізнають знаки зупинки, клеячи різні наклейки на нього.

Наклейки були розроблені для імітації графіті й, таким чином, "ховалися в людській психіці" (Рис 5) [4].



Рис. 5 — приклад наліпки, що допомагає обманути нейронні мережі

На лівому зображені зображені справжні графіті на знаку "Стоп" — те, що більшість людей не вважає підозрілим.

На правому зображені зображені наклейки, що обманюють моделі.

Системи класифікують знак праворуч як знак обмеження швидкості: 45 миль / год!

Документ "Adversarial Patch", опублікований у NIPS 2017, продемонстрував, як згенерувати стікер, який можна розмістити в будь-якому місці в полі зору класифікатора та викликати класифікатор для виведення цільового класу (Рис 6) [5].

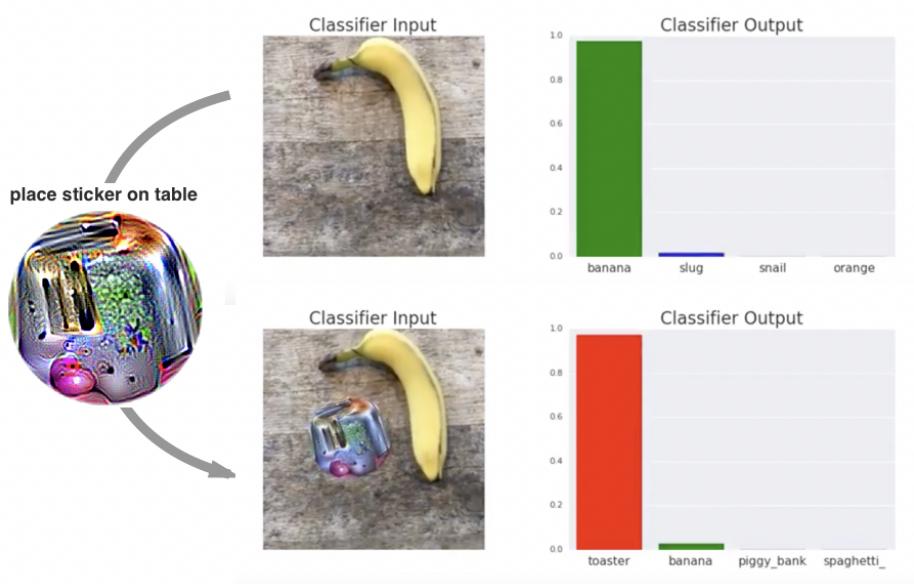


Рис. 6 — приклад наліпки, що допомагає обманути нейронні мережі

Розміщення наклейки з надрукованим тостером недостатньо для того, щоб обдурити мережу, і вона все ще продовжує класифікувати її як банан.

Однак за допомогою ретельно сконструйованого "змагального стікера" легко змусити мережу думати, що це тостер.

Цитуючи авторів, "ця атака була важливою, оскільки зловмиснику не потрібно знати, на який образ вони нападають, будуючи атаку.

Після генерування змагального стікера він може бути широко розповсюджений через інтернет для друкування зловмисниками та використання".

Ці приклади свідчать про те, що наші нейронні мережі все ще досить крихкі, коли явно атакуються противником таким чином.

1.2 Класифікація за наявністю цілі

Змагальні атаки класифікуються на дві категорії — цільові атаки та атаки без цілі [6].

Цільова атака спрямована на те, щоб змусити цільову модель M класифікувала зображення I класу X як Y . Отже, мета націленої атаки — зробити M неправильним класифікатором, розпізнавши клас штучного прикладу I як Y замість справжнього класу X .

З іншого боку, атака без цілі не намагається змусити цільову модель розпізнавати приклад як елемент цільового класу, натомість мета — просто зробити так, щоб цільова модель класифікувала вхідні дані як клас, відмінний від початкового класу X .

У цілому, хоча атаки без цілі не такі корисні як цільові атаки, вони займають набагато менше часу. Цілеспрямовані атаки, хоча й успішніші в зміні прогнозів моделі, коштують дорожче (за часом).

1.3 Види захисту

Навчальний тренінг

Один з найпростіших і найкорстокіших способів захисту від цих атак — удати, що ви є зловмисником, створити ряд змагальних прикладів проти вашої власної мережі, а потім чітко навчити модель не обманюватися на них. Це покращує узагальнення моделі, але не здатне забезпечити значущий рівень стійкості — насправді, це просто закінчується грою, де зловмисники та захисники просто намагаються одне одного обігнати.

Оборонна дистилляція [7]

При оборонній дистилляції ми "навчаємо вторинну модель, що більш стійка в напрямах, які зловмисник, як правило, намагатиметься експлуатувати, ускладнюючи їм виявлення вразливих вхідних параметрів, що призводять до неправильної категоризації". Причина, з якої це працює, полягає в тому, що на відміну від першої моделі, друга модель навчається на "вторинних" вихідних

показниках первинної моделі, а не на "безпосередніх" справжніх ярликах з реальних даних про навчання.

1.4 Класифікація за доступністю внутрішніх даних

Атаки на моделі з відкритим кодом [8]

В умовах доступу до коду противник має доступ до всієї інформації цільової нейронної мережі, включаючи її архітектуру, параметри, градієнти тощо.

Атакувальний модуль може повною мірою використовувати інформацію мережі, щоб ретельно обробляти змагальні приклади.

Атаки на моделі з відкритим кодом були широко вивчені, оскільки розкриття архітектури та параметрів моделі допомагає людям чітко зрозуміти слабкість моделей ГН та надає можливість математично її проаналізувати.

Захист від атак на моделі з відкритим кодом — це властивість, яку ми хочемо мати в моделях ШІ.

Атаки на моделі з закритим кодом [8]

В умовах атаки на моделі з закритим кодом внутрішня конфігурація моделей ГН недоступна для супротивників.

Супротивники можуть подавати лише вхідні дані та отримувати виходи моделей.

Зазвичай напади здійснюють, даючи моделі зразки вхідних даних та спостерігаючи за результатами, щоб використати причинно-наслідковий взаємозв'язок моделі, а також її слабкість.

У порівнянні з атаками з відкритим кодом, напади з закритим кодом є більш практичними в застосуванні, оскільки дизайнери моделей зазвичай не відкривають вихідні параметри своєї моделі з власних причин.

Атаки на альтернативні моделі [8]

У цьому виді атак зловмисник навчає модель, що є аналогом тієї, яку він намагається обдурити.

Після того, як модель-копія буде навчена, зловмисник більше не потребує оригінальної моделі та може виконувати змагальні атаки в умовах з відкритим кодом.

1.5 Постановка задачі

Як практичну частину було вирішено застосувати генетичні алгоритми для змагальних атак з закритим кодом.

Як ціль було обрано застосунок Tesseract як найбільш популярну програму розпізнавання тексту. Це вільна програма, що розроблялася Hewlett-Packard з 1985 до 1994 року, а в наступне десятиріччя залишалася практично без змін. Не так давно Google купив її та відкрив початковий код під ліцензією Apache 2.0 у 2006 році для продовження розробки. Зараз програма вже працює з UTF-8, розпізнає багато мов, серед яких є й українська.

