Informatyka w Medycynie - Projekt 1
Tomograf komputerowy

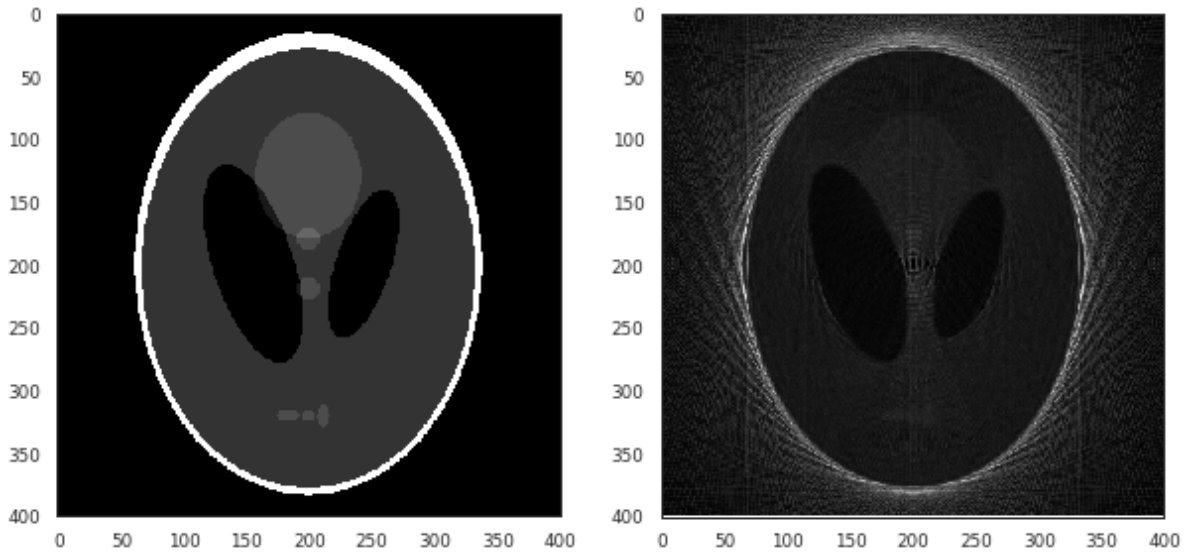
Dariusz Max Adamski 136674, Sławomir Gilewski 142192

{dariusz.adamski,slawomir.gilewski}@student.put.poznan.pl

Data oddania: 28 lutego 2021

Wstęp

W danym sprawozdaniu przedstawiony będzie symulator tomografu komputerowego, z wykorzystaniem transformaty radona. Następnie wygenerowany zostaje obraz wyjściowy, wykonując odwrotną transformatę na uzyskanym wcześniej sinogramie.



Rysunek 1: Przykład działania programu

1 Model tomografu

Zastosowany przez nas model tomografu to model równoległy. Zdecydowaliśmy się na to rozwiązanie, ponieważ było wygodne w implementacji. Po wyznaczaniu współrzędnych punktów odpowiadających emiterowom, wyznaczenie współrzędnych odpowiadających detektorom przebiega analogicznie.

2 Narzędzia

2.1 Język Programowania

Wybraliśmy język programowania Python. Głównie ze względu na jego prostotę i łatwość instalowania i korzystania z dodatkowych bibliotek. Jest to język, w którym obydwoje najbardziej

swobodnie się poruszamy.

