MRI Intelligent Selective Scanning based on Independent Component Analysis (ICA) of the frequency domain raw MRI data (k-space) feed to assess the effect of motion

Team Members:
Malavika Kuchakulla - 1352529 - Computer Science - Undergraduate
Rahul Parasa - 1599185 - Computer Science - Undergraduate
Aashrit Mathur - 1611541- Computer Science- Graduate
Neha Singh - 1618926 - Computer Science - Graduate

Table of Contents:

1. Introduction
2. Methods
3. Softwares and packages
4. Results and Discussions
5. Conclusions
6. Code

Introduction:

This project is about MRI Intelligent Selective Scanning based on Independent Component Analysis (ICA) of the frequency domain of raw MRI data (k-space) to assess the effect of motion. The aim of the project is to identify the important portion of the raw data(frequency domain or k-space) that we need to collect in order to reproduce the image that holds all the important information as in the original raw MRI data. In order to retrieve the significant information from the raw MRI image we are using ICA(Independent Component Analysis) on the K-Space of the raw image.

Independent component analysis (ICA) is a recently developed method in which the goal is to find a linear representation of non-Gaussian data so that the components are statistically independent, or as independent as possible. Such a representation seems to capture the essential structure of the data in many applications, including feature extraction and signal separation.

Methods:

Software: python3,
Packages: tkinter, sklearn, cv2, numpy, PIL

- **tkinter**: Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.Creating a GUI application using Tkinter includes below steps:
    - Import the *Tkinter* module.
    - Create the GUI application main window
    - Add one or more of the above-mentioned widgets to the GUI application.
    - Enter the main event loop to take action against each event triggered by the user.
- **sklearn:** SK learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!
- **cv2**: OpenCV is a cross-platform library using which is used to develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.
- **numpy**: Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.
- **PIL:** The Python Imaging Library (PIL) adds image processing capabilities to the Python interpreter. We have used PIL for displaying images on our Tkinter window.
- Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

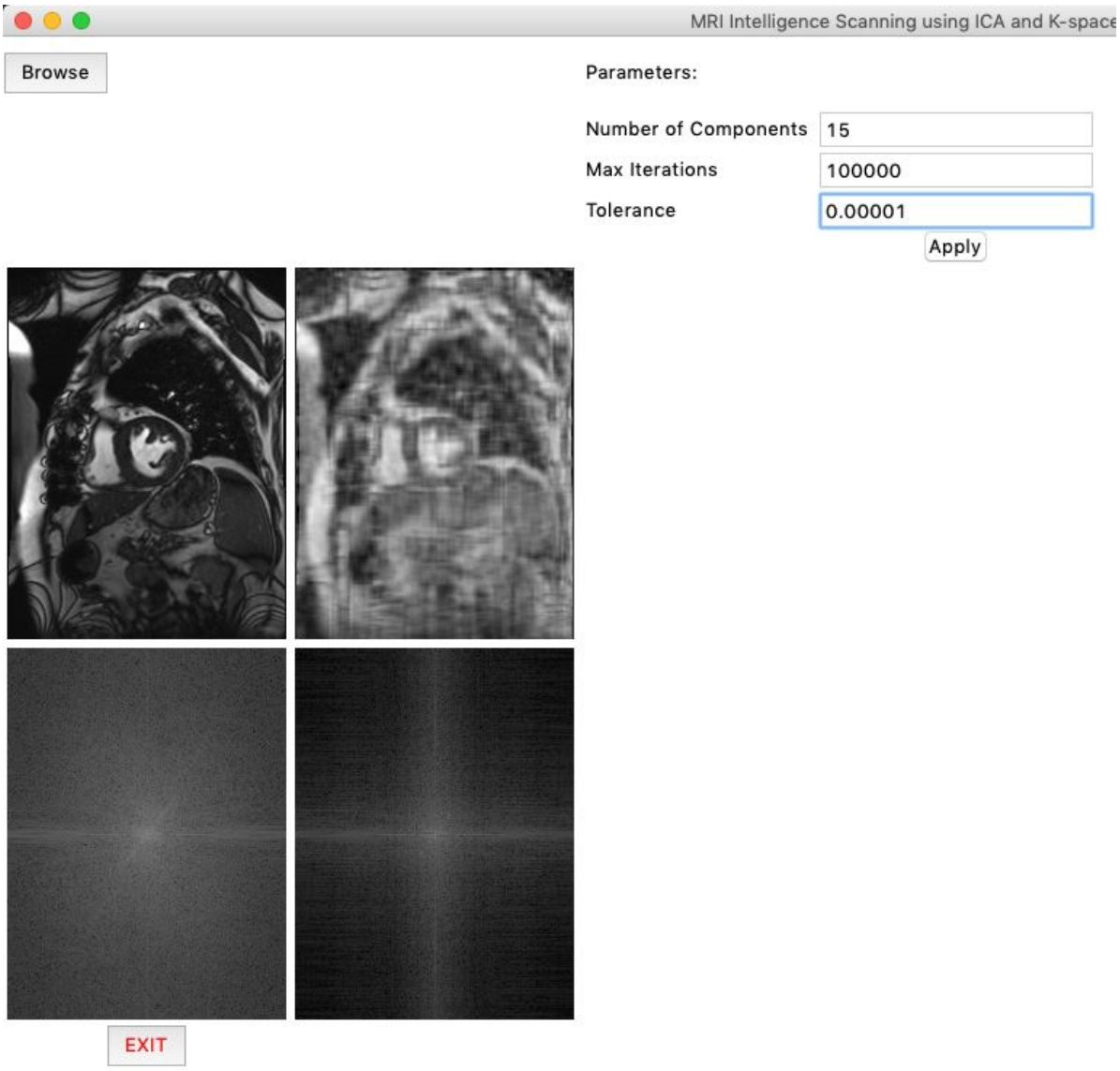
Algorithm: We are using the FastICA algorithm from the sklearn package in our code. FastICA is a fast algorithm for performing Independent Component Analysis on image data. It is a computational method for separating a multivariate signal into additive subcomponents. This is achieved by assuming that the subcomponents are Non-Gaussian signals and that they are statistically independent from each other.

```
Class sklearn.decomposition.FastICA (n_components=None,
algorithm='parallel', whiten=True, fun='logcosh', fun_args=None,
max_iter=200, tol=0.0001, w_init=None, random_state=None)
```

