Inception-Based U-Net in Magnetic Resonance Image Reconstruction

by

Elmira Vafay Eslahi

A Report Submitted in Partial Fulfillment of the Requirements for the

Degree of

MASTER OF ENGINEERING

In The Department of Electrical and Computer Engineering

SUPERVISORY COMMITTEE

Inception-Based U-Net in Magnetic Resonance Image Reconstruction

by

Elmira Vafay Eslahi

University of Victoria, 2022

Supervisory Committee

Dr. Amirali Baniasadi, Department of Electrical and Computer Engineering
Supervisor

Dr. Levi Smith, Department of Electrical and Computer Engineering
Departmental Member

Abstract

Magnetic resonance imaging (MRI) is one of the best imaging techniques that produce high-quality images of objects. The long scan time is one of the biggest challenges in MRI acquisitions. To address this challenge, many researchers have aimed at finding methods to speed up the process. Faster MRI can reduce patient discomfort and motion artifacts. One method to speed up MRI scans is skipping some signals in the k-space. Although the incomplete k-space or sub-sampling causes undersampling artifacts due to missing signals, reconstruction techniques can solve the problem by recovering the missing data. Many reconstruction methods are used in this matter, like deep learning-based MRI reconstruction, parallel MRI, and compressive sensing. Among these techniques, the convolutional neural network (CNN) generates high-quality images with faster scan and reconstruction procedures compared to the other techniques. However, CNN architecture has been an area under study in image reconstruction and needs more investigations [1-2].

In this study, we propose a new deep learning algorithm for MRI reconstructions. The Inception module proposed by Google inspires this algorithm [3]. The proposed architecture in this work creates a better-reconstructed image than the standard U-Net. The U-Net architecture consists of encoding and decoding sections, which are considered to decrease complexity. Decreasing the complexity prevents overfitting in the network and, consequently, improves image quality. In other words, we introduce a new MRI U-Net modification by using the Inception module, which is more flexible and robust compared to the standard U-Net. Mean square error (MSE), normalized mean square error (NMSE), structural similarity index measure (SSIM), and peak signal-to-noise ratio (PSNR) are the metrics that we use to evaluate the model. In this study, we improve scan speed by 3.2 times. We show that our method with a new architecture performs better than the standard U-Net to decrease the error and increase the image quality.

Acronyms

MRI	Magnetic Resonance Imaging
CNN	Convolutional neural network
Conv	Convolution
IFFT	Inverse fast Fourier transform
FFT	Fast Fourier transform
LF	Low frequency
MSE	Mean Square Error
NMSE	Normalized Mean Square Error
SSIM	Structural similarity index measure
PSNR	Peak signal-to-noise ratio

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Magnetic Resonance Imaging (MRI) is an imaging modality that takes advantage of the proton density of the hydrogen atoms inside the objects [4]. MRI acquires signals from spinning protons that are caused by a magnetic field [5]. Although damaging ionizing radiation like x-ray is not used in MRI, the duration of MRI scan time is so long that it limits its application [6-7]. The long time of the MRI scan increases patient discomfort, generates motion artifacts, and increases the medical cost [8]. MRI acquires the data and encodes it in a frequency domain called k-space. MRI acquires each line in k-space at the time. As scanning these lines in k-space individually is time-consuming, researchers have made many efforts to skip some signals in k-space to speed up MRI. Skipping the signals results in a serious consequence of Nyquist criterion violation and causes artifacts in reconstructed images [9]. For dealing with these artifacts, some techniques like Compressed Sensing MRI and Parallel MRI showed good results in reconstructing the original image. Parallel MRI works with an array of multiple receiver coils by receiving much less amount of k-space data [1], [7], [10-11]. MRI acceleration includes two parts of subsampling and reconstruction. The goal of MRI acceleration is to find an optimal reconstruction function *f*. The function *f* represents y_s to y, f: $y_s \rightarrow y$, which y_s is the reconstructed image with undersampling artifacts and *y* is the reconstructed image [12]. In Fig. 1, the fully sampled k-space data is shown as x_{full} and *x* is undersampled data in k-space. If *x* is undersampled, *y* will be corrupted images, and if *x* is fully sampled, then *y* will be an image with no artifact. The inverse Fourier transform of x_{full} produces *y*, which is an artifact-free reconstructed MRI image. This figure shows a general strategy and the flow of the undersampled MRI reconstruction problem [12].

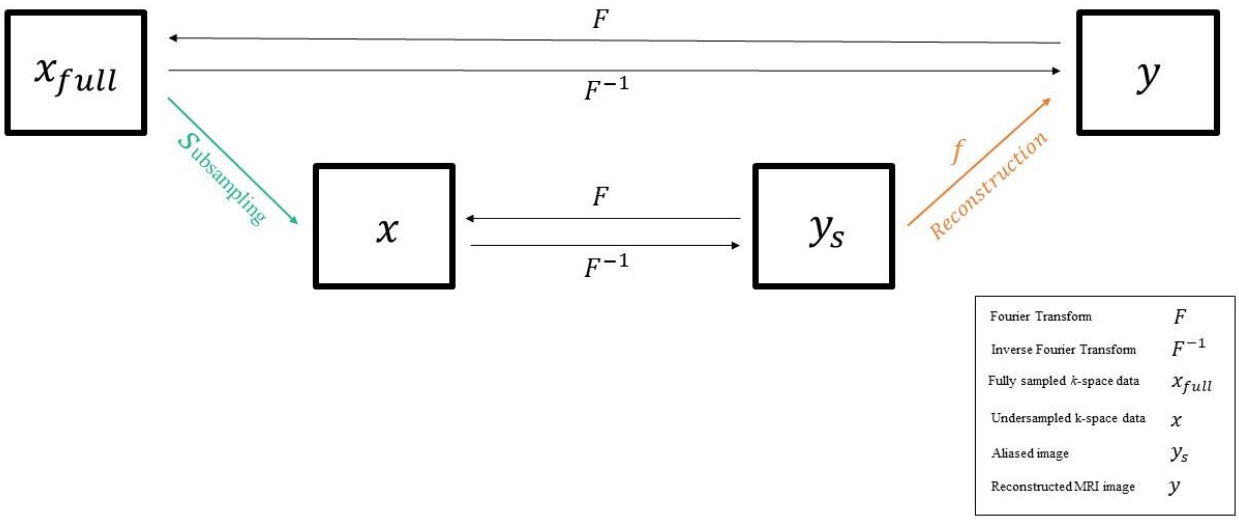


Fig. 1. The strategy of the undersampled MRI reconstruction problem.

In this figure, F represents the Fourier transform and F^{-1} represents the inverse Fourier transform. We can use deep learning models as a reconstruction strategy to remove artifacts from the corrupted images. Undersampled data causes these artifacts in corrupted images. For example, U-Net is one of the convolutional neural networks that is able to reconstruct reference images [12]. On the other hand, U-Net works as the function f with some parameters. The model trains these parameters during the training procedure. This reconstruction function f helps to find a reconstructed image y from the aliased image y_s.

Chapter 2: Background

2.1. Undersampling

In order to accelerate the MRI scan procedure, one of the approaches is undersampling the data [12]. Undersampling reduces the amount of data in k-space and results in information loss. According to the Nyquist criterion, the scan frequency should be twice the expected maximum frequency [9]. However, undersampling does not meet this criterion as for undersampling we skip some of the signals in the k-space. Therefore, it leads to appearing artifacts in images called aliasing artifacts. Recovering the missed

signals can be done in the frequency or image domains. In this study, we focus on the image domain and try to recover reference images from corrupted images caused by undersampling artifacts. In order to obtain images from the frequency domain dataset, we need a transformation technique. As we do the sampling in the Cartesian scheme, we use fast Fourier transformation (FFT) [4].

MRI reconstruction in deep learning-based image reconstruction is a method of reconstructing the original image from undersampled data [4]. There are different methods of reconstructing the images from corrupted images like deep learning-based MRI reconstruction, parallel MRI, and compressive sensing [4], [13]. These image reconstruction methods show that undersampling can speed up the procedure of MRI scans by using a large training dataset [4]. The goal is to minimize the loss function for reconstructing the original image from corrupted images [4].

2.2. U-Net

The architecture of U-Net is for Biomedical Image Segmentation [14]. The U-Net network is a fully Convolutional Neural Network (CNN) that uses kernels. The network trains the weights of the kernels during the training procedure [15]. U-Net is a U-shaped network that includes encoder and decoder sections and uses down-sampling and up-sampling strategies. In the down-sampling operation, we decrease the spatial dimensions and increase the feature channels (the number of filters). However, in the up-sampling procedure, we increase the spatial dimensions, and reduce the number of filters [14]. Fig.2 shows one example of a U-Net architecture.

In this network, there are two 3×3 convolutions in each encoder block. We use an activation function like Rectified Linear Unit (ReLU) after each convolution for better generalization of the training data [16]. The output of each encoder block can get concatenated with the decoder block on the other side of the network [14]. The goal of concatenation or skip connections in U-Net is to prevent information loss and make the output richer. It means that, as the encoding blocks have more information about the

images, the output of the encoding blocks can be concatenated to the input of the decoding blocks with the same size of the images to prevent information loss. Fig. 2 shows the architecture of a U-Net used in MRI reconstructions [12].

