Bilkent University

Department of Electrical and Electronics Engineering

EEE475/575- Medical Image Reconstruction and Processing

Term Project - Deep Learning for Image Reconstruction

Group Members

Elif Ceren Fitoz - 21601060

Emre Dönmez - 21602680

Course Instructor: Emine Ulku Saritas

Due: 14 June 2020

Contents

Abstract

In the field of MRI scanning technology, high-fidelity image reconstructions are possible however more time-consuming. As new techniques emerge, the time it is required to obtain a close to perfect MRI scan image is reducing significantly. In this project, we have proposed a CNN architecture for the cartesian, single-coil undersampled k-space data. We have trained the proposed model with partial k-space data and Poisson disc sampling to compare with the current state-of-art reconstructions which are POCS and Compressed Sensing respectively. We can see under the light of our study and prior studies that CNN architectures is a promising framework in MRI reconstruction, with high-fidelity image reconstruction, near SSIM and PSNR values compared with the other methods.

Video Link

Introduction

In the field of MRI scanning technology, it is possible to do nearly perfect, accurate and high-quality medical scanning. The main drawback of this subject is the time it takes for performing a full scan. As new MRI scanning techniques emerge, the time it is required to obtain a close to perfect MRI scan image is reducing significantly. However, for most of the cases, scanning faster means scanning less data. So, the main goal of Medical Image Reconstruction is to reconstruct images using the least data as possible whilst keeping the accuracy as high as possible.

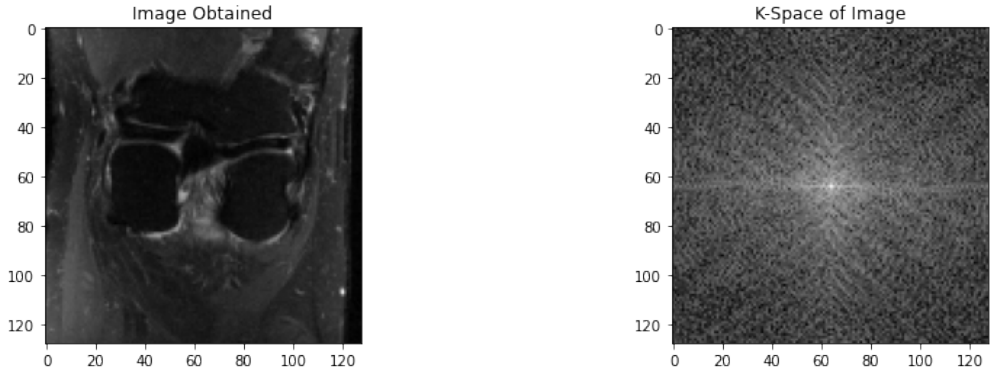
As Machine Learning and Deep Learning become more popular, Medical Image Reconstruction methods involving these subjects also start emerging more and more. Even though the current state-of-the-art methods for Medical Image Reconstruction can result in fast and accurate results, involving Deep Learning in the field of study can raise the performance of reconstructions significantly since Deep Learning is an expanding field itself.

In this project, we have focused on Cartesian sampling strategies from data acquired by a single coil. Our approach was to develop a deep convolutional neural network based on U_{net} structure, that learns with a training dataset. U_{net} architecture and their modifications were concluded to be successfull in image-to-image prediction and reconstruction tasks [1]. The network implemented on image domain and trained with two different undersampling strategies, one with Poisson-disc undersampling with rate of 4 and the other with partial k-space acquisitions with varying proportions. The CNN architecture trained with partial k-space acquisition performance was compared with the iterative partial k-space algorithm with Projection onto Convex Sets (POCS) method. The network trained with Poisson-disc subsampled k-space was compared with state-of-art Compressed Sensing reconstruction. We can see under the light of our study and prior studies that CNN architectures is a promising framework in MRI reconstruction, with high-fidelity image reconstruction, high SSIM and PSNR values compared with the other methods.

Dataset

For this project, the fastMRI single-coil knee dataset from Facebook AI Research and NYU Langone Health was used [2]. The dataset consists 973 training, 199 validation and 108 knee scans in total. The raw multi-coil k-space data was combined to obtain single-coil image, which uses the Root-Sum-of-Squares method. The dataset includes proton-density (PD) and T2 weighted acquisitions both with and without fat suppression. The data is stored in HDF5 files, which includes the matrices with size 320x320 of k-space along with image itself. The resolution is 0.5mmx0.5mm, and slice thickness of 3mm. For a single H5 file, nearly 30-40 slices are present [2]. Finally, the MRI acquisitions present in the dataset consists each coronal, sagittal and axial images of knee. Throughout the project, since the maskings were manually implemented,

solely the part (250 scans) of the training dataset containing the fully-sampled k-space was used and divided into test, training and validation itself. A sample knee-slice PD-weighted, fat-saturated coronal MRI image along with its k-space acquisition are given below.



(a) PD-FS Coronal Knee Slice

(b) k-space of Provided Image

Figure 1: Data Example

Data Pre-Processing

Since the training dataset itself was allocating high memory space (88 GB) and deep learning part of the project was implemented on Colab, several size reduction in H5 files was necessary. Therefore, from the h5 file, only k-space arrays were taken. Further, to ease the computational complexity of training session, the 320x320 sized matrices downsampled to (128,128). 250 scans were modified in that manner and and read from Colab. Then, the slices were separated out and inverse-FFT was taken to obtain slices of images. The process of subsampling can be viewed from Appendix B, Data Preprocessing section.

Methods

Throughout the projects, all examined and implemented algorithms are provided below.

Transition from AUTO-MAP to U-Net

For accelerated k-space data acquisition in MRI, there are different deep learning frameworks.These frameworks can be used in k-space learning, before taking iFFT, image-domain reconstruction, after taking iFFT of the zero-filled k-space data or in an end-to-end image recovery. Automated Transform by Manifold Approximation (AUTOMAP), is an neural network scheme proposed for end-to-end recovery purposes. With fully-connected layers, AUTOMAP learns an optimal reconstruction function, irregardless of the acquisition type [3]. However, although AUTOMAP approach eliminates the Fourier transforming space, it was observed that AUTOMAP is able to successfully recover sufficiently small sized images. Further, in order to obtain a successful implementation, high number of samples are necessary to feed system in training. Finally, considering the available working environments in Colab, fully connected layers requires huge memory spaces [4]. The original implementation uses 3 fully-connected layers, which combined with the convolutional layers, requires more than 200 million parameters to train [3]. Hence, in order to observe and study a successful reconstruction mechanism, we have focused on single domain learning approaches available.

CNN Architecture on K-Space

Our initial strategy was to interpolate the undersampled k-space data first with an iterative CNN algorithm. The diagram is presented below.

(a) Overall Reconstruction Diagram

(b) Convolutional Neural Network Layers

Figure 2: Architecture

The network backbone consists total of 8 convolution layers, where 1-7 uses batch normalization and rectified linear unit (ReLU) for activation. The final layer uses linear activation. The first layer uses kernel of size 5x5, where the other layer kernels are 3x3. Since the k-space component is complex-valued, the input slice is divided into 2, to separate the real and imaginary parts. The convolutional layers 1-5 has 64 features, where as the layers 6, 7 and 8 has 32, 8, 2 respectively. The detailed analysis of the network can be viewed from Appendix A.

After the network, data consistency applied on the reconstructed k-space image, and then inputted again to the network. The procedure is repeated for the specified number of iterations. Finally, from the interpolated k-space that also assures data consistency, 2D inverse Fourier transform is applied to obtain the final reconstructed image.

The training of the network is made by randomized 7000 slices from the pre-processed data. The batch size is set to 100 and the initial learning rate was 0.0001. In order to allocate small memory and make the overall training computationally efficient, Adam algorithm was used for optimization [5]. Overall, the network was trained for total of 10 epochs. The network loss recorded was the Mean Squared Error.

However, throughout the implementation phase, the network structure seems to work poorly with reconstructing undersampled images with artifact, an impulse near the center. In addition, the noise-like artifacts stemming from undersampling was not eliminated which motivated us to abandon this architecture and focus on image-domain learning. The reconstructions made with the network is provided again in Appendix A and the implementation code is provided in Appendix B.

CNN Architecture on Image Domain

After leaving the k-space learning algorithm, we have focused on image-domain learning with the same CNN architecture. The image-domain translated algorithm is presented below.

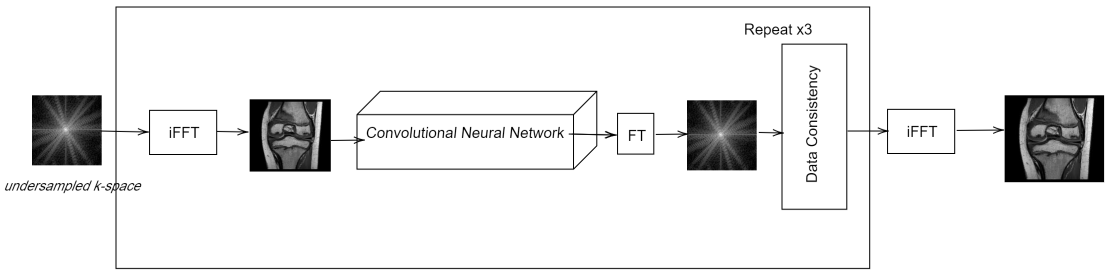


Figure 3: Implemented Architecture

Further, the CNN network is given as:

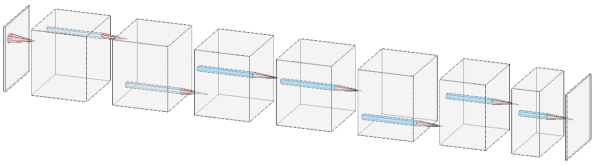


Figure 4: Network implemented

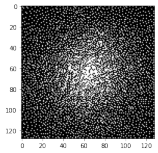
The input size to the network is (128,128,1) since the absolute version of images are fed to network, there is no need to separate real and imaginary components. Then, likewise the k-space domain network implementation, total of 8 convolutional layers were used. The layers 1-7 uses batch normalization along with ReLU activation function. The final convolutional layer uses the linear activation function. Finally, 1-5 layers have 64 features whereas the $6^{th}, 7^{th}$ and 8^{th} layers have 32, 8 and 1 respectively, since the output of the network again should be 128x128. An important point to highlight is the usage of zero-paddings to keep the image size in between the network stable which is different from the U-Net architecture, the max-pooling and upsampling parts were extracted. The model summary is provided below.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 64)	1664
batch_normalization (BatchNo	(None, 128, 128, 64)	256
conv2d_1 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_1 (Batch	(None, 128, 128, 64)	256
conv2d_2 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_2 (Batch	(None, 128, 128, 64)	256
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_3 (Batch	(None, 128, 128, 64)	256
conv2d_4 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_4 (Batch	(None, 128, 128, 64)	256
conv2d_5 (Conv2D)	(None, 128, 128, 32)	18464
batch_normalization_5 (Batch	(None, 128, 128, 32)	128
conv2d_6 (Conv2D)	(None, 128, 128, 8)	2312
batch_normalization_6 (Batch	(None, 128, 128, 8)	32
conv2d_7 (Conv2D)	(None, 128, 128, 1)	73

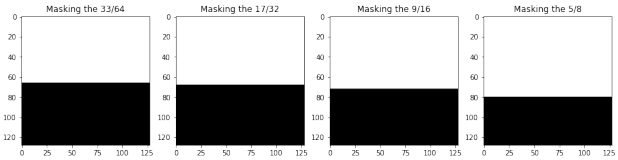
Total params: 171,665
Trainable params: 170,945
Non-trainable params: 720

Figure 5: Network Summary

From the acquired fully-sampled k-space images which were preprocessed and have 128x128 size, the masking masking applied that will be used in the training section. Since we want to analyze the model performance for two different sampling strategies, following masks were applied to the fully-sampled k-space images that will be used in training section.