Приклад роботи програми для зображення (Рис 4.1, 4.2)

1. Introduction

The internet offers a wealth of audio-visual content and communities such as Flickr and YouTube make large amounts of photos and videos publicly available. In many cases, an observer might wonder what elements are visible on a certain shot or movie. Especially for natural scenes, the answer to this question can be difficult because only few landmarks might be easily recognizable by non experts. While the information about the camera position is (at least roughly) known in many cases (photographer's knowledge or camera GPS), the camera orientation is usually unknown (digital compasses have poor accuracy).

(x=648 v=501) — R:160 G:165 B:168

Рис. 7 — приклад зображення з текстом

```
> tesseract 1590850515.png stdout
Warning: Invalid resolution 0 dpi. Using 70 instead.
Estimating resolution as 202
1. Introduction

The internet offers a wealth of audio-visual content
and communities such as Flickr and YouTube make large
amounts of photos and videos publicly available. In many
cases, an observer might wonder what elements are visible
on a certain shot or movie. Especially for natural scenes,
the answer to this question can be difficult because only
few landmarks might be easily recognizable by non experts.
While the information about the camera position is (at least
roughly) known in many cases (photographer's knowledge
or camera GPS), the camera orientation is usually unknown
(digital compasses have poor accuracy).

misc 12:00 1 | nvim 2 / zsh / 1 31 May astadnik
```

Рис. 8 — приклад результату розпізнавання тексту

Як ми бачимо, текст розпізнається коректно.

Завданням є створення ПЗ, яке б змінювало зображення таким чином, щоб текст більше не розпізнавався.

3. Дослідження

3.1 Перша спроба

Було імплементовано алгоритм генетичного навчання. Він був досить класичним [9]:

1. Створити випадкову популяцію $X \in \Re^{N^*w^*h^*d}$, де N - кількість елементів у популяції, а $w * h * d$ — розмірність гена (у цьому випадку — ширина поля з шумом, висота та кількість кольорів відповідно), та заповнити її випадковими значеннями з рівномірного розподілу. $x \sim uniform(min, max), x \in X$. Мінімальне та максимальне значення шуму — гіперпараметр.
2. Порахувати для неї оцінку фітнесу $F = fitness(X)$. Алгоритм залежить від задачі, у нашому випадку це була впевненість розпізнавача в тому, що на зображені є певний текст.
3. Якщо для певного з елементів функція фітнесу свідчить, що цей шум повністю приховує зміст від tesseract — результат повертається, і пошук оголошується виконаним.
4. Створити так званий пул для схрещування $P \in \Re^{\hat{N}^*w^*h^*d}: \hat{N} >> N$, що складатиметься з копій елементів X . Кількість повторів певного елемента X_i в \hat{N} залежить від відносного значення функції фітнесу. Тобто
$$P = \{repeat(x_i, \lfloor \frac{f_i * \hat{N}}{\sum F} \rfloor) \mid x_i \in X, f_i \in F\},$$
 де $repeat(x, n)$ — функція, що повторює x n разів.
5. На основі результатів застосування шумів вибрati елементи для створення наступного покоління. $\hat{X}_1 = sample(P, \hat{N}),$ $\hat{X}_2 = sample(P, N),$ де $sample(P, N)$ — функція, що повертає з P N випадкових елементів.

6. Створити нові шуми шляхом отримання випадкових генів (пікселів) від батьків. Для цього спочатку треба створити маску, яка визначатиме, від якого батька буде взято певний ген. $R = (0, 1)^{N^*w^*h^*d}$, $P_{R_i=1} = 0.5$. Після

цього наступне покоління отримується таким чином:

$$X_{new} = \hat{X}_1 * R + \hat{X}_2 * (1 - R).$$

7. З невеликою ймовірністю замінити деякі гени (пікселі) в новому поколінні на випадкові. $M = (0, 1)^{N^*w^*h^*d}$, $M_{R_i=1} = 0.1$. $M_{R_i=1}$ — параметр, що визначає ймовірність мутації. Тоді можемо застосувати мутацію таким чином: $X_{new} = X_{new} * (1 - M) + rand(min, max) * M$.

8. Повернутись до кроку 2.

Було внесено деякі особливості:

- **Оптимізація виконувалася для кожного слова окремо (рис 9).** Це дозволяло зменшити час роботи та кількість доданого шуму.
- **Як оцінку фітнесу було використано максимальне значення впевненості програми в тому, що в цьому місці є будь-який текст.**
- **У популяції було збережено певну кількість найкращих елементів з минулого, щоб упевнитися, що результат роботи в жодному разі не буде погіршуватися, і гарні рішення не будуть втрачені.**

1. Introduction

The internet offers a wealth of audio-visual content and communities such as Flickr and YouTube make large amounts of photos and videos publicly available. In many cases, an observer might wonder what elements are visible on a certain shot or movie. Especially for natural scenes, the answer to this question can be difficult because only few landmarks might be easily recognizable by non-experts. While the information about the camera position is (at least roughly) known in many cases (photographer's knowledge or camera GPS), the camera orientation is usually unknown (digital compasses have poor accuracy).

(x=664, y=189) — R:154 G:157 B:161

Рис. 9 — приклад зображення з текстом, який було розмічено

Утім, ми розраховували на певні припущення. Основним було те, що з великою ймовірністю, якщо об'єднати 2 шуми, що при додаванні до зображення певним чином зменшують рівень впевненості tesseract, то отриманий шум матиме кращі результати. Перевірити це було можливо, лише завершивши роботу, і виявилося, що у випадку зі складними мережами це не завжди так. Модель майже не навчалась.

3.2 MNIST

Ми вирішили змінити підхід та почати з чогось простішого. Вибір пав на датасет MNIST[10] (рис. 10).

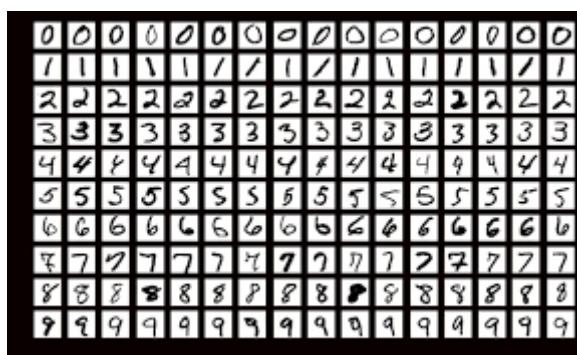


Рис. 10 — приклад зображення з текстом

База даних MNIST (скорочення від Mixed National Institute of Standards and Technology) — об'ємна база даних зразків рукописного написання цифр. Є стандартом, запропонованим Національним інститутом стандартів і технологій США з метою калібрування та зіставлення методів розпізнавання зображень за допомогою машинного навчання, у першу чергу на основі штучних нейронних мереж. База даних містить 60000 зображень для навчання та 10000 зображень для тестування [11].