2.2 Dodatkowe biblioteki

Istotne biblioteki i ich wykorzystanie w naszym programie to:

- Numpy: Obliczenia i operacje na n-wymiarowych macierzach
- Pydicom: Wykorzystana przy zapisie i odczycie obrazów w standardzie DICOM
- Skimage: Wszelkiego rodzaju odczyt, operacje i przekształcenia na obrazach
- Scipy: Funkcje związane z transformatami Fouriera, wykorzystanymi przy filtrowaniu sinogramu w celu redukcji szumu
- Matplotlib: Wszelkiego rodzaju wizualizacje
- Multiprocessing: Urównolegnienie transformaty Radona, w celu jej przyspieszenia

3 Funkcje programu

Opis głównych funkcji programu oraz ilustracja za pomocą fragmentów kodu źródłowego

3.1 pozyskiwanie odczytów dla poszczególnych detektorów

```
def single_radon_transform(detector_count, angle_range, image, radius, center, alpha):
    emitters = emitter_coords(alpha, angle_range, detector_count, radius, center)
    detectors = detector_coords(alpha, angle_range, detector_count, radius, center)
    lines = radon_lines(emitters, detectors)
    result = rescale(np.array([np.sum( image[tuple(line)]) for line in lines]))
    return result
```

Dla konkretnego kroku alpha:

1. emiter_coords, detector_coords: Pozyskujemy współrzędne emiterów oraz detektorów dla wybranych parametrów(ilość emiterów/detektrów, rozpiętość układu) oraz wybranego obrazu (potrzebujemy promień oraz współrzędne środka)
2. radon_lines: Dla uzyskanych współrzędnych emiterów oraz detektorów obliczamy współrzędne wszystkich pixeli leżących na prostych pomiędzy tymi punktami. Wykorzystujemy algorytm Bresenhama do liniowego przejścia po kolejnych pikselach obrazu dyskretnego.
3. Następnie dla pozyskanych wcześniej współrzędnych pikseli, sumujemy ich wartości dla każdej pary emiter/detektor.

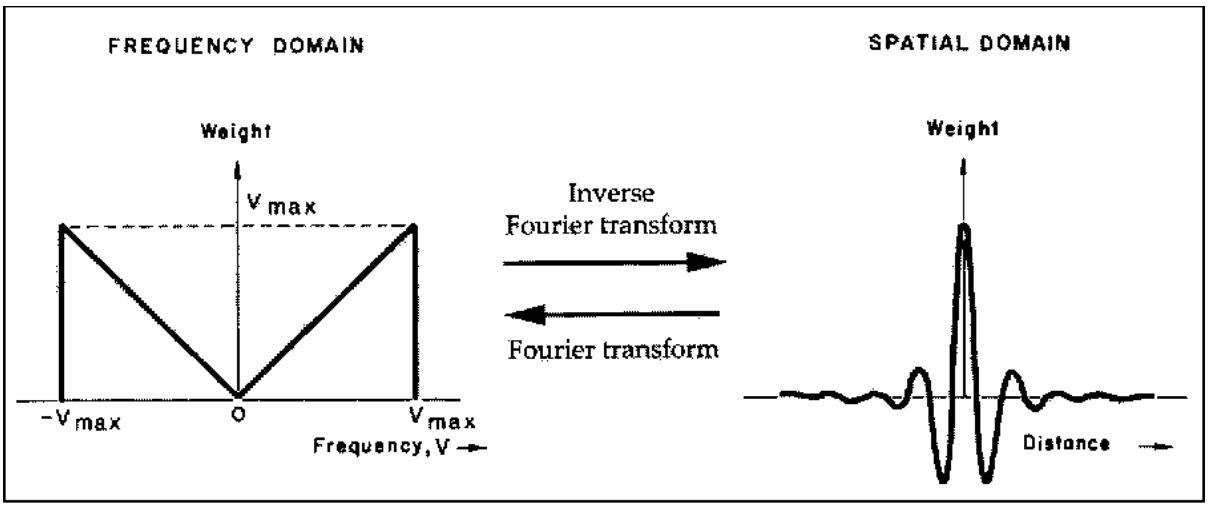
W celu uzyskania całego sinogramu, iterujemy po kolejnych wartościach alfy, wykorzystując powyższy algorytm w funkcji „radon_transform". Warto wspomnieć iż algorytm pracuje na zmodyfikowanym obrazie wejściowym - przed zastosowaniem transformaty, do obrazu dodawany jest padding, aby zapobiec utracie informacji.