- In the project we have used only 3 (n_componenets, max_iter, tol) out of the 9 parameters provided in the FastICA function definition.
- Using FastICA algorithm we are hoping to retrieve the important components of the MRI scans and displaying them with the help of a GUI and also displaying the k-spaces of both the original scans and the resultant image after application of FastICA.

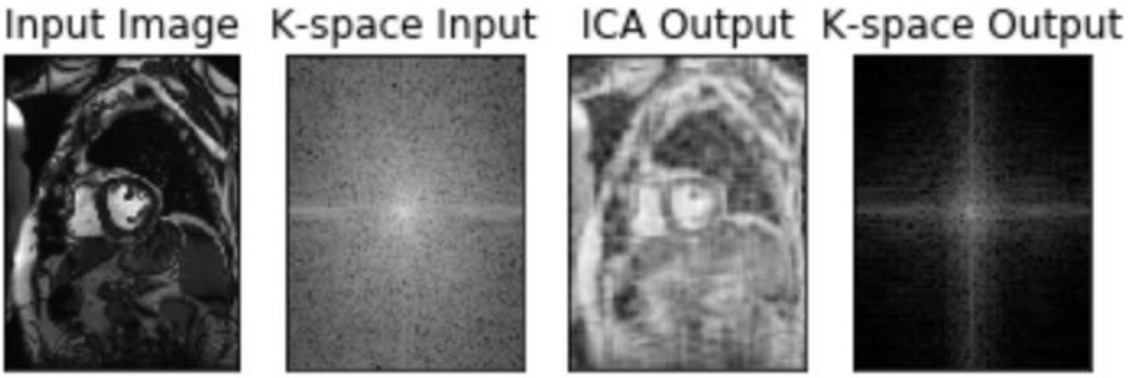
Description: We have implemented a python script that can help browse for an MRI scan of the chest cavity of a human and perform Independent Component Analysis (ICA) using sklearn's FastICA method. It helps provide a contrast between the k-spaces of the input MRI scan and the processed output.