(a) Poisson Disc Mask



(b) Partial k-space Masking

Figure 6: Maskings Used

The following network training sections will be provided in the sub-sections, for the different sampled training data.

Network Trained with Poisson Disc Sampled Images

The first masking is made with Poisson disc sampling method, with an acceleration factor of 4. We can see that the subsampling strategy is successful since it focuses on the origin, were most of the data lies. Further, it is quite common in compressed sensing MRI [6]. Hence, we reduce the scanning time to 1/4th without loosing information severely. However, since the high-frequency components are eliminated, we still have resolution problem to think upon. As an example, the image with poisson subsampling and its original is provided below.

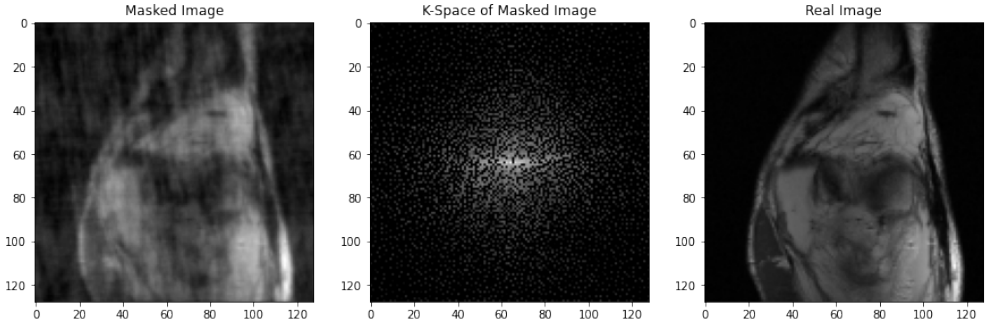


Figure 7: Example Poisson Disc Sampling

We can see the blurring of the image in general, with further noise-like structures occurring in the background. From the fully-sampled k-space images, randomly selected 7000 of them is taken for training process of the model. The masking was applied on top, to obtain undersampled k-space images. We then take its inverse Fourier transform to obtain the undersampled image and train our CNN model with these images, along with fully-sampled ones being their ground truth. The loss function for the model was Mean Squared Error, which can be shown as:

$$MSE = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (x_{ij} - yij)^2 \tag{1}$$

Likewise in the previously made architecture, batch size is set to 100 and the initial learning rate is 0.0001. Again, considering memory allocations and computational efficiency, Adam optimizer used for optimization. Finally, 0.2 portion of the total 7000 samples inputted to the system was used for validation. The overall training was done for 20 epochs. The loss function plot for validation and training set is provided after the partially acquired k-space section.

Network Trained with Partially Acquired k-space Data

For training the network with partially acquired k-space data, following proportions were considered: 33/64, 17/32, 9/16 and 5/8 which are the maskings provided in 6b. For each of the fully acquired k-space data, the uniform random sampling was applied in order to randomly select the mask to choose. Hence, all of the maskings were applied to the data in a random manner, with nearly equal number of times. Again, a sample of the undersampled image along with its k-space and reference is provided below.

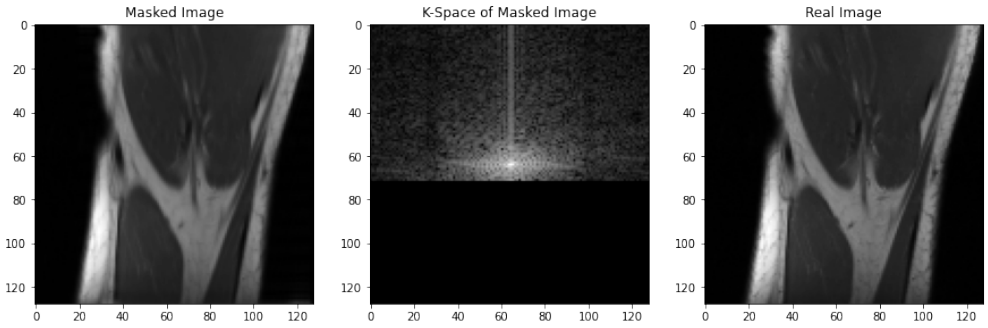
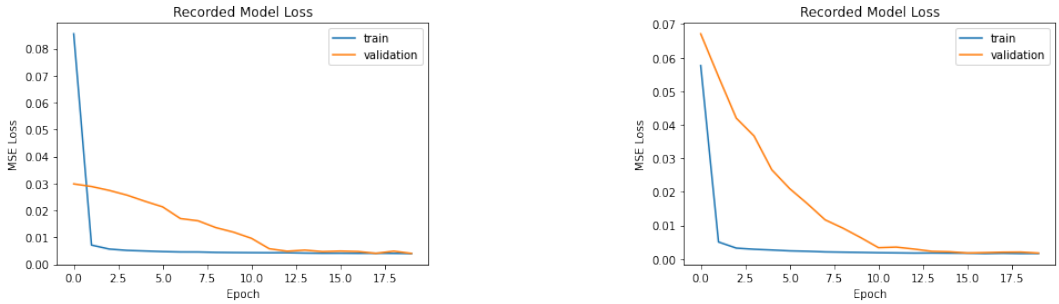


Figure 8: Example Partial k-space Sampling

Finally, again for the total of 20 epochs, following loss functions were observed for training and validation sets, for these two different masking sets.



(a) Loss Function for Poisson Disc Sampled Images

(b) Loss Function for Partial k-space

Figure 9: Recorded Loss Functions

The convergence of loss can be noticed from both plots.

Projection Onto Convex Sets (POCS) Algorithm

POCS is an iterative image reconstruction method. It is mainly used for partial k-space data and can give better results than similar methods such as Homodyne Reconstruction or PCCS as the acquired k-space data fraction decreases. The main goal behind it is to find a point in the intersection of two closed convex sets.

Implementation

The implementation of POCS can be better explained step by step:

1. Get the symmetric central section of the partial k-space data. Fill the rest with zeros to obtain m_k.
2. Take the Inverse 2D FFT of m_k to obtain $m_s(x, y)$ and take it's phase to obtain $p(x, y)$ where:

$$p(x, y) = e^{i\angle m_s} \tag{2}$$

3. Take the Inverse 2D FFT of the partial k-space data to obtain $m_i(x, y)$.
4. Multiply the magnitude of $m_i(x, y)$ with $p(x, y)$ to obtain the Phase Constrained Image data.
5. Finally, take the 2D FFT of Phase Constrained Image Data to obtain the reconstructed k-space data, replace the reconstructed k-space data with the acquired data at sampled k-space locations.
6. Repeat 1-5 using the reconstructed k-space data as the partial data for a number of iterations.

We have found it sufficient to repeat for 5 iterations since POCS algorithm is known to have fast convergence.

Compressed Sensing (CS) Algorithm

Compressed Sensing is a Medical Image Reconstruction method that mainly relies on sparsity. Even though most of the time the image to be reconstructed is not sparse, it often is in another domain. The main idea behind Compressed Sensing is to compress the image by transforming it to a sparse domain, threshold the coefficients of that domain that can be considered noise and inverse transform the resulting sparse domain image to obtain a reconstruction which has smoother features and eliminated artifacts. Addition to this, a data consistency step is added which replaces the acquired pixels of the reconstructed k-space with the sampled k-space data. This algorithm also needs a few iterations to reconstruct the image in higher quality.

Implementation

Compressed sensing method is tested on two different random sampling methods which are Poisson undersampling and a random undersampling mask generated manually using python. The Poisson mask data samples k-space data closer to the center of k-space whereas manually generated mask does not have a specific sampling pattern. In order to generate the Poisson sampling mask, Pythons' sigpy.mri package's function poisson [7] is used which simply generates an undersampling mask of 1s and 0s.

For the sparse domain transform, the Wavelet domain in chosen since it sparsely represents most natural images including our data. To get the forward and inverse Wavelet Transforms of the undersampled data, we have used the PyWavelet [8] library of python which is used for 4 levels of decomposition and reconstruction. After the wavelet transform, the coefficients are thresholded by the 1 / 500th of the maximum coefficient with the following soft thresholding:

$$S_\beta(x) = \frac{x}{|x|} max(0, |x| - \beta) \tag{3}$$

Where x is the coefficient and β is the 1 / 500th of the maximum coefficient. After this we take the inverse Wavelet transform of the resulting image and perform a data consistency step by replacing the reconstructed k-space data with the acquired data at sampled k-space locations. This process is repeated for 1000 iterations. Thus, it is expected to reduce artifacts and generally smooth out the image since by thresholding in wavelet domain, we are actually increasing sparsity and eliminating noise signals.

Results

For both sampling strategy the obtained results are provided below in the subsections.

Poisson Disc Sampling CNN vs. CS Results

For the Poisson Disc sampled images, random test data was reconstructed with the network. Following is the example reconstruction. The SSIM value is recorded to be 0.84 whereas the PSNR is 28.31.

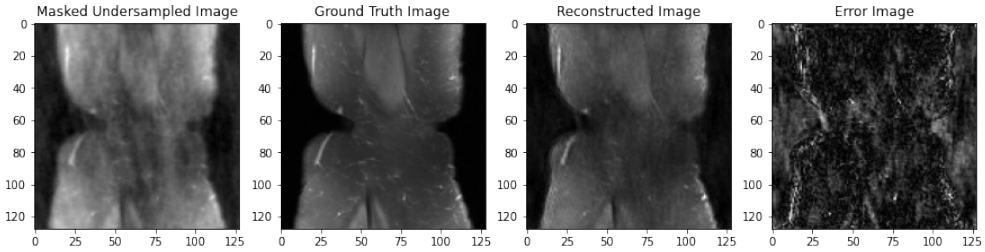


Figure 10: Example Reconstruction with Poisson Disk Sampling, Acceleration Rate=4

Further, CS algorithm on the same test sample provided the following results. The recorded SSIM value was 0.81 and PSNR was 27.51.

