Politechnika Warszawska

Dokumentacja Projektu

Metody wykrywania steganografii

Przedmiot: Kryptografia Stosowana
Kod przedmiotu: 103A-TLTIC-MSP-KRYS
Prowadzący: dr Adam Komorowski

Autorzy:

Safiya Aliaksandrava
Wiktor Ciołek
Ignacy Czajkowski
Urszula Kamińska
Dawid Karpiński

Data:
2 lutego 2025

Spis treści

1 Wstęp

1.1 Steganografia i steganoanaliza

Steganografia to sztuka ukrywania informacji. Jej celem jest przekazanie wiadomości w taki sposób, żeby pozostała niezauważona przez osoby postronne. Tym różni się od kryptografii - ta zabezpiecza informacje przed odczytaniem, natomiast sam komunikat może zostać zauważony, dopóki nie zostanie odkodowany. W steganografii sam fakt przekazywania informacji powinien zostać niezauważony przed podsłuchujących. Sama technika steganografii powstała w starożytności, ale rozwój technologii znacząco wpłynął na rozwój tej dziedziny, dostarczając nowych metod ukrywania informacji. [7, 8]

Nośnik, w którym zostanie ukryta informacja, może być właściwie dowolny i nazywany jest kontenerem. Najpopularniejszymi nośnikami w steganografii są dane multimedialne: obraz, pliki audio i wideo. Równie często wykorzystuje się protokoły IP i VoIP. Zdarzają się również techniki steganograficzne w tekstach, jednak są rzadziej stosowane z powodu różnego rodzaju ograniczeń.
Można ogólnie powiedzieć, że każdy nośnik danych jest potencjalnym kontenerem dla steganografii. Istnieją opisy wykorzystywania do tego plików wykonywalnych *.exe, wiadomości MMS, protokołów UDP czy nawet łańcuchów DNA. [10]

Odpowiedzią na steganografię jest **steganaliza**, zajmująca się wykrywaniem ukrytych wiadomości. Steganaliza ma na celu identyfikację obecności ukrytych danych, a czasem nawet ekstrakcję tych danych, bez wiedzy o kluczu lub algorytmie użytym do osadzenia wiadomości. Steganalizę można podzielić na dwie główne kategorie: steganalizę specyficzną (target) oraz steganalizę uniwersalną (blind). Do pierwszej grupy należeć będą metody ukierunkowane na konkretne algorytmy steganograficzne. Druga grupa zbiera rozwiązania bardziej elastyczne, ale przez to najczęściej mniej dokładne.

Wraz z rozwojem nowych technik steganograficznych, stale potrzebne są nowe i bardziej zaawansowane metody steganalizy – steganaliza jest nieustannie rozwijającą się dziedziną, która odgrywa kluczową rolę w zapewnianiu bezpieczeństwa informacji i walce z cyberprzestępczością. [17]

1.2 Informacje nt. realizacji projektu

Projekt polegał na opracowaniu teoretycznym i implementacji wybranych metod wykrywania ukrytych informacji w obrazach, plikach audio lub innych mediach. W dalszej części dokumentacji zostały przedstawione wybrane techniki steganoanalizy wraz z opisem implementacji. Każdy członek zespołu opracował jedną metodę. Pełny kod źródłowy projektu znajduje się w repozytorium na platformie GitHub `https://github.com/davkk/steganalysis`.

2 Wykrywanie steganografii w obrazach z wykorzystaniem analizy statystycznej

2.1 Steganaliza statystyczna

Proces ukrywania informacji w nośnikach cyfrowych zmienia ich właściwości statystyczne [17]. Nośniki takie jak obrazy, dźwięki czy wideo mają przewidywalne wzorce, które są zakłócane przez steganografię. Dzięki temu stosując odpowiednie metody analizy statystycznej, można wykryć obecność ukrytych danych.

Najpopularniejszym nośnikiem steganograficznym wymienianym w literaturze są obrazy cyfrowe. Poniżej wymieniono często stosowane techniki steganalizy statystycznej w kontekście obrazów cyfrowych [14].

- **Test chi-kwadrat**: Porównuje rozkład obserwowany z oczekiwanym, np. dla steganografii PVD [12].
- **RS steganaliza**: Klasyfikuje grupy pikseli jako „regularne” lub „osobliwe” na podstawie zmian szumu po modyfikacji LSB [13].
- **Raw Quick Pair (RQP)**: Analizuje pary kolorów w obrazach 24-bitowych, które różnią się w bitach LSB.
- **Analiza histogramu**: Wykrywa zmiany w rozkładzie wartości pikseli lub współczynników DCT.

2.2 Analiza histogramów

Steganaliza oparta na histogramach jest statystyczną metodą wykrywania ukrytych wiadomości w obrazach, które zostały zmodyfikowane za pomocą technik takich jak steganografia w najmniej znaczących bitach (LSB). Poniżej znajduje się opis implementacji metody z artykułu A Novel Steganalysis Method Based on Histogram Analysis [4].

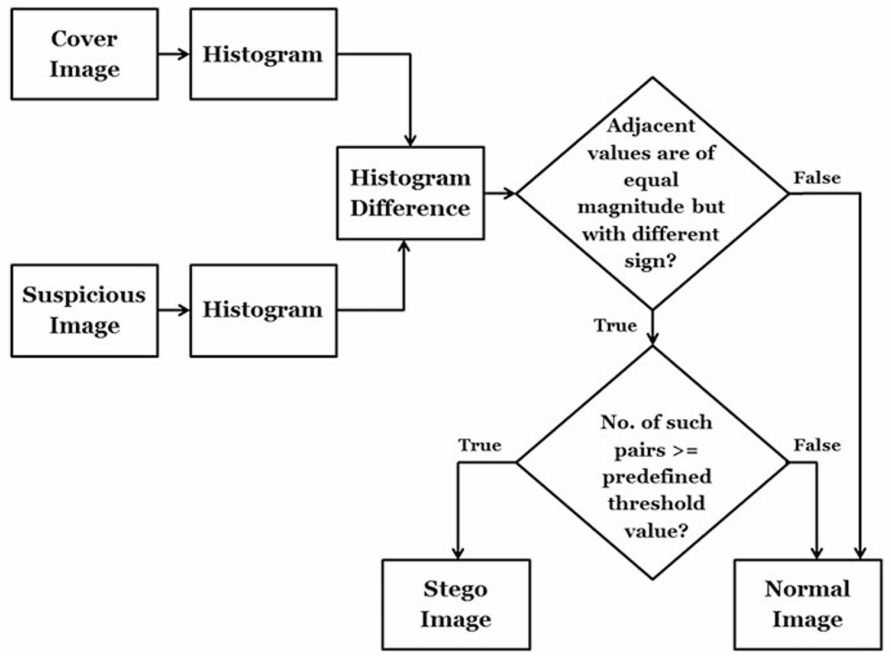
2.2.1 Algorytm

Metoda opiera się na porównaniu histogramów obrazu wyjściowego (cover image) oraz podejrzanego obrazu (suspicious image). Specyficzny wzór różnic histogramów pomaga wykrywać obrazy stego (obrazy zawierające ukryte dane). Metodę zaprojektowano tak, aby była uniwersalna, czyli niezależna od zastosowanego algorytmu steganograficznego.

Algorytm składa się z następujących kroków:

1. Odczytanie podejrzanego i wyjściowego obrazu oraz zapisanie ich wartości intensywności pikseli.

2. Obliczenie histogramów obu obrazów.
3. Analiza różnic histogramów: Szukanie wartości sąsiednich, które mają tę samą wielkość, ale różne znaki.
4. Zliczenie takich par i porównanie wyniku z ustalonym progiem.
5. Na podstawie wyniku określenie, czy obraz podejrzany to obraz stego, czy normalny obraz.



Rysunek 1: Schemat działania algorytmu opartego na histogramach.

2.2.2 Implementacja

Poniżej przedstawiono fragment programu z kodem zaimplementowanego algorytmu. Pełny kod dostępny jest w repozytorium.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

def detect_stego_image(cover_image, suspicious_image, threshold):
    # Step 1: Wczytaj obrazy i zamie  je na warto ci intensywno ci (grayscale)
    cover = np.array(Image.open(cover_image).convert('L')) / 255.0
    suspicious = np.array(Image.open(suspicious_image).convert('L')) / 255.0

```

```
    # Step 2: Oblicz histogramy obu obraz w
    cover_histogram, _ = np.histogram(cover.flatten(), bins=256, range=(0,
1))
    suspicious_histogram, _ = np.histogram(suspicious.flatten(), bins=256,
range=(0, 1))

    # Step 3: Wykres histogram w i r  nice
    plt.bar(range(256), cover_histogram, alpha=0.5, label='Cover Image
Histogram')
    plt.bar(range(256), suspicious_histogram, alpha=0.5, label='Suspicious
Image Histogram')
    plt.legend()
    plt.title("Histogram Comparison")
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    #plt.show()

    # Step 4: R  nice histogram w
    differences = suspicious_histogram - cover_histogram

    # Step 5: Zlicz r  nice z przeciwnym znakiem i t  sam  warto ci
(z tolerancj )
    counter = 0
    tolerance = 1e-6  # Tolerancja dla numerycznych b  d  w
    for i in range(len(differences) - 1):
        if abs(differences[i] + differences[i + 1]) < tolerance and
abs(differences[i]) > 0:
            counter += 1