Також було прийнято рішення використати роботу GenAttack: Practical Black-box Attacks with Gradient-Free Optimization авторів Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, Mani Srivastava [12]. Ця робота присвячена саме використанню генетичного алгоритму для атак на чорний ящик, утім, це використовувалося для інших задач. Їхньою ціллю було приховати зображення предметів або ж символів з датасету MNIST, у той час як моею було розпізнавання тексту за допомогою Tesseract.

Код до цієї роботи знаходився у відкритому доступі. Утім, він був написаний з використанням фреймворку tensorflow, що внеможливидало аналіз внутрішніх процесів. У цьому фреймворку створюється потоковий граф, що дозволяє його цілком виконувати на відеокарті, але вслідкувати за чимось буває досить важко.

Для полегшення роботи аналізу код було повністю переписано на використання бібліотеки numpy, що дозволило швидше експериментувати та перевіряти значення змінних. Це дозволило швидко модифікувати код і спостерігати за впливом певних архітектурних рішень на результат. Перевіривши роботу моделі на нейронній мережі, що розпізнавала зображення з датасету MNIST, ми впевнилися, що модель працює. На малюнку (рис. 12) можна побачити результат роботи моделі, що намагається створити шум, що змусить нейронну мережу думати, що на зображені (рис. 11) написано 9, а не 7 (рис. 4.8). На графіку відтворена залежність упевненості моделі (вертикальна

вісь) у тому, що на зображенні число 9 з кількістю ітерацій (горизонтальна вісь). Як бачимо, впевненість розпізнавача в тому, що на зображенні число 9 зростає від майже 0 до майже 1.

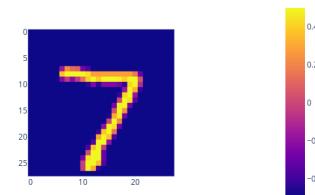


Рис. 11 — приклад символу

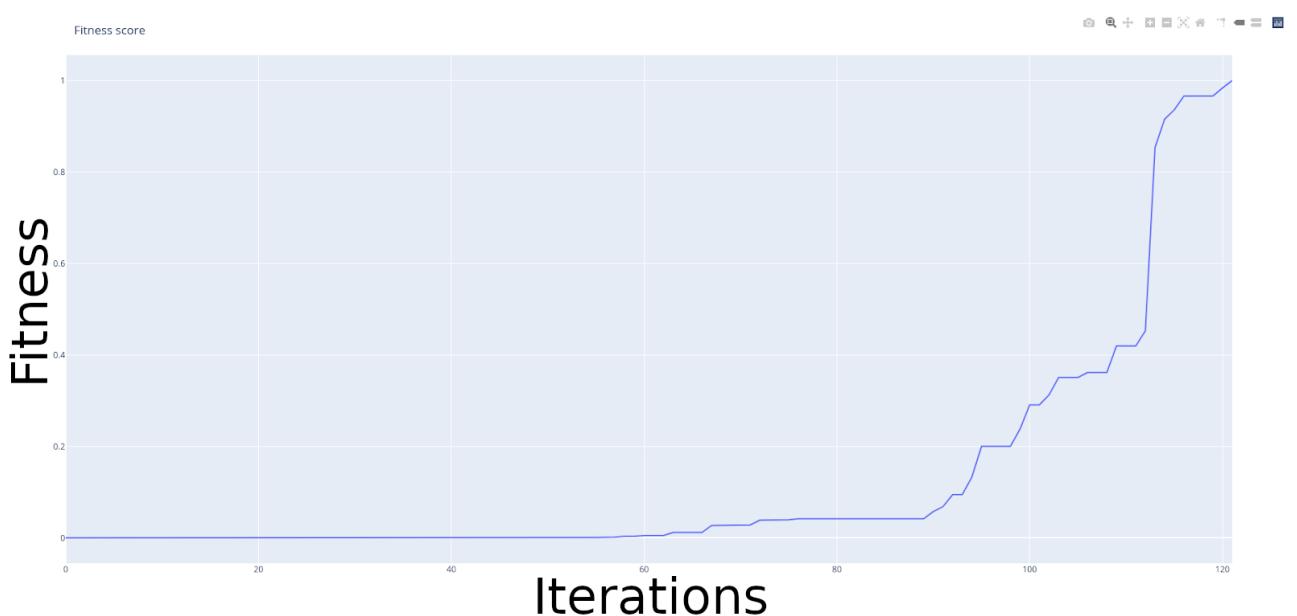


Рис. 12 — графік зміни впевненості моделі (вертикальна вісь), що на зображенні число 9, з часом (горизонтальна вісь)

3.3 Успішна модель

Було залишено обробку зображення частинами та оновлено функції оцінки успішності елемента, і це принесло свої плоди — програма змогла знаходити шум, який змушував розпізнавач "губити" обраний фрагмент тексту.

У результаті було використано наступні параметри:

Розмір популяції = 6

Шанс мутації = 0.30

Максимальне значення шуму = 0.5

Мінімальне значення шуму = -0.5

Множитель шуму (чим менше, тим повільніше змінюється шум) = 1

Звісно, ці параметри абсолютно емпіричні: більші максимальні та мінімальні значення означатимуть більше спотворення зображення, але швидшу швидкість сходження. Було визначено, що при значеннях, що близькі до +1 та -1 відповідно, у більшості випадків результат відсутній. Шанс мутації впливає на логіку алгоритму: при присвоєнні йому значення 1 алгоритм стає випадковим пошуком, при значеннях, що близькі до нуля, повністю зникає можливість створення інформації, відмінної від тієї, що була визначена під час ініціалізації. Якщо користувачу необхідно отримати більш якісний результат (менш помітний шум), ми рекомендуємо спробувати зменшити шанс мутації, та встановити менші значення мінімального та максимального значення шумів. Алгоритм є випадковим, тож врешті-решт він зійдеться, утім, для отримання адекватної швидкості (5-10 сек), було обрано саме ці параметри.

Як критерій оптимізації шуму було використано максимальне значення впевненості програми в тому, що в цьому місці є будь-який текст. Таким чином, мінімізуючи це значення за допомогою генетичних алгоритмів, нейронна мережа перестала бачити на зображенні будь-який текст.

3.4 Результат роботи

Була зроблена візуалізація. Це програма, що відкриває обране зображення з можливістю його анотувати, тобто показати місця, де був знайдений текст, та, власне, сам текст. При натисканні на цю ділянку запускається процес, що знаходить шум, який може приховувати текст, і застосовував його. Після цього зображення можна було зберегти (рис. 13, 14).

1. Introduction

The internet offers a wealth of audio-visual content and communities such as Flickr and YouTube make large amounts of photos and videos publicly available. In many cases, an observer might wonder what elements are visible on a certain shot or movie. Especially for natural scenes, the answer to this question can be difficult because only few landmarks might be easily recognizable by non-experts. While the information about the camera position is (at least roughly) known in many cases (photographer's knowledge or camera GPS), the camera orientation is usually unknown (digital compasses have poor accuracy).