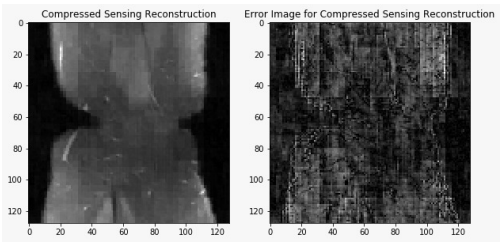


Figure 11: Example CS Reconstruction with Poisson Disk Sampling, Acceleration Rate=4

Although we have not fed the system with poisson disc sampling with acceleration rate of 2, we have tried to see the reconstruction results of the network for the acceleration rate of 2. The following results are on an experimental data reconstruction. The SSIM value was recorded to be 0.93 and PSNR was 30.84.

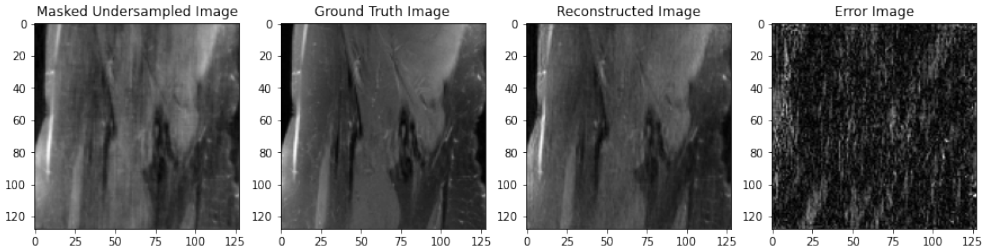


Figure 12: Example Reconstruction with Poisson Disk Sampling, Acceleration Rate=2

The CS algorithm provided the following results. The recorded SSIM value was 0.91 and PSNR was 34.4.

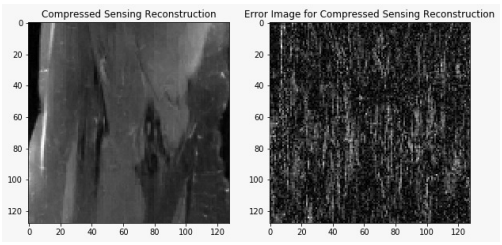
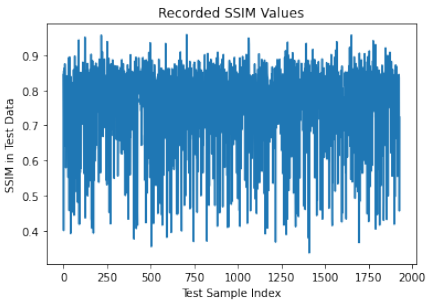
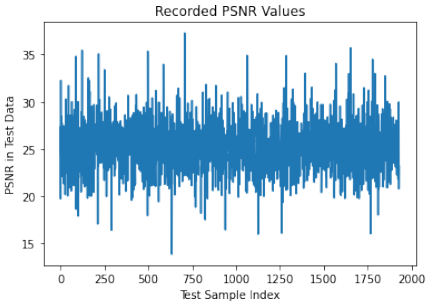


Figure 13: Example CS Reconstruction with Poisson Disk Sampling, Acceleration Rate=2

Finally, for the total of 1929 test data obtained, the network reconstruction was performed and SSIM PSNR values for each is recorded. The following plots shows the results for each test data.



(a) SSIM Records for Each Test Data



(b) PSNR Records for Each Test Data

Figure 14: Recorded Quality Metrics

We can see that the reconstruction network provides for this sampling strategy is not stable in a sense that it can result quite high or low SSIM and PSNR values. The plots above strengthens this observation. The mean of the SSIM for all of the test data is recorded to be 0.75 with a standard deviation of 0.12. For the PSNR metric, the mean was 25.3 and standard deviation was 2.6. However, compared with the state-of-art Compressed Sensing method, we see that reconstruction of both undersampled images yielded higher accuracy with network.

Partial K-Space Acquisition CNN vs. POCS Results

In order to compare, random indexes from the test dataset were taken. For each of the sampling masks, the reconstructions were obtained. Following is an example reconstruction, where the 33/64 portion of the k-space were masked. The SSIM and PSNR values are recorded to be 0.912 and 26.54 respectively.

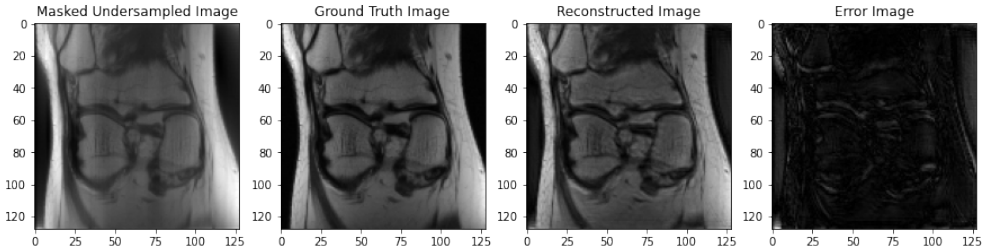


Figure 15: Example Partial Reconstruction with 33/64 of the k-space

Whereas, the iterative POCS algorithm on the same test data provided the below results. The recorded SSIM value was 0.929 and PSNR value was 28.77.

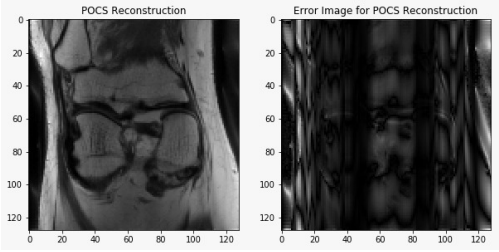
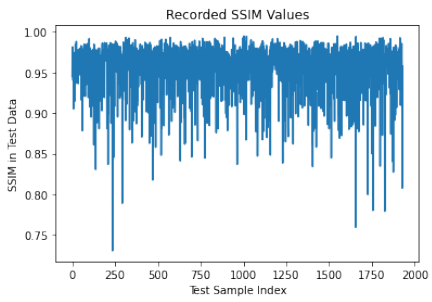


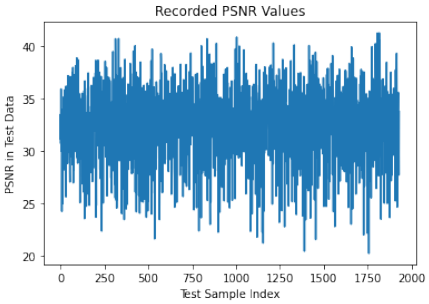
Figure 16: Example POCS Reconstruction with 33/64 of the k-space

It is essential to state that error images are normalized, which magnifies the underlying small artifact patterns in POCS, which is predictable and exemplified in [9]. The results for other test samples that was

masked with other proportions reconstructions made with network and POCS are provided in Appendix section A. Finally, the test data of size 1929 samples were masked with the one of 4 proportions randomly and SSIM and PSNR values were recorded for each. The below plots shows these recorded values for each test data.For the overall performance of the network, the mean SSIM was calculated to be 0.95, whereas the PSNR mean is 32.42. We can see a lower standard deviation of SSIM compared with the previous case, which is 0.029 and 3.45 for PSNR.



(a) SSIM Records for Each Test Data



(b) PSNR Records for Each Test Data

Figure 17: Recorded Quality Metrics

Discussion

With single-coil reconstruction task implemented above, we were able to compare the state-of-art reconstruction for undersampled k-space that does not use deep learning, with a proposed deep learning scheme. After the training process, the network did not propose any computational complexity compared with the other methods.

Considering the networks behavior for Poisson disc sampling, the network was able to slightly outperform the Compressed Sensing Algorithm with higher SSIM values along with higher PSNR value for the test data sampled with acceleration rate of 4. However, it should be noted that on the overall test data, the network was not stabilized enough that it can yield high or low values for a random image.

For partial k-space acquisition scheme, the network provided nearly identical results with POCS algorithm, however POCS were slightly better in reconstruction. It should be noted that the CNN network structure was trained with relatively small training dataset of 7000 -including the validation dataset also- due to several constraints such as memory space and training time. However, the network structure was able to provide highly accurate results for the undersampled images, without compromising the computational complexity after training. This is indeed a motivation to make detailed observations for the network performance and modifying the designed network. Although the network was able to provide high-fidelity images, the model loss function can be analysed in l1-loss and hyperparameters can be tuned. The U_net architecture was modified with eliminating the max-pooling and upsampling sections and may be a subject to further examination of the model. Overall, we have seen that although parallel imaging is a novel technique, single-coil MRI machines are again in use [2]. The single-coil experiment made can be generalized to the parallel imaging scheme with more complex observations. A simple deep learning architecture was able to provide accurate solutions, which suggests that these architectures can potentially accelerate data acquisition further and outperform the reconstruction schemes that are in use today [10].

References

[1] C. M. Hyun, H. P. Kim, S. M. Lee, S. Lee, and J. K. Seo, "Deep learning for undersampled mri reconstruction," *Physics in Medicine & Biology*, vol. 63, no. 13, p. 135007, 2018.

[2] J. Zbontar, F. Knoll, A. Sriram, M. J. Muckley, M. Bruno, A. Defazio, M. Parente, K. J. Geras, J. Katsnelson, H. Chandarana, *et al.*, "fastmri: An open dataset and benchmarks for accelerated mri," *arXiv preprint arXiv:1811.08839*, 2018.

[3] E. J. Topol, "High-performance medicine: the convergence of human and artificial intelligence," *Nature medicine*, vol. 25, no. 1, pp. 44–56, 2019.

[4] Y. Han, L. Sunwoo, and J. C. Ye, "k-space deep learning for accelerated mri," *IEEE transactions on medical imaging*, 2019.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[6] E. Levine, B. Daniel, S. Vasanawala, B. Hargreaves, and M. Saranathan, "3d cartesian mri with compressed sensing and variable view sharing using complementary poisson-disc sampling," *Magnetic resonance in medicine*, vol. 77, no. 5, pp. 1774–1785, 2017.

[7] O. Frank, "sigpy," Oct 2019.

[8] G. R. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O'Leary, "Pywavelets: A python package for wavelet analysis," Apr 2019.

[9] M. Chiew, "A brief overview of some partial fourier mr image reconstruction methods."