    # Step 6: Sprawdzenie warto ci progowej licznika
    if counter > threshold:
        return True  # Stego Image
    else:
        return False  # Normal Image
```

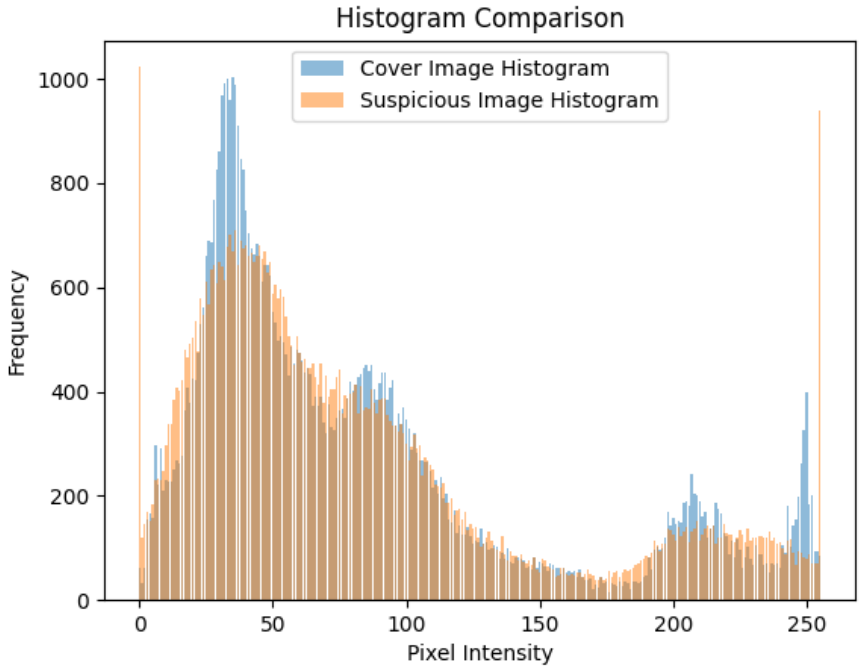
Listing 1: Steganaliza oparta na histogramach w Pythonie

Analizę wykonano dla próbek 20 obrazów stego i 20 obrazów cover pobranych z bazy BOSSBase. W celu sprawdzenia wrażliwości algorytmu na modyfikacje obrazów nie będące działaniami steganograficznymi, pliki cover poddano kompresji oraz wprowadzono losowy szum–w ten sposób otrzymano 20 próbek "compress".

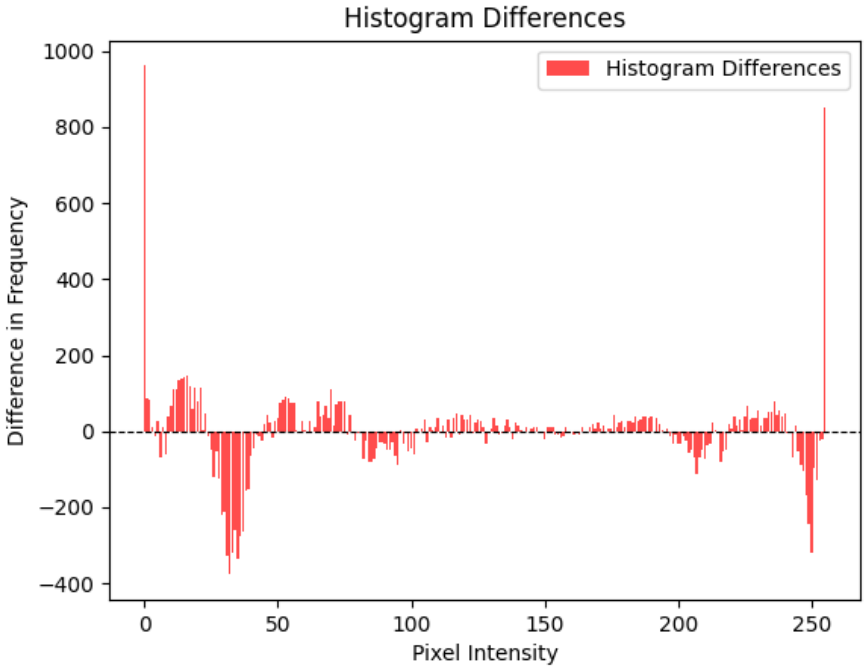
Program sprawdził pary obrazów cover–stego oraz cover–compres i zwrócił wynik detekcji dla różnych wartości progu detekcji.

2.2.3 Wyniki

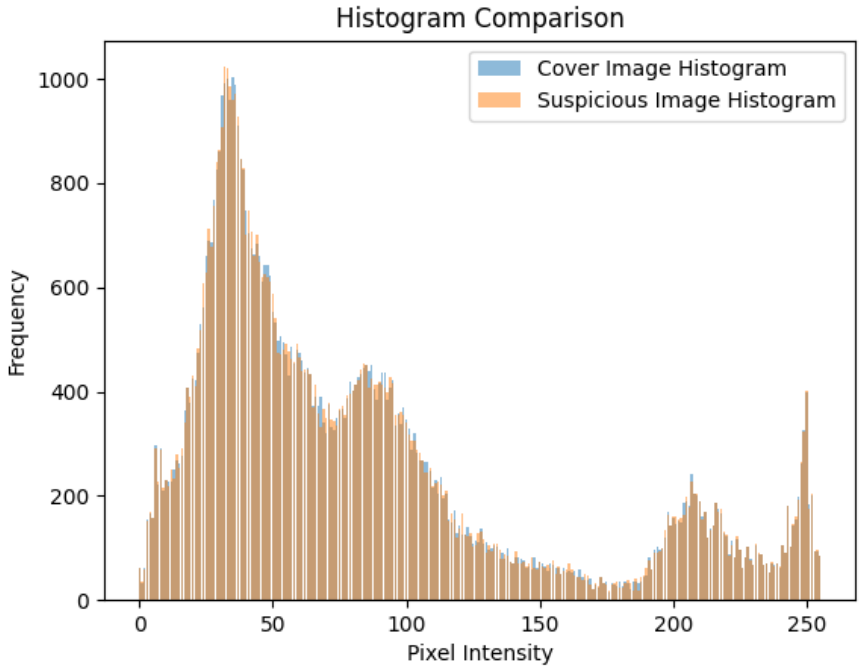
Obrazy histogramów dla jednej z 20 par przedstawiono poniżej.



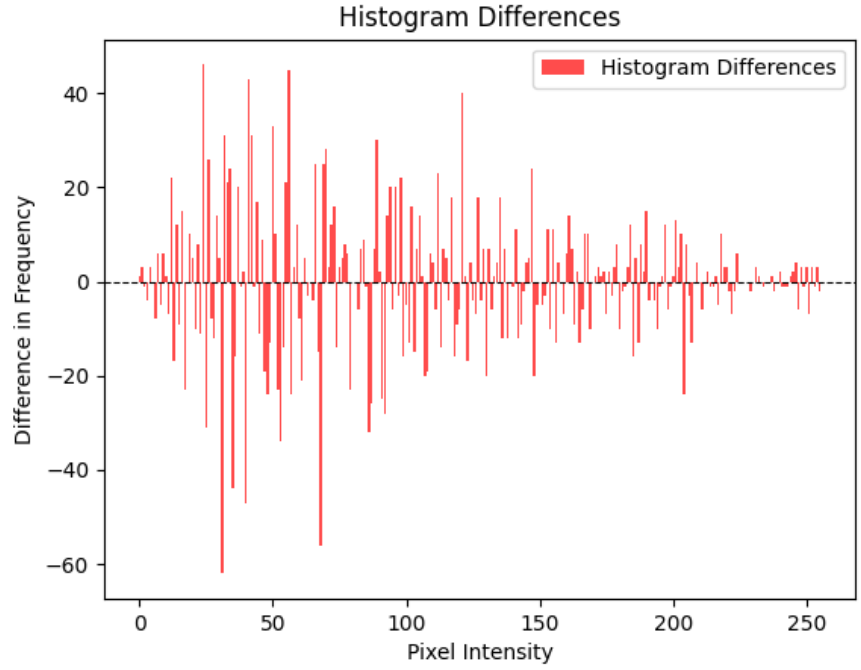
Rysunek 2: Porównanie histogramów obrazu cover i compress.



Rysunek 3: Histogram różnic między obrazami cover i compress.



Rysunek 4: Porównanie histogramów obrazu cover i suspicious.



Rysunek 5: Histogram różnic między obrazami cover i suspicious.

Próg detekcji	Poprawnie zidentyfikowane obrazy compress	Poprawnie zidentyfikowane obrazy stego
5	19 / 20	18 / 20
6	19 / 20	16 / 20
4	18 / 20	19 / 20

Tabela 1: wyniki klasyfikacji dla różnych progów detekcji.

2.2.4 Dyskusja i wnioski

Wyniki wyglądają bardzo dobrze. Program poprawnie zidentyfikował większość próbek. Sam algorytm był łatwy w implementacji oraz intuicyjny. Jest również prosty i nie wymaga zaawansowanych obliczeń, dlatego analiza trwa krótko. Dobrze też wypada wrażliwość na szum–mimo wprowadzenia zmian do obrazów, które nie były ukrytymi wiadomościami, algorytm poprawnie zidentyfikował je jako obrazy nie-stego.

Nie znany jest rodzaj steganografii, który został zastosowany do obrazów, więc nie można jednoznacznie określić uniwersalności metody.

Minusem algorytmu jest to, że opiera się na porównaniu podejrzanego obrazu z oryginałem. W praktyce utrudnieniem może być konieczność posiadania obrazu wzorcowego. Skuteczność zależy od ustawienia wartości progowej, co również może być problematyczne.

2.2.5 Podsumowanie

Algorytm zaproponowany przez autorów artykułu okazał się skuteczny w wykrywaniu obrazów stego. Istnieje też szansa rozwoju go na analizę steganografii nie tylko przestrzennej, ale też czasowej, co pozwoliłoby na rozszerzenie analiz na inne nośniki cyfrowe, takie jak dźwięk czy wideo.

3 Wykrywanie LSB Matching w obrazach za pomocą uczenia maszynowego

3.1 Opis steganografii

Jedną z najpopularniejszych metod steganografii obrazów jest *LSB*[9]. Każda z jej odmian opiera się na modyfikacji najmniej znaczącego bitu informacji opisującej kolor pikseli. Autor ukrywanej w ten sposób wiadomości musi wybrać piksele (w przypadku kodowania RGB dodatkowo kanał), które zmodyfikuje. Odbiorca tej wiadomości może ją odczytać na podstawie zmienionego obrazu oraz reguły opisującej kolejność odczytywania bitów.

LSB zawdzięcza swoją popularność co najmniej dwóm faktom. Po pierwsze jest to prostolinijna idea, łatwa do opisania. Jeszcze prostsze jest odczytanie zakodowanej wiadomości. Należy tu jednak zaznaczyć, że nie należy mylić prostolinijności idei z trywialnością implementacji. Otóż sama implementacja nie musi być trywialna, o czym napiszemy w następnym akapicie dotyczącym różnych wersji *LSB*. Druga ważna cecha tej steganografii to niemalże niemożliwe jej wykrycie za pomocą inspekcji wizualnej. Zawdzięczane jest to temu, że zmieniane są tylko najmniej znaczące bity, które mają najmniejszy wpływ na kolor widoczny na obrazie.

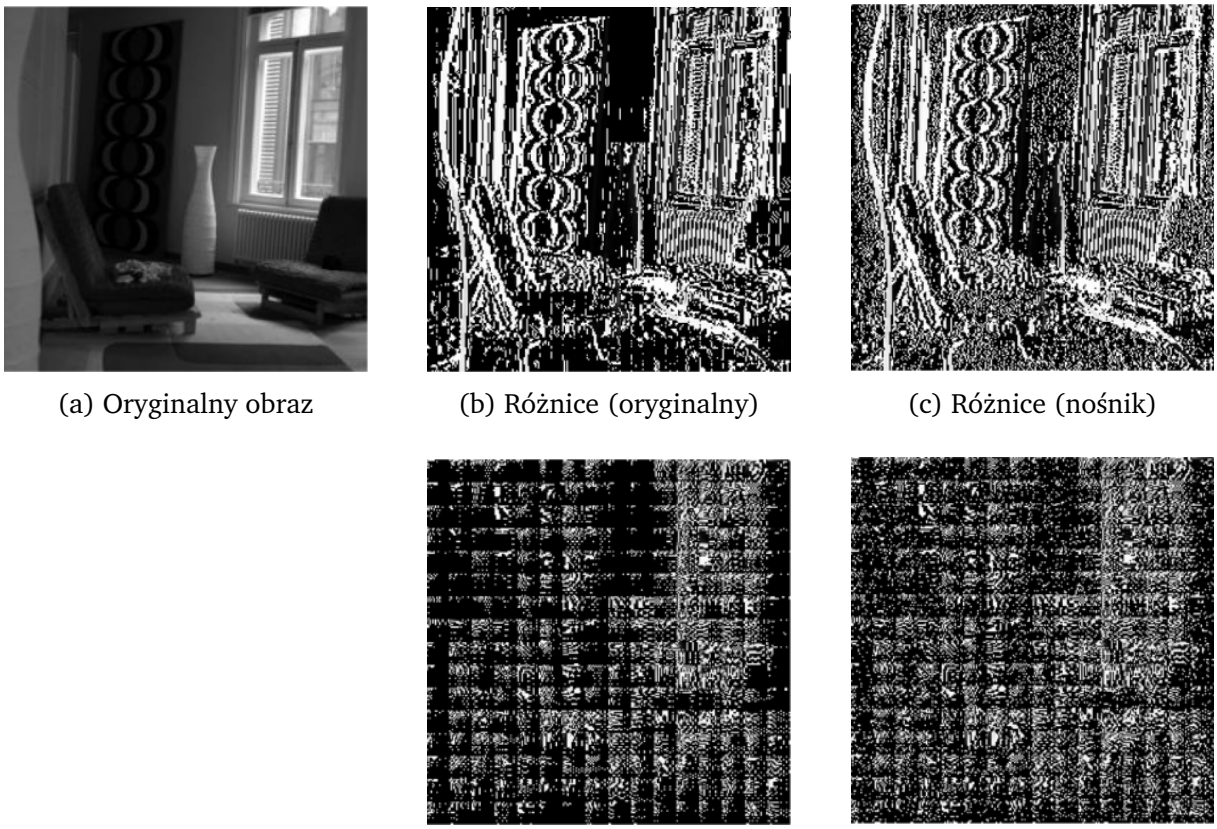
Przejdziemy teraz do omówienia szczególnej wersji *LSB* jaką jest *LSB Matching*. Zakładamy, że mamy wybrany bajt G_i, w którym chcemy zapisać część wiadomości, czyli bit m_i. Jeśli najmniej znaczący bit bajtu jest równy m_i to pozostawiamy go niezmienionego. W przeciwnym przypadku z prawdopodobieństwem $p = 0.5$ dodajemy lub odejmujemy 1 od bajtu G_i. Wyjątkami są przypadki jeśli nowa wartość bajtu nie byłaby dozwolona w danym kodowaniu. Wtedy wybierana jest dozwolona opcja. Np. w kodowaniu 8-bitowym odcieniu szarości (*8-bit grayscale*) musi zachodzić $0 <= G_i <= 255$. [18]

Takie podejście jest adaptacją innej wersji zwanej *LSB Subsitution*, w której najmniej znaczący jest po prostu zamieniany na pożądaną wartość. Taka steganografia jest jednak bardzo podatna na atak statystyczny, który polega na zbadaniu histogramu występujących wartości bajtów i np. wykonaniu testu χ^2. Metoda *LSB Matching* jest odporna na takie ataki i sprawia kłopoty analizom statystycznym przez dywersyfikacje wprowadzanych zmian.[11, 18]

Za pomocą metody *LSB* można przekazać wiadomość o rozmiarze co najwyżej takim, ile bajtów koduje dany obraz. Przykładowo, jeśli jest to obraz trójkolorowy o rozmiarze 256x256 pikseli, można za jego pomocą ukryć wiadomość o długości $3 \cdot 256 \cdot 256 = 192kb$. Jednak im więcej ukrywający wiadomość chce wykorzystać tym bardziej narażony jest na atak. To jaka część obrazu jest wykorzystywana do ukrywania będziemy nazywali jako *usage* $\mu \in [0, 1]$