Рис. 13 — розмічене зображення з доданим до одного слова шумом

```
> tesseract changed_img.png stdout
Warning: Invalid resolution 0 dpi. Using 70 instead.
Estimating resolution as 394

The internet of a wealth of audio-visual content
and communities such as Flickr and YouTube make large
amounts of photos and videos publicly available. In many
cases, an observer might wonder what elements are visible
on a certain shot or movie. Especially for natural scenes,
the answer to this question can be difficult because only
few landmarks might be easily recognizable by non experts.
While the information about the camera position is (at least
roughly) known in many cases (photographer's knowledge
or camera GPS), the camera orientation is usually unknown
(digital compasses have poor accuracy).
```

```
> tesseract 1590850515.png stdout
Warning: Invalid resolution 0 dpi. Using 70 instead.
Estimating resolution as 202
1. Introduction

The internet offers a wealth of audio-visual content
and communities such as Flickr and YouTube make large
amounts of photos and videos publicly available. In many
cases, an observer might wonder what elements are visible
on a certain shot or movie. Especially for natural scenes,
the answer to this question can be difficult because only
few landmarks might be easily recognizable by non experts.
While the information about the camera position is (at least
roughly) known in many cases (photographer's knowledge
or camera GPS), the camera orientation is usually unknown
(digital compasses have poor accuracy).
```

```
/ ^ ~ /a/adversarial_attack * ` master !2 ?1 ✓ / adversarial_attack *
```

Рис. 14 — одне зі слів більше не розпізнається

Як можна побачити, обрані слова перестали розпізнаватись tesseract.

Також є можливість приховати всі слова на зображенні (рис. 15).

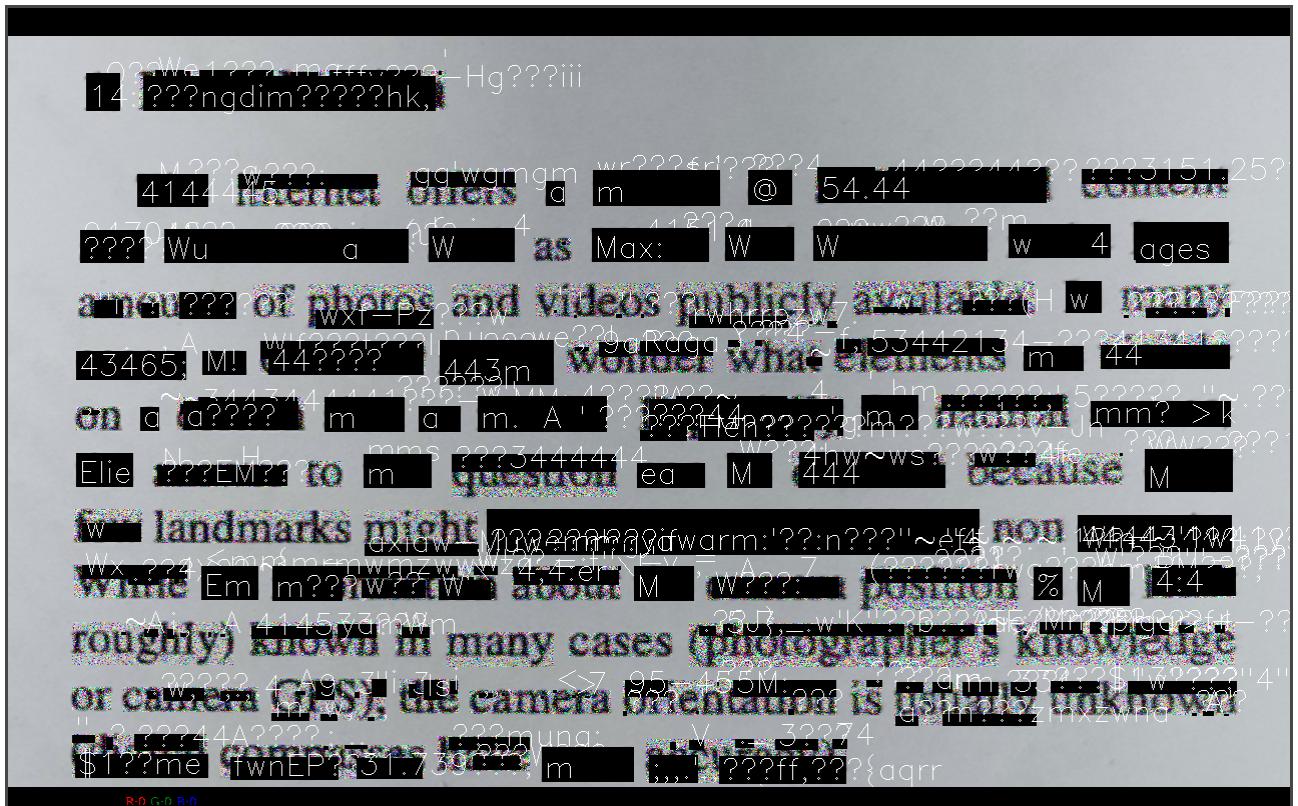


Рис. 15 — приховування всіх слів



Рис. 16 — приклад листа

Tesseract використовує певні адаптивні межі, тож при зменшенні кількості "нормальних" слів збільшується кількість шуму. Через це довелося відкинути ідею повторного використання застосунку для покращення результату.

Іншим прикладом може бути лист про важливість захисту даних (Рис. 16). Як можна побачити (Рис. 17), навіть попри поганий стан листа, програма-роздільник усе одно успішно "читає" майже всі слова.

Після застосування моделі до слова "information" воно стало приховане від розпізнавача (Рис. 18), але не від людського ока (Рис. 19).



Рис. 17 — розпізнавання тексту



Рис. 18 — розпізнавання тексту на зображенні з шумом



Рис. 19 — Текст на зображенні з шумом людина може прочитати

Модель інваріантна до оточення, у якому було зроблене фото. Наприклад, ми можемо приховати назву магазину на вулиці (Рис. 20, 21).