[10] C. M. Sandino, N. Dixit, J. Y. Cheng, and S. S. Vasanawala, "Deep convolutional neural networks for accelerated dynamic magnetic resonance imaging,"

Appendix

Appendix A: Supplementary Images

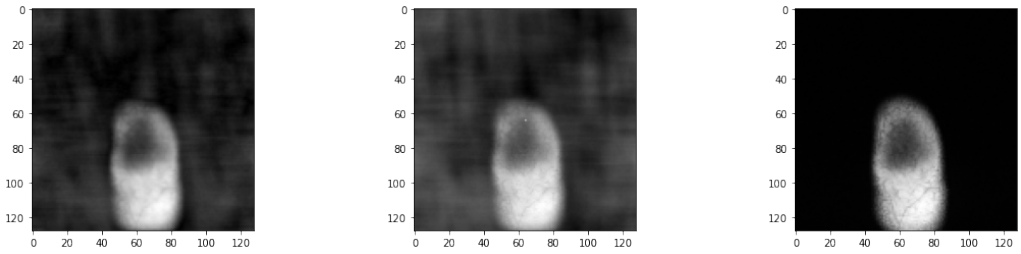
K-Space Learning

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 128)	6528
batch_normalization (BatchNo	(None, 128, 128, 128)	512
conv2d_1 (Conv2D)	(None, 128, 128, 64)	204864
batch_normalization_1 (Batch	(None, 128, 128, 64)	256
conv2d_2 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_2 (Batch	(None, 128, 128, 64)	256
conv2d_3 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_3 (Batch	(None, 128, 128, 64)	256
conv2d_4 (Conv2D)	(None, 128, 128, 64)	36928
batch_normalization_4 (Batch	(None, 128, 128, 64)	256
conv2d_5 (Conv2D)	(None, 128, 128, 32)	18464
batch_normalization_5 (Batch	(None, 128, 128, 32)	128
conv2d_6 (Conv2D)	(None, 128, 128, 8)	2312
batch_normalization_6 (Batch	(None, 128, 128, 8)	32
conv2d_7 (Conv2D)	(None, 128, 128, 2)	146

Total params: 344,794
Trainable params: 343,946
Non-trainable params: 848

Figure 18: Network Summary

An example of reconstructions made:



(a) Undersampled Image (b) Reconstructed Image (c) Reference Image

Figure 19: Example Reconstruction

Additional Results for Partial k-space Reconstruction

The results for other proportional maskings are provided below. The network was able to reconstruct the 17/32 proportional k-space data with SSIM value of 0.94 and PSNR of 26.54.

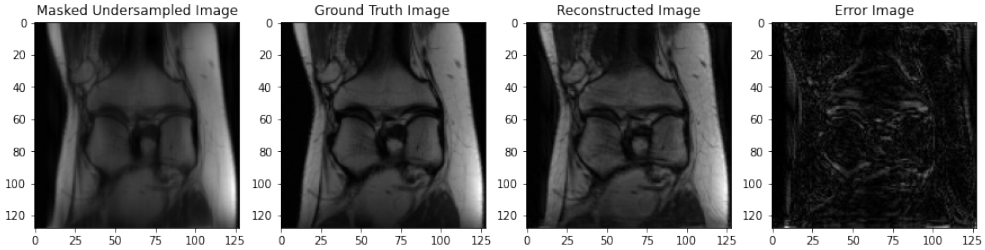


Figure 20: Example Partial Reconstruction with 17/32 of the k-space

Whereas, the POCS algorithm achieved SSIM value of 0.96 and PSNR 35.2. The reconstructions provided below.

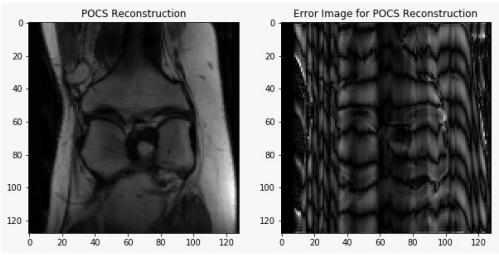


Figure 21: Example POCS Reconstruction with 17/32 of the k-space

For another sampling proportion, we take the CNN reconstruction made for masking the 9/16 portion of data. The reconstruction results can be viewed below. The SSIM and PSNR values were recorded to be 0.937 and 33.93 respectively. The POCS algorithm was able to achieve SSIM and PSNR values of 0.98 and 41.53 respectively.

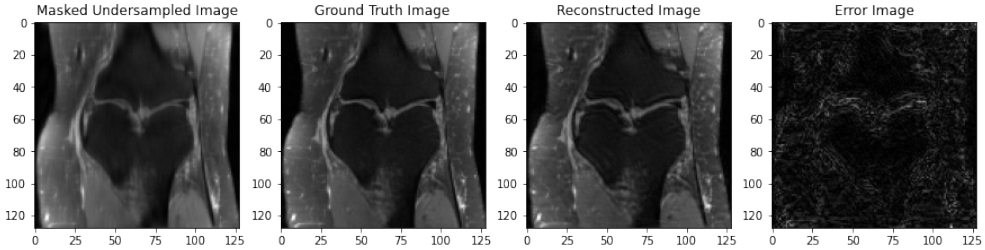


Figure 22: Example Partial Reconstruction with 9/16 of the k-space

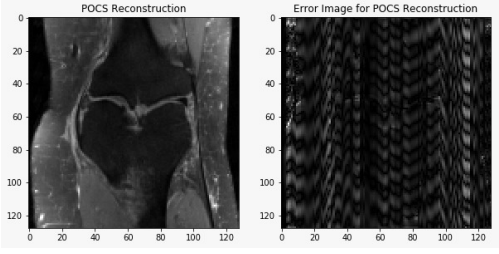


Figure 23: Example POCS Reconstruction with 9/16 of the k-space

Finally, for the 5/8 rate of acquisition, the results obtained from both network and POCS algorithm are provided below. The network was able to achieve 0.96 SSIM value along with 35.7 PSNR. The POCS algorithm on the other hand achieved 0.99 SSIM and 48.9 PSNR.

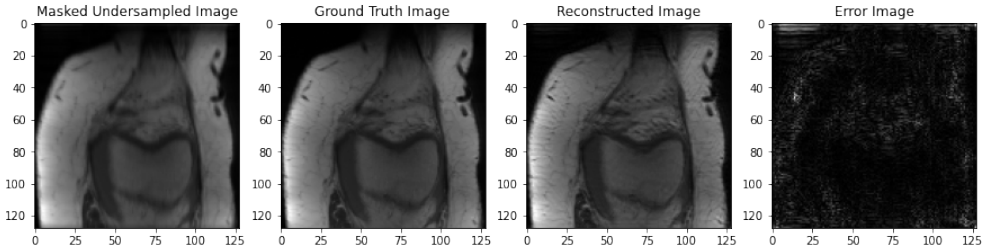


Figure 24: Example Partial Reconstruction with 5/8 of the k-space

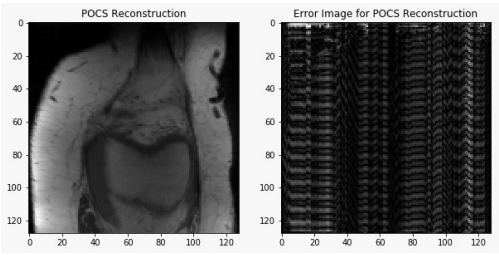


Figure 25: Example POCS Reconstruction with 5/8 of the k-space

Appendix B: Project Codes

Data PreProcessing

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

# In[ ]:


from h5py import File

from numpy import array
from pathlib import Path
import numpy as np
import pandas as pd
import h5py
from skimage.transform import rescale, resize, downscale_local_mean
import os

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
import io
import matplotlib.pyplot as plt
import scipy.fftpack as spfft


# In[2]:


file_path = Path('F:/User/Downloads/knee_singlecoil_data')
df_kspace = []
df_reconim = []


# In[3]:


def centerCrop(data):
    slices,width,height=data.shape
    new_s=320
    left_lim=int((width-new_s)/2)
    top_lim=int((height-new_s)/2)

    data_n=np.zeros((slices,128,128))

    for i in range(slices):
        im=data[i,:,:]
        im=im[left_lim:left_lim+new_s,top_lim:top_lim+new_s]
        im=resize(im,(128,128), anti_aliasing=True)
        #normalize im:
        ma = np.max(im)
        mi = np.min(im)
        im = (im - mi) / (ma - mi)
        data_n[i,:,:]=im
```

```
    return data_n


# In[5]:


counter=1
# Parent Directory
df_kspace = []
df_reconim = []
parent_dir = "C:/Users/user/Desktop"
direct = "UPDATED_DATA3"
fil_path = os.path.join(parent_dir, direct)
try:
    os.mkdir(fil_path)
except OSError as error:
    print(error)

for entry in file_path.iterdir():
    print(entry.name)
    f=h5py.File(str('F:/User/Downloads/knee_singlecoil_data/'+entry.name),'r')
    if counter ==250:
        break
    data_im=np.array(f[list(f.keys())[3]])
    data_im_new=centerCrop(data_im)

    data_k=fftSlices(data_im_new)
    print(data_k.shape)

    hf = h5py.File(fil_path+'/'+str(counter)+'.h5', 'w')
    hf.create_dataset('image', data=data_k)
    hf.close()

    counter=counter+1



# In[18]:


print(len(df_reconim))


# In[6]:


print(data_im.shape)
    checkdata=np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(data_k[15,:,:])))
    print(checkdata.shape)
    plt.imshow(np.abs(checkdata[:,:]))
    plt.gray()
    plt.show()



# In[ ]:
```

```
model = keras.Sequential(
    [
       #layer 1
        layers.Conv2D(128,(3,3),activation="relu",input_shape=(128,128,2)),
        layers.BatchNormalization(),
        layers.Conv2D(128,(3,3),activation="relu"),
        layers.BatchNormalization(),
        layers.Conv2D(128,(3,3),activation="relu"),
        layers.BatchNormalization(),
        layers.Conv2D(128,(3,3),activation="relu"),
        layers.BatchNormalization(),
        layers.Conv2D(128,(3,3),activation="relu")
   ]
)


# In[ ]:


def DataCons(recon,real,mask):
    notmask=np.logical_not(mask).astype(int)
    reconmasked=np.multiply(notmask,reconmasked)
    data_cons=np.sum(reconmasked,real)

    return data_cons


# In[ ]:


def Network(model,mask,input,real):
  recon=input
  for i in range(5):
    out=model.predict(recon)
    recon=DataCons(recon,input,mask)

  recon_im=spfft.ifft2(spfft.fftshift(recon))    #YANLIS DUZELT
  return recon_im


# In[ ]:


def trainNetwork(model, X, Y):
  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
  model.fit(X, y, epochs=50, batch_size=100, verbose=0)



