Comparative MRI Reconstruction using DC-WCNN with SGWT with Enhanced Loss Functions and Uncertainty Estimation

Group 4(Akshat Taparia, Kishan Prajapati, Tanishka Pradhan)
210110014,21d070048,210040159

***Abstract*—Our work aims to enhance MRI reconstruction from undersampled data by comparing advanced wavelet-based and graph-based techniques.We aim to integrate DC-WCNN with SGWT instead of DWT. In particular, the SGWT is introduced to sparsely represent magnetic resonance images in iterative image reconstructions by extending the traditional wavelet transform to signals defined on the vertices of a weighted graph, thus operating in the spectral graph domain. This integration allows for accelerated data acquisition, which significantly reduces MRI scan time. The study evaluates the capacity of each method to capture fine anatomical structures with improved metrics such as PSNR and SSIM. Additionally, a novel uncertainty estimation module is incorporated to map model confidence across different reconstruction methods. Experiments on the Kirby21 dataset demonstrate a considerable reduction in scanning time while achieving high-quality reconstruction, with SGWT and DC-WCNN showing superior performance in retaining structural detail.**

I. Introduction

There is a great surge in recent years in demand for accelerated MRI reconstruction as conventional MRI procedures take so much time to scan, which in turn makes this technique much more necessary because MRI imaging is a non-invasive technique, so it requires high-resolution imaging of soft tissues without causing any type of ionizing radiation for a patient, and is thus greatly useful in complex medical condition diagnosis. However, the scanning time with MRI is quite long as the amount of data that needs to fill k-space is extremely large. It takes every scan point-by-point through k-space, which takes a long time and is painful for the patient. Therefore, the efficiency of MRI is rather low in a clinical setup.

The thought behind establishing CS-MRI was for compressing information in the k-space; hence it means that substantially lesser amounts of data needs to be acquired to get the desired for processing, and that outcome in reducing data acquisition time. CS-MRI mainly relies on the fact that most MRI images are sparse, that is, at most places the image is composed of nearly insignificant detail while a small part makes up most detailed information. These methods acquire only a part of the data in k-space and reconstruct the whole picture using advanced mathematical algorithms; yet, undersampling introduces artifact aliasing, which would degrade the quality of a reconstructed image. The key idea behind CS-MRI is to enforce sparsity, which focuses on picking up the essential data, and incoherent sampling, which samples points somewhat randomly rather than in predictable patterns, thereby reducing aliasing artifacts and improving image reconstruction quality [1] [3].

Recently, the methods of deep learning have gained significant promise to improve MRI reconstruction further. Other conventional machine learning methods include dictionary learning and total variation regularization. These methods are indeed very effective but sometimes are very computationally intensive and often fail to capture the fine details of an image. Especially in the case of MRI reconstruction, Convolutional Neural Networks, particularly U-Net, have gained very widespread applications because they work exceptionally well in learning end-to-end mappings between undersampled images and fully sampled images. The effectiveness of U-Net encoder-decoder architecture with skip connections for representing features hierarchically from medical images is quite very useful. However, down-sampling layers in the U-Net architecture lose valuable high-frequency information since it results in lossy images and incomplete reconstructions of fine structures [1] [2].

To address these vulnerabilities, researchers have incorporated wavelet transforms into deep structures of neural networks. Thus, wavelet-based CNNs have been proposed with structures like the Deep Cascade Wavelet-Based Convolutional Neural Network (DC-WCNN), which is built by replacing the pooling layer of the U-Net with Discrete Wavelet Transform (DWT) and Inverse Wavelet Transform (IWT) layers. Wavelet transforms enable the representation of the image at multiple scales. These could allow the network to perceive fine details along with more general features while losing nothing. DC-WCNN is a multiplicity of wavelet-based layers cascaded in such a way that each step successively refines the reconstruction. DC-WCNN, without making use of conventional pooling and unpooling layers, retains spatial and frequency information at each layer. Due to this reason, its reconstruction quality in terms of clear fine details, noise, and artifacts is improved relative to other CNN-based reconstruction methods [1].

SGWT is a novel method that is considered very promising for MRI reconstruction. Graph-based data extends wavelet transformation. The approach of SGWT applies wavelets in the spectral graph domain such that wavelet analysis may be carried out on data represented by graphs rather than

regular grids. This technique is particularly suited to MRI, as it leverages connectivity information encoded within weighted graph edges instead of relying strictly on spatial information. This allows a sparse representation in the spectral domain and is helpful in representing complicated geometric structures in MRI data. This technique results in superior reconstruction quality, particularly with highly undersampled acquisitions as it better captures fine details and minimizes aliasing artifacts than the standard wavelet approaches [3].

Our current work continues the trend to assess and compare these techniques under one roof. We propose improved loss functions where fine anatomical details are focused upon and data consistency, as well as uncertainty estimation that will help to recover each image with a particular degree of reliability, leading to better interpretation of results and the quality of those features that are critical. This would be necessary for the reconstructed images to be diagnostic clinically and it gives an insight into some areas of the image which may need further scrutiny because of uncertainty in reconstruction.

This comparative analysis through our research will identify the best technique for accelerated MRI reconstruction across a range of sampling rates and noise conditions. Quantifying the trade-offs in terms of reconstruction accuracy, artifact suppression, and computational efficiency, the study will contribute to improvement in clinical MRI practices through solutions that minimize scan time while preserving the high diagnostic quality necessary for patient care.

II. Background and Related Work

A. MRI Reconstruction and Challenges

MRI data undersampling reduces the acquisition time but poses challenges requiring robust methods in order to reconstruct images correctly from partial data. Deep learning models such as U-Net(our baseline) and DC-WCNN, and spectral graph methods (SGWT) combined with classical transforms (Curvelets and Shearlets), improve sparse image representation.

B. Deep Learning Methods

The baseline architecture of U-Net is reliable for image reconstruction in tasks involving image-to-image but sometimes fails to hold fine structures. The improvement in details retention is introduced by replacing the pooling layers with discrete wavelet transforms in DC-WCNN, which reduce the loss of information involved in the process of reconstruction. [1]

C. Graph Fourier Transform (GFT)

The core of GFT is fundamental in spectral graph theory and has been used in the work as a tool to transform graph-based data into the spectral domain, which is very handy for processing MRI data that have complex structures. The GFT helps represent MR images as signals on weighted graphs, so efficient reconstruction becomes possible by exploiting the connectivity between data points in the spatial domain. [3]

III. Mathematical Formulations for MRI Reconstruction

A. Compressed Sensing in MRI

Compressed sensing (CS) accelerates MRI by undersampling in k-space, effectively reducing acquisition time. The undersampling process can be modeled as:

$$\mathbf{y} = \mathcal{U}\mathcal{F}\mathbf{x} + \epsilon \tag{1}$$

where $\mathbf{x} \in \mathbb{C}^n$ represents the MRI image as a vector, $\mathcal{F}$ is the discrete Fourier transform, $\mathcal{U}$ is the undersampling matrix, $\mathbf{y} \in \mathbb{C}^m$ is the sampled k-space data (with $m < n$), and ϵ is the noise vector. The objective is to reconstruct $\mathbf{x}$ from limited measurements

$\mathbf{y}$. [2].

B. CS-MRI Optimization Problem

To reconstruct $\mathbf{x}$ from undersampled data $\mathbf{y}$, we solve a regularized optimization problem:

$$\min_{\mathbf{x}} \lambda\|\Psi\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{y} - \mathcal{U}\mathcal{F}\mathbf{x}\|_2^2 \tag{2}$$

where Ψ is a sparsifying transform (e.g., wavelets or spectral graph wavelets), $\|\cdot\|_1$ enforces sparsity, $\|\cdot\|_2$ enforces fidelity to measurements, and λ is a regularization parameter. Solving this problem with iterative thresholding, such as FISTA, achieves faster convergence and reduced memory usage. We can see that the first and the second term in the equation are L1 and L2 regularization respectively, the reason for choosing them is that they both are convex functions hence minimization problem can be solved efficiently in a polynomial time by convex optimization methods. [2].