3.2 Opis metody wykrywania

To że, *LSB Matching* jest odporne na ataki czysto statystyczne, nie oznacza, że nie pozostawia żadnych śladów. Poniższe rysunki pokazują jakie zmiany są obserwowalne pomiędzy czystymi a naruszonymi obrazami (patrz rysunek 6).



(a) Oryginalny obraz (b) Różnice (oryginalny) (c) Różnice (nośnik)

(d) DCT (oryginalny) (e) DCT (nośnik)

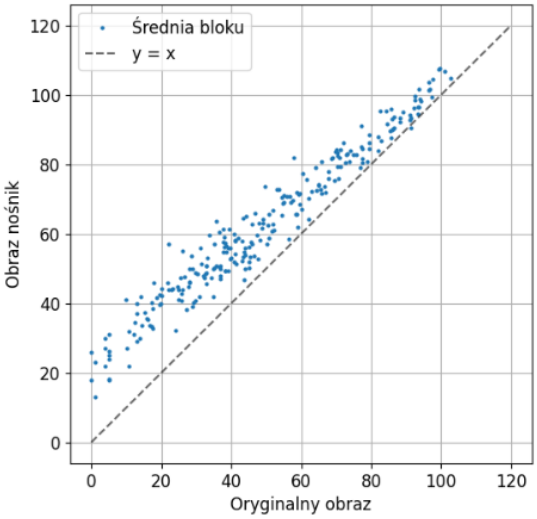
Rysunek 6: Porównanie różnic pomiędzy sąsiadującymi pikselami w poziomie oraz DCT dla obrazu oryginalnego i z zakodowaną wiadomością

Pierwsza przedstawiona transformacja to różnica pomiędzy sąsiadującymi w poziomie pikselami. Na oryginalnych obrazach występują obszary, gdzie kolor jest taki sam, przez co nie występują różnice między sąsiadującymi pikselami. Na naruszonych obrazach w tych samych obszarach występują różnice. Drugą przedstawioną transformacją jest *DCT* (*Discrete Cosine Transform*) opisana wzorem 1. Obraz dzielony jest na bloki o wymiarach NxN (na rysunku i dalej $N = 16$), z których każdy jest poddawany przekształceniu. Interpretacja współczynników DCT jest następująca. Współczynnik $X_{k,\ell}$ odpowiada tym wyższej częstotliwości pionowej (zmianom wartości bajtów w poziomie) im większe k oraz tym większej częstotliwości poziomej im większe ℓ.

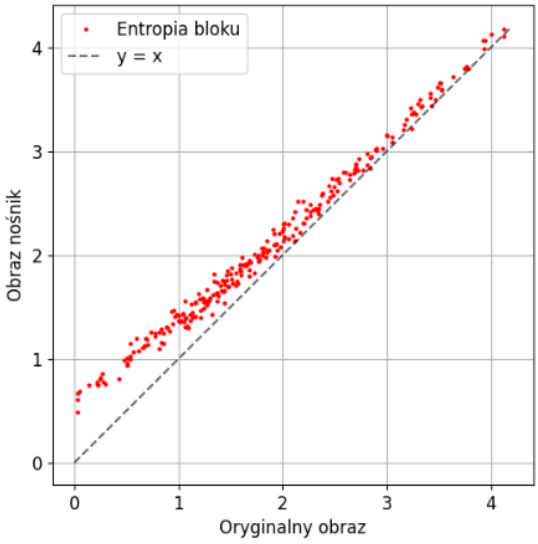
$$X_{k,\ell} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x_{m,n} \cos\left[\frac{\pi}{N}\left(m+\frac{1}{2}\right)k\right] \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\ell\right] \tag{1}$$

Kluczową obserwacją, którą wykorzystamy, jest to, że ukrywanie wiadomości w obrazie zwiększa obecność wysokich częstotliwości przestrzennych, szczególnie w obszarach o oryginalnie małych zmianach koloru.

Poza zmianami w różnicach pomiędzy sąsiadującymi pikselami oraz wysokoczęstotliwościowych składowych *DCT* wskazać należy jeszcze jedną charakterystyczną, mierzalną zmianę wprowadzaną przez *LSB Matching*. Okazuje się, bowiem, że im mniejsza jest średnia oraz entropia Shannona w blokach *DCT*, tym bardziej większa jest ona w odpowiadających blokach z naruszonych obrazów. Poniższy rysunek (patrz rysunek 9) przedstawia porównanie pomiędzy tymi samymi blokami *DCT* wziętymi z oryginalnego obrazu i nośnika. Obserwowalne jest zwiększenie się średniej i entropii Shannona. Choć takie założenie niemalże nigdy nie jest spełnione, warto wspomnieć ten fakt, że gdyby atakujący tę steganografię miał dostęp do obu obrazów i mógł je ze sobą porównać, w większości przypadków na podstawie samych tych wielkości mógłby wskazać ten z zakodowaną wiadomością.



Rysunek 7: Średnia



Rysunek 8: Entropia

Rysunek 9: Zmiany w średniej i entropii Shannona bloków DCT. Porównanie obrazu oryginalnego z nośnikiem. Każdy punkt reprezentuje odpowiadający blok o wymiarach 16x16

Najwyższa pora, aby zacząć mówić o samej metodzie wykrywania *LSB Matching*. Jak pisze na swoim blogu Daniel Lerch, jedyną możliwością wykrycia tej stenografii na podstawie samego podejrzanego obrazu są metody uczenia maszynowego [11]. Omówione powyżej cechy mogą zostać wykorzystane jako dane wejściowe do modelu klasyfikującego obraz jako nośnik bądź czysty. O tym jaki model został wybrany oraz na jaki dokładnie wektor cech został użyty w implementacji powiemy w następnej sekcji.

Zanim przejdziemy dalej warto wspomnieć o ograniczeniach i specyfikacji tej metody. Przede wszystkim opisywana metodologia opiera się na założeniu, że znana jest steganografia, czyli, że jest to *LSB Matching*. Nie można zatem zastosować wytrenowanego modelu do wykrywania innej steganografii. Ponadto, istnieją inne typy opierające się na modyfikacji naj-

mniej znaczącego bitu, które mogą implementować sposób wyboru pikseli, który jest mniej naiwny niż wybór losowy, klasyczny [9]. Do takich należy *LSB Matching Revisited* opisany przez Mielikainena w 2006 oraz jego warianty [15, 5]. Model został wytrenowany przy użyciu obrazów z wiadomościami zakodowanymi w sposób zupełnie losowy i może nie być skuteczny w wykrywaniu obrazów z wiadomością zakodowaną w pikselach wybranych na podstawie wiedzy o obrazie. Kolejnym ograniczeniem metody uczenia maszynowego jest potrzeba przygotowania zbioru obrazów zarówno czystych jak i z ukrytą wiadomością. Dodatkowo metody uczenia maszynowego często mogą wymagać większej mocy obliczeniowej, ze względu na potrzebę obróbki obrazów i ekstrakcji wyżej opisanych cech.

3.3 Implementacja

W ramach implementacji metody wykrywania *klasycznego LSB Matching* wykonano następujące kroki

1. **Przygotowanie danych i implementacja steganografii**
   Skorzystano ze zbioru BOSSbase zawierającego 10 tysięcy obrazów o wymiarach 256x256 pikseli każdy o wielkości bajta [2]. W każdym z obrazów ukryto dodatkowo wiadomość z parametrem $\mu = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$. Otrzymano w ten sposób dla każdego obrazu dodatkowo różne wersję każda z ukrytą wiadomością o innej długości. Wiadomością był losowy ciąg bitów. Ukrywany był on w losowo wybieranych pikselach

