Algorytm usuwający szum ze zrzutów ekranu z gry "Wiedźmin 3: Dziki Gon"

Dokumentacja projektu

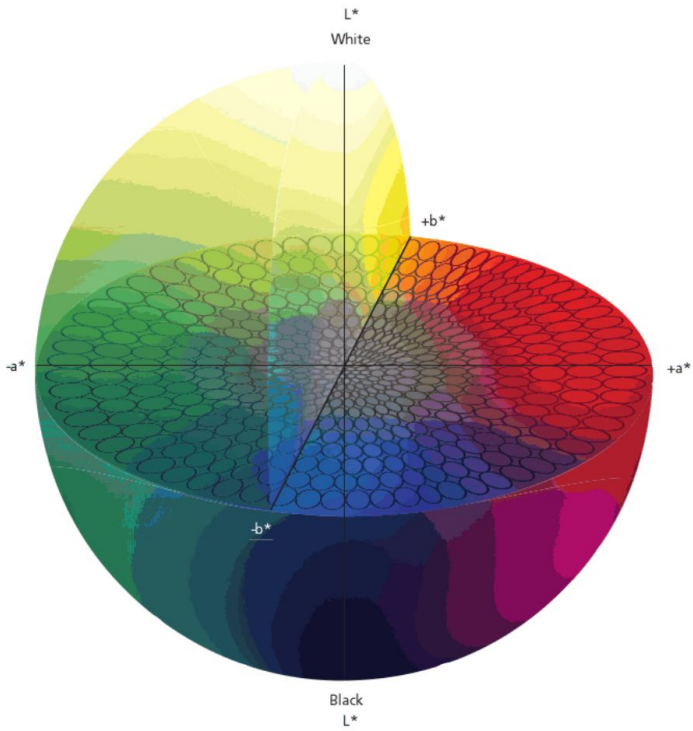
Maciej Biesek, Natalia Postawa

Spis treści:

Opis rozwiązywanego problemu

Celem projektu było stworzenie algorytmu usuwającego szumy z sześćdziesięciu zrzutów ekranu z gry *Wiedźmin 3 : Dziki Gon*. Wejściowe obrazy pochodzą z konkursu zaproponowanego przez międzynarodową konferencję *The Second International Conference on Intelligent Desision Science (IDS)*.

Na większości zdjęć szumy są łatwo zauważalne w postaci czerwonych, zielonych oraz niebieskich pikseli pojawiających się w różnych miejscach zrzutu. Oczywistym jest, że szumy te zostały sztucznie nałożone na zdjęcia na potrzeby stworzenia bazy pod konkurs. Podczas planowania prac nad projektem ustaliliśmy, że w pierwszej kolejności skupimy się na pracy badawczej i dowiemy się jakie są rodzaje szumów oraz jak je rozpoznawać i usunąć. Dodatkowo, musieliśmy zdecydować, z jakiej przestrzeni barw będziemy korzystać. Podstawowym modelem jest RGB, ale jest on mało intuicyjny dla ludzkiego oka, więc ostatecznie zdecydowaliśmy się na model LAB, który jest chyba najbardziej zbliżony do sposobu postrzegania kolorów przez człowieka.



Model LAB ma trzy kanały odpowiadające za: L - jasność (luminancja), a – barwa od zielonej do magenty, b – barwa od niebieskiej do żółtej.

Dodatkowym argumentem przemawiającym za jego wyborem, była łatwość nałożenia metryki na wartości modelu - było to dla nas istotne, ze względu na stosowane w algorytmach porównywanie kolorów pikseli. Zastosowaliśmy podstawową metrykę euklidesową:

$$\Delta E = \sqrt{(\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2}$$

bowiem badania wykazały, że można przyjąć, iż standardowy obserwator zauważa różnicę barw następująco:

- 0 < ΔE < 1 - nie zauważa różnicy,
- 1 < ΔE < 2 - zauważa różnicę jedynie doświadczony obserwator,
- 2 < ΔE < 3,5 - zauważa różnicę również niedoświadczony obserwator,
- 3,5 < ΔE < 5 - zauważa wyraźną różnicę barw,
- 5 < ΔE - obserwator odnosi wrażenie dwóch różnych barw.

W drugim etapie pracy pracowaliśmy nad stworzeniem algorytmu, którego zadaniem będzie usuwanie szumu z danego zdjęcia. Podeszliśmy do tego zadania od kilku różnych stron, w efekcie czego powstało kilka różnych sposobów na jego rozwiązanie. Niniejszy dokument jest opisem wszystkich zastosowanych algorytmów i ich rezultatów.