C. DC-WCNN

Let $x \in \mathbb{C}^N$ be the target image we want to reconstruct from undersampled k-space measurements, where $y \in \mathbb{C}^M$, with $M \ll N$. These measurements are related by $y = F_u x$, where F_u denotes the undersampling Fourier operator. The zero-filled reconstruction provides an estimate, but reconstruction of x from y is challenging with the sub-Nyquist sampling. We present a solution in an optimization form using WCNN:

$$\arg\min_{x} \|x - f_{\text{wcnn}}(x_u)\|_2^2 + \|F_u x - y\|_2^2 \tag{3}$$

Here, $x_{\text{wcnn}} = f_{\text{wcnn}}(x_u)$, where f_{wcnn} represents the forward mapping of WCNN parameterized by its weights. This parameter, λ (regularization factor) , depends on the noise level in y. The strategy aims to reconstruct x from WCNN within the image domain without prior knowledge about k-space data.

To match the output of WCNN with the k-space data acquired, we add a data fidelity (DF) operation in k-space after every WCNN unit. The f_{df} operation is given by:

$$x_{\text{rec}} = x_{\text{wcnn}}(k) + \frac{x_{\text{wcnn}}(k) + x_u(k)}{1 + \lambda} \tag{4}$$

Here, $x_{\text{wcnn}} = F_f x_{\text{wcnn}}$ and $x_u = F_f x_u$ are the Fourier-transformed WCNN output and undersampled data, respectively. The set Ω contains indices of the known k-space data,

F_f is the Fourier encoding matrix, and x_{rec} represents the corrected k-space data. The final reconstructed image x_{rec} is obtained by applying the inverse Fourier transform:

$$x_{\text{rec}} = F_f^H x_{\text{rec}} \tag{5}$$

Our proposed DC-WCNN architecture consists of a series of N_c cascaded WCNN and DF units, as:

$$x_n = \text{WCNN}_n(x_{\text{df},n-1}) + x_{\text{df},n-1} \tag{6}$$

$$x_{\text{df},n} = \text{DF}_n(x_n) \tag{7}$$

where WCNN_n and DF_n denote the n-th WCNN block and DF unit, respectively, for $n = 1, 2, \ldots, N_c$, with $x_{\text{df},0} = x_u$ and $x_{\text{rec}} = x_{\text{df},N_c}$ as the output of the last DF unit.

In each WCNN unit, the 2D Discrete Wavelet Transform (DWT) and inverse wavelet transform (IWT) layers are given by:

$$(X_1, X_2, \ldots, X_K) = \text{DWT}(X_{\text{in}}) \tag{8}$$

$$X = \text{IWT}(X_1, X_2, \ldots, X_K) \tag{9}$$

where X_{in} denotes the input image or an intermediate feature map, K is the number of subbands (example, $K = 4$ for approximation, horizontal, vertical, and diagonal subbands), X_k for $k = 1, 2, \ldots, K$ are the subband coefficients, and X is the output of wavelet recomposition. [1]

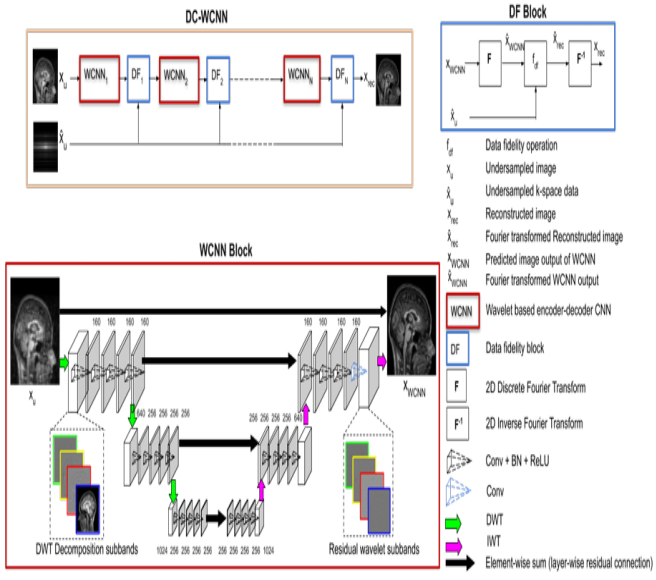


Fig. 1: DC-WCNN Architecture [1]

D. Graph Fourier Transform (GFT)

The GFT of a function f defined on the vertices of a weighted graph G uses the eigenvalues and eigenvectors of the graph Laplacian $\mathcal{L}$. The graph Laplacian $\mathcal{L}$ for a graph $G = (V, E)$ with vertices V and edges E is given by:

$$\mathcal{L} = D - A \tag{10}$$

where D is the degree matrix and A is the adjacency matrix of G. For a function $f \in \mathbb{R}^N$ on the vertices of G, the GFT is defined as:

$$\hat{f}(l) = \langle f, \chi_l \rangle = \sum_{n=1}^{N} f(n)\chi_l(n) \tag{11}$$

where χ_l are the eigenvectors of $\mathcal{L}$ with corresponding eigenvalues λ_l, arranged as $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$. The inverse GFT allows for reconstruction in the spatial domain:

$$f(n) = \sum_{l=0}^{N-1} \hat{f}(l)\chi_l(n) \tag{12}$$

Inverse Graph Transform is given by:

$$f(n) = \sum_{l=0}^{N-1} \hat{f}(l)\chi_l(n) \tag{13}$$

Due to the fact that GFT represents signals as graph harmonics, GFT boasts very high suitability for MRI data undersampling representation because spatial relationships represented by edges of the graphs are most important to recover high-quality images from these limited measurements, and with their leverages, an error that GFT can make because of undersampling artifacts comes down. [3]

E. Spectral Graph Wavelet Transform (SGWT)

The spectral graph wavelet transform represents MRI data sparsely using graph structures, extending wavelet theory to graph domains by employing graph Laplacian eigenfunctions. For a function f on graph vertices, the SGWT wavelet coefficient at scale s and position n is:

$$W_f(s, n) = \langle \psi_{s,n}, f \rangle = \sum_{l=0}^{N-1} \mathcal{G}(s\lambda_l)\hat{f}(l)\chi_l(n) \tag{14}$$

where λ_l and χ_l are eigenvalues and eigenvectors of the graph Laplacian $\mathcal{L}$, and $\mathcal{G}$ is a kernel function acting as a band-pass filter in the spectral domain. The SGWT allows for adaptive sparse representation of the MR image. [3]

F. Fast Spectral Graph Wavelet Transform (SGWT)

Fast Spectral Graph Wavelet Transform is a computation scheme that effectively applies the Spectral Graph Wavelet Transform on a large dataset or a large graph with a huge number of vertices. Traditional wavelet transforms are extended into graphs in order to provide sparse representations for MRI data and thus save the computation speed. [3]

The SGWT is built upon the GFT by defining a wavelet kernel $\mathcal{G}$ that acts as a band-pass filter. This kernel is applied

to the eigenvalues of the graph Laplacian, allowing for multi-scale analysis on graph-based data. For a wavelet operator $\mathcal{T}_\mathcal{G}$ at scale t, the SGWT of a function f on graph vertices n is:

$$\mathcal{T}_\mathcal{G} f(n) = \sum_{l=0}^{N-1} \mathcal{G}(t\lambda_l)\hat{f}(l)\chi_l(n) \tag{15}$$

where $\mathcal{G}(t\lambda_l)$ modulates each Fourier mode, controlled by the scale parameter t.