```python
import random
import numpy as np
from PIL import Image

def embedd_random_msg(image:Image, usage:float|int):
    assert 0 <= usage <= 1
    image:np.ndarray = np.array(image)
    height, width = image.shape
    msg = np.random.randint(2, size=int(height * width * usage))
    indices = np.random.choice(height * width, len(msg),
   replace=False)
    positions = np.unravel_index(indices, (height, width))
    for pos, bit in zip(zip(*positions), msg):
        grey_byte = image[pos]
        if grey_byte & 1 != bit:
            if grey_byte == 255:  # Avoid overflow
                grey_byte = 254
            elif grey_byte == 0:  # Avoid underflow
                grey_byte = 1
            else:
                if random.random() < 0.5:
                    grey_byte -= 1
                else:
                    grey_byte += 1
        image[pos] = grey_byte

```

```
    return Image.fromarray(image)

```

Listing 2: Funkcja ukrywająca wiadomość za pomocą *LSB Matching*.

2. **Przygotowanie funckji do ekstrakcji cech**
   Jako wektor cech do uczenia modelu wybrano:

   - średnia wartość współczynników z każdego z $256 \cdot 256/(N = 16)^2 = 256$) bloków *DCT*
   - entropia współczynników z każdego z bloków *DCT*
   - 1 współczynnik bloku *DCT* odpowiadający najwyższym częstotliwościom przestrzennym

   Łącznie $3 \cdot 256 = 768$ elementów wektora cech dla danego obrazu.

```
def dct(X:np.ndarray, N:int = 16) -> np.ndarray:
    # source: https://dev.to/marycheung021213 \
    #   /understanding-dct-and-quantization-in-jpeg-compression-1col
    DCT=np.zeros((N,N))
    for m in range(N):
        for n in range(N):
            if m==0:
                DCT[m][n]=math.sqrt(1/N)
            else:

   DCT[m][n]=math.sqrt(2/N)*math.cos((2*n+1)*math.pi*m/(2*N))
    y_size, x_size = X.shape
    X = np.pad(X, constant_values=0, pad_width=((0, (N-y_size%N)%N),
   (0, (N-x_size%N)%N)))
    y_size, x_size = X.shape
    X_dct = np.zeros_like(X)
    for i in range(0, y_size, N):
        for j in range(0, x_size, N):
            X_dct[i:i+N, j:j+N] = DCT @ X[i:i+N, j:j+N] @ DCT.T
    blocks = X_dct.reshape(-1, N, N)