3.2 Filtrowanie sinogramu

W celu uzyskania lepszych wartości wynikowych, przed przystąpieniem do odwrotnej transformaty radona zostaje zastosowany splot na naszym sinogramie.

```
def filter_sinogram(sinogram):
    n = sinogram.shape[0]
    filter = 2 * np.abs(
```

```
fftfreq(n).reshape(-1, 1))
result = ifft(fft(sinogram, axis=0) * filter, axis=0)
result = clip(np.real(result), 0, 1)
return result
```



Rysunek 2: Po lewej stronie „ramp filter” w dziedzinie częstotliwości, a po prawej w dziedzinie przestrzennej

1. Najpierw tworzymy nasz kernel/filtr w dziedzinie częstotliwości (frequency domain). Po przeprowadzonych eksperymentach, doszliśmy do wniosku, że najlepiej wypada tutaj „ramp filter”.
2. Następnie przekształcamy nasz sinogram do dziedziny częstotliwości za pomocą funkcji fft.
3. Przekształcony sinogram przemnażamy przez stworzony wcześniej filtr.
4. Wynikowy sinogram przekształcamy z powrotem do dziedziny przestrzennej (spatial domain). Ucinane są również wartości poniżej 0 oraz powyżej 1 w sinogramie. Uznaliśmy, iż rozmiar maski najlepiej pozostawić taki, jak rozmiar sinogramu, czyli ilość detektorów transformaty.

3.3 Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe

```
def single_inverse_radon_transform(image, tmp, single_alpha_sinogram, alpha,
detector_count, angle_range, radius, center):
    emitters = emitter_coords(alpha, angle_range, detector_count, radius, center)
    detectors = detector_coords(alpha, angle_range, detector_count, radius, center)
    lines = radon_lines(emitters, detectors)
    for i, line in enumerate(lines):
        image[tuple(line)] += single_alpha_sinogram[i]
        tmp[tuple(line)] += 1
```

1. Początek funkcji przebiega analogicznie do wcześniej omówionej transformaty radona. Dla konkretnego kroku alpha: pozyskujemy współrzędne emiterów oraz detektorów a następnie dla uzyskanych punktów obliczamy współrzędne wszystkich pixeli leżących na prostych pomiędzy tymi nimi.

2. Następnie dla każdej pary emiter/detektor, dodajemy wartość odpowiadającego punktu sinogramu do pikseli leżących na linii pomiędzy nimi.

3. W macierzy tmp dla każdego pixela przechowywana jest liczba linii, które przez niego przeszły. Macierz ta wykorzystana jest później w celu normalizacji obrazu wynikowego.

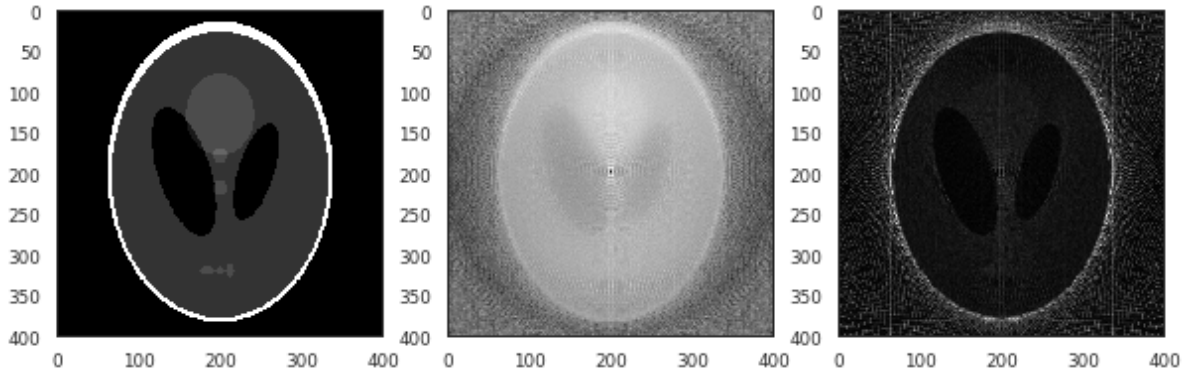
W celu uzyskania całego obrazu wynikowego, iterujemy po kolejnych wartościach alfy, wykorzystując powyższy algorytm w funkcji „inverse_radon".
Po przejściu przez wszystkie wartości alpha, funkcja ta wykorzystuje wspomnianą wcześniej macierz tmp w celu normalizacji obrazu.
Na koniec wycinany jest obraz wynikowy, w celu odrzucenia dodanego wcześniej paddingu.