a) Fast Computation using Chebyshev Polynomial Approximation: To make SGWT computationally feasible for large MRI datasets, Chebyshev polynomials are used to approximate the kernel $\mathcal{G}$. [3] The Chebyshev approximation reduces the complexity of evaluating $\mathcal{G}(t\lambda)$, allowing the SGWT to be computed without explicitly performing the eigendecomposition of $\mathcal{L}$. Given a polynomial of degree K, the Chebyshev approximation for $\mathcal{G}$ is:

$$\mathcal{G}(t\lambda) \approx \sum_{k=0}^{K} c_k T_k \left(\frac{2\lambda}{\lambda_{\max}} - 1 \right) \tag{16}$$

where T_k are the Chebyshev polynomials and c_k are the expansion coefficients. This approach ensures that the SGWT can be computed efficiently, even for high-resolution MRI images. [3]

G. Reconstruction Quality Metrics

For quantitative assessment of reconstruction, we use Root Mean Square Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), Relative L2 Norm Error (RLNE), and Structural Similarity Index (SSIM):

a) Root Mean Square Error (RMSE)::

$$\text{RMSE} = \sqrt{\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2}{M \times N}} \tag{17}$$

where $\mathbf{x}$ is the ground truth image, $\hat{\mathbf{x}}$ the reconstructed image, and $M \times N$ the image dimension.

b) Peak Signal-to-Noise Ratio (PSNR)::

$$\text{PSNR} = 20 \log_{10} \left(\frac{\text{MAX}}{\sqrt{\text{MSE}}} \right) \tag{18}$$

where MAX is the maximum pixel value. Higher PSNR indicates better reconstruction quality.

c) Relative L2 Norm Error (RLNE)::

$$\text{RLNE} = \frac{\|\hat{\mathbf{x}} - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} \tag{19}$$

Smaller RLNE values indicate more accurate reconstructions.

d) Structural Similarity Index (SSIM)::

$$\text{SSIM}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{(2\mu_x\mu_{\hat{x}} + c_1)(2\sigma_{x\hat{x}} + c_2)}{(\mu_x^2 + \mu_{\hat{x}}^2 + c_1)(\sigma_x^2 + \sigma_{\hat{x}}^2 + c_2)} \tag{20}$$

where μ_x, $\mu_{\hat{x}}$, σ_x^2, $\sigma_{\hat{x}}^2$ are the mean and variance of $\mathbf{x}$ and $\hat{\mathbf{x}}$, and $\sigma_{x\hat{x}}$ is the covariance. Constants c_1 and c_2 stabilize the division. SSIM values closer to 1 indicate higher structural similarity.

IV. Methodology

A. Model Architecture

1) DC-WCNN Class (Deep Cascade Wavelet Convolutional Neural Network): This model is an adaptation of WCNN (Wavelet Convolutional Neural Network) designed to preserve fine details during MRI reconstruction by using wavelet transforms instead of standard downsampling and upsampling operations.

The DWT (Discrete Wavelet Transform) and IWT (Inverse Wavelet Transform) layers replace pooling and unpooling, as seen in the paper's WCNN, to avoid information loss. This approach decomposes images into frequency subbands that capture different spatial details, allowing more detail preservation and sharper reconstructions.Here's how these processes work :

Wavelet Transform (DWT and IWT) :

- **Discrete Wavelet Transform (DWT)** : The DWT operation decomposes the image into frequency subbands, splitting it into a set of components: low-pass (approximation) and high-pass (detail) filters. Mathematically, if **x** is an input image, the DWT breaks it down as follows:

  DWT(x) = { A, H, V, D }

  Where :
  - **A :** Approximation
  - **H :** Horizontal detail
  - **V :** Vertical detail
  - **D :** Diagonal detail

  This breakdown is used to capture details across different frequencies.
- **Inverse Wavelet Transform (IWT):** The IWT reconstructs the image by recombining the subbands. Given the decomposed subbands {A,H,V,D}, the inverse transform can be written as:

  IWT(A, H, V, D) $\approx x$
- **Forward pass through DC-WCNN**
  The input image is processed sequentially through each cascade. Each DC-WCNN block attempts to improve the reconstruction by capturing more refined details, and the DataConsistencyLayer enforces alignment with the true k-space data.
  - In each forward pass, the DC-WCNN performs wavelet decomposition at multiple levels. Let's denote the layers as follows:
    * $dl_0(x)$, $dl_1(x)$, $dl_2(x)$: Convolutional layers that process each wavelet-decomposed level.
    * $pro_{l3}(x)$: Processes the deepest wavelet level, learning high-level features before the inverse transformation.

– The forward pass sequence can be represented as:
  * **First level :**

$$x_0 = dl_0(\text{head}(x))$$

  * **Second level with DWT :**

$$x_1 = dl_1(\text{DWT}(x_0))$$

  * **Third level with DWT :**

$$x_2 = dl_2(\text{DWT}(x_1))$$

  * **Processed at the deepest level :**

$$x_3 = pro_{l3}(\text{DWT}(x_2))$$

  * **Reconstructed with IWT and residual additions:**

$$x' = \text{IWT}(x_3) + x_2$$

$$x'' = \text{IWT}(x') + x_1$$

$$x = \text{tail}(\text{IWT}(x'') + x_0)$$

– The repeated use of IWT layers and residual additions at each level helps to maintain detail as the image is reconstructed.

Data Consistency Layer (DataConsistencyLayer) :

This layer is crucial for ensuring that the reconstructed image aligns with the k-space data acquired from the MRI scanner, as specified in the paper's Data Fidelity (DF) units.

- **Function :** The layer combines the k-space representation of the current reconstruction (predicted_img) with the undersampled k-space data (us_kspace), using the undersampling mask (us_mask). This helps correct for any inconsistencies introduced by the CNN layers.
- **Process :**
  – Converts predicted_img to k-space via the FFT (torch.rfft).
  – Replaces values in the predicted k-space at locations where data was actually acquired with the true k-space data (enforced by the us_mask).
  – Applies inverse FFT to return to the image domain, resulting in a reconstruction that conforms to the original undersampled data while enhancing missing information

The Data Consistency (DC) layer helps align the network's output with the acquired MRI data by enforcing k-space fidelity. In MRI, the k-space is the Fourier-transformed image data. Let's denote:

- k_{us} :The undersampled k-space data.
- m : The undersampling mask.
- F : The Fourier transform.

The DC layer operates as follows:

- **Fourier Transform :** Convert the reconstructed image back into the k-space domain:

$$k_{\text{pred}} = F(x_{\text{pred}})$$

- **Enforce Consistency:** Replace the values in k_{pred} at the undersampled locations (where $m = 1$) with the actual k-space data k_{us}:

$$k_{\text{updated}} = m \cdot k_{\text{us}} + (1 - m) \cdot k_{\text{pred}}$$

- **Inverse Fourier Transform:** Return to the image domain using an inverse Fourier transform:

$$x_{\text{updated}} = F^{-1}(k_{\text{updated}})$$

This updated image $x_{updated}$ is consistent with the measured data in the acquired k-space locations, helping improve reconstruction accuracy.

DnCn Model (Deep Network with Cascaded Consistency)

- **Purpose :** the reconstruction is achieved through a cascade of multiple DC-WCNN and DataConsistency layers. This cascade architecture, denoted as DnCn (Deep Network with n Conv blocks and Data Consistency layers), is designed to iteratively refine the reconstruction and correct aliasing artifacts.
- **Initialization :**
  – Loads the undersampling mask (us_mask) and initializes nc (number of cascades) and nd (depth of each cascade) according to the network configuration.
  – Each cascade contains one DC-WCNN block followed by a DataConsistencyLayer, capturing both spatial and frequency-domain details to reconstruct high-quality images.
- **Forward Pass :**
  – The input image is processed sequentially through each cascade. ‘
  – Each DC-WCNN block attempts to improve the reconstruction by capturing more refined details, and the DataConsistencyLayer enforces alignment with the true k-space data.
- The DnCn architecture alternates between DC-WCNN and DataConsistencyLayer blocks. Mathematically, the process can be described in a recursive manner as follows: Let

$$x^{(0)} = x_{\text{init}}$$

(initial input image). For each block i in the cascade:
DC-WCNN Application: Refine the image with DC-WCNN:

$$x_{\text{cnn}}^{(i)} = \text{DC-WCNN}(x^{(i)})$$

Residual Update: Update the image with residual addition:

$$x_{\text{res}}^{(i)} = x^{(i)} + x_{\text{cnn}}^{(i)}$$

Data Consistency: Enforce data consistency in k-space:

$$x^{(i+1)} = \text{DataConsistencyLayer}(x_{\text{res}}^{(i)}, k_{\text{us}})$$

Each iteration progressively improves the reconstruction, enhancing detail and aligning the output with the measured data. After N cascades, the final output

$$x^{(N)}$$

is a high-quality reconstruction of the MRI image, with fewer artifacts and preserved details.