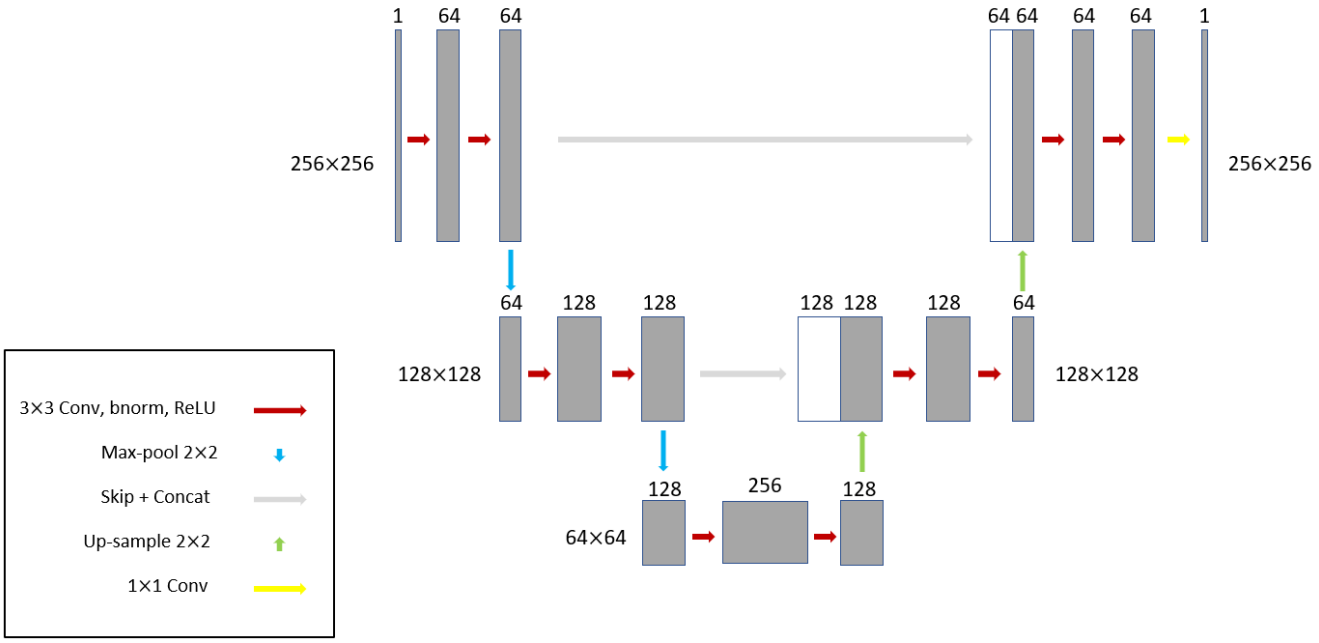


Fig. 2. The architecture of the U-Net. [12].

2.3. Inception Module

In deep learning study, there is always a question of which combination of convolutions can be used to get the best result [17]. Different architectures use convolutions with different kernel sizes to extract features from the images. Each kernel can play a different role in designing the networks. For example, 1×1 convolution means convolution with a kernel size 1×1 is fast and needs less memory but cannot solve complicated problems. On the other hand, 5×5 convolutions involve so many multiplications. They need more parameters but is computationally expensive with regards to the number of parameters it adds to the function. Therefore, using a combination of convolutions and pooling layers in various convolutional blocks can benefit from all the convolutions used in the network [18]. This combination of convolutions and max-pooling layers helps to reduce computation costs with much better memory.

The Inception module was first proposed in GoogLeNet [3]. After that, researchers used this module in different applications, including Alzheimer's disease diagnosis [19]. The idea behind of the Inception module is to make the architecture wider instead of deeper. Deeper architectures tend to overfit, by combining different convolutions in just one layer, the model is able to extract more information from the images.

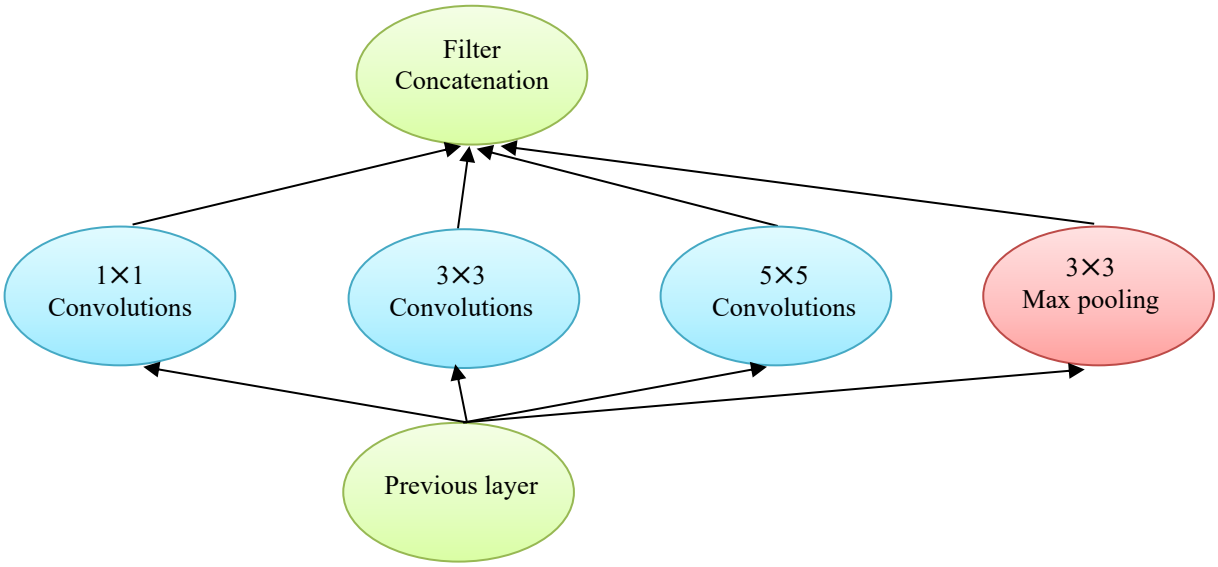


Fig. 3. The Inception Block.

Fig. 3 illustrates one example of Inception block [12]. This block includes:

a) 1×1 convolution.

b) 1×1 convolution comes with 3×3 convolutions.

c) 1x1 convolution comes with 5×5 convolutions

d) 3×3 max-pool layer comes with a 1×1 convolution

Using the combination of the convolutions and pooling layers in a block helps to extract more information from the images, which leads to better reconstructed images [18]. Fig.3 is one of the examples of an Inception block and can be the combination of any of convolution layers or max pooling layers. For example, in this study we combine 3×3 convolutions with 5×5 convolutions.

Chapter 3: Methodology

3.1. Proposed Methodology

In MRI reconstructions, researchers use different deep learning algorithms [4]. U-Net is one of the practical and straightforward techniques in MRI reconstruction as it removes most of the folding artifacts. In this study, we use the Inception module integrated with U-Net. Inception module has been studied in Image classification and object detection, however, to the best of our knowledge, researchers have not studied the application of the Inception module in U-Net for MRI reconstruction.

3.2. Proposed Methodology for MRI Reconstructions

In this study, we use an inspired Inception module integrated with the U-Net model for the architecture. Combining different convolutions in various convolutional blocks helps to reduce computation costs and increase the quality of the undersampled image [18]. Fig. 4 shows the architecture of the proposed model. The architecture shows how we use the Inception module in the U-Net. We combine 5×5 convolutions with 3×3 convolutions into two blocks. Then, the combination of convolution layers can be concatenated or summed.