    return blocks

def features_from_image(image:np.ndarray|Image.Image):
    X_dct = dct(np.array(image))
    X = []
    for block in X_dct:
        X.append(np.mean(block))
        X.append(entropy(block))
        for i in range(1, 2): # high freqs
            for j in range(1, 2):
                X.append(block[-i, -j])
    return X

```

Listing 3: Funkcja wykonujące transformację *DCT* oraz oraz funkcja do ekstrakcji cech z pozyskanych w tenb sposób bloków

3. **Wybór algorytmu i uczenie modelu**
   Jako model do klasyfikacji obrazu wybrano *las losowy*. *Las losowy* jest *ensemblem* składającym się z drzew decyzyjnych konstruowanych na podstawie losowo wybieranych cech. Ostatecznie, klasyfikacja polega na demokratycznym wyborze większościowym spośród wszystkich estymatorów. W implementacji użyto n=100 drzew decyzyjnych.[3]
   Do uczenia użyto po 100 obrazów czystych i nośników oraz z każdego przygotowanego wykorzystania μ, czyli łącznie 2000 obrazów

Kod wykorzystany do implementacji znajduje się w repozytorium w folderze *\image_lsbm_ml*. Wymagania środowiska Python znajdują się w pliku `pyproject.toml`.

3.4 Wyniki, interpretacja i wnioski

Model został przetestowany na zbiorze testowym oraz dla różnych stopni wykorzystania pikseli μ. Utworzony w ten sposób wykres przedstawiony jest na rysunku 10. Czułość i swoistość są obliczane według odpowiednio wzorów 2 i 3, gdzie TP - poprawnie wskazane steganografie, FP - fałszywie wskazane steganografie, FN - fałszywie odrzucone steganografie, TN - poprawnie odrzucone steganografie.

$$C = \frac{TP}{TP + FN} \tag{2}$$

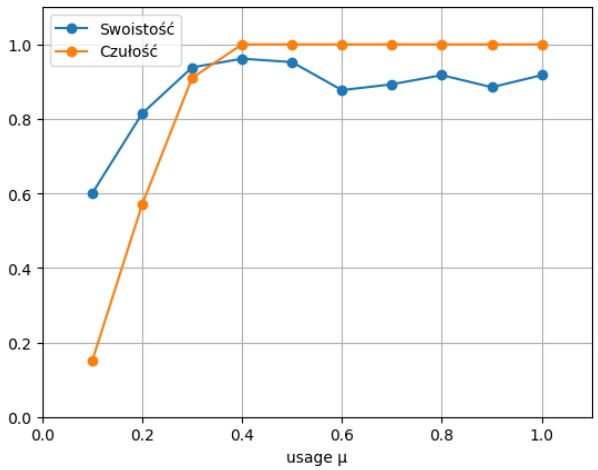
$$S = \frac{TP}{TP + FP} \tag{3}$$

Im wyższa czułość tym rzadziej model nie zauważa steganografii. Jest to cecha pożądana ze względu na potrzebę wskazania obrazów, które mogą potencjalnie przenosić informacje. Zauważamy, że im więcej pikseli zostało wykorzystanych do przekazania wiadomości, tym lepiej model radzi sobie ze wskazywaniem ich. Przy wykorzystaniu pikseli μ większym niż około 40% każda steganografia jest zauważona. Model nie radzi sobie z mniej zmodyfikowanymi obrazami co jest zgodne z intuicją.

Im wyższa z kolei swoistość tym rzadziej model wszczyna fałszywy alarm. Jest to znów pożądana cecha szczególnie, gdy chcemy wszcząć zaawansowane procedury przy wykryciu steganografii. Ta wielkość rośnie dla małych μ a następnie oscyluje w okolicy 0.9.

Wyniki modelu można ocenić pozytywnie ze względu na wysokie wartości zarówno swoistości jak i czułości. Słaba skuteczność w przypadku małych μ nie jest zatrważający szczególnie jeśli weźmie się pod uwagę, że przy tak małej dostępnej długości wiadomości niewiele da

się przekazać. Jednocześnie warto przyznać, że jest to niedoskonałość i dalsze badania udoskonalające proces uczenia i model mogłyby postawić za cel zmienienie zniwelowanie tego efektu.



Rysunek 10: Czułość i swoistość wytrenowanego modelu *lasu losowego* w funkcji wykorzystania pikseli μ

4 Wykrywanie Echo Hiding poprzez analizę cepstrum mocy sygnału

4.1 Echo Hiding

Jednym z najczęstszych podejść w dziedzinie steganografii audio jest metoda zwana *Echo Hiding*, *EH*. Metoda ta wykorzystuje właściwości dźwięku, dzięki którym dodatkowe echo można wprowadzić w sposób niesłyszalny dla ludzkiego ucha, jednocześnie kodując w nim wiadomość [6].

Podstawowa idea EH polega więc na dodaniu niewielkiego, kontrolowanego opóźnienia do oryginalnego sygnału audio. W celu zakodowania wiadomości, sygnał audio $x(n)$ poddawany jest operacji splotu z funkcją impulsową $h(n)$, która definiuje właściwości echa. Matematycznie proces ten można zapisać jako:

$$y(n) = h(n) * x(n), \tag{4}$$

gdzie operator $*$ oznacza splot, a $h(n)$ jest funkcją kernela zdefiniowaną jako:

$$h(n) = \delta(n) + a\delta(n-d). \tag{5}$$

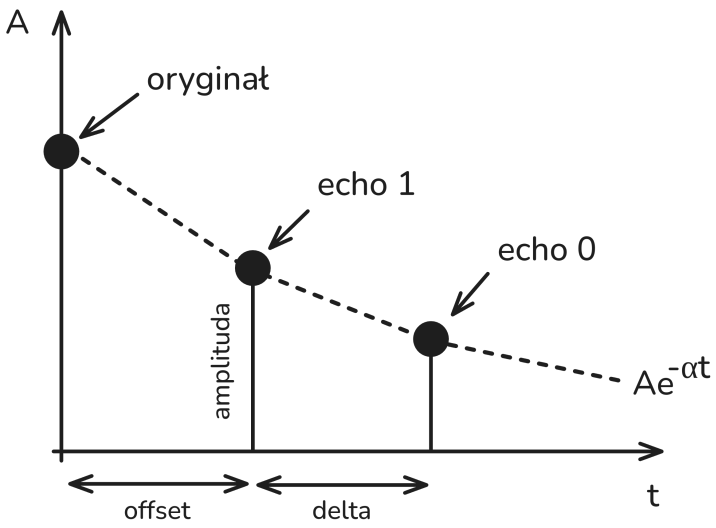
W powyższym równaniu $\delta(n)$ oznacza deltę Diraca, a jest współczynnikiem tłumienia echa ($0 < a < 1$), a d to opóźnienie echa wyrażone w ilości próbek.

Wartości d oraz a są zwykle dobierane w taki sposób, aby echo pozostało niesłyszalne, jednocześnie zapewniając wystarczające właściwości by umożliwić dekodowanie ukrytej wiadomości.

Przed kodowaniem, wejściowy sygnał audio dzieli się na segmenty o stałej długości N, w taki sposób, by każdy zawierał echo reprezentujące jeden bit wiadomości. Zamienia się więc ją na reprezentację binarną, a następnie koduje poprzez użycie dwóch różnych opóźnień: d_0, dla bitu "0", oraz d_1, dla bitu o wartości "1".

Zatem, osadzane dane są definiowane za pomocą czterech głównych parametrów echa (11):

1. amplituda echa (A),
2. współczynnik zanikania (α),
3. opóźnienie dla bitu "1" (*offset*),
4. opóźnienie dla bitu "0" ($offset + delta$).



Rysunek 11: Parametry EH.

4.2 Dekodowanie

Opisany poniżej proces dekodowania został wykonany na przykładzie ukrytego echo o parametrach: $d_1 = 48$ próbek, $d_0 = 96$ próbek, i długości segmentu $N = 1078$ próbek, w pliku audio z nagraną ludzką mową.

4.2.1 Analiza cepstralna i cepstrum mocy

Najczęstszą procedurą wykorzystywaną do wykrywania echa jest analiza cepstralna. Obliczenie tzw. cepstrum $C_y(n)$ pozwala zidentyfikować opóźnienia dodanego echa, ponieważ zawiera wyraźne szczyty w odpowiadających im pozycjach.

Cepstrum $C_y(n)$ definiowane jest jako odwrotna transformata Fouriera logarytmu widma amplitudowego sygnału:

$$C_y(n) = \mathscr{F}^{-1}\big(\ln|\mathscr{F}(y(n))|\big), \tag{6}$$

gdzie $\mathscr{F}$ oznacza transformatę Fouriera, $y(n)$ to sygnał z ukrytym echem, a $\mathscr{F}^{-1}$ oznacza odwrotną transformatę Fouriera.

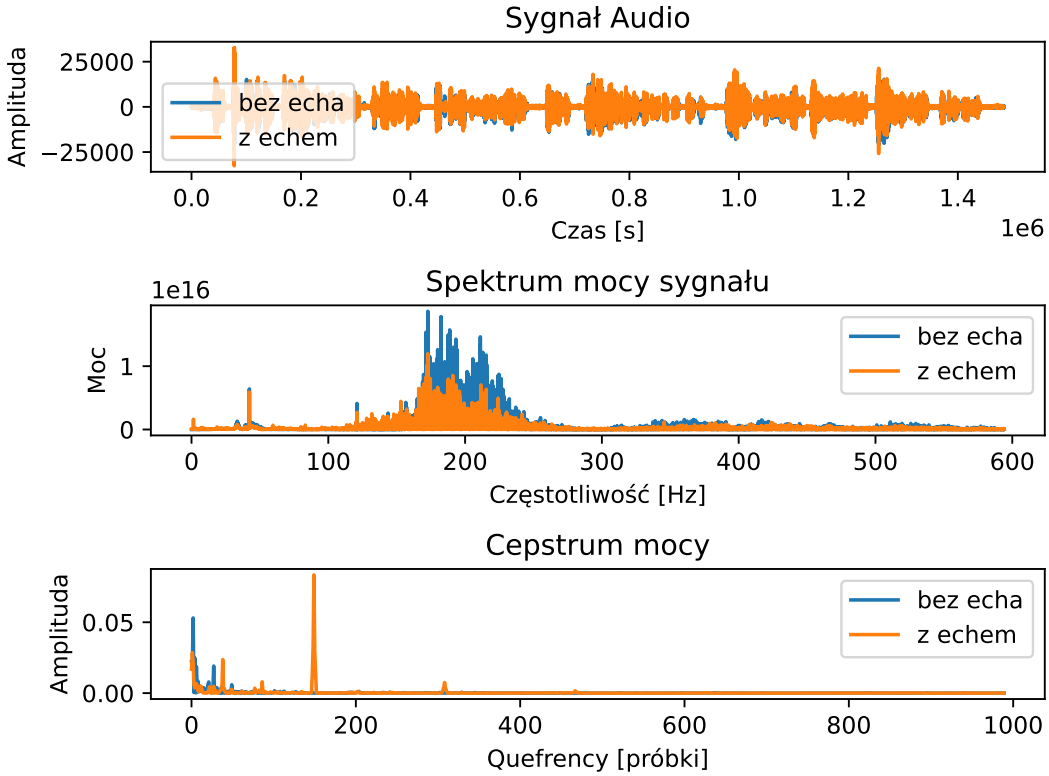
Można również zdefiniować tzw. cepstrum mocy (ang. *power cepstrum*), które daje lepsze wyniki niż samo cepstrum [16]. Oblicza się je jako odwrotną transformatę Fouriera logarytmu mocy widma sygnału:

$$P_y(n) = \left|\mathscr{F}^{-1}\big(\ln|\mathscr{F}(y(n))|^2\big)\right|^2, \tag{7}$$