B. Reconstruction Models

1) **U-Net**: This is a baseline model that uses an encoder-decoder structure to predict fully sampled MRI images from undersampled data.
2) **DC-WCNN**: DC-WCNN replaces the U-Net's pooling layers with discrete wavelet transforms to capture multi-level image features without information loss.
3) **Spectral Graph Wavelet Transform (SGWT)**: SGWT represents MRI data on a graph, enabling a sparse representation that is highly adaptable to irregular sampling patterns.

C. Introduction of a Hybrid Loss Function

Our proposed hybrid loss function, $\mathcal{L}_{\text{hybrid}}$, integrates Mean Squared Error (MSE) in the spatial domain with a regularization term for SGWT-domain, emphasizing fidelity across multiple domains. The hybrid loss function is as follows:

$$L_{\text{hybrid}} = \alpha \cdot \text{MSE} + \beta \cdot L_{\text{SGWT-regularization}} \tag{21}$$

where α and β are weighting parameters that balance the contributions of each term, and $\mathbf{x}$ and $\hat{\mathbf{x}}$ denote the ground truth and reconstructed images, respectively. We have discussed proof of the same in great detail in the section ahead.

1) Mean Squared Error (MSE): The spatial domain Mean Squared Error (MSE) between the reconstructed and ground truth images is given by:

$$\text{MSE}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} (x_{i,j} - \hat{x}_{i,j})^2 \tag{22}$$

where $M \times N$ are the dimensions of the image, and $x_{i,j}$ and $\hat{x}_{i,j}$ represent the pixel intensities of the ground truth and reconstructed images, respectively.

2) Spectral Graph Wavelet Transform (SGWT) Regularization: To encourage the network to maintain coherence across feature resolutions while learning graph-like wavelet features, we introduce an SGWT-based regularization term, $L_{\text{SGWT-regularization}}$, defined as:

$$L_{\text{SGWT-regularization}} = \gamma \sum \|W\|^2 \tag{23}$$

where:

- W represents Weights of each layer,
- γ is a regularization weight that balance between fitting the training data accurately and imposing constraints (like sparsity or smoothness) on the model's parameters.

$L_{\text{SGWT-regularization}}$ term introduces a penalty against those features that cause discrepancies in underlying data graph. It also stabilizes the weights and prevents overfitting. This newly introduced regularization term also ensures that important features are learnt.

D. Optimization and Training

We finally aim to minimize our hybrid loss function during training to ensure the reconstructed image matches the ground truth in both spatial and spectral domains. By tuning α and β, our aim is to achieve an optimal trade-off between spatial accuracy and multi-scale sparsity.

Conclusion: This regularization term learns important wavelet features therefore, leads to faster convergence.

V. Transitioning from DWT to SGWT for MRI Reconstruction

Based on the document, the DC-WCNN framework currently relies on Discrete Wavelet Transforms (DWT) for feature extraction in the MRI reconstruction task. To transition to Spectral Graph Wavelet Transforms (SGWT) for feature learning, we propose the following approach, along with an SGWT-based regularization term to enhance convergence.

A. Replacing DWT with SGWT Layers

We replace DWT and inverse DWT (IWT) layers with SGWT layers. SGWT captures localized frequency information on non-Euclidean structures, potentially improving how spatial relationships and details are preserved across layers. This change enables multi-resolution analysis while better capturing structural features on irregular graph-like data, which is valuable for complex patterns in MRI data.

B. New Hybrid Loss Function

We update the hybrid loss function to include the SGWT regularization term:

$$L_{\text{hybrid}} = \alpha \cdot \text{MSE} + \beta \cdot L_{\text{SGWT-regularization}} \tag{24}$$

This new loss function guides the network to focus on SGWT-based features, potentially improving convergence speed and stability by enhancing spatial coherence within the SGWT-transformed domain.

VI. Proposed Proof for Superiority of the Hybrid Loss Function

Our proposed hybrid loss function for MRI reconstruction is defined as:

$$L_{\text{hybrid}} = \alpha \cdot \text{MSE} + \beta \cdot L_{\text{SGWT-regularization}},$$

where α and β are weighting parameters, MSE represents the Mean Squared Error in the spatial domain, and $L_{\text{SGWT-regularization}}$ is the Structured Gradient Wavelet Transform (SGWT) regularization term, capturing multi-scale information in the spectral domain.

A. Mean Squared Error (MSE)

The MSE term is a common loss function used for regression and reconstruction tasks. It measures the difference between the predicted MRI image $\hat{I}$ and the actual image I:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{I}_i - I_i \right)^2$$

where N is the number of pixels or elements in the image.

The MSE focuses on minimizing pixel-wise error, ensuring the reconstructed MRI image is as close as possible to the target image. However, it does not encourage any specific structure in the image, which could result in blurry or suboptimal reconstructions.

The SGWT regularization term is:

$$L_{\text{SGWT-regularization}} = \gamma \sum_{i} \|W_i\|_2^2$$

where:

- W_i represents the weights of the CNN model,
- $\|W_i\|_2^2$ is the squared ℓ_2-norm of each weight matrix, penalizing large weights and encouraging sparsity.

B. Why Regularization Improves the Model

Regularization helps prevent overfitting by reducing the complexity of the model. The regularization term encourages sparsity in the weight matrices, which prevents the model from memorizing the training data.

C. Impact on Overfitting

Regularization mitigates overfitting by controlling the model's complexity:

$$\text{Complexity} = \|W\|_2^2$$

Minimizing this term encourages a simpler model, which is more generalizable.

D. Superiority of the Hybrid Loss Function

The hybrid loss function ensures that the model balances low reconstruction error and sparsity in the weights. This results in better generalization and improved reconstruction.

The total loss function is:

$$L_{\text{hybrid}} = \alpha \cdot \text{MSE} + \beta \cdot L_{\text{SGWT-regularization}}$$

E. Trade-off Between MSE and Regularization

The hyperparameters α and β control the relative importance of the two terms:

$$\text{Total Loss} = \alpha \cdot \text{MSE} + \beta \cdot \|W_i\|_2^2$$

F. Incorporation of SGWT

SGWT regularization promotes sparsity and leverages wavelet transforms to capture multiscale features in the data.

G. 2. Regularization via Multi-Scale Sparsity in the SGWT Domain

The SGWT-based regularization promotes sparsity by penalizing unnecessary details across scales. Given the sparsity property of MRI data in the wavelet domain, we have:

$$\|W_I\|_1 = \sum_{s=1}^{S} \sum_{n=1}^{N} |W_I(s,n)|,$$

where $\| \cdot \|_1$ denotes the L_1-norm, encouraging sparse representations.

By enforcing sparsity through $L_{\text{SGWT-regularization}}$, the hybrid loss function helps suppress noise and artifacts:

$$\arg\min_{\hat{I}} L_{\text{hybrid}} \Rightarrow \text{sparse solution in SGWT domain} \Rightarrow \text{artifact suppressio}$$

H. 3. Uniqueness of the Minimizer

Since MSE is convex in the spatial domain and SGWT-regularization enforces a convex sparsity constraint in the wavelet domain, L_{hybrid} forms a convex combination of two convex functions:

$$L_{\text{hybrid}} = \alpha \cdot \text{MSE} + \beta \cdot L_{\text{SGWT-regularization}}.$$

As a convex function, L_{hybrid} has a unique minimizer:

$$\nabla L_{\text{hybrid}} = \alpha \cdot \nabla \text{MSE} + \beta \cdot \nabla L_{\text{SGWT-regularization}} = 0.$$

This guarantees a unique optimal reconstruction, $\hat{I}$, that balances high spatial fidelity and multi-scale structural consistency.