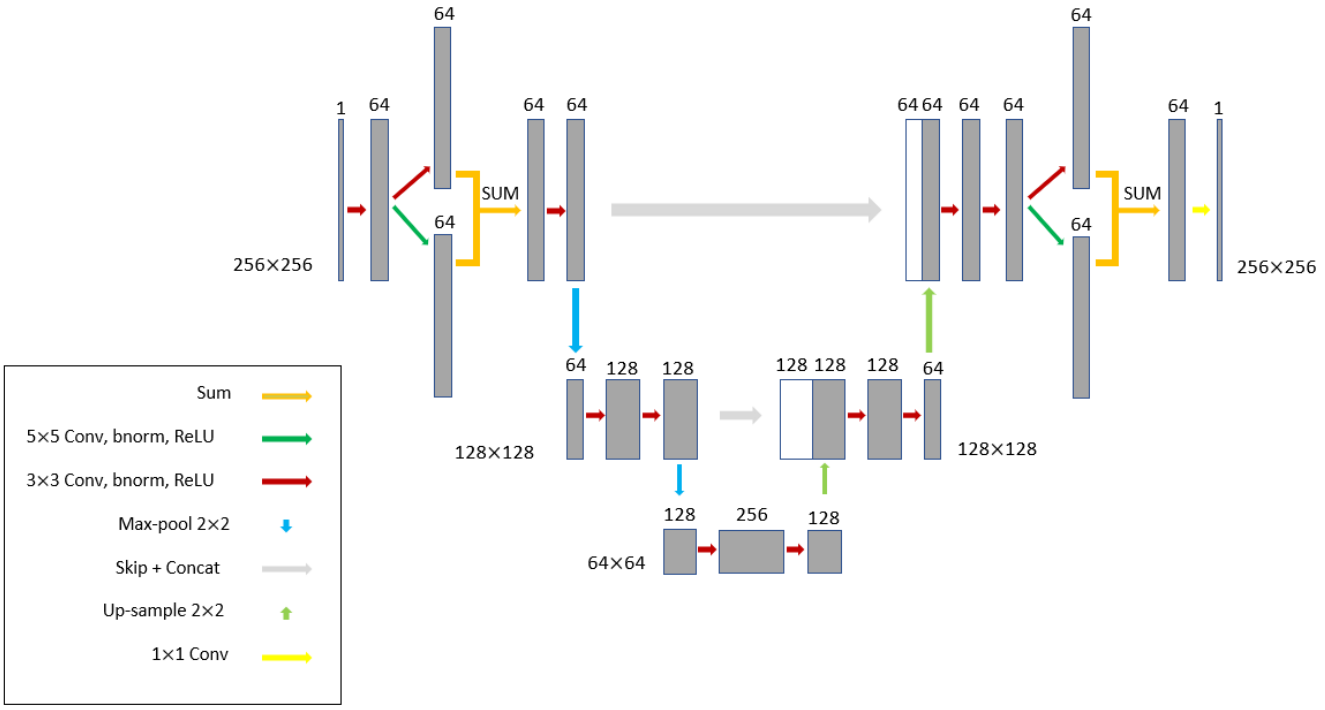


Fig. 4. The architecture of proposed methods.

Fig. 4 shows what the inspired Inception module looks like. Instead of having a simple 3×3 convolution, we have a combination of 3×3 and 5×5 convolutions. The architecture seems complicated, but in the end, the total number of parameters, errors, and image distortion is smaller than the standard U-Net.

3.3. Dataset

The dataset is obtained from the NYU fastMRI database [21-23]. NYU fastMRI investigators provided data but did not participate in the analysis or writing of this report [21-23]. The dataset includes raw k-space data in two types of MRI scans: knee MRIs and brain (neuro) MRIs. The brain dataset comprises 6,970 T1, T2, and FLAIR fully sampled k-space and consists of multi-channel data with the sizes 320, 320, 16, 16 (phase encoding, frequency encoding, slices, channels) [21-23]. The number of each imaging is shown in table 1. In this study, we just use the T2 brain dataset.

Field Strength	1.5T	3T	
T1	382	409	
T1 post contrast	849	646	
T2	1655	2524	
FLAIR	126	411	
Total	3012	3990	7002

Table 1. The fastMRI NYU Brain dataset *(https://fastmri.med.nyu.edu/)*

3.4. Data Preparation

This study uses the T2 brain single channel dataset to implement our methods. The T2 dataset includes 2250 fully sampled k-spaces. We use IFFT to produce the images from fully sampled data and generate the target images or the ground truth. On the encoding section of the architecture, we need corrupted images for the inputs. So, we do undersampling on the fully sampled data to reduce the amount of data for generating the corrupted images. For undersampling, there are different techniques to store data in the K-space like Cartesian, Radial and Spiral. We use Cartesian sub-sampling which fills the K-space row by row. The center signals have the highest energy and the most important information of the image. Therefore, we add the high-energy signals or namely low-frequency signals, in the center of the k-space to preserve the information on the location of the small anomalies [12]. Fig. 5 show the differences between low frequency signal and high frequency signals in K-space.

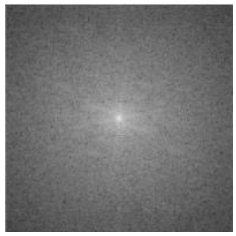
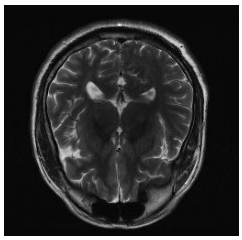
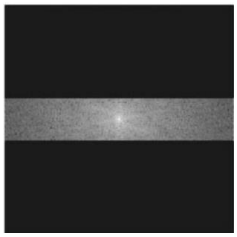
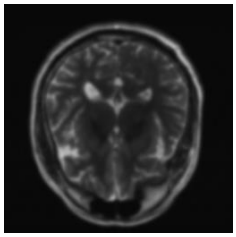
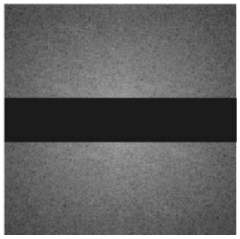
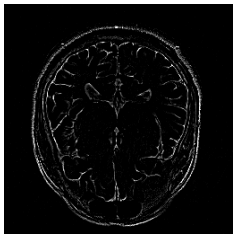
Original image in K-space	Original image in image domain
	
Center of K-Space	Low Freq Only
	
Periphery of K-Space	High Freq Only
	

Fig. 5. Low Frequency and High Frequency Components in K-space

Fig. 5 shows the differences between low frequency components and high frequency components in the k-space. Low frequency components are located in the center of k-space and high frequency components are located in the peripheries of k-space. The center of k-space has information about contrast, brightness, and general shapes, while peripheries have information about edges and details. So, the center of k-space has more energy and information about images.

As the images are large with the size of 320×320, in order to reduce the computation time and cost, the magnitude images are cropped to 256 by 256. We do sub-sampling in the phase encoding direction by 4 as the goal is to speed up the scan by 4. Therefore, we generate 64×256 images from 256×256 images. In the k-space matrix, there are two directions which are called phase and frequency directions. The phase-encoding direction is along the y-axis in k-space and the frequency-encoding direction is along the x-axis in K-space. Once we add 8% and the other time, 20% of the acquired lines in the phase encoding direction to the center as the low frequency. We need these lines in the phase encoding direction to recognize the location of the missed small object due to the aliasing effect. The data will be zero-filled after undersampling to increase the dimensions. Then we use IFFT to produce the corrupted images with an undersampling artifact [4]. The obtained aliased images will work as the inputs, and the targets are the images constructed from IFFT of the full k-space [21-23], [4].

3.5. Implementation & Evaluation

3.5.1. Implementation:

In this study, we use Python 3.8 for the programming and implement CNN with TensorFlow 2.4 [4], [24]. The long run time is one of the challenges in this work which per run took around 3 days. We used a personal computer with Processor Intel Core i7-4790 CPU 3.60 GHz, 16GB Ram.

After taking the Fast Fourier Transform (FFT), we apply undersampling by 4 with a uniform undersampling pattern [4]. In the center of the k-space, 8% of the phase encoding signals that are equal to the low-frequency signals will remain during the undersampling. The decoding section is the mirror of the encoding section, but instead of max-pooling, we have upsampling layers. The upsampling layers

add feature maps, which help the U-Net network to have the same size in both the encoder and decoder path [25].

We double the number of layers in the encoder to add the decoder section to the architecture to complete the network. To encode the input, we combine the CNN kernels or filters with the input elements.

The kernel size is odd because we can have a central pixel in the input image and can decode the other pixels. Also, to reduce the dimensionality of the network, we use max-pooling for encoding the inputs. The reason for max-pooling is to prevent overfitting the model and to reduce the computational cost. We use a max filter in max-pooling. Max filter means the maximum number of each patch of the feature map.

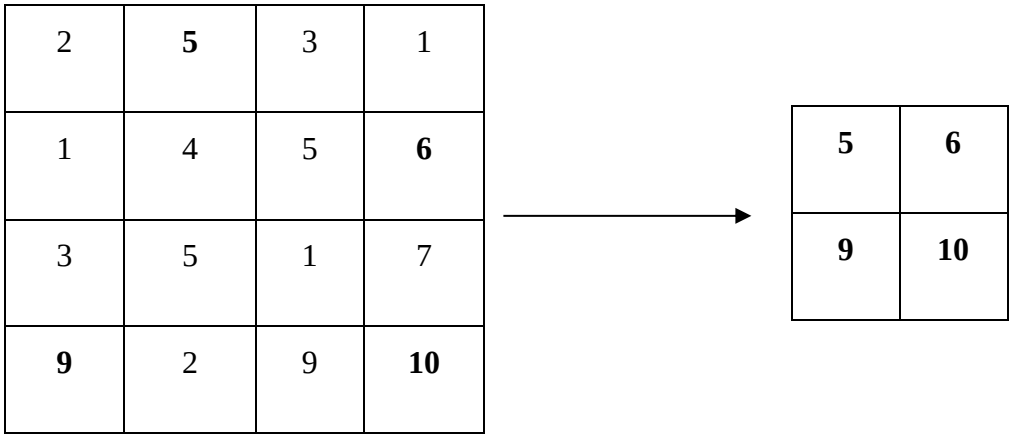


Fig. 6. Max pooling procedure.

Fig. 5 is an example of the max-pooling procedure and shows how we can distribute the max-pooling layers in the network. By adding the max-pooling with stride (2,2), the size of the image will be half in both height and width [4].