Przykładowy zaszumiony zrzut ekranu prezentuje się następująco:

Szczegółowy opis zastosowanego algorytmu

Pierwszy z algorytmów, który wydawał się najbardziej intuicyjny, to filtracyjne usuwanie szumów. Polega to na tym, że dla każdego piksela badamy jego najbliższe otoczenie (nakładamy filtr o oknie np. 3x3) i w zależności od tego, jakie kolory mają jego sąsiedzi, odpowiednio zmieniamy kolor tego piksela, stosując np. uśrednianie tych wartości lub zliczanie najczęściej występującego koloru. Można dodać dodatkowy warunek, np. branie pod uwagę tylko tych pikseli z otoczenia, których kolor znacząco różni się od koloru aktualnie badanego piksela. Można również warunkować decyzję, czy piksel ten jest zaszumiony, czy nie, ustalając, że nadaje się do zmiany, gdy stosunek różniących się od niego pikseli do całkowitej liczby pikseli z sąsiedztwa jest większy niż jedna trzecia. Próbowaliśmy wszystkich tych kombinacji, efekty poniżej:

Filtr 3x3, ze wszystkimi sąsiadami branymi pod uwagę:

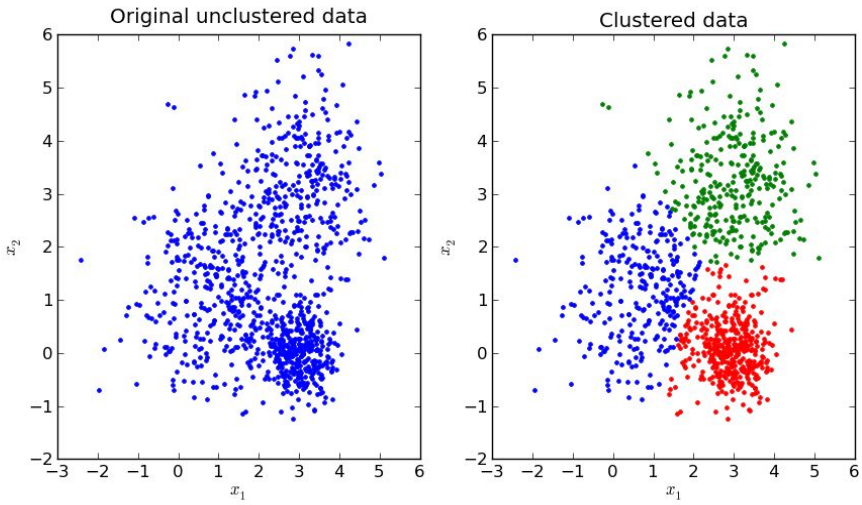
Filtr 3x3, analizie zostały poddane tylko te piksele, które znacząco różnią się od tego, który jest aktualnie badany:

Dla porównania: zastosowanie filtru 5x5 z braniem pod uwagę różniących się pikseli

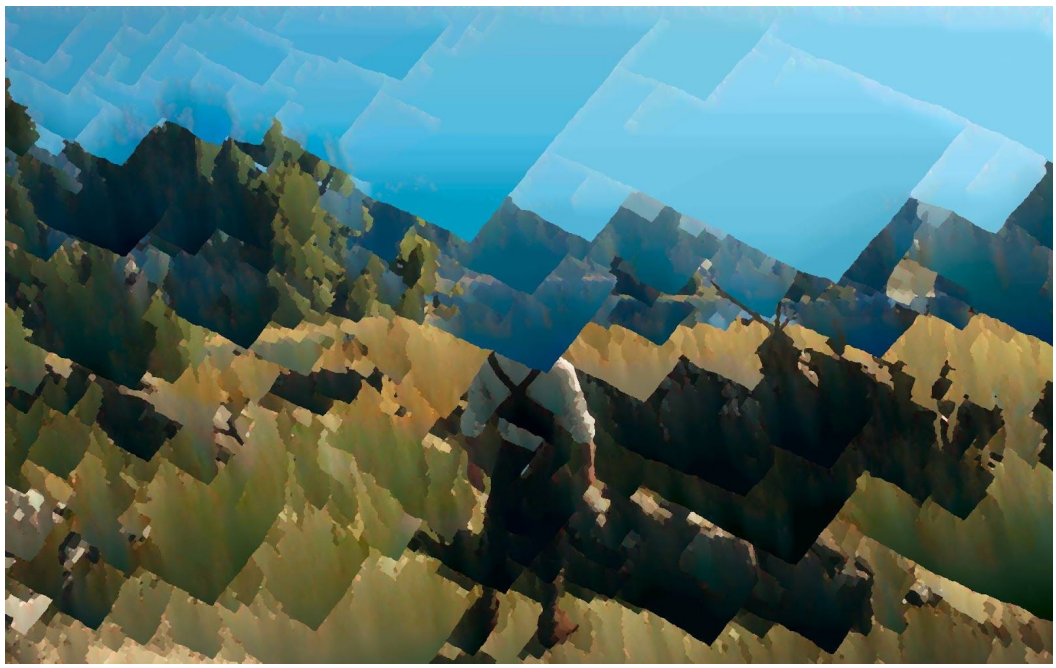
Jak można zauważyć, zastosowanie tego filtru nie jest najlepszym rozwiązaniem - co prawda w jakimś stopniu udaje się odszumić obraz, jednak kosztem jego rozmazania.