# In[4]:


def fftSlices(data_im):
    kspace_im=np.zeros(data_im.shape,dtype = complex)
    slices=data_im.shape[0]
    print(slices)
    for i in range(slices):
```

```
        kspace_im[i,:,:]=spfft.fftshift(spfft.fft2(spfft.ifftshift(data_im[i,:,:])))

    return kspace_im



# In[6]:


fu=h5py.File(str('C:/Users/user/Desktop/UPDATED_DATA2/15.h5'),'r')

datam=np.array(fu[list(fu.keys())[0]])
print(datam.shape)
```

k-space Learning

```
# -*- coding: utf-8 -*-
"""kspacelearning.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/10DCQplOKamsteDYn6scA5-9_0j_FqfSZ

Imports:
"""

import h5py
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
from google.colab import files
import io
import matplotlib.pyplot as plt
import scipy.fftpack as spfft
from pathlib import Path
from os import listdir, walk
from os.path import isfile, join, isdir, abspath
import glob
!pip install sigpy
import sigpy.mri as mri
import random

from tensorflow.keras.callbacks import ModelCheckpoint

from skimage.measure import compare_ssim
import argparse
import imutils
import cv2

from google.colab import drive
drive.mount('/content/drive')

root='/content/drive/My Drive/UPDATED_DATA3'
kspace_images = []
count=1
```

```
for folder in glob.glob(root + '/*'):
  if count==250:
    break
  f=h5py.File(folder,'r')
  data_k=np.array(f[list(f.keys())[0]])
  slices, weight, height=data_k.shape
  print(data_k.shape)
  for i in range(slices):
    kspace_images.append(data_k[i,:,:])
  count=count+1

dat=kspace_images[60]
print(dat.shape)
plt.figure()
plt.imshow(abs(dat))
real_im=spfft.ifftshift(spfft.ifft2(spfft.fftshift(dat)))
plt.figure()
plt.imshow(abs(real_im))

"""Obtain & Preprocess Images"""

mask = mri.poisson([128,128], 4, crop_corner=False)
plt.figure()
plt.imshow(abs(mask), cmap = 'gray')
plt.show()

"""SUBSAMPLE THE IMAGES:"""

random.shuffle(kspace_images)
tt_gt = np.zeros((8000,128,128,2))
for i in range(8000):
  tt_gt[i,:,:,0]=np.real(kspace_images[i])
  tt_gt[i,:,:,1]=np.imag(kspace_images[i])

train_gt = tt_gt[0:7000,:,:,:]
masked_train = np.zeros((7000,128,128,2))

for i in range(len(train_gt)):
  full_k=train_gt[i,:,:,:]
  masked_train[i,:,:,0]=np.multiply(full_k[:,:,0],mask)
  masked_train[i,:,:,1]=np.multiply(full_k[:,:,1],mask)

kspace_images=[]

print(len(masked_train))
bib=masked_train[420]
plt.figure()
plt.imshow(abs(bib), cmap = 'gray')
print(bib.shape)
bib_im=np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(bib)))
plt.figure()
plt.imshow(abs(bib_im), cmap = 'gray')



n1 = data.get('reconstruction_rss')
print(n1)
arr=n1[25,:,:]
```

```
# Plot the grid
plt.imshow(abs(arr))
plt.gray()
plt.show()

"""PREPROCESS THE DATA:"""

model = keras.Sequential(
    [
        #layer 1
        layers.Conv2D(64,(5,5),activation="relu",input_shape=(128,128,2), padding='same'),
        layers.BatchNormalization(),
        #layer 2
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 3
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 4
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 5
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 1:
        layers.Conv2D(32,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 2:
        layers.Conv2D(8,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 3:
        layers.Conv2D(2,(3,3),activation="linear", padding='same')
   ]
)

model.summary()

def DataCons(reconn,real,mask):
  notmask=np.logical_not(mask).astype(int)
  recon=reconn[0,:,:,:]
  reconmasked=np.zeros(recon.shape)
  data_cons=np.zeros(recon.shape)
  for i in range(2):
    reconmasked[:,:,i]=np.multiply(notmask,recon[:,:,i])
    data_cons[:,:,i]= reconmasked[:,:,i] + real[0,:,:,i]


  recon_dat_fin=np.zeros((1,128,128,2))
  recon_dat_fin[0,:,:,0]=data_cons[:,:,0]
  recon_dat_fin[0,:,:,1]=data_cons[:,:,1]

  return recon_dat_fin

def Network(model,mask,inputt):
  recon=inputt
  for i in range(5):
    out=model.predict(recon)
```

```
    print(out.shape)
    recon=DataCons(out,inputt,mask)


  recon_out=np.zeros((128,128), dtype=complex)
  recon_out=recon[0,:,:,0] + 1j*recon[0,:,:,1]
  recon_im=np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(recon_out)))
  return recon_im

def trainNetwork(model, X, Y):
  filepath="weights.best2.hdf5"
  checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=
      True, mode='max')
  callbacks_list = [checkpoint]
  opt = keras.optimizers.Adam(learning_rate=0.0001)
  model.compile(loss='mean_squared_error', optimizer='adam',metrics=['MeanSquaredError'])
  history = model.fit(X, Y, epochs=10, batch_size=100, validation_split = 0.2, verbose=1,
      callbacks=callbacks_list)
  return history

a=np.ones((1,128,128,2))
model.predict(a)

"""TRAIN NETWORK:"""

history = trainNetwork(model,masked_train,train_gt)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

test_gt = tt_gt[7000:len(tt_gt),:,:,:]

"""TESTING NETWORK:"""

test_sample=tt_gt[7999,:,:,:]
print(test_sample.shape)
masked_test=np.zeros((1,128,128,2))
masked_test[0,:,:,0]=np.multiply(test_sample[:,:,0],mask)
masked_test[0,:,:,1]=np.multiply(test_sample[:,:,1],mask)

recon_im=Network(model,mask,masked_test)
ma=np.max(recon_im)
mi=np.min(recon_im)
```

```
recon_im=(recon_im-mi)/(ma-mi)



plt.imshow(np.abs(recon_im))
plt.gray()
plt.show()

real_im=np.zeros((128,128), dtype=complex)
real_im=test_sample[:,:,0]+ 1j*test_sample[:,:,1]
plt.imshow(np.abs(real_im))
plt.gray()
plt.show()
real_im=np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(real_im)))

plt.imshow(np.abs(real_im))
plt.gray()
plt.show()

masked_im=masked_test[0,:,:,0]+ 1j*masked_test[0,:,:,1]
masked_im=np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(masked_im)))
plt.imshow(np.abs(masked_im))
plt.gray()
plt.show()

masked_im=masked_test[0,:,:,0]+ 1j*masked_test[0,:,:,1]
masked_im=np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(masked_im)))
plt.imshow(np.abs(masked_im))
plt.gray()
plt.show()

def QA(recon, ref):
  recon_im=(recon-np.min(recon))/(np.max(recon)-np.min(recon))
  ref_im=(recon-np.min(recon))/(np.max(recon)-np.min(recon))
  err_im=recon_im - ref_im

  plt.figure()
  plt.imshow(np.abs(err_im))
  plt.gray()
  plt.show()


  #SSIM and PSNR
  ssim_none = ssim(recon_im, ref_im, vmin=0, vmax=1)

filepath="poisson_test_data.hdf5"

a = np.array([1+2j, 3+5j])
b=np.real(a)
c=np.imag(a)
print(b)
print(c)
d=masked_train[3]
print(d[:,:,1])

a=np.array([1,0,0,1])
a=np.logical_not(a).astype(int)
a
  recon_im=spfft.ifft2(spfft.fftshift(data_cons))
```

```
print(masked_train_arr.shape)
```

Image-Domain Learning-Poisson Sampling

```
# -*- coding: utf-8 -*-
"""ImageDomain_Dl.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1wrSEVmOhKhqt1cDim4A4u0KtnZFu-fk7

Imports:
"""

import h5py
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
from google.colab import files
import io
import matplotlib.pyplot as plt
import scipy.fftpack as spfft
from pathlib import Path
from os import listdir, walk
from os.path import isfile, join, isdir, abspath
import glob
!pip install sigpy
import sigpy.mri as mri
import random

from tensorflow.keras.callbacks import ModelCheckpoint
from skimage import data, img_as_float
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import peak_signal_noise_ratio as psnr
import argparse
import imutils
import cv2
from math import log10, sqrt

from google.colab import drive
drive.mount('/content/drive')

root='/content/drive/My Drive/UPDATED_DATA3'
images = []
count=1
for folder in glob.glob(root + '/*'):
  if count==250:
    break
  f=h5py.File(folder,'r')
  data_k=np.array(f[list(f.keys())[0]])
  slices, weight, height=data_k.shape
  print(data_k.shape)
  for i in range(slices):
    data_im=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(data_k[i,:,:]))))
```

```
    #normalize im:
    ma = np.max(data_im)
    mi = np.min(data_im)
    data_im = (data_im - mi) / (ma - mi)
    images.append(data_im)
  count=count+1

dat=images[60]
print(dat.shape)
plt.figure()
plt.imshow(abs(dat))
plt.gray()
plt.title('Image Obtained')
print(np.max(dat))
print(np.min(dat))
real_im=spfft.fftshift(spfft.fft2(spfft.ifftshift(dat)))
plt.figure()
plt.imshow(abs(np.log(real_im)+1))
plt.gray()
plt.title('K-Space of Image')

mask = mri.poisson([128,128], 4, crop_corner=False)
plt.figure()
plt.imshow(abs(mask), cmap = 'gray')
plt.show()

random.shuffle(images)
images=np.array(images)
print(images.shape)

train_gt =images[0:7000,:,:,np.newaxis]
test_gt = images[7000:len(images),:,:,np.newaxis]
masked_train = np.zeros((7000,128,128,1))

for i in range(len(train_gt)):
  full_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(train_gt[i,:,:,0])))
  masked_traink=np.multiply(full_k,mask)
  masked_im=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_traink))))
  #normalize im:
  ma = np.max(masked_im)
  mi = np.min(masked_im)
  masked_im = (masked_im - mi) / (ma - mi)
  masked_train[i,:,:,0]=masked_im

images=[]

print(masked_train.shape)
print(train_gt.shape)
print(test_gt.shape)

f, (ax1, ax2, ax3)  = plt.subplots(1,3)
f.set_figheight(15)
f.set_figwidth(15)

ax1.imshow(masked_train[80,:,:,0], cmap = 'gray')
ax1.set_title('Masked Image')

maskedk=spfft.fftshift(spfft.fft2(spfft.ifftshift(masked_train[80,:,:,0])))
```

```
ax2.imshow(np.log(np.abs(masked_traink)+1),cmap = 'gray')
ax2.set_title('K-Space of Masked Image')

ax3.imshow(train_gt[80,:,:,0], cmap = 'gray')
ax3.set_title('Real Image')

model = keras.Sequential(
    [
        #layer 1
        layers.Conv2D(64,(5,5),activation="relu",input_shape=(128,128,1), padding='same'),
        layers.BatchNormalization(),
        #layer 2
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 3
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 4
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 5
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 1:
        layers.Conv2D(32,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 2:
        layers.Conv2D(8,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 3:
        layers.Conv2D(1,(3,3),activation="linear", padding='same')
   ]
)

def DataCons(reconn,real,mask):
  notmask=np.logical_not(mask).astype(int)
  recon_k=np.zeros((128,128),dtype=complex)
  real_k=np.zeros((128,128),dtype=complex)
  recon_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(reconn[0,:,:,0])))
  real_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(real[0,:,:,0])))

  reconmasked=np.multiply(notmask,recon_k)
  recon_corr=real_k+reconmasked
  image=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(recon_corr))))
  #normalize im:
  ma = np.max(image)
  mi = np.min(image)
  image = (image - mi) / (ma - mi)
  image=image[np.newaxis,:,:,np.newaxis]
  return image

def trainNetwork(model, X, Y):
  filepath="/content/drive/My Drive/PROJECT_RESULTS/IMAGEDOMAIN_best3.hdf5"
  checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True
      , mode='min')
  callbacks_list = [checkpoint]
  opt = keras.optimizers.Adam(learning_rate=0.0001)
  model.compile(loss='mean_squared_error', optimizer='adam',metrics=['MeanSquaredError','
```