Also, in the implementation procedure, we skip layers with concatenation in the network. The skipped layer can be summation or concatenation. Concatenation means feeding a layer's output to another layer's input. When using a skip connection, we connect the encoder layers to the corresponding layer in the upsampling path. We also have batch normalization before we pass it to the activation function. Batch normalization makes our network more stable, and our model can learn faster due to scaling the values down to between -1 and 1.

First, we speed up the model by 2.5 times and then by 3.2 times, with 1000 epochs. Early stopping with patience 20 is considered as part of our code to stop the run process at the best epoch.

3.5.2. Evaluation:

We evaluate the model with 4 metrics. The metrics are mean square error (MSE), normalized mean square error (NMSE), structural similarity index (SSIM), and peak signal-to-noise ratio (PSNR). Eq.1 shows the MSE formula [4].

$$MSE = \frac{1}{MN} ||\hat{y}_{mn} - y_{mn}||_2^2 \quad \text{Eq. 1.}$$

y_{mn}, is the ground truth and $\hat{y}_{mn}$ is the reconstructed image which *m* and *n* are pixel locations, with size *M*×*N.* MSE of an image is the pixel-wised error estimation and derived from y_{mn} [4]. Eq. 2 shows the NMSE formula [4].

$$NMSE(y_{mn}, \hat{y}_{mn}) = \frac{MSE(y_{mn}, \hat{y}_{mn})}{MSE(y_{mn}, 0)} \quad \text{Eq. 2.}$$

MSE (y_{mn}, $\hat{y}_{mn}$) is the MSE of the reference and the reconstructed image, and MSE (y_{mn},0) is the MSE of the reference and zero. The MSE and NMSE define the error, and the lower the MSE and NMSE, the higher the image quality.

The other metric which shows the similarities of the reference and the reconstructed images, is SSIM, and measures image similarities from brightness, structure, and contrast. Equation below shows the formula of SSIM [4].

$$SSIM(x,y) = \frac{(2\mu_x\mu_y+C_1)(2\sigma_{xy}+C_2 2)}{(\mu_x^2\mu_y^2+C_1)(\sigma_x^2\sigma_y^2+C_2)} \quad \text{Eq. 3.}$$

In this equation, μ_x and μ_y show the average in *x*, and average in *y* respectively. σ_x^2 and σ_y^2 show the variance of *x* and the variance of *y*, and σ_{xy} shows the covariance of *x* and *y. C1* and *C2* are the stabilize variables and are constants [4].

Another metric is PSNR which measures the quality of the image. The higher the value of PSNR the higher the image quality. Eq. 4 shows the formula of PSNR [4].

$$PSNR = 10\ log\ \frac{{x_{max}}^2}{MSE} \quad \text{Eq. 4.}$$

In this formula, x_{max} is the maximum possible pixel value in the image and Eq. 1 shows the MSE formula.

Chapter 4: Experimental Results

Table 2 shows the comparison between the standard U-Net network with the proposed method at the low frequency of 8% and 20% by 4 different metrics.

	Model/Metrics	**MSE (1e-3)**	**NMSE (1e-2)**	**SSIM (1e-1)**	**PSNR**
LF 8/100	**U-Net**	1.5	4.05	8.85	28.6
	Proposed model	1.4	3.78	8.90	28.9
LF 20/100	**U-Net**	0.69	1.80	9.430	32.25
	Proposed model	0.68	1.75	9.432	32.25

Table 2. Quantitative comparison.

In MSE and NMSE, the lower the value shows, the lower the error and, consequently, the higher the image quality. The proposed model decreases the value of MSE and NMSE in both low frequencies. Also, SSIM measures image similarities in brightness, structure, and contrast. The higher the value, the higher the similarities. The comparison clearly shows that we increase the value of SSIM in the proposed method. The last metric in the table is PSNR which measures the quality of the image. The higher the value of PSNR, the higher the image quality. The value of PSNR is the same in LF of 20%, but by decreasing the low frequency, which leads to the higher speed, the value of PSNR increases. Overall, by looking at the quantitative comparison table, we can easily conclude that the proposed model decreases the error and increases the image quality compared to the standard U-Net.

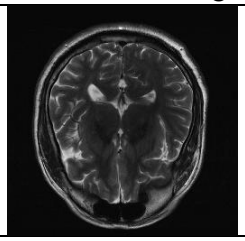
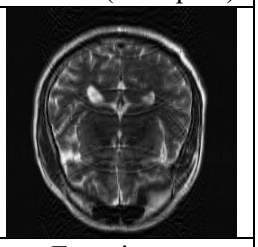
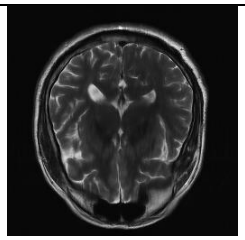
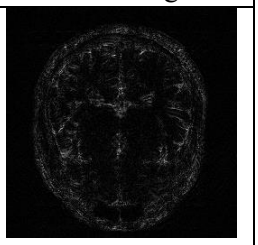
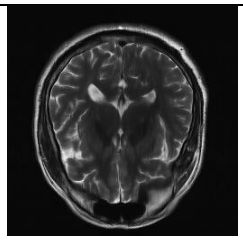
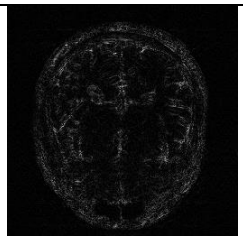
	Ground-truth image	Aliased (corrupted)
		
	Reconstructed	Error image
U-Net model LF 20/100		
Proposed model LF 20/100		

Fig. 7. The comparison in LF 20%.

Fig. 6 shows the comparison between the U-Net model and the proposed model at the low frequency of 20%. We compare the Ground truth image, the reconstructed image, and the error image in these two networks.

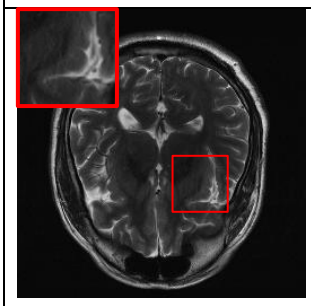
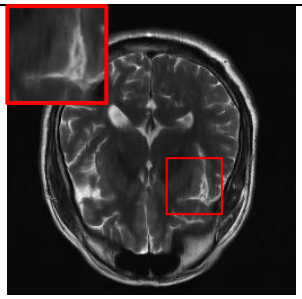
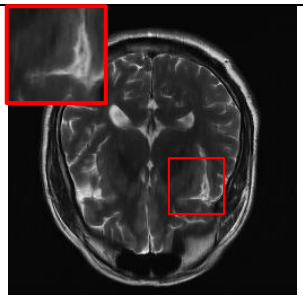
Ground-truth image	U-Net model-LF 20/100 NMSE: 1.80e-2	Proposed model-LF 20/100 NMSE: 1.75e-2
		
a	b	c

Fig. 8. Reconstruction results in LF 20%.

Fig. 7 shows the reconstruction results by (a) Ground truth (original), (b) U-Net and (c) proposed model with the low frequency (LF) of 20%. Red boxes show the enlarged view of the images. The NMSE shows the normalized mean square error value in both models. The lower the value, the lower the error. The enlarged images and the value of NMSE show that the similarity is higher, and the error is less in the proposed model compared to the U-Net.

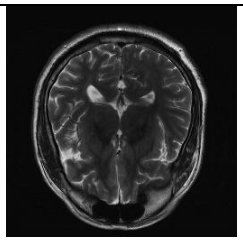
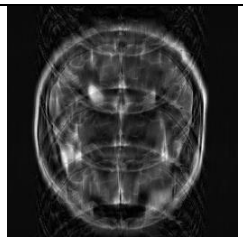
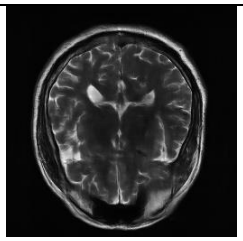
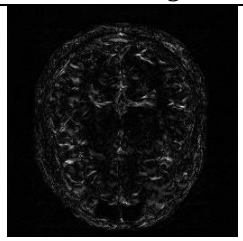
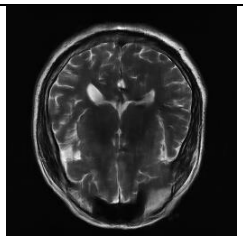
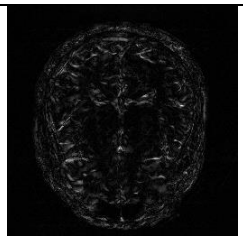
	Ground-truth image	Aliased (corrupted)
		
	Reconstructed	Error image
U-Net model LF 8/100		
Proposed model LF 8/100		

Fig. 9. The comparison in LF 8%.

Fig.8 shows the comparison between the Ground truth image (original), the reconstructed image and the error image. This comparison is between the U-Net model and the proposed model at the low frequency of 8%.