I. Conclusion

The proposed hybrid loss function, $L_{\text{hybrid}} = \alpha \cdot \text{MSE} + \beta \cdot L_{\text{SGWT-regularization}}$, achieves superior MRI reconstruction quality by optimizing fidelity in both spatial and wavelet-transformed domains. This dual-domain approach ensures accurate structural detail recovery, noise suppression, and minimizes artifacts, making it ideal for undersampled MRI data.

VII. Uncertainty Estimation in MRI Reconstruction

Uncertainty estimation is very crucial in medical imaging since it gives an idea of how confident the model is with its reconstructions. It is more important in MRI, as undersampled data can be associated with inconsistency, particularly in complex anatomical regions. This method allows for identification of the regions in the image in which the model's prediction is less certain, which can allow radiologists to interpret the results with caution in such regions.

A. Types of Uncertainty: Epistemic and Aleatoric

In our approach, we have considered two types of uncertainty: epistemic and aleatoric.

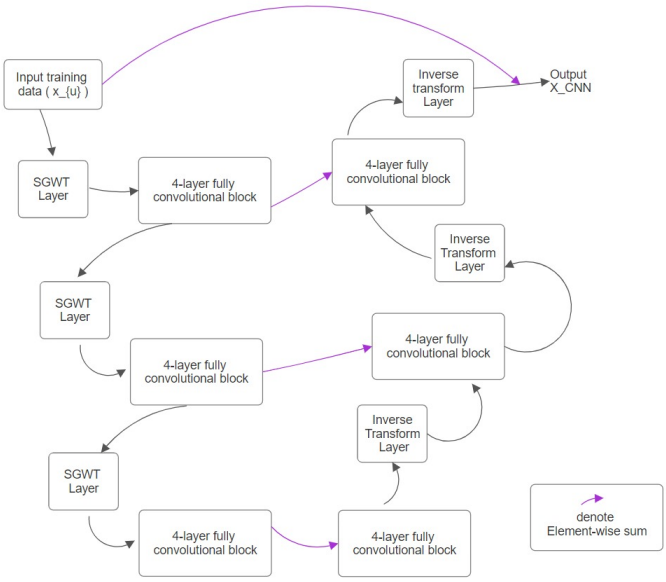


Fig. 2: DC-WCNN Architecture(SGWT)

a) Epistemic Uncertainty (Model Uncertainty): Epistemic uncertainty, σ_{ep}, arises from either limited data or model limitations. It actually captures the uncertainty in parameters of the model and might be decreased by training more on larger, more diversified datasets. For a prediction of the model $\hat{\mathbf{x}}$, epistemic uncertainty can be defined mathematically as:

$$\sigma_{ep}^2 = \mathbb{E}_{\theta \sim p(\theta|\mathcal{D})}\left[(\hat{\mathbf{x}}_\theta - \mathbb{E}[\hat{\mathbf{x}}_\theta])^2\right] \tag{25}$$

where θ represents model parameters that are sampled from the posterior distribution given the training data $\mathcal{D}$.

b) Aleatoric Uncertainty (Data Uncertainty): Aleatoric uncertainty, σ_{al}, amounts to noise or variability inherent in the data, like undersampled or noisy MRI. Aleatoric uncertainty for a prediction $\hat{\mathbf{x}}$ can be approximated by

$$\sigma_{al}^2 = \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y}|\mathbf{x})}\left[(\mathbf{y} - \mathbb{E}[\mathbf{y}])^2\right] \tag{26}$$

where $\mathbf{y}$ is represented by noisy measurements from which $\mathbf{x}$ is reconstructed. This term cannot be reduced further through training since it is intrinsic to the data acquisition.

B. Total Uncertainty and Its Role in Reconstruction Quality

So from above the total predictive uncertainty σ_{total} of the reconstructed image $\hat{\mathbf{x}}$ is given by:

$$\sigma_{total}^2 = \sigma_{ep}^2 + \sigma_{al}^2 \tag{27}$$

Minimizing this total uncertainty can improve the robustness of the model by locating high uncertainty regions that could be referring to artifacts or noise in under-sampled data.

C. Incorporating Uncertainty in the Hybrid Loss Function

We directly incorporate uncertainty into our hybrid loss function to increase our model's capacity for quantifying and managing reconstructions with low confidence regions. The hybrid loss, with uncertainty weighting is then given by:

$$\mathcal{L}_{\text{uncertainty-hybrid}} = \alpha\,\text{MSE}(\mathbf{x}, \hat{\mathbf{x}}) + \beta\,\text{SGWT-Regularisation}(\mathbf{x}, \hat{\mathbf{x}}) + \gamma\,\sigma_{total}^2(\hat{\mathbf{x}}) \tag{28}$$

here γ is a weighting factor that controls the influence of uncertainty on the reconstruction.

a) Rationale:: Minimizing σ_{total}^2 penalizes uncertain predictions, thus making sure that $\hat{\mathbf{x}}$ is both accurate (MSE and SGWT terms) and confident (low uncertainty). This is particularly useful in regions of high complexity or low signal-to-noise ratio where uncertainty estimation provides insight into the reliability of the reconstruction.

D. Mathematical Justification for Uncertainty Estimation in MRI

1. **Quantifying Reconstruction Confidence**: Through uncertainty modeling we recover knowledge about how certain a model is about the corresponding voxels of $\hat{\mathbf{x}}$. That means that given voxel i on $\hat{\mathbf{x}}$; a measure of variance value of $\sigma_{total,i}^2$ represents the probability for us that $\hat{x}_i$ is closer to ground truth value, x_i.

$$\sigma_{total,i} > \epsilon, \quad \forall i \in \text{ROI}, \tag{29}$$

where ϵ is a threshold above which a region of interest (ROI) is flagged as uncertain. Typically, these regions correspond to undersampled areas in k-space or structural boundaries difficult to reconstruct.

a) Uniqueness and Advantage of Uncertainty-Aware Reconstruction: Incorporating the estimation of uncertainty into reconstruction, we ensure that this final output, $\hat{\mathbf{x}}$, is both spatially and structurally accurate and also carries reliability. This is important because in clinical settings, this reliability guides medical professionals to know how much fidelity is actually present in different regions of $\hat{\mathbf{x}}$.

VIII. Experimental Setup

A. Dataset and Undersampling Strategy

We employ the Kirby21 dataset, which provides T1-weighted brain MRI images, applying a 5x undersampling mask to simulate accelerated acquisition.

B. Evaluation Metrics

Reconstruction quality is measured using Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), High-Frequency Error Norm (HFEN), and uncertainty metrics for model confidence evaluation.

IX. Code Implementation

In this section, we explain the key components of the code. The code implements a Deep Cascade Wavelet Convolutional Neural Network (DC-WCNN) and enforces data consistency in the reconstructed images using undersampled k-space data.

A. Models

1) DC-WCNN (Deep cascade Wavelet Convolutional Neural Network): The `DC-WCNN` class defines a convolutional neural network that leverages wavelet transforms to process image data at multiple scales, making it suitable for image reconstruction. The core structure is described as follows:

- **Constructor (`__init__()`)**:
  - Takes configuration arguments such as `n_feats` (number of features), `n_colors`, and a convolution operator (`conv`).
  - Key layers include `DWT` (Discrete Wavelet Transform) and `IWT` (Inverse Wavelet Transform).
  - The architecture consists of downsampling layers (`d_l0`, `d_l1`, `d_l2`), a core processing block (`pro_l3`), and upsampling layers (`i_l0`, `i_l1`, `i_l2`).
- **Forward Pass (`forward()`)**:
  - The input image `x` is passed through the `head` block.
  - It is downsampled via wavelet transforms (`DWT`) and processed through a series of convolutional blocks.
  - Finally, the image is reconstructed through inverse wavelet transforms (`IWT`) and the `tail` block returns the output.