```
def power_cepstrum(audio):
    spectrum = np.abs(np.fft.fft(audio)) ** 2
    log_spectrum = np.log(spectrum + np.finfo(float).eps)
    power_cepstrum = np.abs(np.fft.ifft(log_spectrum)) ** 2
```

```
5     return power_cepstrum
```

Listing 4: Fragment kodu liczącego cepstrum mocy.

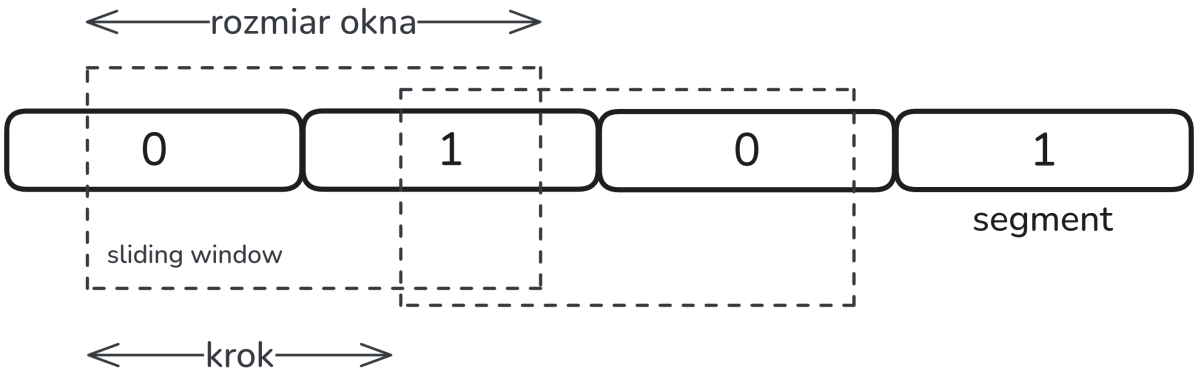


Rysunek 12: Porównanie reprezentacji sygnału z echem i bez echa.

4.2.2 Estymacja opóźnień echa

Dokonując steganalizy jako potencjalny atakujący, nie mamy pojęcia o użytych parametrach do kodowania echem. Natomiast, możliwa jest ich estymacja poprzez wykorzystanie analizy cepstrum mocy z przesuwanym oknem (ang. *sliding window*) [19].

W tej metodzie sygnał analizowany jest segment po segmencie za pomocą okna o określonym rozmiarze τ, które przesuwa się po sygnale z krokiem k (13). Dla każdego segmentu obliczane jest cepstrum mocy, a następnie lokalizowane są szczyty odpowiadające potencjalnym opóźnieniom d_0 i d_1. Po przesunięciu okna przez cały sygnał tworzy się histogram lokalizacji pików w celu identyfikacji najbardziej prawdopodobnych wartości opóźnień.



Rysunek 13: Wizualizacja algorytmu okna przesuwnego.

Algorytm został zaimplementowany w klasie `EchoDetector`, której wejściowe parametry to:

- `window_size` – długość okna analizy w próbkach,
- `step_size` – krok przesuwania okna w próbkach,
- `min_delay` – minimalny czas opóźnienia, który będzie analizowany,
- `max_delay` – maksymalny czas opóźnienia, który będzie analizowany.

Cepstrum mocy jest obliczane dla kolejnych segmentów sygnału, wyznaczanych za pomocą okna przesuwnego, z dodatkowo aplikowaną funkcją Hamminga. Metoda `iter_cepstrums` generuje cepstrum dla kolejnych okien:

```
def iter_cepstrums(self, audio, *, window_size=None, step_size=None):
    window_size = window_size or self.window_size
    step_size = step_size or self.step_size
    for idx in range(0, (audio.size - window_size) + 1, step_size):
        window = audio[idx : idx + window_size]
        window = window * np.hamming(window_size)
        yield power_cepstrum(window)
```

Listing 5: Generator liczący cepstrum mocy dla poszczególnych okien przesuwnych.

Na podstawie obliczonego cepstrum, metoda `estimate_delays` wyznacza histogram lokalizacji maksimów w przedziale `min_delay`–`max_delay`. Dwa najczęściej występujące maksima (`d0` i `d1`) są uznawane za potencjalne opóźnienia echa:

```
counts, bins = np.histogram(
    peak_locations,
    bins=(self.max_delay - self.min_delay),
    range=(self.min_delay, self.max_delay),
)
top_2 = np.argsort(counts)[-2:]
d0, d1 = bins[top_2]
```

Listing 6: Tworzenie histogramu z lokalizacji pików.

4.2.3 Współczynnik CPLAR

Jedną z kluczowych wielkości oceniających jakość detekcji opóźnień jest współczynnik **CPLAR** (*Cepstrum Peak Location Aggregation Rate*) [19]. Jest definiowany jako stosunek liczby okien cepstrum, w których piki zostały poprawnie zlokalizowane w pozycjach d_0 lub d_1, do całkowitej liczby analizowanych okien.

```
cplar = sum(counts[top_2]) / len(peak_locations)
```

Listing 7: Obliczanie współczynnika CPLAR.

4.2.4 Estymacja długości segmentu

Po otrzymaniu opóźnień echa można wyznaczyć długość segmentu w przybliżony sposób, poprzez iterowanie po potencjalnych długościach i wyznaczanie na podstawie każdego segmentu średnią sumę wartości cepstrum w lokalizacjach `d0` i `d1` [19]:

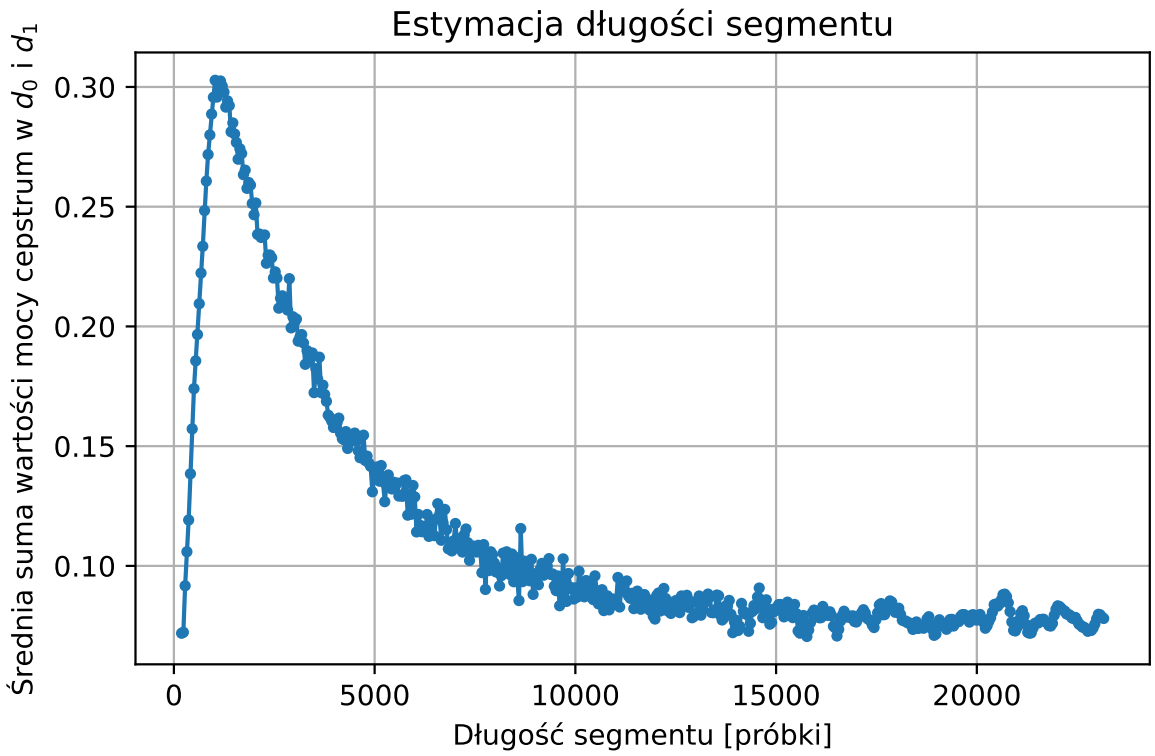
```
avg_sum = np.mean(
    [
        cepstrum[d0] + cepstrum[d1]
        for cepstrum in self.iter_cepstrums(
            audio,
            window_size=est_length,
            step_size=est_length,
        )
    ]
)
```

Listing 8: Średnia z sum wartości pików z każdego segmentu.

Ostateczna, estymowana długość segmentu to ta, dla której średnia suma wartości mocy w lokalizacjach `d0` i `d1` jest największa (14):

```
return lengths[np.argmax(avg_sums)]
```

Listing 9: Branie długości, której odpowiada średnia suma o maksymalnej wartości



Rysunek 14: Wykres średniej sumy pików od badanej, potencjalnej długości segmentu.

Na koniec, znając już wszystkie potrzebne parametry można obliczyć długości segmentów N, z jakimi zakodowana jest wiadomość.