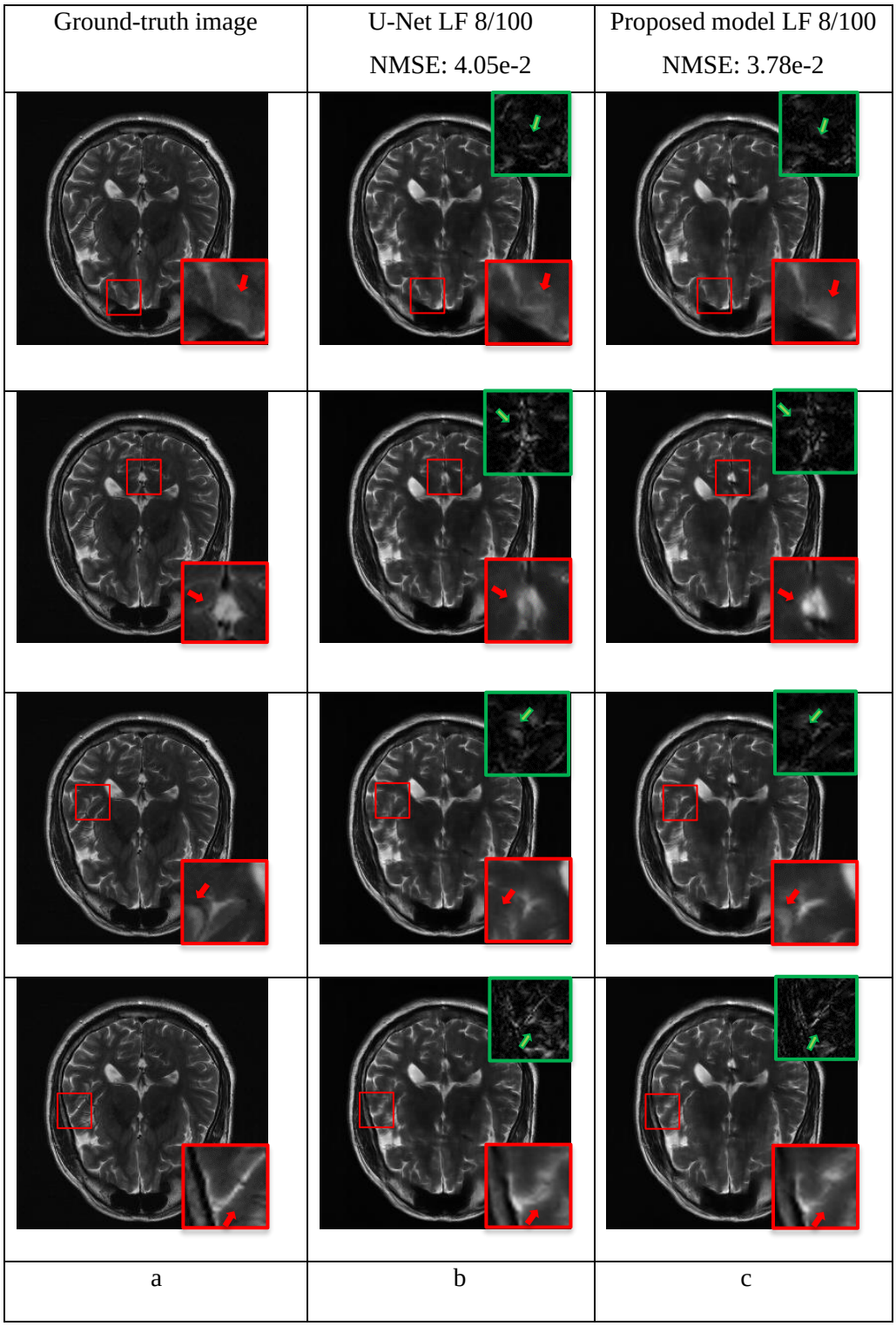


Fig. 10. Reconstruction results in LF 8%.

Fig. 9 shows the reconstruction results by (a) Ground truth (original), (b) U-Net, and (c) proposed model at various views of reconstruction. We compare these results in a low frequency (LF) of 8%. Red boxes

show the enlarged view of the images, and the green boxes illustrate the enlarged view of the error images. The enlarged images indicate a higher similarity and less error in the proposed model than the U-Net.

In this study, we also compare the training and validation loss curves. Fig.10. shows the comparison of learning curves in both frequencies. The vertical axis shows training and validation loss, and the horizontal axis illustrates the number of epochs. In U-Net, the validation loss is higher than the training loss in higher iterations. We conclude that the model is slightly leaning toward overfitting, especially at higher speeds. Overfitting means that the model learns the noise and details to the extent that it impacts the model's performance on new data. In contrast, the proposed model reduces the validation loss in higher iterations compared to U-Net. The reason for the better performance in the proposed model is that by combining 5×5 convolutions with 3×3 convolutions, the model involves more multiplications. These multiplications help to extract more information from the images and result in less validation loss in the new architecture.

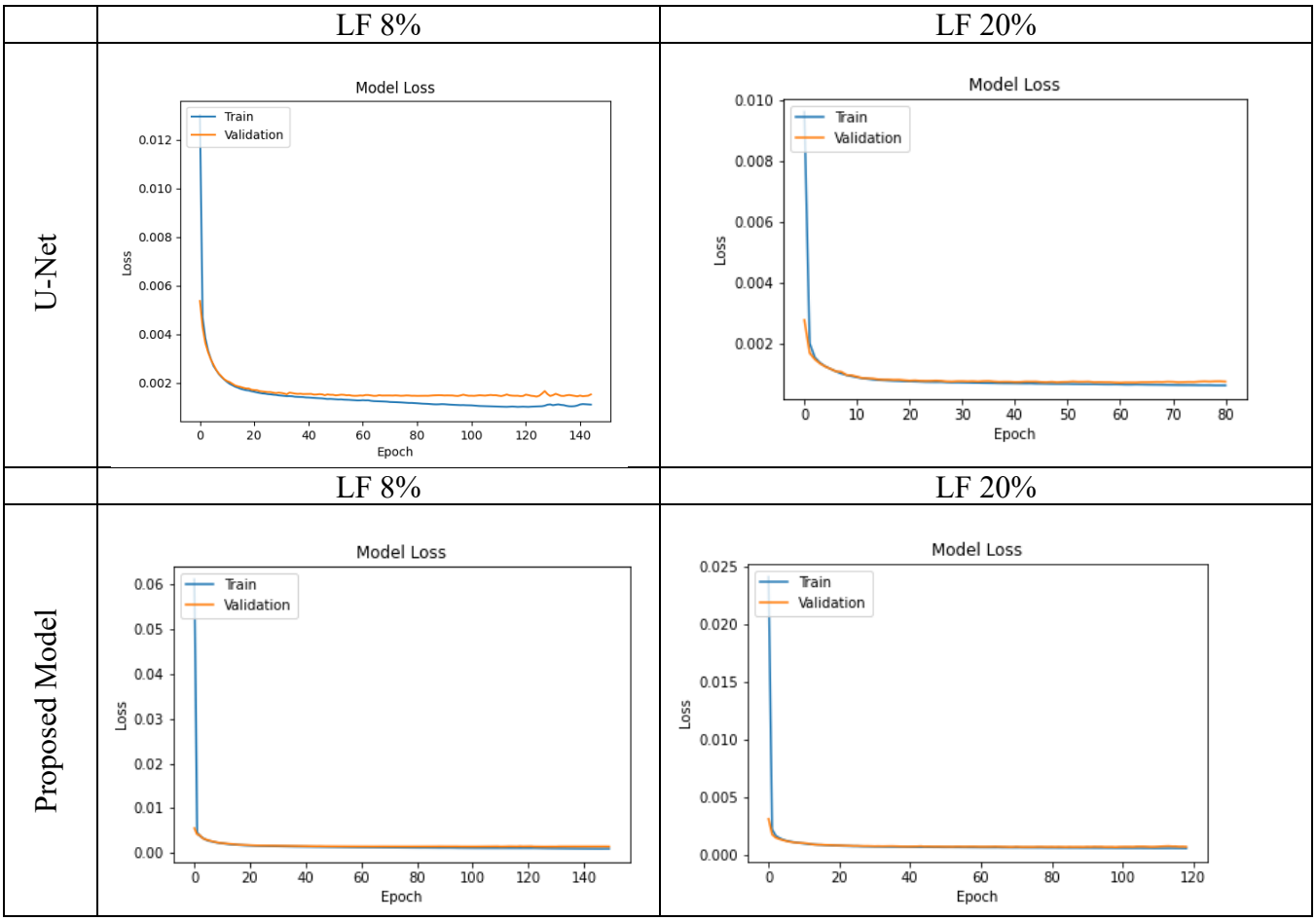


Fig. 11. Learning curves comparison.

Chapter 5: Conclusion

One of the biggest challenges in MRI reconstruction is choosing an optimal architecture. There are many deep learning methods to reconstruct the corrupted images [8], [13]. The different size of the input images, different number of training datasets, and the computer hardware specifications can make this choice challenging [12]. In this study, we implemented the U-Net model and the Inception U-Net model using deep learning methods in MRI reconstruction and then compared these two models. The model's architecture differs in different datasets as it has a different kernel size and a different number of outputs [4]. So, when the dataset type and dimensions change, the model should be updated accordingly. The

proposed model is trained for brain T2 images with 2200 datasets; however, it works for any brain dataset with the same data size.