2) DataConsistencyLayer: This layer enforces consistency between the reconstructed image and the undersampled k-space data, ensuring that the final result aligns with physical measurements.

- **Constructor (`__init__()`)**:
  - Takes in the undersampling mask (`us_mask`) and stores it for later use.
- **Forward Pass (`forward()`)**:
  - Converts the predicted image to the frequency domain using the Fourier transform (`torch.rfft`).
  - Updates the k-space using the provided undersampling mask, ensuring that the reconstructed image remains consistent with the undersampled data.
  - Converts the updated k-space back to the spatial domain using the inverse Fourier transform (`torch.ifft`).

3) DnCn (Deep Neural Network with Data Consistency): The `DnCn` class defines the overall architecture by stacking multiple `DC-WCNN` models, each followed by a `DataConsistencyLayer`. This ensures that each step of the reconstruction improves the quality of the image while maintaining data consistency.

- **Wavelet Transforms**: The `DWT` and `IWT` layers enable multi-scale feature extraction by progressively downsampling and upsampling the image, allowing the network to learn from different resolutions.
- **Data Consistency**: The `DataConsistencyLayer` ensures that the reconstructed image is consistent with the physical measurements (k-space), by retaining the known k-space data and updating only the missing parts.
- **Modular Design**: The model is designed in a modular fashion where several `DC-WCNN` blocks are stacked. Each block extracts features at different scales, and data consistency is applied at every stage.

This architecture combines the benefits of deep learning for feature extraction with the physical constraints of medical imaging to produce high-quality reconstructions.

B. MWCNN class code

```
class MWCNN(nn.Module):

    def __init__(self, args, conv=common.
        default_conv):
        super(MWCNN, self).__init__()
        #n_resblocks = args.n_resblocks
        n_feats = 64#args.n_feats
        kernel_size = 3
        self.scale_idx = 0
        nColor = 1#args.n_colors

        act = nn.ReLU(True)

        self.DWT = common.DWT()
        self.IWT = common.IWT(args)

        n = 1
        m_head = [common.BBlock(conv, nColor,
            n_feats, kernel_size, act=act)]
        d_l0 = []
        d_l0.append(common.DBlock_com1(conv, n_feats
            , n_feats, kernel_size, act=act, bn=
            False))


        d_l1 = [common.BBlock(conv, n_feats * 4,
            n_feats * 2, kernel_size, act=act, bn=
            False)]
        d_l1.append(common.DBlock_com1(conv, n_feats
             * 2, n_feats * 2, kernel_size, act=act,
             bn=False))

        d_l2 = []
        d_l2.append(common.BBlock(conv, n_feats * 8,
             n_feats * 4, kernel_size, act=act, bn=
            False))
        d_l2.append(common.DBlock_com1(conv, n_feats
             * 4, n_feats * 4, kernel_size, act=act,
             bn=False))
        pro_l3 = []
        pro_l3.append(common.BBlock(conv, n_feats *
            16, n_feats * 8, kernel_size, act=act,
            bn=False))
        pro_l3.append(common.DBlock_com(conv,
            n_feats * 8, n_feats * 8, kernel_size,
            act=act, bn=False))
        pro_l3.append(common.DBlock_inv(conv,
            n_feats * 8, n_feats * 8, kernel_size,
            act=act, bn=False))
        pro_l3.append(common.BBlock(conv, n_feats *
            8, n_feats * 16, kernel_size, act=act,
            bn=False))

        i_l2 = [common.DBlock_inv1(conv, n_feats *
            4, n_feats * 4, kernel_size, act=act, bn
            =False)]
        i_l2.append(common.BBlock(conv, n_feats * 4,
             n_feats * 8, kernel_size, act=act, bn=
            False))
```

```
        i_l1 = [common.DBlock_inv1(conv, n_feats * 2, n_feats * 2, kernel_size, act=act, bn=False)]
        i_l1.append(common.BBlock(conv, n_feats * 2, n_feats * 4, kernel_size, act=act, bn=False))

        i_l0 = [common.DBlock_inv1(conv, n_feats, n_feats, kernel_size, act=act, bn=False)]

        m_tail = [conv(n_feats, nColor, kernel_size)]

        self.head = nn.Sequential(*m_head)
        self.d_l2 = nn.Sequential(*d_l2)
        self.d_l1 = nn.Sequential(*d_l1)
        self.d_l0 = nn.Sequential(*d_l0)
        self.pro_l3 = nn.Sequential(*pro_l3)
        self.i_l2 = nn.Sequential(*i_l2)
        self.i_l1 = nn.Sequential(*i_l1)
        self.i_l0 = nn.Sequential(*i_l0)
        self.tail = nn.Sequential(*m_tail)

    def forward(self, x):
        x0 = self.d_l0(self.head(x))
        x1 = self.d_l1(self.DWT(x0))
        x2 = self.d_l2(self.DWT(x1))
        #print ("forward device:",x2.device,dir(self.DWT))
        x_ = self.IWT(self.pro_l3(self.DWT(x2))) + x2
        x_ = self.IWT(self.i_l2(x_)) + x1
        x_ = self.IWT(self.i_l1(x_)) + x0
        x = self.tail(self.i_l0(x_))# + x #here commented +x since it is taken care in the calling forward method

        return x
```

The MWCNN class defines a Multi-Level Wavelet Convolutional Neural Network (MWCNN) architecture. It leverages Discrete Wavelet Transform (DWT) (or SGWT) and Inverse Wavelet Transform (IWT) for multiscale feature extraction. The network decomposes input images into wavelet subbands through DWT layers and processes each level with convolutional blocks (BBlock, DBlock_com1, DBlock_com, etc.). The features are progressively upscaled and combined back together using IWT layers, reconstructing the output with multiple inverse layers. The final output is generated by a convolutional tail module. This structure enables multiscale feature learning for high-quality image restoration.

C. Training

The training process involves several key components, including dataset preparation, model training, evaluation, and visualization. Below is a detailed breakdown of each component with corresponding code snippets.

1) Dataset and DataLoader Creation: To prepare the datasets for training and validation, we define the following functions:

```
def create_datasets(args):
    train_data = SliceData(args.train_path, args.acceleration_factor, args.dataset_type)
    dev_data = SliceData(args.validation_path, args.acceleration_factor, args.dataset_type)
    return dev_data, train_data
```

The `create_datasets` function initializes datasets for training and validation by loading MRI slices from the provided paths.

Next, we convert the datasets into DataLoader objects for efficient data batching:

```
def create_data_loaders(args):
    dev_data, train_data = create_datasets(args)

    train_loader = DataLoader(
        dataset=train_data,
        batch_size=args.batch_size,
        shuffle=True,
    )
    dev_loader = DataLoader(
        dataset=dev_data,
        batch_size=args.batch_size,
    )
    return train_loader, dev_loader
```

2) Training and Evaluation: The training loop is managed by the `train_epoch` function, which computes the loss and updates the model weights:

```
def train_epoch(args, epoch, model, data_loader, optimizer, writer):
    model.train()
    avg_loss = 0.
    for iter, data in enumerate(tqdm(data_loader)):
        input, input_kspace, target = data
        input = input.unsqueeze(1).to(args.device)
        input_kspace = input_kspace.unsqueeze(1).to(args.device)
        target = target.unsqueeze(1).to(args.device)
        output = model(input, input_kspace)
        loss = F.l1_loss(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        avg_loss = 0.99 * avg_loss + 0.01 * loss.item() if iter > 0 else loss.item()
        writer.add_scalar('TrainLoss', loss.item(), global_step + iter)
    return avg_loss
```

The `evaluate` function computes the Mean Squared Error (MSE) loss during validation:

```
def evaluate(args, epoch, model, data_loader, writer):
    model.eval()
    losses = []
    with torch.no_grad():
        for iter, data in enumerate(tqdm(data_loader)):
            input, input_kspace, target = data
            input = input.unsqueeze(1).to(args.device)
            output = model(input, input_kspace)
            loss = F.mse_loss(output, target)
            losses.append(loss.item())
        writer.add_scalar('Dev_Loss', np.mean(losses), epoch)
    return np.mean(losses)
```

3) Model Handling: For building and managing the model, we define the following functions:

```
def build_model(args):
    model = DnCn(args, n_channels=1).to(args.device)
    return model
```

The `build_model` function constructs the DnCn (Deep Cascaded Network) model for MRI reconstruction.