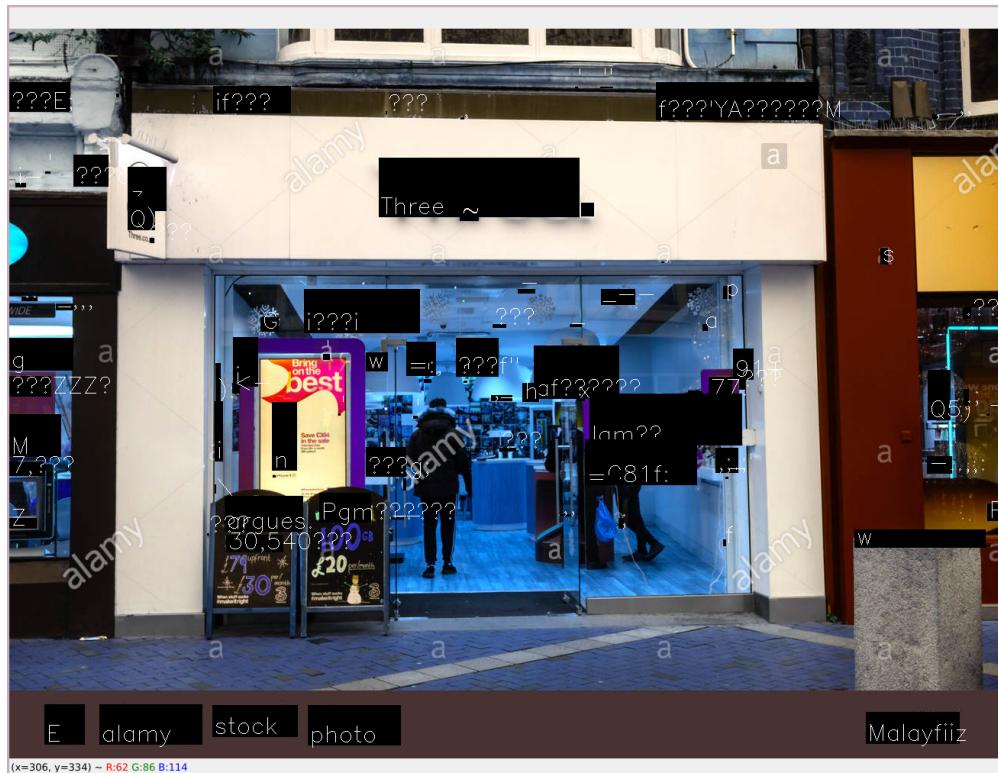


Рис. 20 — магазин

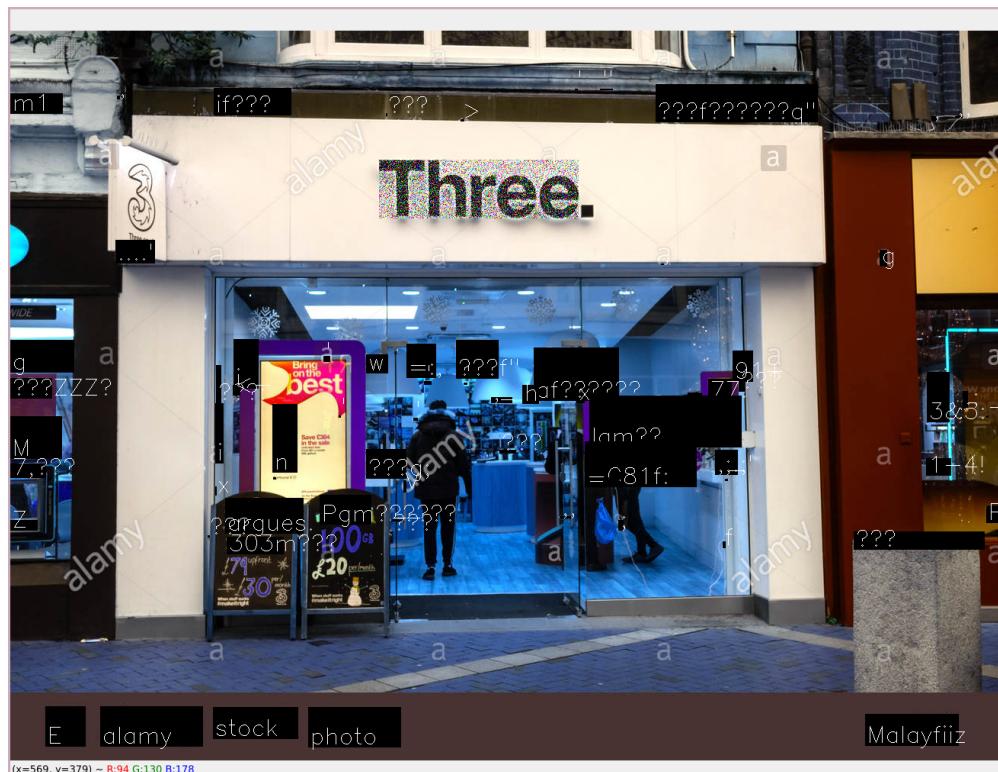


Рис. 21 — називу магазину приховано, попри чіткий чорний текст на світловому фоні

Висновки

Результатом цієї роботи є модель, що здатна приховувати текст на зображенні від програм-роздільників шляхом додавання шуму, який не перешкоджає сприйняттю його людиною.

У межах цієї роботи було розроблено модель на основі генетичних алгоритмів для створення шуму, який приховує текст на зображенні від програм-роздільників. Було досліджено вплив параметрів на результат і швидкодію, і використано ті, що дозволяють досягти найкращих результатів. Модель не потребує доступу до внутрішнього стану програми-роздільника, а працює лише з результатом її роботи.

Також було написано програму, яка, використовуючи створену модель, дозволяє вибірково приховувати текст на зображенні, та зберігати зображення для подальшого використання. Ця програма може бути використана як у ручному режимі для вирізання важливої інформації, а також автоматично для обробки зображень перед подальшим використанням.

Напрямки подальших досліджень:

- Визначити інваріантність моделі до програми-роздільника.
Перевірити, чи необхідно створювати шум для різних роздільників та моделей окремо.
- Оптимізувати швидкодію.
- Протестувати наявні методи покращення результатів роботи генетичних алгоритмів.

Література

1. <https://cds.nyu.edu/adversarial-attacks-poster/>
2. <https://arxiv.org/pdf/1801.00553>
3. <http://www.cs.cmu.edu/~sbhagava/papers/face-rec-ccs16.pdf>
4. <https://arxiv.org/pdf/1707.08945>
5. <https://arxiv.org/abs/1712.09665>
6. <https://www.novetta.com/2019/10/adversarial-attacks-part2/>
7. <https://arxiv.org/abs/1511.04508>
8. https://www.researchgate.net/publication/340249406_A_comparative_study_of_white-box_and_black-box_adversarial_attacks_to_the_deep_neural_networks_with_different_architectures
9. https://en.wikipedia.org/wiki/Genetic_algorithm
10. <http://yann.lecun.com/exdb/mnist/>
11. <http://yann.lecun.com/exdb/mnist/>
12. Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, Mani Srivastava: GenAttack: Practical Black-box Attacks with Gradient-Free Optimization (2018)

Додатки

Додаток А

attack.py

```
import time
from multiprocessing import cpu_count
from multiprocessing.pool import Pool

import numpy as np
import pytesseract
from PIL import Image
import pickle
import atexit

from genattack_tf2 import GenAttack2
```

```

pop_size = 6
test_size = 1000
mutation_rate = 0.30
alpha = 0.5
adaptive = False
max_steps = 10000
# max_steps = 1
mutation_rate = 0.3
# eps = 0.30
eps = 1
output_dir = 'mnist_output'
temp = 0.1

rez = []

def save_rez():
    with open(f'rez_ps_{str(pop_size)}_mr_{str(mutation_rate)}.pickle', 'wb') as f:
        pickle.dump(rez, f)

def score(img: np.ndarray, c=6):
    img = Image.fromarray(((np.squeeze(img) + 0.5) * 255).astype(np.uint8))
    d_ = pytesseract.image_to_data(
        img, output_type=pytesseract.Output.DICT, config=f'--psm {c} --oem 0')
    return max([c for c in d_['conf'] if isinstance(c, int)] + [0])

def pred(imgs):
    with Pool(cpu_count()) as p:
        return np.array(p.map(score, imgs))
    # return np.array([score(i, c) for i in imgs])

def gen_noise(img, pop_size_ = None, mutation_rate_ = None):
    pop_size_ = pop_size_ or pop_size
    mutation_rate_ = mutation_rate_ or mutation_rate
    image_shape = img.shape
    image_channels = 3

    attack = GenAttack2(pred=pred,
                        pop_size=pop_size_,
                        mutation_rate=mutation_rate_,
                        eps=eps,
                        max_steps=max_steps,
                        alpha=alpha,
                        image_shape=image_shape,