```
      Accuracy'])
  history = model.fit(X, Y, epochs=20, batch_size=100, validation_split = 0.2, verbose=1,
      callbacks=callbacks_list)
  return history

history = trainNetwork(model,masked_train,train_gt)

# plot loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Recorded Model Loss')
plt.ylabel('MSE Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()

def Network(model,mask,inputt):
  recon=inputt
  for i in range(3):
    out=model.predict(recon)
    recon=DataCons(out,inputt,mask)
  return recon

"""TEST:"""

test_sample=test_gt[400,:,:,:]
masked_test=np.zeros((1,128,128,1))
test_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(test_sample[:,:,0])))
mask1=mri.poisson([128,128], 4, crop_corner=False)
masked_testk=np.multiply(test_k,mask1)
masked_test[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk))))


recon_im=Network(model,mask1,masked_test)
ssim_res,psnr_res,err_im= QA(recon_im[0,:,:,0],test_sample[:,:,0])

print('SSIM Achieved: '+ str(ssim_res))
print('PSNR Achieved:'+ str(psnr_res))

f, (ax1, ax2, ax3, ax4)  = plt.subplots(1,4)
f.set_figheight(15)
f.set_figwidth(15)
ax1.imshow(np.abs(masked_test[0,:,:,0]), cmap='gray')
ax1.set_title('Masked Undersampled Image')

ax2.imshow(np.abs(test_sample[:,:,0]), cmap='gray')
ax2.set_title('Ground Truth Image')

ax3.imshow(np.abs(recon_im[0,:,:,0]), cmap='gray')
ax3.set_title('Reconstructed Image')

ax4.imshow(np.abs(err_im), cmap='gray')
ax4.set_title('Error Image')

#WITH MASKING 2:
mask2 = mri.poisson([128,128], 2, crop_corner=False)
test_sample2=test_gt[110,:,:,:]
test_k2=spfft.fftshift(spfft.fft2(spfft.ifftshift(test_sample2[:,:,0])))
```

```
masked_test2=np.zeros((1,128,128,1))
masked_testk2=np.multiply(test_k2,mask2)
masked_test2[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk2))))

recon_im2=Network(model,mask2,masked_test2)
ssim_res2,psnr_res2,err_im2= QA(recon_im2[0,:,:,0],test_sample2[:,:,0])
print('SSIM Achieved: '+ str(ssim_res2))
print('PSNR Achieved:'+ str(psnr_res2))

f, (ax11, ax22, ax33, ax44)  = plt.subplots(1,4)
f.set_figheight(15)
f.set_figwidth(15)
ax11.imshow(np.abs(masked_test2[0,:,:,0]), cmap='gray')
ax11.set_title('Masked Undersampled Image')

ax22.imshow(np.abs(test_sample2[:,:,0]), cmap='gray')
ax22.set_title('Ground Truth Image')

ax33.imshow(np.abs(recon_im2[0,:,:,0]), cmap='gray')
ax33.set_title('Reconstructed Image')

ax44.imshow(np.abs(err_im2), cmap='gray')
ax44.set_title('Error Image')

def QA(recon, ref):
  recon_im=(recon-np.min(recon))/(np.max(recon)-np.min(recon))
  ref_im=(ref-np.min(ref))/(np.max(ref)-np.min(ref))
  err_im=np.abs(recon_im - ref_im)

  #SSIM and PSNR
  ssim_res = ssim(recon_im, ref_im, vmin=0, vmax=1)
  psnr_res= psnr(ref_im, recon_im)

  return ssim_res, psnr_res, err_im

np.save('/content/drive/My Drive/PROJECT_RESULTS/poisson_test_data.h5', test_gt)
np.save('/content/drive/My Drive/PROJECT_RESULTS/poisson_train_data.h5', train_gt)

ssims=np.zeros((len(test_gt),1))
psnrs=np.zeros((len(test_gt),1))
for i in range(len(test_gt)):
  testdata=test_gt[i,:,:,:]
  perct=np.random.randint(4)
  masked_test=np.zeros((1,128,128,1))
  test_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(testdata[:,:,0])))
  mask1= mri.poisson([128,128], 4, crop_corner=False)
  masked_testk=np.multiply(test_k,mask1)
  masked_test[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk))))

  reconn=Network(model,mask1,masked_test)
  ssims[i,0],psnrs[i,0],err_im= QA(reconn[0,:,:,0],testdata[:,:,0])

plt.plot(ssims)
plt.title('Recorded SSIM Values')
plt.ylabel('SSIM in Test Data')
plt.xlabel('Test Sample Index')
plt.show()
```

```
plt.plot(psnrs)
plt.title('Recorded PSNR Values')
plt.ylabel('PSNR in Test Data')
plt.xlabel('Test Sample Index')
plt.show()

#MEAN VALUES:
ssim_mean=np.mean(ssims)
print('SSIM mean: '+str(ssim_mean))
print('SSIM STD:'+ str(np.std(ssims)))
psnr_mean=np.mean(psnrs)
print('PSNR mean: '+str(psnr_mean))
print('SSIM STD:'+ str(np.std(psnrs)))
```

Image-Domain Learning-Partial k-space

```
# -*- coding: utf-8 -*-
"""ImageDomainPartial.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1QHatOKJwFlalVGn1TGOqUM4DS1Wkwycs
"""

import h5py
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
from google.colab import files
import io
import matplotlib.pyplot as plt
import scipy.fftpack as spfft
from pathlib import Path
from os import listdir, walk
from os.path import isfile, join, isdir, abspath
import glob
!pip install sigpy
import sigpy.mri as mri
import random

from tensorflow.keras.callbacks import ModelCheckpoint
from skimage import data, img_as_float
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import peak_signal_noise_ratio as psnr
import argparse
import imutils
import cv2
from math import log10, sqrt

from google.colab import drive
drive.mount('/content/drive')

root='/content/drive/My Drive/UPDATED_DATA3'
images = []
count=1
```

```
for folder in glob.glob(root + '/*'):
  if count==250:
    break
  f=h5py.File(folder,'r')
  data_k=np.array(f[list(f.keys())[0]])
  slices, weight, height=data_k.shape
  print(data_k.shape)
  for i in range(slices):
    data_im=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(data_k[i,:,:]))))
    #normalize im:
    ma = np.max(data_im)
    mi = np.min(data_im)
    data_im = (data_im - mi) / (ma - mi)
    images.append(data_im)
  count=count+1

dat=images[60]
print(dat.shape)
plt.figure()
plt.imshow(abs(dat))
plt.gray()
plt.title('Image Obtained')
print(np.max(dat))
print(np.min(dat))
real_im=spfft.fftshift(spfft.fft2(spfft.ifftshift(dat)))
plt.figure()
plt.imshow(abs(np.log(real_im)+1))
plt.gray()
plt.title('K-Space of Image')

mask = np.zeros((128,128,4))
percentage=np.array([66, 68, 72, 80])
for j in range(4):
  percnt=percentage[j]
  for i in range(percnt):
    mask[i,:,j]=1

f, (ax1, ax2, ax3, ax4)  = plt.subplots(1,4)
f.set_figheight(15)
f.set_figwidth(15)
ax1.imshow(abs(mask[:,:,0]), cmap = 'gray')
ax1.set_title('Masking the 33/64')

ax2.imshow(abs(mask[:,:,1]), cmap = 'gray')
ax2.set_title('Masking the 17/32')

ax3.imshow(abs(mask[:,:,2]), cmap = 'gray')
ax3.set_title('Masking the 9/16')

ax4.imshow(abs(mask[:,:,3]), cmap = 'gray')
ax4.set_title('Masking the 5/8')

random.shuffle(images)
images=np.array(images)
print(images.shape)

train_gt =images[0:7000,:,:,np.newaxis]
test_gt = images[7000:len(images),:,:,np.newaxis]
```

```
masked_train = np.zeros((7000,128,128,1))

for i in range(len(train_gt)):
  perct=np.random.randint(4)
  full_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(train_gt[i,:,:,0])))
  masked_traink=np.multiply(full_k,mask[:,:,perct])
  masked_im=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_traink))))
  #normalize im:
  ma = np.max(masked_im)
  mi = np.min(masked_im)
  masked_im = (masked_im - mi) / (ma - mi)
  masked_train[i,:,:,0]=masked_im

images=[]

print(masked_train.shape)
print(train_gt.shape)
print(test_gt.shape)

f, (ax1, ax2, ax3)  = plt.subplots(1,3)
f.set_figheight(15)
f.set_figwidth(15)

ax1.imshow(masked_train[80,:,:,0], cmap = 'gray')
ax1.set_title('Masked Image')

maskedk=spfft.fftshift(spfft.fft2(spfft.ifftshift(masked_train[80,:,:,0])))
ax2.imshow(np.log(np.abs(masked_traink)+1),cmap = 'gray')
ax2.set_title('K-Space of Masked Image')

ax3.imshow(train_gt[80,:,:,0], cmap = 'gray')
ax3.set_title('Real Image')

model = keras.Sequential(
    [
        #layer 1
        layers.Conv2D(64,(5,5),activation="relu",input_shape=(128,128,1), padding='same'),
        layers.BatchNormalization(),
        #layer 2
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 3
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 4
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #layer 5
        layers.Conv2D(64,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 1:
        layers.Conv2D(32,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 2:
        layers.Conv2D(8,(3,3),activation="relu", padding='same'),
        layers.BatchNormalization(),
        #deconv 3:
        layers.Conv2D(1,(3,3),activation="linear", padding='same')
```