3.4 Wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego

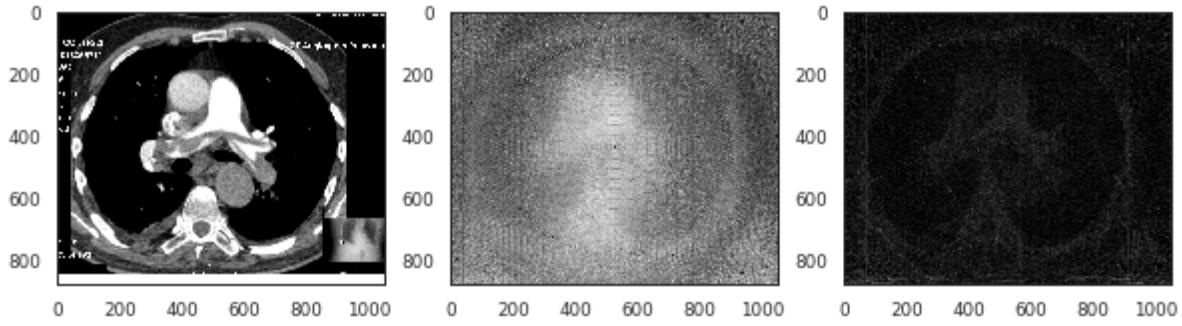
Przeprowadziliśmy eksperyment sprawdzający wpływ poszczególnych parametrów transformaty na jakość odtworzonego obrazu. Jako miary użyliśmy błędu średniokwadratowego (RMSE). Zgodnie z poleceniem ‚jakop parametry domyślne przyjęliśmy 180 detektorów, 180 skanów oraz rozpiętość wachlarza równą 180. Jeśli nie powiedziano inaczej, do testów użyliśmy obrazu „saddle-pe", widoczny na rysunku 4.

Na rysunku 5 przedstawiony jest błąd przy zmieniającej się odpowiednio liczbie skanów, detektorów i rozwartości kątowej (wynikowe obrazki widoczne na rysunkach na ostatniej stronie). Według nas zwiększanie każdego z parametrów zwiększa jakość obrazu, przy czym większa liczba skanów i detektorów redukuje szum/ziarnistosć obrazu. Jednak na wykresach RMSE niestety nie widać znaczących zmian. Według nas jest to spowodowane dużą ilością szumu w obrazach wyjściowych.

Dodatkowo sprawdziliśmy wpływ filtrowania sinogramu na jakość obrazu. Wyniki są przedstawione na rysunkach 3 i 4, oraz w tabeli 1. Według nas filtrowanie zdecydowanie poprawia odzwierciedlenie obrazu, w szczególności jasność elementów na obrazie jest bardziej zbliżona do oryginału. Zredukowana jest też okrągła poświata. Filtrowanie na obrazie „shepp-logan" zdecydowanie zredukowało błąd, ale przy „saddle-pe" niestety już nie (jednak według nas obraz jest odrobinę ostrzejszy).



Rysunek 3: Wizualizacja efektów filtrowania sinogramu (shepp-logan). Po lewej stronie oryginalny obrazek, na środku efekt odwrotnej transformaty Radona bez filtrowania, a po prawej z filtrowaniem



Rysunek 4: Wizualizacja efektów filtrowania sinogramu (saddle-pe)

Tablica 1: Wyniki eksperymentu

obrazek	rysunek	RMSE (bez filtrowania)	RMSE (z filtrowaniem)
shepp-logan	rysunek 3	0.482961	0.195934 (↓)
saddle-pe	rysunek 4	0.365851	0.371541 (↑)

3.5 Obsługa plików DICOM

Do obsługi plików DICOM wykorzystaliśmy bibliotekę `pydicom`.