GUI :



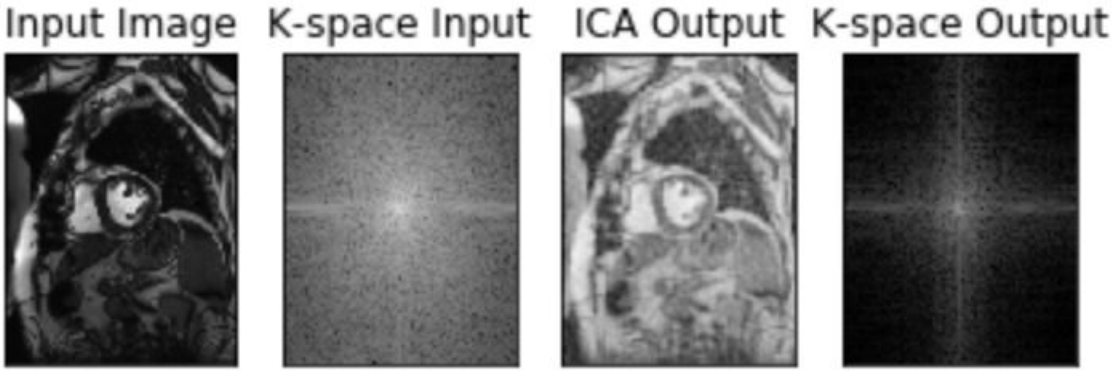
Results and Discussion:

We are using an MRI scan of a human chest cavity as input. On applying FFT on the input followed by FFTshift, we retrieve the K-space of the input image. We proceed by applying the FastICA algorithm on the original input image and obtain an image that has all the significant components. This is followed by generating a K-space for the resultant output image.



The above output is based on the following parameters values:

- no_of_componenets = 15
- max_iterations = 100000
- tolerance = 0.000001



The above output is based on the following parameter values:

- no_of_componenets = 25
- max_iterations = 100000
- tolerance = 0.000001

As we can observe the 'ICA Output' image for the second result looks a bit more detailed and clearer than the one shown in the first result. The same can be observed with 'K-space Output' images. The 'K-space Output' for the second case appears to contain much more high-intensity information than the first case, as we increase the no_of_ components from 15 to 25. So as we increase the no_of_components to add more details to the input image.
As we try to increase the no_of_components, FastICA might require much more max_iterations to converge to the final output, than was initially planned. Also, having a very low tolerance might result in increasing the max_iterations as the FastICA algorithm will not allow high value errors to pass through to the following iterations.

Future Work:

For the future, this python script can be evolved to browse and import multiple MRI scans together and display their corresponding ICA outputs. In this case, both input and output can be displayed as videos or GIFs to view a series of scans over time.

Conclusions:

We have successfully utilized the FastICA algorithm on the 25 MRI scans in order to retrieve the significant components of the images which will help doctors to fasten the treatment process by retrieving enough information in a short amount of time.