We integrated the architecture of the standard U-Net with an inspired Inception module proposed by Google [3]. As deeper models have the probability of overfitting, we made the architecture wider instead of deeper by adding 5×5 convolutions to the blocks. The new architecture with different kernel sizes prevents overfitting than U-Net in the same speed of 3.2 times as the model can extract more information from the images. In this study, the value of MSE and NMSE are decreased by 6.7%, the value of SSIM is decreased by 0.6% and the value of PSNR is decreased by 1.05% compared to U-Net. In conclusion, according to the results and analytical quantification, we showed that the proposed method eliminates more folding artifacts compared to the widely used state-of-the-art U-Net. Consequently, removing more artifacts leads to better-visualized images and lower reconstruction errors.

Acknowledgement

I would like to express my special gratitude to my supervisor Dr. Amirali Baniasadi for his guidance and support in completing my project, and I offer my sincere appreciation for the learning opportunities he provided.

I would also like to extend my gratitude to my sister Dr. Samira Vafay Eslahi, for her inspiration in the selection process of the project topic and her guidance and support throughout the project.

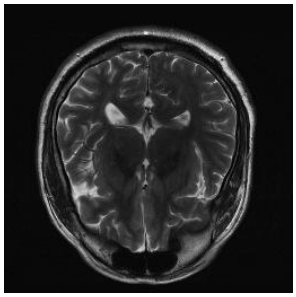
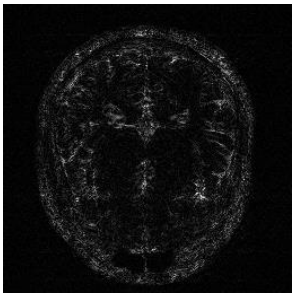
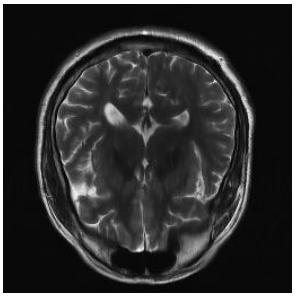
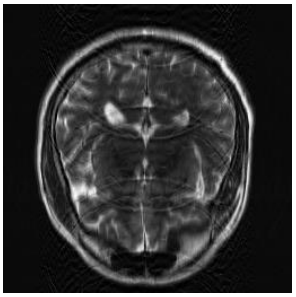
Appendix A. U-Net Network LF 20%

MSE of the Best Model and Reference: 0.0006914402183601 45 +/- 0.0004115333431301 8855
NMSE of the Best Model and Reference: 0.017906741572268053 +/- 0.006678765106087649
SSIM of the Best Model and Reference: 0.943156506369748 +/- 0.014432305725706682
PSNR of the Best Model and Reference: 32.257816269166796 +/- 2.336328521780247
Model: "U-Net"

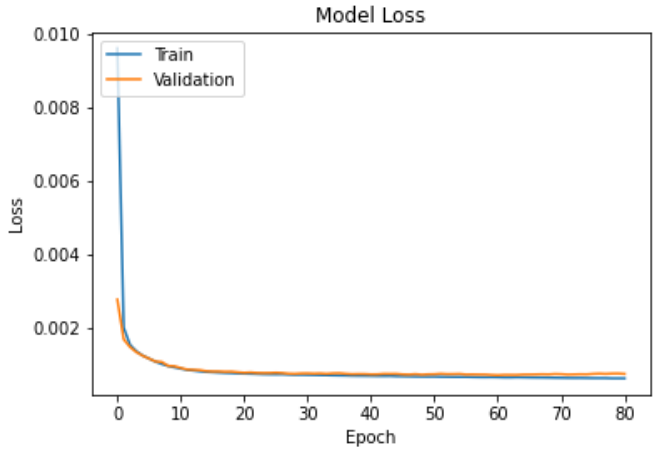
_____
_____
_____
Layer (type) Output Shape Param # Connected to
==========
==========
=======
input_1 (InputLayer) [(None, 256, 256, 1 0 []
)]
conv2d (Conv2D) (None, 256, 256, 64 640 ['input_1[0][0]']
)
conv2d_1 (Conv2D) (None, 256, 256, 64 36928 ['conv2d[0][0]']
)
max_pooling2d (MaxPooling2D) (None, 128, 128, 64 0 ['conv2d_1[0][0]']
)
9
conv2d_2 (Conv2D) (None, 128, 128, 12 73856 ['max_pooling2d[0][0]']
8)
conv2d_3 (Conv2D) (None, 128, 128, 12 147584 ['conv2d_2[0][0]']
8)
max_pooling2d_1 (MaxPooling2D) (None, 64, 64, 128) 0 ['conv2d_3[0][0]']
conv2d_4 (Conv2D) (None, 64, 64, 256) 295168 ['max_pooling2d_1[0][0]']
conv2d_5 (Conv2D) (None, 64, 64, 128) 295040 ['conv2d_4[0][0]']
up_sampling2d (UpSampling2D) (None, 128, 128, 12 0 ['conv2d_5[0][0]']
8)
concatenate (Concatenate) (None, 128, 128, 25 0 ['conv2d_3[0][0]',
6) 'up_sampling2d[0][0]']
conv2d_6 (Conv2D) (None, 128, 128, 12 295040 ['concatenate[0][0]']
8)
conv2d_7 (Conv2D) (None, 128, 128, 64 73792 ['conv2d_6[0][0]']
)
up_sampling2d_1 (UpSampling2D) (None, 256, 256, 64 0 ['conv2d_7[0][0]']
)
concatenate_1 (Concatenate) (None, 256, 256, 12 0 ['conv2d_1[0][0]',
8) 'up_sampling2d_1[0][0]']
conv2d_8 (Conv2D) (None, 256, 256, 64 73792 ['concatenate_1[0][0]']
)
conv2d_9 (Conv2D) (None, 256, 256, 64 36928 ['conv2d_8[0][0]']
)

conv2d_10 (Conv2D) (None, 256, 256, 1) 65 ['conv2d_9[0][0]']
==========
==========
=======
Total params: 1,328,833
Trainable params: 1,328,833
Non-trainable params: 0
_____

Images:

Ground truth	Error	Reconstructed	Aliased
			

Learning Curve:



Appendix B. U-Net Network LF 8%

MSE of the Best Model and Reference: 0.0014898736257789853 +/- 0.0020788849676932845
NMSE of the Best Model and Reference: 0.04047889486564833 +/- 0.038599621343708326
SSIM of the Best Model and Reference: 0.8864673644665217 +/- 0.037527210565490354
PSNR of the Best Model and Reference: 28.59774678039657 +/- 4.449971265340075
Model: "U-Net"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 1)	0	
conv2d (Conv2D)	(None, 256, 256, 64)	640	input_1[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 64)	36928	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 128	73856	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 128	147584	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 256)	295168	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 128)	295040	conv2d_4[0][0]
up_sampling2d (UpSampling2D)	(None, 128, 128, 128	0	conv2d_5[0][0]
concatenate (Concatenate)	(None, 128, 128, 256	0	conv2d_3[0][0] up_sampling2d[0][0]

conv2d_6 (Conv2D) (None, 128, 128, 128 295040 concatenate[0][0]

conv2d_7 (Conv2D) (None, 128, 128, 64) 73792 conv2d_6[0][0]

up_sampling2d_1 (UpSampling2D) (None, 256, 256, 64) 0 conv2d_7[0][0]

concatenate_1 (Concatenate) (None, 256, 256, 128 0 conv2d_1[0][0]
up_sampling2d_1[0][0]

conv2d_8 (Conv2D) (None, 256, 256, 64) 73792 concatenate_1[0][0]

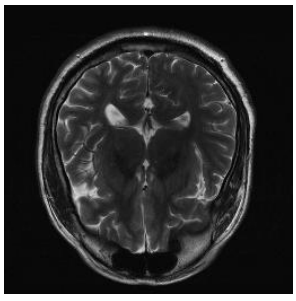
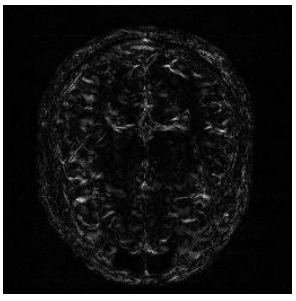
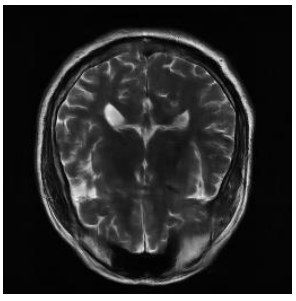
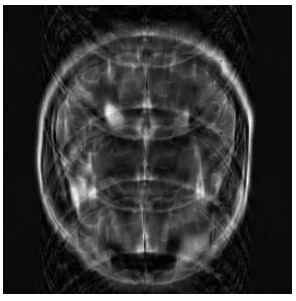
conv2d_9 (Conv2D) (None, 256, 256, 64) 36928 conv2d_8[0][0]

conv2d_10 (Conv2D) (None, 256, 256, 1) 65 conv2d_9[0][0]