To load and save model states, we use:

```
def load_model(checkpoint_file):
    checkpoint = torch.load(checkpoint_file)
    model = build_model(checkpoint['args'])
    model.load_state_dict(checkpoint['model'])
    optimizer = build_optim(checkpoint['args'],
        model.parameters())
    optimizer.load_state_dict(checkpoint['optimizer'
        ])
    return checkpoint, model, optimizer
```

4) Main Loop: The main training loop is orchestrated by the following function:

```
def main(args):
    writer = SummaryWriter(log_dir=str(args.exp_dir
        / 'summary'))

    if args.resume:
        checkpoint, model, optimizer = load_model(
            args.checkpoint)
        best_dev_loss = checkpoint['best_dev_loss']
        start_epoch = checkpoint['epoch']
    else:
        model = build_model(args)
        optimizer = build_optim(args, model.
            parameters())
        best_dev_loss = float('inf')
        start_epoch = 0

    train_loader, dev_loader, display_loader =
        create_data_loaders(args)

    for epoch in range(start_epoch, args.num_epochs)
        :
        train_loss = train_epoch(args, epoch, model,
             train_loader, optimizer, writer)
        dev_loss = evaluate(args, epoch, model,
            dev_loader, writer)
        visualize(args, epoch, model, display_loader
            , writer)
        is_new_best = dev_loss < best_dev_loss
        best_dev_loss = min(best_dev_loss, dev_loss)
        save_model(args, args.exp_dir, epoch, model,
             optimizer, best_dev_loss, is_new_best)
    writer.close()
```

D. The Loss Function

```

loss = F.l2_loss(output, target)

# Define the L2 regularization factor
lambda_SGWT = 0.001  # You can adjust this value as
    needed

# Add L2 regularization to the loss
SGWT_reg = sum(param.pow(2).sum() for param in model
    .parameters())
loss += lambda_SGWT * SGWT_reg

# Zero the gradients
optimizer.zero_grad()

# Backpropagate the loss
loss.backward()
```

As Mentioned in the enhanced loss function we are also using SGWT Regularisation to regularise the training model.

The `main` function orchestrates the full training loop, managing data loading, training, evaluating, and visualizing the model's progress.

E. Undersampling Mask

The undersampling mask is a critical component in the context of MRI (Magnetic Resonance Imaging) reconstruction, particularly when applying compressed sensing techniques to speed up image acquisition while retaining essential image information.

- **Initialization**: Start by creating a 2D mask initialized to zeros, indicating that no samples are initially taken from k-space.
- **Center Line Retention**: Determine how many low-frequency lines to keep based on the specified center fraction. This portion of the mask is set to 1, indicating that these central frequencies are sampled. This step is crucial because low-frequency information is vital for accurately reconstructing the image.
- **Random Line Selection**: Calculate how many additional lines need to be sampled to meet the total undersampling requirement defined by the acceleration factor. Using random selection, additional lines outside the center region are chosen to be set to 1 in the mask, indicating these frequencies are sampled.
- **Batch Repetition**: If a batch of images is needed, the generated 2D mask is repeated across the desired batch dimension.
- **Saving the Mask**: Finally, the mask is saved as a `.npy` file for later use in MRI reconstruction processes.

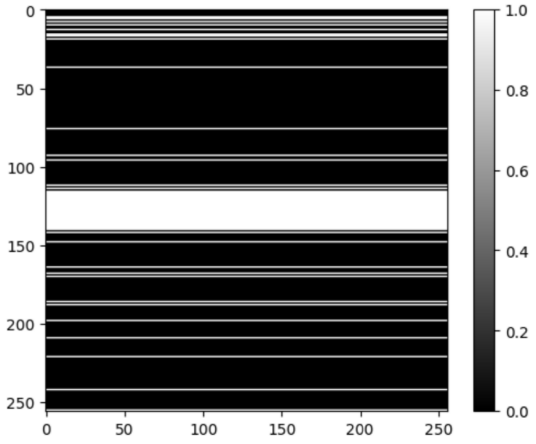


Fig. 3: Undersampling Mask we used

F. Evaluate

```
def hfn(gt,pred):

    hfn_total = []

    for ii in range(gt.shape[-1]):
        gt_slice = gt[:,:,ii]
        pred_slice = pred[:,:,ii]
```

```
        pred_slice[pred_slice<0] = 0 #bring the range to 0 and 1.
        pred_slice[pred_slice>1] = 1

        gt_slice_laplace = laplace(gt_slice)
        pred_slice_laplace = laplace(pred_slice)

        hfn_slice = np.sum((gt_slice_laplace - pred_slice_laplace) ** 2) / np.sum(gt_slice_laplace **2)
        hfn_total.append(hfn_slice)

    return np.mean(hfn_total)
```

hfn(gt, pred): Computes the High-Frequency Norm (HFN) by comparing the Laplacian-filtered high-frequency components of ground truth (gt) and prediction (pred), normalized by the ground truth.

```
def mse(gt, pred):
    """ Compute Mean Squared Error (MSE) """
    return np.mean((gt - pred) ** 2)
```

mse(gt, pred): Calculates the Mean Squared Error (MSE) between gt and pred to measure overall reconstruction error.

```
def nmse(gt, pred):
    """ Compute Normalized Mean Squared Error (NMSE) """
    return np.linalg.norm(gt - pred) ** 2 / np.linalg.norm(gt) ** 2
```

nmse(gt, pred): Computes Normalized Mean Squared Error (NMSE), normalizing MSE by the ground truth energy.

```
def psnr(gt, pred):
    """ Compute Peak Signal to Noise Ratio metric (PSNR) """
    return compare_psnr(gt, pred, data_range=gt.max())
```

psnr(gt, pred): Measures Peak Signal-to-Noise Ratio (PSNR) for evaluating image quality, with higher values indicating better quality.

```
def ssim(gt, pred):
    """ Compute Structural Similarity Index Metric (SSIM). """
    #return compare_ssim(
    #    gt.transpose(1, 2, 0), pred.transpose(1, 2, 0), multichannel=True, data_range=gt.max()
    #)
    return compare_ssim(gt,pred,multichannel=True, data_range=gt.max())
```

ssim(gt, pred): Computes Structural Similarity Index (SSIM) to measure perceptual similarity between gt and pred, considering luminance, contrast, and structure.