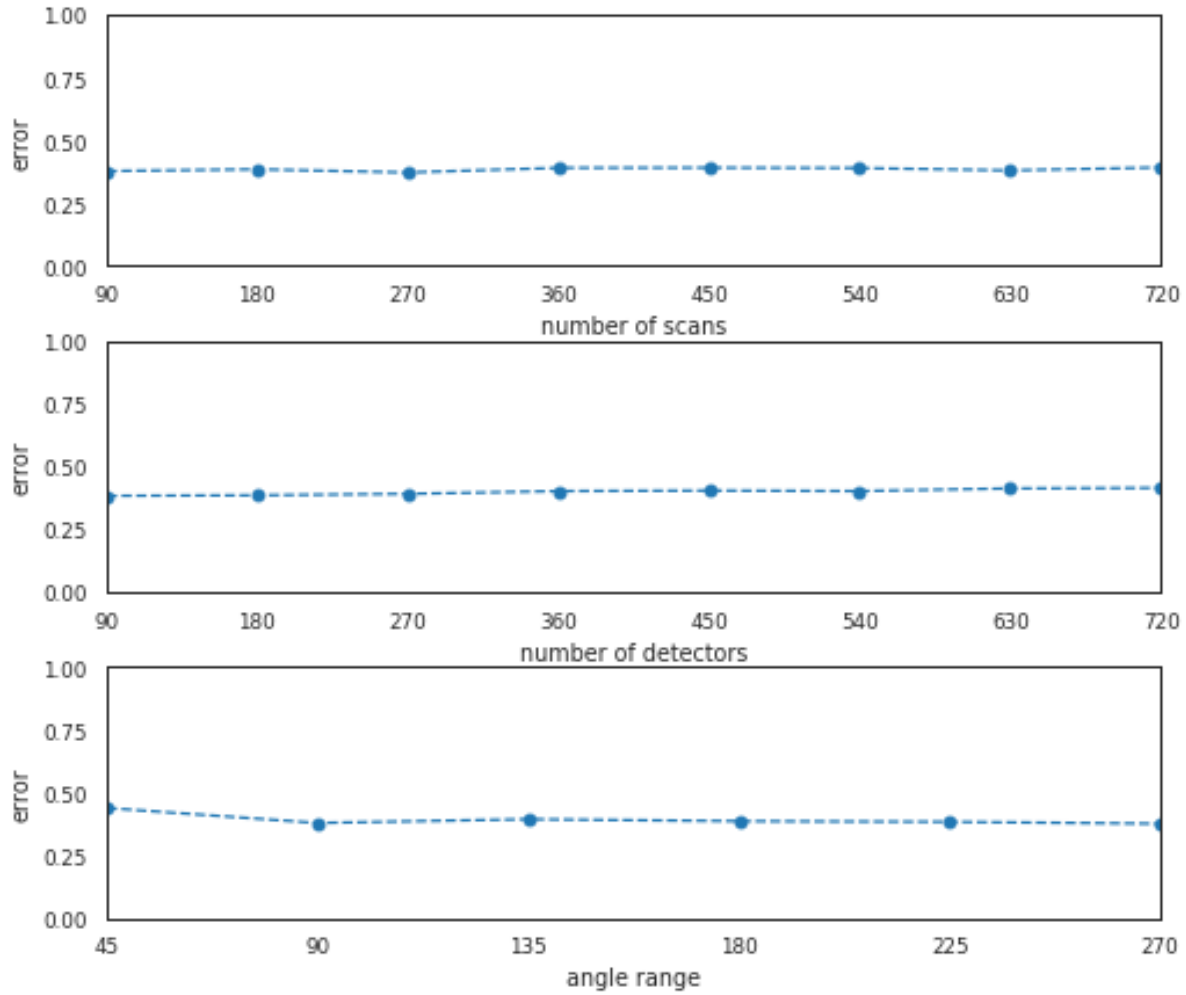
W naszym programie odczyt jest realizowany przez funkcję `read_dicom`, która przyjmuje ścieżkę do pliku który chcemy otworzyć i zwraca obraz w postaci 2-wymiarowej macierzy oraz słownik metadanych, które zawierają na przykład imię pacjenta.

```
image, meta = read_dicom('test.dcm')
```