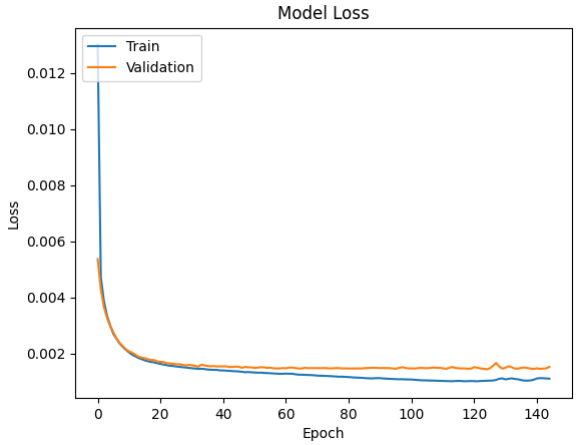
==========

Total params: 1,328,833
Trainable params: 1,328,833
Non-trainable params: 0

Images:

Ground truth	Error	Reconstructed	Aliased
			

Learning Curve:



Appendix C. Proposed Network LF 20%

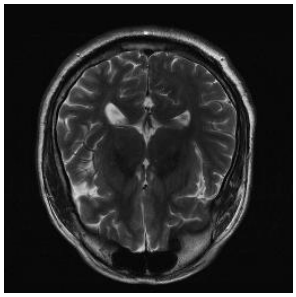
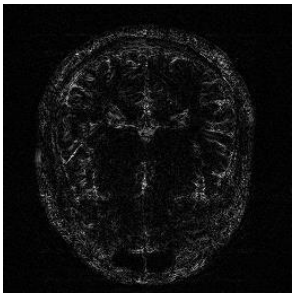
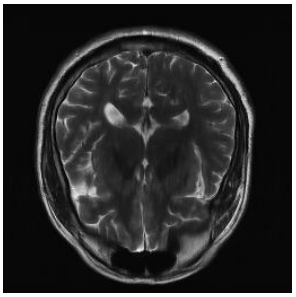
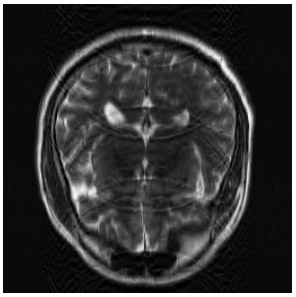
MSE of the Best Model and Reference: 0.0006813238540971933 +/- 0.0003785764493505038
NMSE of the Best Model and Reference: 0.017773853730724903 +/- 0.006094107992586631
SSIM of the Best Model and Reference: 0.9431945701315723 +/- 0.014291313684075714
PSNR of the Best Model and Reference: 32.248736314894174 +/- 2.208159860666277
Model: "U-Net"

_____
_____
_____
Layer (type) Output Shape Param # Connected to
==========
==========
=======
input_1 (InputLayer) [(None, 256, 256, 1 0 []
)]
9
conv2d (Conv2D) (None, 256, 256, 64 640 ['input_1[0][0]']
)
conv2d_1 (Conv2D) (None, 256, 256, 64 1664 ['input_1[0][0]']
)
lambda (Lambda) (None, 256, 256, 64 0 ['conv2d[0][0]',
) 'conv2d_1[0][0]']
conv2d_2 (Conv2D) (None, 256, 256, 64 36928 ['lambda[0][0]']
)
max_pooling2d (MaxPooling2D) (None, 128, 128, 64 0 ['conv2d_2[0][0]']
)
conv2d_3 (Conv2D) (None, 128, 128, 12 73856 ['max_pooling2d[0][0]']

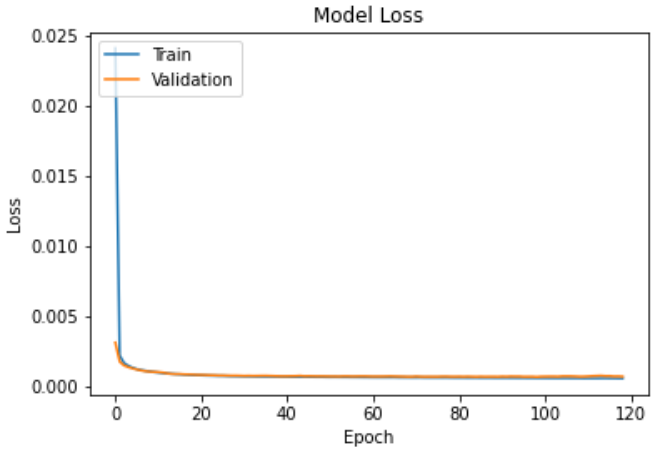
```
8)
conv2d_4 (Conv2D) (None, 128, 128, 12 147584 ['conv2d_3[0][0]']
8)
max_pooling2d_1 (MaxPooling2D) (None, 64, 64, 128) 0 ['conv2d_4[0][0]']
conv2d_5 (Conv2D) (None, 64, 64, 256) 295168 ['max_pooling2d_1[0][0]']
conv2d_6 (Conv2D) (None, 64, 64, 128) 295040 ['conv2d_5[0][0]']
up_sampling2d (UpSampling2D) (None, 128, 128, 12 0 ['conv2d_6[0][0]']
8)
concatenate (Concatenate) (None, 128, 128, 25 0 ['conv2d_4[0][0]',
6) 'up_sampling2d[0][0]']
conv2d_7 (Conv2D) (None, 128, 128, 12 295040 ['concatenate[0][0]']
8)
conv2d_8 (Conv2D) (None, 128, 128, 64 73792 ['conv2d_7[0][0]']
)
up_sampling2d_1 (UpSampling2D) (None, 256, 256, 64 0 ['conv2d_8[0][0]']
)
concatenate_1 (Concatenate) (None, 256, 256, 12 0 ['conv2d_2[0][0]',
8) 'up_sampling2d_1[0][0]']
conv2d_9 (Conv2D) (None, 256, 256, 64 73792 ['concatenate_1[0][0]']
)
conv2d_10 (Conv2D) (None, 256, 256, 64 36928 ['conv2d_9[0][0]']
)
conv2d_12 (Conv2D) (None, 256, 256, 1) 65 ['conv2d_10[0][0]']
==================================================================================
==================
======
Total params: 1,330,497
Trainable params: 1,330,497
Non-trainable params: 0
__________________________________________________________________________________
_________
10

_____
None
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\ops
\nn_ops.py:5214: tensor_shape_from_node_def_name (from
tensorflow.python.framework.graph_util_impl) is deprecated and will be removed in a future
version.
Instructions for updating:
Use `tf.compat.v1.graph_util.tensor_shape_from_node_def_name`
FLOPS: 43.8 G
Load_time (S): 104.83322811126709
Reconstruction_time (S): 70.46426272392273
Elapsed time (S): 152406.56579256058
2022-07-05 10:50:50.296124: W tensorflow/stream_executor/platform/default/dso_loader.cc:64]
Could not load dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll not found
```

2022-07-05 10:50:50.298076: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)
2022-07-05 10:50:50.329865: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: DESKTOP-U9B8QS1
2022-07-05 10:50:50.329984: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: DESKTOP-U9B8QS1
2022-07-05 10:50:50.378789: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-07-05 10:51:09.102623: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.
2022-07-05 10:51:09.971770: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.
2022-07-05 10:51:16.407591: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.
2022-07-05 10:51:18.733050: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.
2022-07-05 10:51:31.932536: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 16777216 exceeds 10% of free system memory.
2022-07-07 05:09:21.934173: I tensorflow/core/grappler/devices.cc:66] Number of eligible GPUs (core count >= 8, compute capability >= 0.0): 0
2022-07-07 05:09:21.936255: I tensorflow/core/grappler/clusters/single_machine.cc:358] Starting new session
2022-07-07 05:09:21.966640: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:1164] Optimization results for grappler item: graph_to_optimize
function_optimizer: function_optimizer did nothing. time = 0.002ms.
function_optimizer: function_optimizer did nothing. time = 0.001ms.

Images:

Ground truth	Error	Reconstructed	Aliased
			

Learning Curve:



Appendix D. Proposed Network LF 8%

MSE of the Best Model and Reference: 0.0014144318814627074 +/- 0.0006216548416073484
NMSE of the Best Model and Reference: 0.037880745893396565 +/- 0.010379807724381438
SSIM of the Best Model and Reference: 0.8896085502624047 +/- 0.022926424225715402
PSNR of the Best Model and Reference: 28.87151327548195 +/- 1.802804519482574
Model: "U-Net"