```
    ]
)

model.summary()

def DataCons(reconn,real,mask):
  notmask=np.logical_not(mask).astype(int)
  recon_k=np.zeros((128,128),dtype=complex)
  real_k=np.zeros((128,128),dtype=complex)
  recon_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(reconn[0,:,:,0])))
  real_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(real[0,:,:,0])))

  reconmasked=np.multiply(notmask,recon_k)
  recon_corr=real_k+reconmasked
  image=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(recon_corr))))
  #normalize im:
  ma = np.max(image)
  mi = np.min(image)
  image = (image - mi) / (ma - mi)
  image=image[np.newaxis,:,:,np.newaxis]
  return image

def trainNetwork(model, X, Y):
  filepath="/content/drive/My Drive/PROJECT_RESULTS/IMAGEDOMAIN_partial_best.hdf5"
  checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True
      , mode='min')
  callbacks_list = [checkpoint]
  opt = keras.optimizers.Adam(learning_rate=0.0001)
  model.compile(loss='mean_squared_error', optimizer='adam',metrics=['MeanSquaredError','
      Accuracy'])
  history = model.fit(X, Y, epochs=20, batch_size=100, validation_split = 0.2, verbose=1,
      callbacks=callbacks_list)
  return history

history = trainNetwork(model,masked_train,train_gt)

# plot history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Recorded Model Loss')
plt.ylabel('MSE Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()

def Network(model,mask,inputt):
  recon=inputt
  for i in range(3):
    out=model.predict(recon)
    recon=DataCons(out,inputt,mask)
  return recon

test_sample=test_gt[540,:,:,:]
masked_test=np.zeros((1,128,128,1))
test_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(test_sample[:,:,0])))
mask1=mask[:,:,0]
masked_testk=np.multiply(test_k,mask1)
masked_test[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk))))
```

```
recon_im=Network(model,mask1,masked_test)
ssim_res,psnr_res,err_im= QA(recon_im[0,:,:,0],test_sample[:,:,0])

print('SSIM Achieved: '+ str(ssim_res))
print('PSNR Achieved:'+ str(psnr_res))

f, (ax1, ax2, ax3, ax4)  = plt.subplots(1,4)
f.set_figheight(15)
f.set_figwidth(15)
ax1.imshow(np.abs(masked_test[0,:,:,0]), cmap='gray')
ax1.set_title('Masked Undersampled Image')

ax2.imshow(np.abs(test_sample[:,:,0]), cmap='gray')
ax2.set_title('Ground Truth Image')

ax3.imshow(np.abs(recon_im[0,:,:,0]), cmap='gray')
ax3.set_title('Reconstructed Image')

ax4.imshow(np.abs(err_im), cmap='gray')
ax4.set_title('Error Image')

#WITH MASKING 2:
mask2 = mask[:,:,1]
test_sample2=test_gt[800,:,:,:]
test_k2=spfft.fftshift(spfft.fft2(spfft.ifftshift(test_sample2[:,:,0])))
masked_test2=np.zeros((1,128,128,1))
masked_testk2=np.multiply(test_k2,mask2)
masked_test2[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk2))))

recon_im2=Network(model,mask2,masked_test2)
ssim_res2,psnr_res2,err_im2= QA(recon_im2[0,:,:,0],test_sample2[:,:,0])
print('SSIM Achieved: '+ str(ssim_res2))
print('PSNR Achieved:'+ str(psnr_res2))

f, (ax11, ax22, ax33, ax44)  = plt.subplots(1,4)
f.set_figheight(15)
f.set_figwidth(15)
ax11.imshow(np.abs(masked_test2[0,:,:,0]), cmap='gray')
ax11.set_title('Masked Undersampled Image')

ax22.imshow(np.abs(test_sample2[:,:,0]), cmap='gray')
ax22.set_title('Ground Truth Image')

ax33.imshow(np.abs(recon_im2[0,:,:,0]), cmap='gray')
ax33.set_title('Reconstructed Image')

ax44.imshow(np.abs(err_im2), cmap='gray')
ax44.set_title('Error Image')

#WITH MASKING 3:
mask3 = mask[:,:,2]
test_sample3=test_gt[750,:,:,:]
test_k3=spfft.fftshift(spfft.fft2(spfft.ifftshift(test_sample3[:,:,0])))
masked_test3=np.zeros((1,128,128,1))
masked_testk3=np.multiply(test_k3,mask3)
masked_test3[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk3))))
```

```
recon_im3=Network(model,mask3,masked_test3)
ssim_res3,psnr_res3,err_im3= QA(recon_im3[0,:,:,0],test_sample3[:,:,0])
print('SSIM Achieved: '+ str(ssim_res3))
print('PSNR Achieved:'+ str(psnr_res3))

f, (ax111, ax222, ax333, ax444)  = plt.subplots(1,4)
f.set_figheight(15)
f.set_figwidth(15)
ax111.imshow(np.abs(masked_test3[0,:,:,0]), cmap='gray')
ax111.set_title('Masked Undersampled Image')

ax222.imshow(np.abs(test_sample3[:,:,0]), cmap='gray')
ax222.set_title('Ground Truth Image')

ax333.imshow(np.abs(recon_im3[0,:,:,0]), cmap='gray')
ax333.set_title('Reconstructed Image')

ax444.imshow(np.abs(err_im3), cmap='gray')
ax444.set_title('Error Image')

#WITH MASKING 4:

mask4 = mask[:,:,3]
test_sample4=test_gt[210,:,:,:]
test_k4=spfft.fftshift(spfft.fft2(spfft.ifftshift(test_sample4[:,:,0])))
masked_test4=np.zeros((1,128,128,1))
masked_testk4=np.multiply(test_k4,mask4)
masked_test4[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk4))))

recon_im4=Network(model,mask4,masked_test4)
ssim_res4,psnr_res4,err_im4= QA(recon_im4[0,:,:,0],test_sample4[:,:,0])
print('SSIM Achieved: '+ str(ssim_res4))
print('PSNR Achieved:'+ str(psnr_res4))

f, (ax1111, ax2222, ax3333, ax4444)  = plt.subplots(1,4)
f.set_figheight(15)
f.set_figwidth(15)
ax1111.imshow(np.abs(masked_test4[0,:,:,0]), cmap='gray')
ax1111.set_title('Masked Undersampled Image')

ax2222.imshow(np.abs(test_sample4[:,:,0]), cmap='gray')
ax2222.set_title('Ground Truth Image')

ax3333.imshow(np.abs(recon_im4[0,:,:,0]), cmap='gray')
ax3333.set_title('Reconstructed Image')

ax4444.imshow(np.abs(err_im4), cmap='gray')
ax4444.set_title('Error Image')



#50

def QA(recon, ref):
  recon_im=(recon-np.min(recon))/(np.max(recon)-np.min(recon))
  ref_im=(ref-np.min(ref))/(np.max(ref)-np.min(ref))
  err_im=np.abs(recon_im - ref_im)
```

```
  #SSIM and PSNR
  ssim_res = ssim(recon_im, ref_im, vmin=0, vmax=1)
  psnr_res= psnr(ref_im, recon_im)

  return ssim_res, psnr_res,err_im

np.save('/content/drive/My Drive/PROJECT_RESULTS/partial_test_data.h5', test_gt)
np.save('/content/drive/My Drive/PROJECT_RESULTS/partial_train_data.h5', train_gt)

ssims=np.zeros((len(test_gt),1))
psnrs=np.zeros((len(test_gt),1))
for i in range(len(test_gt)):
  testdata=test_gt[i,:,:,:]
  perct=np.random.randint(4)
  masked_test=np.zeros((1,128,128,1))
  test_k=spfft.fftshift(spfft.fft2(spfft.ifftshift(testdata[:,:,0])))
  mask1=mask[:,:,perct]
  masked_testk=np.multiply(test_k,mask1)
  masked_test[0,:,:,0]=np.abs(spfft.ifftshift(spfft.ifft2(spfft.fftshift(masked_testk))))

  reconn=Network(model,mask1,masked_test)
  ssims[i,0],psnrs[i,0],err_im= QA(reconn[0,:,:,0],testdata[:,:,0])

plt.plot(ssims)
plt.title('Recorded SSIM Values')
plt.ylabel('SSIM in Test Data')
plt.xlabel('Test Sample Index')
plt.show()

plt.plot(psnrs)
plt.title('Recorded PSNR Values')
plt.ylabel('PSNR in Test Data')
plt.xlabel('Test Sample Index')
plt.show()

#MEAN VALUES:
ssim_mean=np.mean(ssims)
print('SSIM mean: '+str(ssim_mean))
print('SSIM STD:'+ str(np.std(ssims)))
psnr_mean=np.mean(psnrs)
print('PSNR mean: '+str(psnr_mean))
print('PSNR STD:'+ str(np.std(psnrs)))
```

CS

```
import pywt
import numpy as np
import h5py
import os
import matplotlib.pyplot as plt
from skimage import metrics
from matplotlib.colors import LogNorm
import sigpy.mri as mri
import cv2
from skimage.transform import resize
```

```
def ifft2c(im):
    return np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(im)))

def fft2c(d):
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(d)))

def normalize(image):
    xmax, xmin = np.abs(image).max(), np.abs(image).min()
    image_n = (np.abs(image) - xmin) / (xmax - xmin)
    return image_n

def wavelet_mix(coeffs):
    cA4, (cH4, cV4, cD4), (cH3, cV3, cD3), (cH2, cV2, cD2), (cH1, cV1, cD1) = coeffs

    trans1 = np.concatenate((cA4, cH4), axis=1)
    trans2 = np.concatenate((cV4, cD4), axis = 1)
    transform1 = np.concatenate((trans1, trans2), axis=0)

    trans1 = np.concatenate((transform1, cH3), axis=1)
    trans2 = np.concatenate((cV3, cD3), axis = 1)
    transform2 = np.concatenate((trans1, trans2), axis=0)

    trans1 = np.concatenate((transform2, cH2), axis=1)
    trans2 = np.concatenate((cV2, cD2), axis = 1)
    transform3 = np.concatenate((trans1, trans2), axis=0)

    trans1 = np.concatenate((transform3, cH1), axis=1)
    trans2 = np.concatenate((cV1, cD1), axis = 1)
    transform4 = np.concatenate((trans1, trans2), axis=0)

    return transform4

def wavelet_sep(transform):
    size = len(transform)
    le4 = int(size / (2**4))
    le3 = int(size / (2**3))
    le2 = int(size / (2**2))
    le1 = int(size / (2))

    cA4 = transform[:le4,:le4]
    cV4 = transform[le4:le4*2,:le4]
    cH4 = transform[:le4,le4:le4*2]
    cD4 = transform[le4:le4*2,le4:le4*2]

    cV3 = transform[le3:le3*2,:le3]
    cH3 = transform[:le3,le3:le3*2]
    cD3 = transform[le3:le3*2,le3:le3*2]

    cV2 = transform[le2:le2*2,:le2]
    cH2 = transform[:le2,le2:le2*2]
    cD2 = transform[le2:le2*2,le2:le2*2]

    cV1 = transform[le1:le1*2,:le1]
    cH1 = transform[:le1,le1:le1*2]
    cD1 = transform[le1:le1*2,le1:le1*2]

    coeffs = cA4, (cH4, cV4, cD4), (cH3, cV3, cD3), (cH2, cV2, cD2), (cH1, cV1, cD1)
    return coeffs
```