Z tego względu zastanawialiśmy się w jaki sposób poprawić skuteczność lub lepiej wybierać zmienione piksele.

Postanowiliśmy wykorzystać analizę skupień, a konkretnie **metodę k-średnich**. W tym podejściu również analizie poddajemy sąsiedztwo tego piksela, dla którego stosunek różniących się pikseli do wszystkich pikseli z sąsiedztwa jest duży. Pomysł polegał na tym, że dla każdego piksela dzielimy piksele z jego sąsiedztwa na klastry.



Minusem tego sposobu jest fakt, że w implementacji KMeans z biblioteki pythonowej sklearn.cluster odgórnie trzeba ustalić wartość parametru k. Poniżej znajdują się próby zastosowania k-średnich przy odszumianiu jednego ze zdjęć:

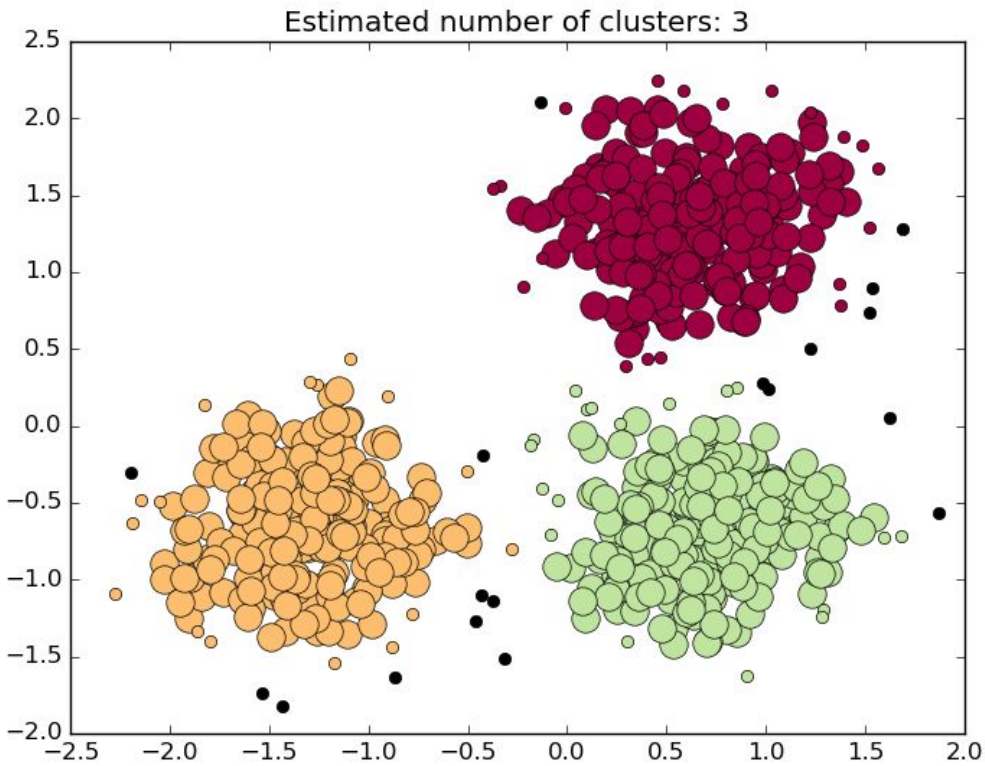


Oraz, po zastosowaniu stałej jako 2 (w myśl zasady, że chcemy wydobyć dominujący i resztę świata, więc potrzebujemy tylko 2 klastry):

Poważne minusy przemawiające za niechęcią do k-średnich są: konieczność ustalenia odpowiedniej wartości k oraz to, że algorytm każdy piksel stara się