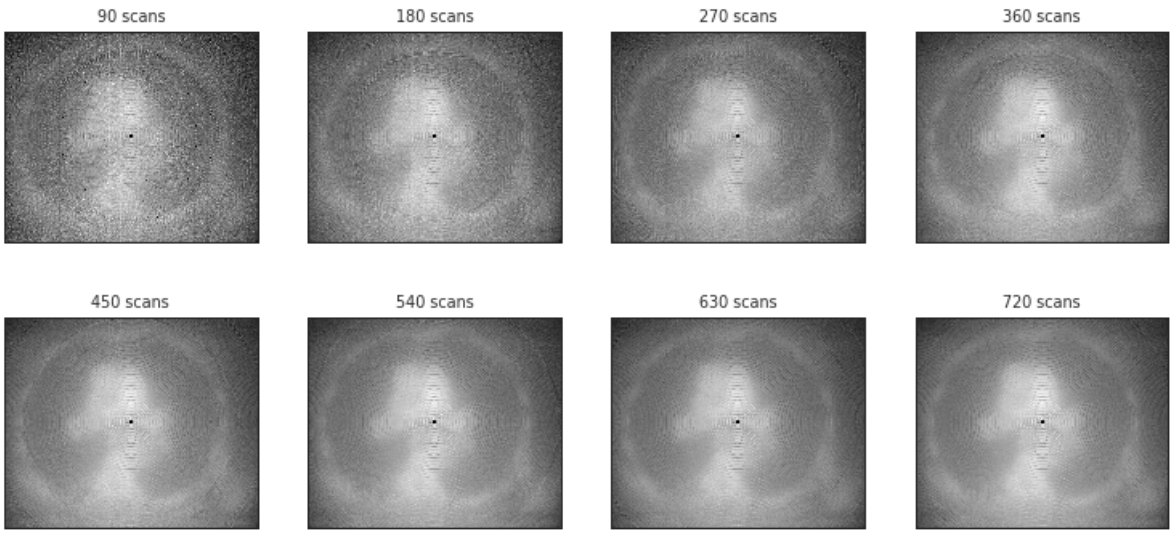
Pliki DICOM zapisujemy przy użyciu funkcji `write_dicom`, która przyjmuje ścieżkę pod którą będzie zapisany plik, obrazek w formie 2-wymiarowej macierzy oraz słownik metadanych. Gdy nie podamy żadnych metadanych, w pliku zostaną umieszczone tylko te, aby plik był zgodny ze standardem DICOM.

```
write_dicom('test.dcm', image, dict(
    PatientName='Doe^John',
    PatientID='666',
    ImageComments='No comment :)',
    StudyDate='20200213',
))
```