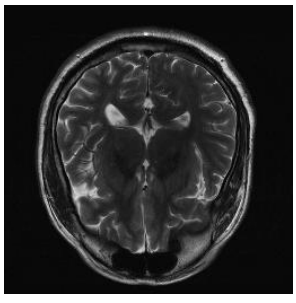
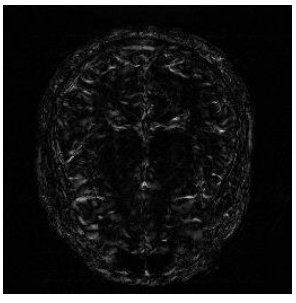
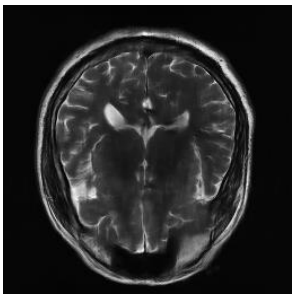
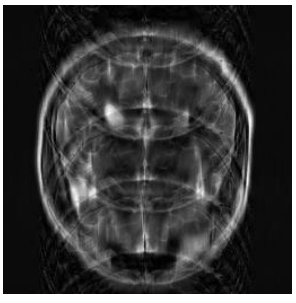
_____
_____
_____
Layer (type) Output Shape Param # Connected to
==========
==========
======
input_1 (InputLayer) [(None, 256, 256, 1 0 []
)]
9
conv2d (Conv2D) (None, 256, 256, 64 640 ['input_1[0][0]']
)
conv2d_1 (Conv2D) (None, 256, 256, 64 1664 ['input_1[0][0]']
)
lambda (Lambda) (None, 256, 256, 64 0 ['conv2d[0][0]',
) 'conv2d_1[0][0]']
conv2d_2 (Conv2D) (None, 256, 256, 64 36928 ['lambda[0][0]']
)
max_pooling2d (MaxPooling2D) (None, 128, 128, 64 0 ['conv2d_2[0][0]']
)
conv2d_3 (Conv2D) (None, 128, 128, 12 73856 ['max_pooling2d[0][0]']
8)
conv2d_4 (Conv2D) (None, 128, 128, 12 147584 ['conv2d_3[0][0]']

8)
max_pooling2d_1 (MaxPooling2D) (None, 64, 64, 128) 0 ['conv2d_4[0][0]']
conv2d_5 (Conv2D) (None, 64, 64, 256) 295168 ['max_pooling2d_1[0][0]']
conv2d_6 (Conv2D) (None, 64, 64, 128) 295040 ['conv2d_5[0][0]']
up_sampling2d (UpSampling2D) (None, 128, 128, 12 0 ['conv2d_6[0][0]']
8)
concatenate (Concatenate) (None, 128, 128, 25 0 ['conv2d_4[0][0]',
6) 'up_sampling2d[0][0]']
conv2d_7 (Conv2D) (None, 128, 128, 12 295040 ['concatenate[0][0]']
8)
conv2d_8 (Conv2D) (None, 128, 128, 64 73792 ['conv2d_7[0][0]']
)
up_sampling2d_1 (UpSampling2D) (None, 256, 256, 64 0 ['conv2d_8[0][0]']
)
concatenate_1 (Concatenate) (None, 256, 256, 12 0 ['conv2d_2[0][0]',
8) 'up_sampling2d_1[0][0]']
conv2d_9 (Conv2D) (None, 256, 256, 64 73792 ['concatenate_1[0][0]']
)
conv2d_10 (Conv2D) (None, 256, 256, 64 36928 ['conv2d_9[0][0]']
)
conv2d_12 (Conv2D) (None, 256, 256, 1) 65 ['conv2d_10[0][0]']
==========
==========
======
Total params: 1,330,497
Trainable params: 1,330,497
Non-trainable params: 0
_____
_____
10

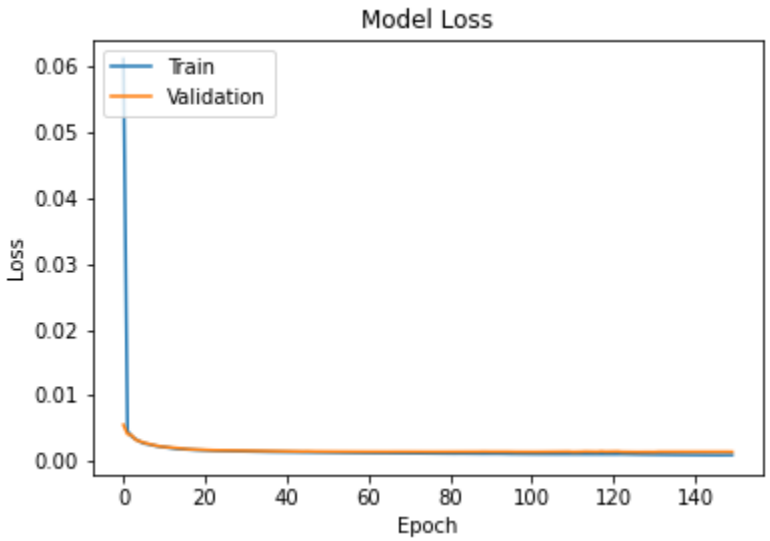
_____
None
2022-07-08 00:46:45.308296: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll not found
2022-07-08 00:46:45.310445: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)
2022-07-08 00:46:45.335153: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: DESKTOP-U9B8QS1
2022-07-08 00:46:45.335317: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: DESKTOP-U9B8QS1
2022-07-08 00:46:45.373631: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-07-08 00:46:50.567739: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.

2022-07-08 00:46:52.293610: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.
2022-07-08 00:46:55.713834: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.
2022-07-08 00:46:55.864008: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 786432000 exceeds 10% of free system memory.
2022-07-08 00:47:09.606382: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 536870912 exceeds 10% of free system memory.
2022-07-10 04:00:01.922725: I tensorflow/core/grappler/devices.cc:66] Number of eligible GPUs (core count >= 8, compute capability >= 0.0): 0
2022-07-10 04:00:01.925816: I tensorflow/core/grappler/clusters/single_machine.cc:358] Starting new session
2022-07-10 04:00:01.954435: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:1164] Optimization results for grappler item: graph_to_optimize
function_optimizer: function_optimizer did nothing. time = 0.001ms.
function_optimizer: function_optimizer did nothing. time = 0ms.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\ops\nn_ops.py:5214: tensor_shape_from_node_def_name (from tensorflow.python.framework.graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.tensor_shape_from_node_def_name`
FLOPS: 43.8 G
Load_time (S): 72.07578301429749
Reconstruction_time (S): 62.011056900024414
Elapsed time (S): 184464.88881516457

Images:

Ground truth	Error	Reconstructed	Aliased
			

Learning Curve:



References

[1] D. L. Donoho, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[2] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html.

[3] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[4] S. V. Eslahi, J. Tao, and J. Ji, "ERNAS: An Evolutionary Neural Architecture Search for Magnetic Resonance Image Reconstructions," *arXiv preprint arXiv:2206.07280*, 2022.

[5] P. C. Lauterbur, "Image formation by induced local interactions: examples employing nuclear magnetic resonance," *nature*, vol. 242, no. 5394, pp. 190–191, 1973.

[6] R. W. Brown, Y.-C. N. Cheng, E. M. Haacke, M. R. Thompson, and R. Venkatesan, *Magnetic resonance imaging: physical principles and sequence design*. John Wiley & Sons, 2014.

[7] D. K. Sodickson and W. J. Manning, "Simultaneous acquisition of spatial harmonics (SMASH): fast imaging with radiofrequency coil arrays," *Magn Reson Med*, vol. 38, no. 4, pp. 591–603, 1997.

[8] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 58, no. 6, pp. 1182–1195, 2007.

[9] H. Nyquist, "Certain topics in telegraph transmission theory," *Transactions of the American Institute of Electrical Engineers*, vol. 47, no. 2, pp. 617–644, 1928.

[10] D. J. Larkman and R. G. Nunes, "Parallel magnetic resonance imaging," *Physics in Medicine & Biology*, vol. 52, no. 7, p. R15, 2007.

[11] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, and P. Boesiger, "SENSE: sensitivity encoding for fast MRI," *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 42, no. 5, pp. 952–962, 1999.

[12] 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.

[13] M. A. Griswold *et al.*, "Generalized autocalibrating partially parallel acquisitions (GRAPPA)," *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 47, no. 6, pp. 1202–1210, 2002.

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, 2015, pp. 234–241.

[15] Y. Han and J. C. Ye, "Framing U-Net via deep convolutional framelets: Application to sparse-view CT," *IEEE Trans Med Imaging*, vol. 37, no. 6, pp. 1418–1429, 2018.

[16] J. K. Seo and E. J. Woo, *Nonlinear inverse problems in imaging*. John Wiley & Sons, 2012.

[17] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning (Cambridge, MA 2016), 2–8; Y," *LeCun, Y. Bengio and G. Hinton,"Deep Learning*, vol. 251, 2015.

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[19] A. Shahina and N. Khan, "Detection of Alzheimer's Disease on Brain MRI using Inception V3 Network".

[20] A. Kumthekar and G. R. Reddy, “An integrated deep learning framework of U-Net and inception module for cloud detection of remote sensing images,” *Arabian Journal of Geosciences*, vol. 14, no. 18, pp. 1–13, 2021.
[21] J. Zbontar *et al.*, “fastMRI: An open dataset and benchmarks for accelerated MRI,” *arXiv preprint arXiv:1811.08839*, 2018.
[22] F. Knoll *et al.*, “fastMRI: A publicly available raw k-space and DICOM dataset of knee images for accelerated MR image reconstruction using machine learning,” *Radiol Artif Intell*, vol. 2, no. 1, 2020.
[23] M. J. Muckley *et al.*, “Results of the 2020 fastmri challenge for machine learning mr image reconstruction,” *IEEE Trans Med Imaging*, vol. 40, no. 9, pp. 2306–2317, 2021.
[24] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems.” 2015.
[25] K. P. Pruessmann, M. Weiger, M. B. Scheidegger, and P. Boesiger, “SENSE: sensitivity encoding for fast MRI,” *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 42, no. 5, pp. 952–962, 1999.