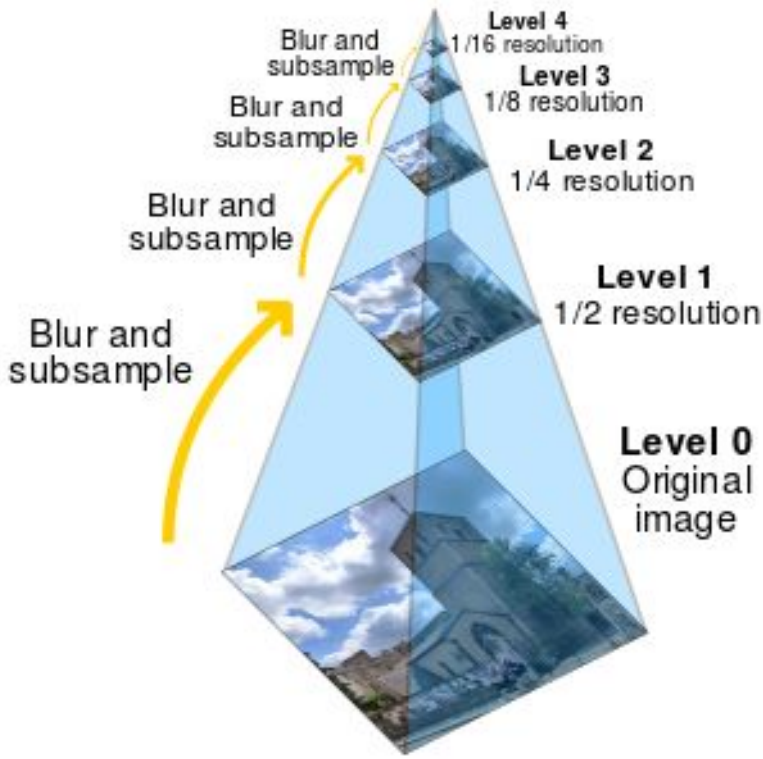
dopasować do najbliższego klastra. W efekcie zaszumiony piksel może zostać rozpoznany jako normalny.

Postanowiliśmy podążać tym tropem i znaleźliśmy **algorytm DBSCAN**, który działa podobnie jak k-means, z tym, że sam dobiera odpowiednią wartość k oraz nie stara się na siłę dopasować elementów do któregoś zbioru - gdy uznaje, że nie pasuje do żadnego z nich, wtedy wrzuca go do wora “nierozpoznane”.



Pomysł nasz był taki, żeby uznać piksele z tego wora jako te, które trzeba odszumić. Niestety, metoda ta powoduje duży narzut obliczeniowy, w związku z tym nie udało się do tej pory uzyskać sensowych wyników analizy zrzutów.

Kolejnym pomysłem było wykorzystanie **piramidy obrazów**. Pomysł opiera się na obserwacji, że im bardziej oddalimy się od obrazu, tym mniej szumów jesteśmy w stanie dojrzeć. Bazując na tej idei, budujemy piramidę obrazów.



W każdym kroku algorytmu nakładamy na obraz maskę Gaussa, która wygładza go oraz zmniejsza rozmiar obrazu o połowę. W tym celu używamy wielokrotnie funkcji pyrDown() i za jej pomocą otrzymujemy kolejne warstwy piramidy. W naszym przypadku postanowiliśmy wykorzystać piramidę złożoną z 6 warstw.
W następnym kroku rozszerzamy obrazy do wyjściowego rozmiaru używając funkcji pyrUp(), otrzymując piramidę, w której wszystkie obrazy są takiego samego rozmiaru. Na dnie piramidy jest obraz najbardziej wyraźny i najbardziej zaszumiony, a na górze obraz bez szumów, ale bardzo rozmazany.

Aby odszumić nasz obraz wyjściowy, przechodzimy piksel po pikselu i zamieniamy je na średnią arytmetyczną wziętą ze wszystkich poziomów piramidy. W rezultacie otrzymujemy mniej ostry obraz, ale z mniejszą liczbą szumów.

Pomysłem na ulepszenie tej metody było zastosowanie znanego już sposobu rozpoznawania, czy dany piksel jest zaszumiony, czy też nie. Efekt można zobaczyć poniżej:

Niestety, również i tutaj, dokonujemy odszumienia zdjęcia kosztem straty jakości tekstur i rozmazania całego obrazka.