```
num_bits = result.size // segment_len
bits = np.zeros(num_bits).astype(np.uint8)

for idx, cepstrum in zip(
    range(num_bits),
    detector.iter_cepstrums(
        result,
        window_size=segment_len,
        step_size=segment_len,
    ),
):
    peak_one = d1 < cepstrum.size and cepstrum[d1] or 0
    peak_zero = d0 < cepstrum.size and cepstrum[d0] or 0
    bits[idx] = int(peak_one < peak_zero)
```

Listing 10: Ostateczna procedura dekodowania wiadomości.

4.3 Podsumowanie

W ramach projektu zaimplementowano algorytmy pozwalające na detekcję opóźnień d_0 i d_1, oraz oszacowanie długości segmentów N. Kluczowe równania, takie jak obliczanie cepstrum mocy oraz proces analizy z przesuwanym oknem, zostały zaimplementowane w środowisku Python.

5 Wykrywanie LSB-substitution na podstawie wzorców współczynników DCT obrazów

5.1 Zarys metody

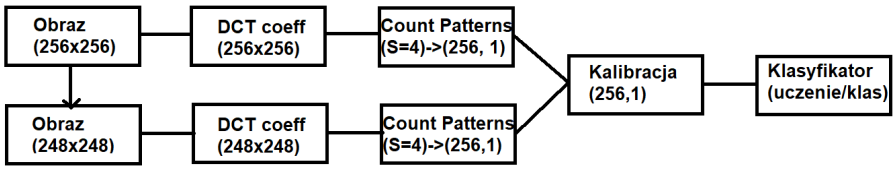
Ta część projektu polegała na zaimplementowaniu metody detekcji steganografii opisanej w pracy [1], działającej na obrazach. Metoda opiera się na wyznaczeniu cech obrazu służących jako dane wejściowe klasyfikatora. Cechy są wyznaczane poprzez porównywanie wzorców występujących we współczynnikach DCT obrazu przed i po operacji croppingu. Okazuje się że obrazy poddane steganografii (stego-images) mają inny rozkład cech niż oryginalne obrazy (cover-images), co pozwala na skuteczną klasyfikację.

5.2 Wykorzystane Dane

Metoda ta w ogólności znajduje zastosowanie w tak zwanej Blind-Steganalysis, czyli próbie detekcji steganografii przy braku wiedzy o metodzie wykorzystywanej do ukrycia wiadomości. W mojej implementacji skupiłem się na detekcji wiadomości ukrytych za pomocą prostego LSB-substitution, w obrazach o wymiarach 256x256 pikseli, w odcieniu szarości. W tym celu zaimplementowałem algorytm przeprowadzający LSB-substitution dla obrazów (.png), a następnie dla zbioru obrazów o podanych wymiarach, wygenerowałem ich wersje stego, gdzie ukrywałem wiadomości, o długości odpowiadającej użyciu około 15% dostępnego nośnika.

5.3 Opis metody

Ogólny schemat działania algorytmu prezentuje diagram 15.



Rysunek 15: Schemat uczenia/klasyfikacji.

W fazie uczenia obrazy wczytywane są wraz z odpowiadającymi im klasami (stego albo cover). Następnie wyznaczane są współczynniki DCT oraz zliczane są ilości występujących w nich wzorców. Te same operacje powtarzane są dla "okrojonej" (cropped) wersji obrazu. Poprzez kalibrację, polegającej na porównaniu ilości wzorców w oryginalnym obrazie i jego

okrojonej wersji wyznaczany jest wektor cech, który przekazywany jest do klasyfikatora. W fazie uczenia, do klasyfikatora przekazywana jest również klasa obrazu, a parametry klasyfikatora są aktualizowane na podstawie danych. W fazie klasyfikacji, wektor cech bez klasy przekazywany jest do nauczonego już klasyfikatora w celu wykonania predykcji klasy.

Okrojenie obrazu (Image Cropping), polega na stworzeniu nowego obrazu będącego wycinkiem poprzedniego. W zaimplementowanej metodzie, obraz "obcina" się o 4 piksele wzdłuż każdej krawędzi, otrzymując tym samym nowy obraz o wymiarach:

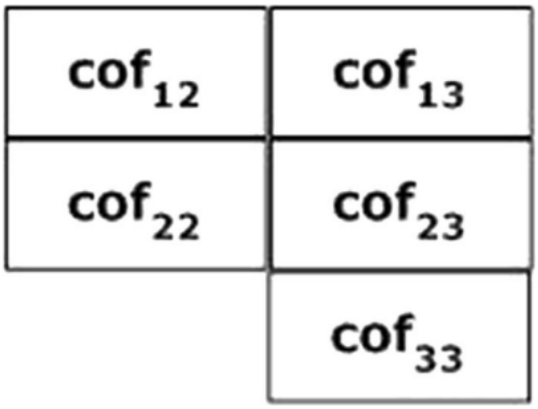
$$(H_{crop}, W_{crop}) = (H - 8, W - 8) \tag{8}$$

gdzie H i W to wymiary oryginalnego obrazu.

Następnie, dla obu obrazów wyznacza się ich współczynniki DCT. Z otrzymanych macierzy współczynników, dla obu obrazów zlicza się wzorce zdefiniowane przez następujący algorytm opisany w pracy [1]. Dany jest parametr S algorytmu. Różnica między dwoma współczynnikami x, y zdefiniowana jest jako:

$$d(x, y) = \max(|x - y|, S - 1) \tag{9}$$

co zapewnia ograniczenie górne odległości. Zgodnie z oryginalną pracą, w implementacji zastosowano wartość parametru $S = 4$. Po całej macierzy współczynników obrazu iteruje się ramką w postaci pokazanej na rysunku 16.

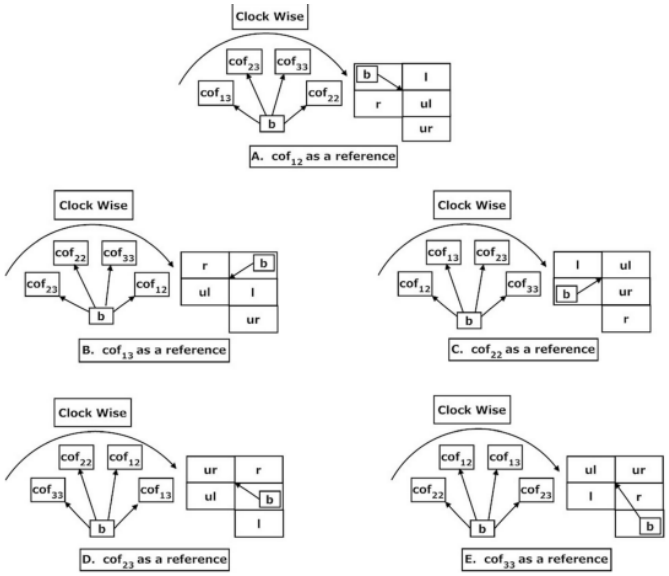


Rysunek 16: Ramka schematów. Źródło:DOI: 10.1007/s11042-017-4676-z

Dla danej pozycji ramki wyznacza się dwie wartości referencyjne (minimalną oraz maksymalną wartość współczynnika w ramce). Następnie dla każdej wartości referencyjnej oblicza się odległości między wartością referencyjną a pozostałymi wartościami w ramce, oraz tworzy się wzór P będący 4-elementowym wektorem postaci:

$$P = (d(l, b), d(ul, b), d(ur, b), d(r, b)) \tag{10}$$

gdzie b oznacza wartość referencyjną, a l, ul, ur, r to wartości pozostałych współczynników ramki. Kolejność tych współczynników zależy od tego, na jakiej pozycji znajdował się współczynnik referencyjny. Wpływ pozycji współczynnika referencyjnego na kolejność współrzędnych wektora P pokazuje rysunek 17.



Rysunek 17: Pozycja współczynnika referencyjnego, a wektor wzorca. Źródło:DOI: 10.1007/s11042-017-4676-z

Tak ustalony schemat P mapujemy na liczbę postaci:

$$M(P,S) = P[1]S^3 + P[2]S^2 + P[3]S + P[4] \tag{11}$$

gdzie $P[i]$ to i-ta współrzędna wektora schematu. W efekcie dla danej pozycji ramki otrzymujemy 2 wartości: $M(P_{max}, S)$ oraz $M(P_{min}, S)$.

Zliczanie wzorców dla danego obrazu polega na inicjalizacji wektora T o długości S^4 (w naszym wypadku 256) zerami. Podczas iteracji ramki po macierzy współczynników dla każdej pozycji wyznaczamy dwie wartości M_{max} oraz M_{min}, zgodnie z opisaną metodą, a następnie aktualizujemy wektor T:

$$T[M_{max}] \mathrel{+}= 1 \wedge T[M_{min}] \mathrel{+}= 1 \tag{12}$$

Kalibracja polega na otrzymaniu wektora cech na podstawie wektorów T_{org} i T_{crop}, odpowiednio obrazu oryginalnego i okrojonego. Zauważmy, że różnica w wymiarach tych obrazów nie ma wpływu na wymiary wektorów T_{org}, T_{crop} i oba te wektory są długości S^4. Wektor F cech obrazu jest zdefiniowany jako:

$$F[i] = \frac{T_{org}[i]}{T_{crop}[i]} i = 0, ..., S^4 - 1 \tag{13}$$

Następnie przeprowadzamy jeszcze normalizację wektora cech:

$$F[i] = \frac{F[i] - min}{max - min} \tag{14}$$

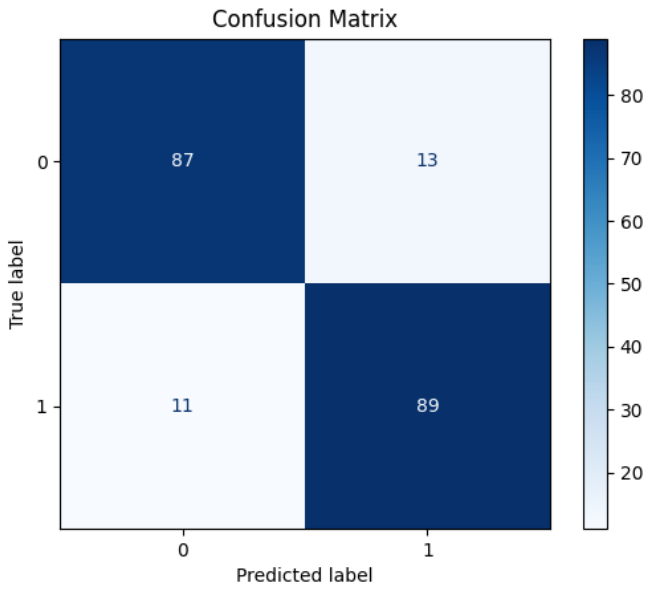
gdzie min i max to odpowiednio najmniejsza i największa wartość wektora cech przed normalizacją. Tak otrzymany wektor cech obrazu przekazujemy do klasyfikatora w celu uczenia bądź klasyfikacji.

5.4 Klasyfikator oraz Wyniki

W implementacji użyłem klasyfikatora SVC (Support Vector Classifier), o kernelu Gaussowskim. W celu uzyskania optymalnych parametrów przeprowadziłem przeszukanie (GridSearch) po siatce:

$$C \in \{2^{-5}, 2^{-4}, ..., 2^{15}\} \wedge \gamma \in \{2^{-15}, 2^{-14}, ..., 2^{3}\} \tag{15}$$

Zbiór danych (500 obrazów cover i 500 obrazów stego) podzieliłem na zestaw uczący/testowy w proporcji 800/200, z zachowaniem balansu klas (po 400 obrazów stego i cover dla zestawu uczącego i po 100 dla zestawu testowego). Przeszukiwanie siatki parametrów przeprowadzane było przez cross-walidację z parametrem `kFold = 5`. Na rysunku 18 przedstawiono macierz pomyłek dla zestawu testowego.