```

```

        image_channels=image_channels,
        temp=temp,
        adaptive=adaptive)

start_time = time.time()
result = attack.attack(img)
end_time = time.time()
attack_time = (end_time-start_time)
if result is not None:
    adv_img, query_count, margin_log = result
    rez.append(margin_log)
    # if len(rez) >= 200:
    #     print('bye')
    #     exit()
    # final_pred = sess.run(test_pred, feed_dict={
    #                         test_in: [adv_img}})[0]
else:
    print('Attack failed')
return result

import atexit
atexit.register(save_rez)

```

genattack_tf2.py

```

import time
import random
import numpy as np
from PIL import Image
from scipy.special import softmax
from sklearn.preprocessing import OneHotEncoder


def imresize(img, size):
    resized_img = Image.fromarray(img).resize(size=size)
    resized_img = np.array(resized_img)
    return resized_img


class GenAttack2(object):
    def mutation_op(self, cur_pop, idx, step_noise=0.01, p=0.005):
        perturb_noise = np.random.uniform(low=-step_noise, high=step_noise,
                                           size=cur_pop.shape)
        mutated_pop = perturb_noise * \
            (np.random.uniform(size=cur_pop.shape) < p).astype(np.float32) +
        cur_pop
        return mutated_pop

```

```

    def attack_step(self, idx, success, orig_copies, cur_noise, prev_elite,
margin_log, best_win_margin, cur_plateau_count, num_plateaus):
        noise_shape = self.image_shape
        cur_pop = np.clip(
            cur_noise + orig_copies, self.box_min, self.box_max)
        # List of predictions of shape (n, 10)
        pop_preds = self.pred(cur_pop)

        # Test if there is a succesfull result
        success_pop = np.equal(pop_preds, 0).astype(np.int32)
        success = np.max(success_pop, axis=0)
        # if success:
        #     __import__('ipdb').set_trace()

        # the goal is to maximize this loss
        loss = -np.log(pop_preds+1e-30)

        # Difference between prediction of the target label and the maximum
        one
        win_margin = np.min(pop_preds, axis=0)

        # If prob of target label became larger
        if np.less(win_margin, best_win_margin):
            new_best_win_margin, new_cur_plateau_count = win_margin, 0
        else:
            new_best_win_margin, new_cur_plateau_count = best_win_margin,
        cur_plateau_count+1,

        # If probability of target is very small, bigger threshold
        if np.greater(win_margin, -0.40):
            plateau_threshold = 100
        else:
            plateau_threshold = 300

        # If plateau lasts for long enough
        if np.greater(new_cur_plateau_count, plateau_threshold):
            new_num_plateaus, new_cur_plateau_count = num_plateaus+1, 0,
        else:
            new_num_plateaus, new_cur_plateau_count = num_plateaus,
        new_cur_plateau_count

        if self.adaptive:
            step_noise = np.maximum(self.alpha,
                                   0.4*np.power(0.9,
                                   new_num_plateaus.astype(np.float32)))
            if np.less(idx, 10):

```

```

        step_p = 1.0
    else:
        step_p = np.maximum(self.mutation_rate,
                            0.5*np.power(0.90,
new_num_plateaus.astype(np.float32)))
    else:
        step_noise = self.alpha
        step_p = self.mutation_rate

    step_temp = self.temp

    # If found a solution
    if np.equal(success, 1):
        # Preserve it
        elite_idx = np.expand_dims(
            np.argmax(success_pop).astype(np.int32), axis=0)
    else:
        # Save the best entity
        elite_idx = np.expand_dims(
            np.argmax(loss, axis=0).astype(np.int32), axis=0)
elite = cur_noise[elite_idx]

    # Get fitness for each value. If step_temp is smaller, then the
function
    # will be steeper
    select_probs = softmax(np.squeeze(loss) / step_temp)

    # Might output a single item repeated
parents = np.random.choice(
    len(select_probs), p=select_probs, size=2*self.pop_size-2)

    # First half of parents
parent1 = cur_noise[parents[:self.pop_size-1]]
    # Second half of parents
parent2 = cur_noise[parents[self.pop_size-1:]]
    # Parent probabilities
pp1 = select_probs[parents[:self.pop_size-1]]
pp2 = select_probs[parents[self.pop_size-1:]]
pp2 = pp2 / (pp1+pp2)
    # Expand shape of probabilities to (n, 1, 1, 1), and then repeat them
so
    # they will match shape of noise
pp2 = np.tile(np.expand_dims(pp2, (1, 2, 3)),
              (1, noise_shape[0], noise_shape[1], self.image_channels))
    # Boolean mask which determines which parents' genes should be taken
xover_prop = (np.random.uniform(
    size=parent1.shape) > pp2).astype(np.float32)

```

```

        child� = parent1 * xover_prop + parent2 * (1-xover_prop)
        # Index of iteration
        idx += 1
        print(idx, np.max(loss), win_margin, step_p, step_noise,
              new_cur_plateau_count, pop_preds)
        margin_log = np.concatenate([margin_log, [[win_margin]]], axis=0)
        # Apply mutation to the population
        mutated_child� = self.mutation_op(
            child�, idx=idx, step_noise=self.eps*step_noise, p=step_p)
        # Add elite to the population
        new_pop = np.concatenate((mutated_child�, elite), axis=0)
        return idx, success, orig_copies, new_pop, np.reshape(elite,
                  (noise_shape[0], noise_shape[1], self.image_channels)),
    margin_log, new_best_win_margin, new_cur_plateau_count, new_num_plateaus

    def __init__(self, pred, pop_size=6, mutation_rate=0.001,
                 eps=0.15, max_steps=10000, alpha=0.20,
                 image_shape=299,
                 image_channels=3,
                 temp=0.3,
                 adaptive=False):
        self.eps = eps
        self.pop_size = pop_size
        self.pred = pred
        self.alpha = alpha
        self.temp = temp
        self.max_steps = max_steps
        self.mutation_rate = mutation_rate
        self.image_shape = image_shape
        noise_shape = self.image_shape
        self.image_channels = image_channels
        self.adaptive = adaptive
        self.input_img = np.zeros(
            (1, self.image_shape[0], self.image_shape[1],
            self.image_channels), dtype=np.float32)
        # copies of original image
        self.pop_orig = np.zeros(
            (self.pop_size, self.image_shape[0], self.image_shape[1],
            image_channels), dtype=np.float32)
        self.pop_noise = np.zeros(
            (self.pop_size, noise_shape[0], noise_shape[1],
            self.image_channels), dtype=np.float32)

        self.init_success = 0
        self.box_min = np.tile(np.maximum(
            self.input_img-eps, -0.5), (self.pop_size, 1, 1, 1))
        self.box_max = np.tile(np.minimum(