```
def regularized_wavelet(imr, beta):

    coeffs = pywt.wavedec2(imr, 'db1', level=4)
    transform = wavelet_mix(coeffs)

    for i in range(transform.shape[0]):
        for j in range(transform.shape[1]):
            transform[i][j] = (transform[i][j] / abs(transform[i][j])) * max(0, abs(
                transform[i][j]) - beta)

    coef = wavelet_sep(transform)
    recon = pywt.waverec2(coef, 'db1')
    return recon

def compressed_sensing(Mu,beta,mask,numiter):
    inv_mask = abs(1 - mask)
    datak = np.zeros(Mu.shape) + 0j;
    datak[:,:] = Mu[:,:];
    for i in range(numiter):
        imr = ifft2c(datak)
        imth = regularized_wavelet(imr, beta);
        datak = fft2c(imth);
        datak = (datak * inv_mask) + Mu;
        imth = ifft2c(datak);
    return imth

# Load test Data
poisson_test = np.load('poisson_test_data.h5.npy', allow_pickle=True)
poisson_test = np.reshape(poisson_test, (1929, 128, 128))
print(poisson_test.shape)


# In[4]:


# generate 2 and 4 acceleratio masks
np.random.seed(7)
mask_test_2 = mri.poisson([128,128], 2, crop_corner=False)
mask_test_4 = mri.poisson([128,128], 4, crop_corner=False)

# Generate Undersampled k-space data with 4 acc
img_pos = 400
kdata = fft2c(poisson_test[img_pos])
Mu_4 = kdata * mask_test_4
imu_4 = ifft2c(Mu_4)

# Do Compressed Sensing Reconstruction
# measure the right beta
coeffs = pywt.wavedec2(imu_4, 'db1', level=4)
transform = wavelet_mix(coeffs)
beta_4 = abs(np.max(transform) * (1/500))

# Start Algorithm
cs_recon_4 = compressed_sensing(Mu_4, beta_4, mask_test_4, 1000)

#Normalize images
undersamp_n = normalize(abs(imu_4))
```

```
cs_n = normalize(abs(cs_recon_4))
ref_n = normalize(abs(poisson_test[img_pos]))

error = np.abs(cs_n) - np.abs(ref_n)

# Show results
f, (ax1, ax2)  = plt.subplots(1,2)
f.set_figheight(10)
f.set_figwidth(10)
ax1.imshow(abs(cs_n), cmap='gray')
ax1.set_title('Compressed Sensing Reconstruction')

ax2.imshow(abs(error), cmap='gray')
ax2.set_title('Error Image for Compressed Sensing Reconstruction')
plt.show()

# Check PSNR / SSIM
print('For Poisson Undersampled(Acc = 4) Data:')
psnr = metrics.peak_signal_noise_ratio(ref_n, cs_n, data_range=1)
print('Compressed Sensing Reconstruction PSNR value: ' + str(psnr))

ssim = metrics.structural_similarity(ref_n, cs_n, data_range=1)
print('Compressed Sensing Reconstruction Sensing SSIM value: ' + str(ssim))

psnr_r_undersamp = metrics.peak_signal_noise_ratio(ref_n, undersamp_n, data_range=1)
print('Zero Fill PSNR value: ' + str(psnr_r_undersamp))

ssim_r_undersamp = metrics.structural_similarity(ref_n, undersamp_n, data_range=1)
print('Zero Fill SSIM value: ' + str(ssim_r_undersamp))

# Generate Undersampled k-space data with 2 acc
img_pos = 110
kdata = fft2c(poisson_test[img_pos])
Mu_2 = kdata * mask_test_2
imu_2 = ifft2c(Mu_2)

# Do Compressed Sensing Reconstruction
# measure the right beta
coeffs = pywt.wavedec2(imu_2, 'db1', level=4)
transform = wavelet_mix(coeffs)
beta_2 = abs(np.max(transform) * (1/500))

# Start Algorithm
cs_recon_2 = compressed_sensing(Mu_2, beta_2, mask_test_2, 1000)

#Normalize images
undersamp_n = normalize(abs(imu_2))
cs_n = normalize(abs(cs_recon_2))
ref_n = normalize(abs(poisson_test[img_pos]))

error = np.abs(cs_n) - np.abs(ref_n)

# Show results
f, (ax1, ax2)  = plt.subplots(1,2)
f.set_figheight(10)
f.set_figwidth(10)
ax1.imshow(abs(cs_n), cmap='gray')
ax1.set_title('Compressed Sensing Reconstruction')
```

```
ax2.imshow(abs(error), cmap='gray')
ax2.set_title('Error Image for Compressed Sensing Reconstruction')
plt.show()

# Check PSNR / SSIM
print('For Poisson Undersampled(Acc = 2) Data:')
psnr = metrics.peak_signal_noise_ratio(ref_n, cs_n, data_range=1)
print('Compressed Sensing Reconstruction PSNR value: ' + str(psnr))

ssim = metrics.structural_similarity(ref_n, cs_n, data_range=1)
print('Compressed Sensing Reconstruction Sensing SSIM value: ' + str(ssim))

psnr_r_undersamp = metrics.peak_signal_noise_ratio(ref_n, undersamp_n, data_range=1)
print('Zero Fill PSNR value: ' + str(psnr_r_undersamp))

ssim_r_undersamp = metrics.structural_similarity(ref_n, undersamp_n, data_range=1)
print('Zero Fill SSIM value: ' + str(ssim_r_undersamp))
```

POCS

```
import numpy as np
import h5py
import os
import matplotlib.pyplot as plt
from skimage import metrics
from matplotlib.colors import LogNorm
from skimage.transform import resize

def ifft2c(im):
    return np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(im)))

def fft2c(d):
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(d)))

def pocs(partial_k, ratio_partial, n_turns, ref_im):
    dim = partial_k.shape[0]

    upper_th = int((1 - ratio_partial) * dim)
    lower_th = int(ratio_partial * dim)

    ms_k = np.zeros((dim,dim)) + 0j
    ms_k[upper_th:lower_th,:] = partial_k[upper_th:lower_th,:]

    proj_data = np.zeros((dim,dim)) + 0j
    ms_im = ifft2c(ms_k)
    p_xy = np.exp(1j * np.angle(ms_im))

    for i in range(n_turns):
        proj_data[:lower_th,:] = partial_k[:lower_th,:]

        proj_xy = ifft2c(proj_data)
        proj_xy = abs(proj_xy) * p_xy

        # take 2D FT and start over:
        proj_data = fft2c(proj_xy)

    proj_data[:lower_th,:] = partial_k[:lower_th,:]
```

```
    final_image = ifft2c(proj_data)

    return proj_data, final_image

def normalize(image):
    xmax, xmin = np.max(image), np.min(image)
    image_n = (np.abs(image) - xmin) / (xmax - xmin)
    return image_n

def QA(recon, ref):
    recon_im=(recon-np.min(recon))/(np.max(recon)-np.min(recon))
    ref_im=(ref-np.min(ref))/(np.max(ref)-np.min(ref))
    err_im=np.abs(recon_im - ref_im)

    #SSIM and PSNR
    ssim_res = metrics.structural_similarity(np.abs(recon_im), ref_im, vmin=0, vmax=1)
    psnr_res= metrics.peak_signal_noise_ratio(ref_im, np.abs(recon_im))

    return ssim_res, psnr_res, err_im

# Load test Data
pocs_test = np.load('partial_test_data.h5.npy', allow_pickle=True)
pocs_test = np.reshape(pocs_test, (1929, 128, 128))
print(pocs_test.shape)

# Generate Partial K-data
img_pos = [540, 800, 750, 210]
ratio_num = [66, 68, 72, 80]
denom = 128
numiter = 20
for i in range(4):
    ratio = ratio_num[i] / denom
    ender = int(ratio * 128)
    kdata = fft2c(pocs_test[img_pos[i]])

    partial_kdata = np.zeros(kdata.shape)
    partial_kdata = 0j + partial_kdata
    partial_kdata[:ender,:] = kdata[:ender,:]

    partial_img = ifft2c(partial_kdata)

    #Perform POCS reconstruction
    pocs_kdata, pocs_recon = pocs(partial_kdata, ratio, numiter, pocs_test[img_pos[i]])

    #Normalize images
    partial_n = normalize(np.abs(partial_img))
    pocs_n = normalize(np.abs(pocs_recon))
    ref_n = normalize(np.abs(pocs_test[img_pos[i]]))

    error = np.abs(pocs_n) - np.abs(ref_n)

    # Display Results
    f, (ax1, ax2)  = plt.subplots(1,2)
    f.set_figheight(10)
    f.set_figwidth(10)
    ax1.imshow(np.abs(pocs_n), cmap='gray')
    ax1.set_title('POCS Reconstruction')
```

```
    ax2.imshow(np.abs(error), cmap='gray')
    ax2.set_title('Error Image for POCS Reconstruction')


    # Check PSNR / SSIM
    print('For {} / 128 Partial K-space Data:'.format(str(ratio_num[i])))
    psnr = metrics.peak_signal_noise_ratio(ref_n, pocs_n, data_range=1)
    print('POCS Reconstruction PSNR value: ' + str(psnr))

    ssim = metrics.structural_similarity(ref_n, pocs_n, data_range=1)
    print('POCS Reconstruction Sensing SSIM value: ' + str(ssim))

    psnr_r_undersamp = metrics.peak_signal_noise_ratio(ref_n, partial_n, data_range=1)
    print('Zero Fill PSNR value: ' + str(psnr_r_undersamp))

    ssim_r_undersamp = metrics.structural_similarity(ref_n, partial_n, data_range=1)
    print('Zero Fill SSIM value: ' + str(ssim_r_undersamp))
    print("")
plt.show()
```