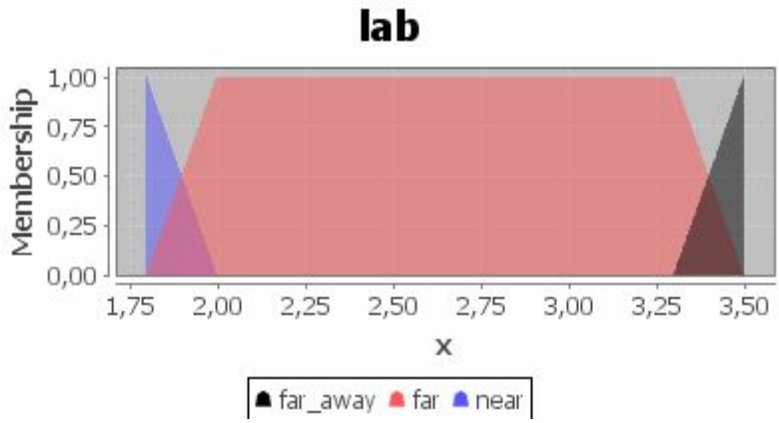
Ostatnim etapem naszego algorytmu jest wyostrzenie obrazu, który otrzymaliśmy za pomocą piramid. Wyostrzenie otrzymujemy korzystając ze wzoru:

Obraz wyostrzony = Obraz oryginalny + (Oryginalny - Rozmyty) * Stała

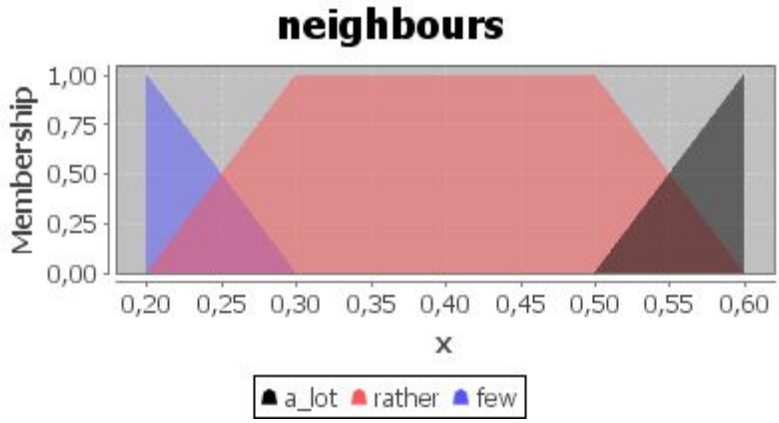
Do uzyskania obrazu rozmytego używamy funkcji GaussianBlur(). Po odszumieniu i wyostrzeniu końcowy obraz wygląda następująco:

We wszystkich wspomnianych podejściach korzystamy ze zmiennych lingwistycznych, które określają co znaczy, że dwa piksele są od siebie odległe oraz co znaczy, że dany piksel posiada w swoim sąsiedztwie dużo różniących się od niego pikseli.

```
# lab expression
near = E(ltrapezoid(1.8, 2), "near")
far = E(trapezoid(1.8, 2, 3.3, 3.5), "far")
far_away = E(rtrapezoid(3.3, 3.5), "far away")
```



```
# neighbours expression
few = E(ltrapezoid(0.2, 0.3), "few")
rather = E(trapezoid(0.2, 0.3, 0.5, 0.6), "rather")
a_lot = E(rtrapezoid(0.5, 0.6), "a lot")
```




```
not_similiar = R(lab = far | far_away)
is_near = R(lab = near)
is_noised = R(neighbours = a_lot | rather)
```

Opis wykonanej implementacji

Implementacja została wykonana w języku Python w wersji 2.7, przy użyciu takich bibliotek, jak: OpenCV, Numpy, sklearn, scipy, frules oraz wielu innych, mniej znaczących. Dzięki zastosowaniu OpenCV oraz numpy możliwa była analiza zdjęć, a frules przysłużyło się do odpowiedniego zamodelowania zmiennych lingwistycznych, które to z kolei były niezbędne z punktu widzenia powodzenia projektu.

Repozytorium kodu znajduje się tutaj: klik

Wyniki testów skuteczności zastosowanego algorytmów

Nie udało nam się zastosować żadnego narzędzia, które sprawdzałoby w jakiś sposób stopień odszumienia zdjęcia. Skupiliśmy się bardziej na sprawdzeniu kolejnych metod, które przychodziły nam do głowy. Jedyną miarą oceny była nasza subiektywna opinia (co w sumie nie jest tak bardzo oderwane od rzeczywistości,

skoro model LAB został stworzony po to, aby odwzorowywać ludzką percepcję barw). Na tej podstawie jednomyślnie stwierdziliśmy, że algorytm piramidalny najlepiej spełnia swoje zadanie, mimo rozmycia w pewien sposób badanego obrazu. Myślimy, że być może połączenie ze sobą którychś z przedstawionych metod da pozytywny rezultat. Zamierzamy to sprawdzić dla zaspokojenia własnej ciekawości, już nie w ramach zaliczenia przedmiotu.