```

```

        self.input_img+eps, 0.5), (self.pop_size, 1, 1, 1))
self.margin_log = np.zeros((1, 1), dtype=np.float32)
self.i = 0

# Variables to detect plateau
self.best_win_margin = -1
self.cur_plateau_count = 0
self.num_plateaus = 0

def attack_main(self):
    while np.logical_and(np.less_equal(self.i, self.max_steps),
                         np.equal(self.init_success, 0)):
        (self.i, self.init_success, self.pop_orig,
         self.pop_noise, self.pop_noise[0], self.margin_log,
         self.best_win_margin, self.cur_plateau_count,
         self.num_plateaus) = self.attack_step(self.i, self.init_success,
                                             self.pop_orig,
                                             self.pop_noise,
                                             self.pop_noise[0], self.margin_log,
                                             self.best_win_margin,
                                             self.cur_plateau_count,
                                             self.num_plateaus)
    return (self.i, self.init_success, self.pop_orig, self.pop_noise,
            self.pop_noise[0], self.margin_log, self.best_win_margin,
            self.cur_plateau_count, self.num_plateaus)

def initialize(self, img):
    # Add dimension to the input_img (from (28, 28, 1) to (1, 28, 28, 1))
    self.input_img = np.expand_dims(img, axis=0)
    # Stack input images (from (1, 28, 28, 1) to (n, 28, 28, 1))
    self.pop_orig = np.tile(self.input_img, [self.pop_size, 1, 1, 1])
    # Init initial population.
    self.pop_noise = self.mutation_op(
        self.pop_noise, idx=self.i, p=self.mutation_rate,
        step_noise=self.eps)
    self.margin_log = np.zeros((1, 1), dtype=np.float32)
    self.best_win_margin = np.array(-1.0, dtype=np.float32)
    self.cur_plateau_count = np.array(0, dtype=np.int32)
    self.num_plateaus = 0
    # print('Population initialized')

def attack(self, input_img):
    self.initialize(input_img)
    (num_steps, success, copies, final_pop, adv_noise,
     log_hist, _, _, _) = self.attack_main()
    if success:
        adv_img = np.clip(np.expand_dims(input_img,

```

```

axis=0)+np.expand_dims(adv_noise, axis=0),
                           self.box_min[0:1], self.box_max[0:1])

        # Number of queries = NUM_STEPS * (POP_SIZE -1 ) + 1
        # We subtract 1 from pop_size, because we use elite mechanism, so
one population
        # member is copied from previous generation and no need to
re-evaluate it.
        # The first population is an exception, therefore we add 1 to
have total sum.
        query_count = num_steps * (self.pop_size - 1) + 1
        return adv_img[0], query_count, log_hist[1:, :]
    else:
        return None

```

helpers.py

```

"""This is the module with miscellaneous functions"""
import argparse


def parse_args() -> argparse.Namespace:
    """Parse arguments from the command line

    This function parses arguments and returns a class-like object with
values

    Returns:
        argparse.Namespace: [TODO:description]
    """
    parser = argparse.ArgumentParser(description="This program allows user to
detect\
                                         and recognize printed text on images, as
well as\
                                         get the contents of the image as a
text")
    parser.add_argument("-i", "--image", default=None, help="Path to the
image", required=True)
    # parser.add_argument("-d", "--display", action="store_true",
    #                     default=False, help="Display the image")
    parser.add_argument("-a", "--annotate", action="store_true",
                        default=False, help="Annotate the image")
    parser.add_argument("-p", "--pop_size", default=None, help="Population
size")
    parser.add_argument("-m", "--mutation_rate", default=None,
                        help="Mutation rate")
    return parser.parse_args()

```

img_ocr.py

```
"""
This module contains ImgOCR class
"""

from dataclasses import astuple
from typing import List

import cv2
import pixcat
import pytesseract
from PIL import Image
from tqdm import tqdm

from text_item import TextItem
from img import Img
import numpy as np
from attack import gen_noise
from multiprocessing import Pool
from imageio import imwrite

class ImgOCR(Img):
    """This class is capable of interacting with pytesseract"""

    def __init__(self, *, img=None, path=None):
        """Initialize the class

        Args:
            path (str, optional): path to the image
        """
        super().__init__(img=img, path=path)
        self.data = None
        self.string = None
        self.data_to_process = None
        self.tqdm = None

    def mouse_callback(self, event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            if self.data is None:
                return
            for d in self.data:
                l, t, w, h, _, text, _, _, _ = astuple(d)
                if l <= x <= l + w and t <= y <= t + h:
                    img_with_noise = gen_noise((d.img.astype(np.float) / 255
- 0.5)
                    if img_with_noise:
                        img_with_noise = (img_with_noise[0] + 0.5) * 255
                        self.img[t:t+h, l:l+w] = img_with_noise
```

```

        self.compute_text_data()
        img = self._annotate(show_text=True)
        self.update(img=img)

    def compute_text_data(self, filter_data=True, add_img=True) ->
List[TextItem]:
    """Get list of TextItem from pytesseract for the image

    This functions provides data about the text on the image (it's
bounding boxes,
text, confidences etc)

    Args:
        img (): input image
    """
    print('Computing data...')
    img = Image.fromarray(self.img)
    data = pytesseract.image_to_data(
        img, output_type=pytesseract.Output.DICT, config=f'--psm 6 --oem
0')
    # 'level',# Page, block, paragraph, line, word
    # 'page_num', 'block_num', 'par_num', 'line_num', 'word_num',
    # 'left', 'top', 'width', 'height', 'conf', 'text',

    # Convert dict of lists to list of dicts
    data = [dict(zip(data, t)) for t in zip(*data.values())]

    # Build TextItem objects
    keys = ['left', 'top', 'width', 'height', 'conf', 'text']
    self.data = [TextItem(**{k:d[k] for k in keys}) for d in data]

    if filter_data:
        self.data = list(
            filter(lambda i: i.conf != -1 and i.text != '', self.data))
    if add_img:
        for v in tqdm(self.data, leave=False):
            l, t, w, h = v.left, v.top, v.width, v.height
            v.img = self.img[t:t+h, l:l+w]
    print('Data is computed')
    return self.data

def get_str(self):
    """Generate string from image

    This function extracts text from the image
    """
    img = Image.fromarray(cv2.cvtColor(self.img, cv2.COLOR_BGR2RGB))

```

```

        self.string = pytesseract.image_to_string(img, config=f'--psm 6 --oem
0')
        return self.string

    def show(self, *, annotate=False, method=None):
        """Display the image

        This function displays images in a various ways

        Args:
            annotate (bool, optional): Set to true to display image with
            additional data
            method (string, optional): kitty, jupyter or None
        """
        img = self.__annotate(show_text=True) if annotate else self.img

        if img is None:
            return None
        if method == "kitty":
            pixcat.Image(Image.fromarray(img)).thumbnail(
                512).show(align="left")
        elif method == "jupyter":
            return Image.fromarray(img)
        else:
            cv2.namedWindow('image')
            cv2.setMouseCallback('image', self.mouse_callback)
            cv2.imshow('image', img)
            # wait = p.apply_async(cv2.waitKey, args=(0, ))
            # key = None
            while True:
                # if wait.ready():
                #     key = wait.get()
                #     wait = p.apply_async(cv2.waitKey, args=(0, ))
                key = cv2.waitKey(1)

                if key == 27:
                    break
                if key == 97:
                    annotate = not annotate
                    img = self.__annotate(show_text=True) if annotate else
self.img
                    cv2.imshow('image', img)
            elif key == 114:
                self.compute_text_data()
                img = self.__annotate(show_text=True) if annotate else
self.img
                self.update(img=img)