Code:

```
from sklearn.decomposition import FastICA
from matplotlib import pyplot as plt
import numpy as np
import pylab as pl
import cv2 as cv

import tkinter as tk
from tkinter import *
from tkinter import filedialog
from PIL import Image, ImageTk

window = tk.Tk()  # window is the main window which pops up when
the program runs

class gui:
    # ICA algorithm
    def compute_ica_output(self, input_img_path, no_of_components,
max_iterations, tolerance):
        image_name = input_img_path
        input_image = cv.imread(image_name, 0)
        original_image = input_image
       # performing FFT to obtain K-space of input
        input_image = np.fft.fft2(input_image)
        input_image = np.fft.fftshift(input_image)
        img = np.uint8(np.log(np.abs(input_image) + 1) * 10)
        kspace_image = img

        img = np.fft.ifftshift(input_image)
        img = np.fft.ifft2(img)                          # performing
IFFT to get the input back and perform ICA on it
        img = np.uint8(np.log(np.abs(img) + 1) * 10)

     # perform ICA using 3 parameters: number of components,
maximum iterations to converge and
        # tolerance of error at each iteration
        ica = FastICA(n_components = no_of_components, max_iter =
max_iterations, tol = tolerance)
        ica.fit(img)
        image_ica = ica.fit_transform(img)
        image_restored = ica.inverse_transform(image_ica)

        imagefcs = np.zeros((image_restored.shape[0],
image_restored.shape[1]), np.float)
```

```
       # performing a full contrast stretch for cler visibility of
the output
        for i in range(image_restored.shape[0]):
            for j in range(image_restored.shape[1]):
                imagefcs[i, j] = ((np.abs(image_restored[i, j]) -
image_restored.min()) * 255) / (image_restored.max() -
image_restored.min())
        ica_image = imagefcs

        imagefcs = np.fft.fft2(image_restored)
        imagefcs = np.fft.fftshift(imagefcs)

        output_kspace = np.uint8(np.log(np.abs(imagefcs) + 1) *
10)
        kspace_output = output_kspace

        return original_image, kspace_image, ica_image,
kspace_output

    def page1(self, window):
        self.window = window
        window.title('MRI Intelligence Scanning using ICA and
K-spaces')
        window.geometry("1250x750")

        # Browse Button
        browse_button = tk.Button(window, text = "Browse", command
= lambda: self.browsefunc(window), padx=10, pady=5)
        browse_button.grid(row = 0, column = 0, pady = 10, sticky
= tk.W)

        # Heading for the input parameters
        op_label = tk.Label(window, text="""Parameters: """)
        op_label.grid(row=0, column=3, sticky=tk.W)

        # Exit Button
        quit_button = tk.Button(window, text="EXIT",fg='red',
command=window.destroy, padx=10, pady=5)
        quit_button.grid(row=7, column=0)

        # Input Parameters
        # ---------------- LABELS ----------------
        # Label for Number of components
        InputLabel1 = tk.Label(window, text="Number of
Components")
        InputLabel1.grid(row=1, column=3, sticky=tk.W)
```

```
        # Label for Number of Iterations
        InputLabel2 = tk.Label(window, text="Max Iterations")
        InputLabel2.grid(row=2, column=3, sticky=tk.W)

        # Label for Tolerance
        InputLabel3 = tk.Label(window, text="Tolerance")
        InputLabel3.grid(row=3, column=3, sticky=tk.W)

        # ---------------- ENTRIES ----------------
        # Entry field for Number of components
        self.InputField1 = tk.Entry(window, textvariable =
"components")
        self.InputField1.grid(row = 1, column = 4, sticky = tk.E +
tk.W)

        # Entry field for Number of Iterationss
        self.InputField2 = tk.Entry(window, textvariable =
"iterations")
        self.InputField2.grid(row = 2, column = 4, sticky = tk.E +
tk.W)

        # Entry field for Tolerance
        self.InputField3 = tk.Entry(window, textvariable =
"tolerance")
        self.InputField3.grid(row = 3, column = 4, sticky = tk.E +
tk.W)

      # Button for the ICA function to be applied on the input
image.
        process_button = tk.Button(window, text = "Apply", command
= lambda: self.apply())
        process_button.grid(row=4, column=4)

       # Displays the image as a label on the main window once the
browse button is clicked
    def browsefunc(self, window):
        filename = filedialog.askopenfile()

   # storing the browsed file name for passing to
compute_ica_output
        self.input_img_path = filename.name

        self.image = Image.open(filename.name)
        self.image.thumbnail((256, 256))
```

```
        self.photo = ImageTk.PhotoImage(self.image)
        self.labelPhoto = Label(window, image = self.photo)
        self.labelPhoto.grid(row = 5, column = 0)

   # Displays the final output images after applying ICA algorithm
    def apply(self):
        self.components = int(float(self.InputField1.get()))
        self.iterations = int(float(self.InputField2.get()))
        self.tolerance = float(self.InputField3.get())

        self.original_image, self.kspace_image, self.ica_image,
self.kspace_output = self.compute_ica_output(
            self.input_img_path, self.components, self.iterations,
self.tolerance)

        # displaying K-space of input image
        self.photo1 =
ImageTk.PhotoImage(Image.fromarray(self.kspace_image))
        self.labelPhoto = Label(window, image = self.photo1)
        self.labelPhoto.grid(row = 6, column = 0)

      # displaying output image after performing ICA on input
image
        self.photo2 =
ImageTk.PhotoImage(Image.fromarray(self.ica_image))
        self.labelPhoto = Label(window, image = self.photo2)
        self.labelPhoto.grid(row = 5, column = 1)

        # displaying K-space of output image
        self.photo3 =
ImageTk.PhotoImage(Image.fromarray(self.kspace_output))
        self.labelPhoto = Label(window, image = self.photo3)
        self.labelPhoto.grid(row = 6, column = 1)

g = gui()
g.page1(window)
window.mainloop()
```

To execute the above Python script, please navigate to the “MedicalImaging” folder using the command prompt and run the following command:

```
python3 -m ICA_on_MRI_scans.py
```

This will open create a window which allows the user to browse a particular MRI scan from 25 available scans and type in the parameter values. Once the value input is given, hit the “Apply” button. This will display the processed ICA output on the corresponding MRI input scan and the k-spaces for the input and output images.