Rysunek 18: Macierz pomyłek dla najlepszego klasyfikatora. Klasa 0 odpowiada obrazom cover, a klasa 1 obrazom stego.

Osiągnięto precyzję na poziomie **88%**, w tym czułość na poziomie **89%** i swoistość **87%**.

6 Implementacja metody LSB Replacing w obrazach oraz weryfikacja za pomocą standardowych i niestandardowych testów

6.1 Technika zastępowania najmniej znaczącego bitu

LSB replacement technique (pol., *Technika zastępowania najmniej znaczącego bitu* lub *technika zamiany najmniej znaczącego bitu*) jest metodą steganograficzną polegającą na modyfikacji najmniej znaczących bitów pikseli obrazu w celu ukrycia informacji. Technika ta opiera się na algorytmie, który jest wystarczająco szybki, prosty i efektywny.

6.2 Algorytm

- Przekształcamy wybrany obrazek (w którym chcemy ukryć wiadomość) do postaci binarnej, czyli ciągu zer (0) i jedynek(1);
- dla każdego piksela obrazu przechodzimy przez każdy kanał kolorów RGB,
- modyfikujemy najmniej znaczący bit danego kanału w pikselu tak, aby pasował do bitu wiadomości, która chcemy ukryć;
- w sposób, opisany powyżej, ukrywamy dane;
- odpowiednio w odwrócony sposób odszyfrujemy zakodowana wcześniej wiadomość.

6.3 Wyjaśnienie implementacji oraz statystycznych testów weryfikacyjnych

6.3.1 Implementacja algorytmu LSB (szyfrowanie wraz z deszyfrowaniem)

Implementację algorytm zaczęto od napisanie funkcji, wykonującej zaszyfrowanie tekstu w wybranym obrazie. Wiadomość tekstu została przekształcona na 8-bitowy ciąg binarny. Następnie, przechodząc przez każdy piksel odpowiedniego koloru w odcienie RGB, zmieniono najmniej znaczący jego bit na odpowiedni bit ukrywanej wiadomości. Następnym krokiem w odwrotnej kolejności zaimplementowano funkcje deszyfrującą. Jako wyniki deszyfrowania otrzymujemy równie tą wiadomość, którą została zaszyfrowana.

6.3.2 Weryfikacja zaimplementowanej metody testami χ^2 i autokorelacji

Pierwszy test statystyczny, który przeprowadzono to standardowy test χ^2, który ocenia zmienność histogramu obrazu przed ukryciem w nim wiadomości i odpowiednio po ukryciu.

Test χ^2 nie wykazał znacznych różnic w przypadku wejściowego i wyjściowego obrazu, co jest charakterne dla zastosowanej metody LSB z tego powodu, że zmienia ona tylko najmniej znaczące bity, co powoduje szczególnie bardzo małe zmiany w obrazie, a z kolei metoda χ^2 jest czuła na duże zmiany.

Drugi test, który przeprowadzono to test autokorelacji, który wykrywa, jak zmienia się położenie pikseli w obrazu po zakodowaniu wiadomości. Ten test policzył korelację równą 0,9932. Co wciąż oznacza duże podobieństwo pomiędzy obrazem wejściowym a wyjściowym, jednak już wykrywa powstałe zmiany.

Źródła

[1] Rabee A., Mohamed M., and Mahdy Y. “Blind JPEG steganalysis based on DCT coefficients differences”. In: *Multimedia Tools and Applications* 77 (2018), pp. 7763–7777. DOI: 10.1007/s11042-017-4676-z.

[2] *BOSSbase*. URL: https://www.kaggle.com/datasets/lijiyu/bossbase. (dostęp: 2 lutego 2025).

[3] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[4] Bismita Choudhury, Rig Das, and Arup Baruah. “A Novel Steganalysis Method Based on Histogram Analysis”. In: (2015). Ed. by Hamzah Asyrani Sulaiman et al., pp. 779–789.

[5] Mansoor Fateh, Mohsen Rezvani, and Yasser Irani. “A New Method of Coding for Steganography Based on LSB Matching Revisited”. In: *Security and Communication Networks* 2021.1 (2021), p. 6610678. DOI: https://doi.org/10.1155/2021/6610678. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/6610678. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/6610678.

[6] Daniel F. Gruhl, Anthony Lu, and Walter Bender. “Echo Hiding”. In: *Information Hiding*. 1996. URL: https://api.semanticscholar.org/CorpusID:14156378.

[7] Neil F. Johnson and Sushil Jajodia. “Exploring steganography: Seeing the unseen”. In: *Computer* 31.2 (1998), pp. 26–34. DOI: 10.1109/MC.1998.4655281.

[8] Md Khalid, Kamiya Arora, and Naina Pal. “A Crypto-Steganography: A Survey”. In: *International Journal of Advanced Computer Science and Applications* 5 (Aug. 2014). DOI: 10.14569/IJACSA.2014.050722.

[9] Meike Helena Kombrink, Zeno Jean Marius Hubert Geradts, and Marcel Worring. “Image Steganography Approaches and Their Detection Strategies: A Survey”. In: *ACM Comput. Surv.* 57.2 (Oct. 2024). ISSN: 0360-0300. DOI: 10.1145/3694965. URL: https://doi.org/10.1145/3694965.

[10] Grzegorz Kozieł. “Współczesne techniki steganograficzne w dźwięku”. In: 2014. URL: https://api.semanticscholar.org/CorpusID:61019036.

[11] Daniel Lerch. *LSB Matching and Matrix Embedding*. URL: https://daniellerch.me/image-stego-lsbm/. (dostęp: 2 lutego 2025).

[12] Wen-Bin Lin, Tai-Hung Lai, and Chao-Lung Chou. “Chi-square-based steganalysis method against modified pixel-value differencing steganography”. In: *Arabian Journal for Science and Engineering* 46 (Apr. 2021). DOI: 10.1007/s13369-021-05554-2.

[13] Xiangyang Luo, Bin Liu, and Fenlin Liu. “Improved RS Method for Detection of LSB Steganography”. In: (2005). Ed. by Osvaldo Gervasi et al., pp. 508–516.

[14] Kristian D. Michaylov and Dipti K. Sarmah. “Steganography and steganalysis for digital image enhanced Forensic analysis and recommendations”. In: *Journal of Cyber Security Technology* 0.0 (2024), pp. 1–27. DOI: 10.1080/23742917.2024.2304441. eprint: https://doi.org/10.1080/23742917.2024.2304441. URL: https://doi.org/10.1080/23742917.2024.2304441.

[15] J. Mielikainen. “LSB matching revisited”. In: *IEEE Signal Processing Letters* 13.5 (2006), pp. 285–287. DOI: 10.1109/LSP.2006.870357.

[16] Chen Ning and Zhu Jie. “A Novel Echo Detection Scheme based on Autocorrelation of Power Cepstrum”. In: *2007 Second International Conference on Communications and Networking in China*. 2007, pp. 554–558. DOI: 10.1109/CHINACOM.2007.4469451.

[17] Arooj Nissar and A.H. Mir. “Classification of steganalysis techniques: A study”. In: *Digital Signal Processing* 20.6 (2010), pp. 1758–1770. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2010.02.003. URL: https://www.sciencedirect.com/science/article/pii/S1051200410000412.

[18] Tanmoy Sarkar and Sugata Sanyal. *Steganalysis: Detecting LSB Steganographic Techniques*. 2014. arXiv: 1405.5119 [cs.MM]. URL: https://arxiv.org/abs/1405.5119.

[19] Chunhui Xie, Yimin Cheng, and Yangkun Chen. “An active steganalysis approach for echo hiding based on Sliding Windowed Cepstrum”. In: *Signal Process.* 91.4 (Apr. 2011), pp. 877–889. ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2010.09.006. URL: https://doi.org/10.1016/j.sigpro.2010.09.006.