```

```

        elif key == 115:
            imwrite('changed_img.png', self.img)
        elif key == 112 and not self.data_to_process:
            self.data_to_process = self.compute_text_data()
            self.tqdm = tqdm(total=len(self.data_to_process))

        if self.data_to_process:
            d = self.data_to_process.pop()
            self.tqdm.update()
            if not self.data_to_process:
                self.data_to_process = None
                self.tqdm = None
            if d.conf < 10:
                continue

            l, t, w, h, _, text, _, _, _ = astuple(d)
            img_with_noise = gen_noise((d.img.astype(np.float) / 255
- 0.5)
            if img_with_noise:
                img_with_noise = (img_with_noise[0] + 0.5) * 255
                self.img[t:t+h, l:l+w] = img_with_noise
            else:
                print('SHIIIT')

            self.compute_text_data()
            img = self.__annotate(show_text=True)
            cv2.imshow('image', img)

            cv2.destroyAllWindows()
        return None

def process(self, pop_size_=None, mutation_rate_=None):
    for d in tqdm(self.compute_text_data()):
        if d.conf < 10:
            continue
        l, t, w, h, _, text, _, _, _ = astuple(d)
        img_with_noise = gen_noise((d.img.astype(np.float) / 255) - 0.5,
            pop_size_, mutation_rate_)
        if img_with_noise:
            img_with_noise = (img_with_noise[0] + 0.5) * 255
            self.img[t:t+h, l:l+w] = img_with_noise
        else:
            print('SHIIIT')

def __annotate(self, *, show_text=False, show_confidence=False):
    """This function annotates the image with bounding boxes and textes

```

```
or confidences
```

Args:

```
    show_text (bool, optional): set to True to show text
    show_confidence (bool, optional): set to True to show confidence
```

"""

```
    img_ = self.img.copy()
    for v in self.data:
        l, t, w, h, conf, text, _, _, _ = astuple(v)
        cv2.rectangle(img_, (l, t), (l+w, t+h), (0, 0, 0), -2)
        text_to_show = ""
        if show_text:
            text_to_show += text
        if show_confidence:
            text_to_show += conf
        font_scale = (w * h) / (img_.shape[0] * img_.shape[1])
        cv2.putText(img_, str(text_to_show), (l+2, t+h-4),
cv2.FONT_HERSHEY_SIMPLEX,
                    1, (255, 255, 255), 1)
    return img_
```

img.py

"""

```
This module contains a base class which implements simple image
"""
```

```
import cv2
import pixcat
from PIL import Image
```

```
class Img():
```

```
    """This class is capable of storing and managing the image"""

    def __init__(self, *, img=None, path=None):
        """
```

Initialize the class

```
        :param img np.array: raw image
        :param path str: path to image
        """
```

```
        if img:
            self.img = img
        elif path:
            self.load_img(path)
        else:
            raise RuntimeError("You should provide the image")
```

```

def load_img(self, path: str):
    """Load image from the disk"""
    self.img = cv2.cvtColor(cv2.imread(path), cv2.COLOR_BGR2RGB)

# pylint: disable=too-many-arguments
def mouse_callback(self, event, x, y, flags, param):
    """Callback for using with cv2.imshow"""

def update(self, *, img=None, method=None):
    img = img if img is not None else self.img
    cv2.imshow('image', img)

def show(self, *, img=None, method=None):
    """Display the image

This function displays images in a various ways

Args:
    method (string, optional): kitty, jupyter or None
    """
    img = img if img is not None else self.img
    if img is None:
        return None
    if method == "kitty":
        pixcat.Image(Image.fromarray(img)).thumbnail(
            512).show(align="left")
    elif method == "jupyter":
        return Image.fromarray(img)
    else:
        cv2.namedWindow('image')
        cv2.setMouseCallback('image', self.mouse_callback)
        cv2.imshow('image', img)
        while True:
            key = cv2.waitKey(0)
            if key == 27:
                break
        cv2.destroyAllWindows()
    return None

def resize(self, resize):
    """
Handy function to resize image

If resize argument is float, it specifies the scaling percent. If
it's a tuple,
    it specifies the dimensions of the resulting image. Additionally, if

```

```
the 2-nd parameter
    equals -1, it is calculated so the proportions of the image are
preserved.
```

```
Args:
    resize (float or tuple, optional): look at the description
"""
if isinstance(resize, float):
    assert 0 < resize <= 1
    width = int(self.img.shape[1] * resize)
    height = int(self.img.shape[0] * resize)
    dim = (width, height)
elif isinstance(resize, (tuple, list)):
    if resize[1] == -1:
        dim = (resize[0], self.img.shape[0] *
               resize[0] // self.img.shape[1])
    else:
        dim = resize
else:
    raise RuntimeError("Wrong resize argument: " + str(resize))
self.img = cv2.resize(self.img, dim)
```

main.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""This is the main module"""

from helpers import parse_args
from img_ocr import ImgOCR
# from attack import gen_noise

def main():
    """Main function"""
    args = parse_args()

    img = ImgOCR(path=args.image)
    # img.resize((1280, -1))
    data = img.compute_text_data()
    # img.process(pop_size_=int(args.pop_size),
    mutation_rate_=float(args.mutation_rate))
    img.show(annotate=True)

    # with open('img.pickle', 'wb') as f:
    #     pickle.dump(img, f)
```

```

if __name__ == "__main__":
    # with launch_ipdb_on_exception():
    main()

text_item.py

"""
This module contains the text_item class
"""

from dataclasses import dataclass, field

import numpy as np
from PIL import Image

from img import Img


@dataclass
class TextItem(Img):
    """
    This class represents the text item got from the tesseract.

    This class has bbox data, text, confidence and image. Also it might have
    a
    noise which is automatically applied to image
    """

    left: int
    top: int
    width: int
    height: int
    conf: float
    text: str
    _img: np.array = field(default=None, init=False, repr=False)
    has_img: bool = field(init=False)
    _noise: np.array = field(default=None, init=False, repr=False)

    def get_pil(self) -> Image:
        """Return PIL.Image"""
        return Image.fromarray(self.img)

    @property
    def img(self) -> np.array:
        """Get image with noise if noise is set"""
        if self._img is None:
            raise RuntimeError("Img is not setted")
        if self.noise is not None:
            return self._img + self.noise
        return self._img

```

```
@img.setter
def img(self, img: np.array):
    self._img = img
    self.has_img = True

@property
def noise(self) -> np.array:
    return self._noise

@noise.setter
def noise(self, noise: np.array):
    s = self._img + noise
    self._noise = np.clip(noise, None, (255 - self._img))
```