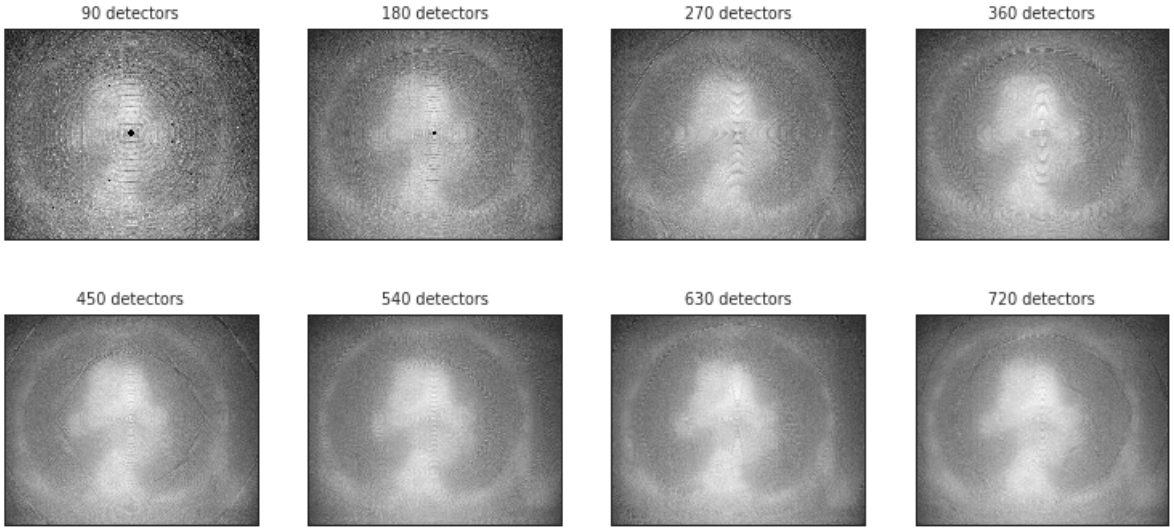
Poprawność zapisu plików DICOM zweryfikowaliśmy przy pomocy przeglądarki `https://www.imaios.com/en/Imaios-Dicom-Viewer`.



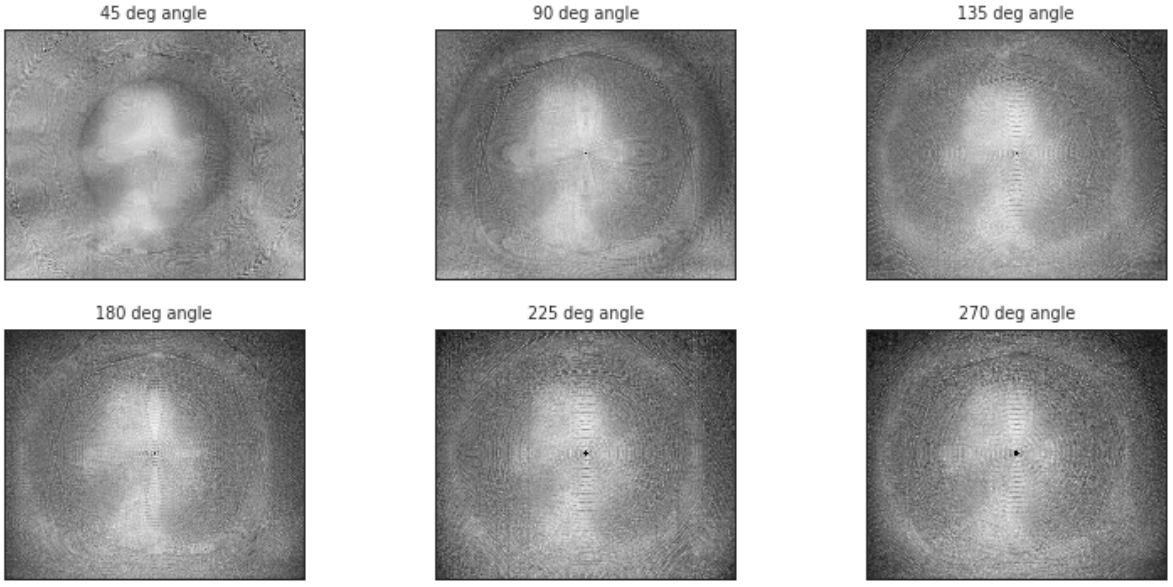
Rysunek 5: Wartość RMSE przy zmienającej się liczbie skanów, detektorów i rozwartości kątowej



Rysunek 6: Zmiana liczby skanów



Rysunek 7: Zmiana liczby detektorów



Rysunek 8: Zmiana rozwartości kąta