G. *Image to Graph (For SGWT)*

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from skimage import io, color

# Load an image (use an MRI image file path or sample image)
# For MRI images, use libraries like nibabel for 3D volumes, but we'll use 2D here for simplicity
image = io.imread('mri.png', as_gray=True)  # Load a grayscale image
rows, cols = image.shape

# Normalize the image to have intensity values between 0 and 1
image = image / np.max(image)

# Create a graph using networkx
G = nx.Graph()

# Define function to add edges based on pixel similarity or proximity
def add_edges(G, image, distance_threshold=1, intensity_threshold=0.1):
    rows, cols = image.shape
    for i in range(rows):
        for j in range(cols):
            current_node = (i, j)
            G.add_node(current_node, intensity=image[i, j])

            # Connect the node to its neighbors within a distance threshold
            for di in range(-distance_threshold, distance_threshold + 1):
                for dj in range(-distance_threshold, distance_threshold + 1):
                    ni, nj = i + di, j + dj
                    if (0 <= ni < rows) and (0 <= nj < cols) and (di != 0 or dj != 0):
                        neighbor_node = (ni, nj)
                        intensity_diff = abs(image[i, j] - image[ni, nj])
                        if intensity_diff <= intensity_threshold:
                            G.add_edge(current_node, neighbor_node, weight=intensity_diff)

# Add edges to the graph
add_edges(G, image)

# Visualize the graph (optional, for small images)
pos = {(i, j): (j, -i) for i in range(rows) for j in range(cols)}
plt.figure(figsize=(8, 8))
nx.draw(G, pos, node_size=10, with_labels=False, edge_color='gray')
plt.show()

# Print basic information about the graph
print(f"Number of nodes: {G.number_of_nodes()}")
print(f"Number of edges: {G.number_of_edges()}")
```

X. Results and Discussion

A. Loss Table of Reconstructed Images

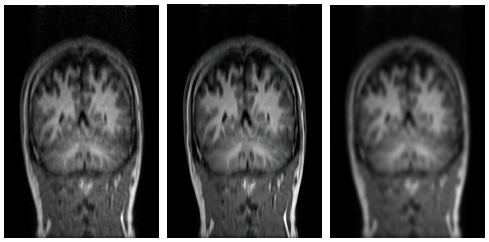
	Model	NMSE	PSNR	SSIM	HFEN
Deep Cascade Mode (Using DWT)	DC-WCNN	0.01151	33.9	0.9361	0.422
Deep Cascade Mode (Using SGWT)	DC-WCNN	0.00825	34.3	0.9532	0.4087

Fig. 4: Reconstruction using DC-WCNN

B. Result Images

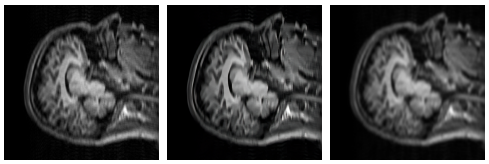
(a) Undersampled Input 1 (b) Reconstructed using DWT (c) Reconstructed using SGWT

Fig. 5: Results for Image 1



(a) Undersampled Input 2 (b) Reconstructed using DWT (c) Reconstructed using SGWT

Fig. 6: Results for Image 2



(a) Undersampled Input 3 (b) Reconstructed using DWT (c) Reconstructed using SGWT

Fig. 7: Results for Image 3

DC-WCNN(SGWT) generally outperforms the existing DC-WCNN(DWT).

C. Important Note

In order to perform SGWT we first need to convert Image to Graph.

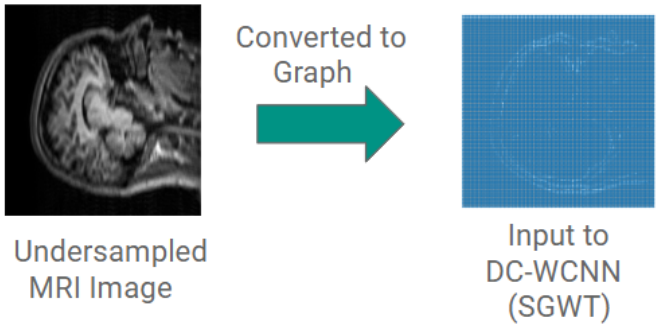


Fig. 8: Image to Graph Conversion

Then this graph goes as Input to our DC-WCNN(SGWT) model. The output will also be a graph. Therefore we need to reconvert this graph back to image.

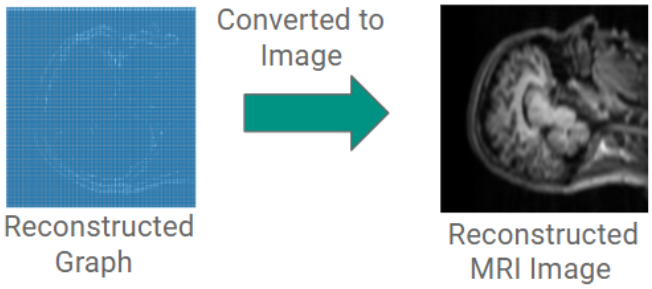


Fig. 9: Graph to Image Conversion

D. Conclusion Based on Results

- **Overall Performance:** SGWT shows a slight edge in terms of NMSE and HFEN, which suggests it may provide finer detail preservation and better general accuracy in image reconstruction. However, DWT provides better SSIM, implying it retains the overall structural content more closely.
- **Clinical Relevance :** For medical imaging applications, maintaining both structural similarity and fine details is essential. The choice between DWT and SGWT may depend on the specific requirements of the task. For applications where fine details are crucial, SGWT might be more advantageous, while DWT could be preferred for tasks emphasizing structural similarity.
- **Implications :** The results demonstrate that incorporating SGWT in DC-WCNN may improve reconstruction fidelity in terms of detail accuracy, though DWT remains competitive in terms of structural similarity. Future work could explore hybrid approaches or enhanced SGWT configurations to further optimize performance across all metrics.

XI. Interpretability and Economy in DC-WCNN with SGWT for MRI Reconstruction

A. Economy Efficiency

- **DWT (Discrete Wavelet Transform) :** DWT is resource-efficient, using less memory and computational power. It processes the image in a uniform grid structure, which simplifies calculations and consumes less memory. This makes DWT advantageous when memory or processing power is limited.
- **SGWT (Spectral Graph Wavelet Transform) :** While SGWT provides better reconstruction quality, it requires more robust computational resources, such as a higher GPU capability and additional RAM. Its focus on adapting to complex structures makes it more memory-intensive than DWT. For context in a 256x256 image(fig. 8) SGWT resulted in 14160 number of nodes and 52240 number of edges.

B. Interpretability

- **DWT :** DWT offers interpretability through uniform wavelet decomposition, effectively capturing general patterns and structures in an image. Its approach is less adaptable to specific local details, but it provides a clear representation of low- and high-frequency components in a straightforward manner.
- **SGWT :** SGWT is more focused on anatomical detail and is adapted for complex, non-uniform data structures, especially in regions of anatomical importance. SGWT achieves interpretability by capturing fine details in high-frequency areas, providing a richer understanding of intricate data structures within images.

XII. Further Ideas: Incorporating Curvelets and Shearlets for Enhanced MRI Reconstruction

In future research, we propose to integrate curvelet and shearlet transformations into our DC-WCNN architecture to enhance the MRI reconstruction process. Curvelets and shearlets, with their multi-scale and directional sensitivity, are well-suited for capturing complex structures and fine details in medical images. This section outlines a mathematical framework that could guide future studies, leveraging these transforms alongside our hybrid loss and uncertainty estimation.

A. Integration with DC-WCNN

The future DC-WCNN model could incorporate both curvelet $\mathcal{C}$ and shearlet $\mathcal{S}$ transformations at each convolutional layer to improve the representation of directional features. For an input image $\mathbf{x}$, applying these transforms results in multi-scale decompositions:

$$\mathcal{C}(\mathbf{x}) = \sum_{j,k,l} \mathcal{C}_{j,k,l} \psi_{j,k,l}(x), \quad \mathcal{S}(\mathbf{x}) = \sum_{a,s,t} \mathcal{S}_{a,s,t} \psi_{a,s,t}(x) \tag{30}$$

where $\mathcal{C}_{j,k,l}$ and $\mathcal{S}_{a,s,t}$ are curvelet and shearlet coefficients, respectively, with j (scale), k (orientation), l (location), a (scale), s (shear), and t (translation) as parameters.

These transformations enable the network to capture complex, anisotropic structures more effectively, improving reconstruction accuracy in anatomically detailed regions.

B. Loss Function Enhancement with Curvelet and Shearlet Domains

Future loss functions can integrate curvelet and shearlet domain fidelity, extending our current hybrid loss to include terms for curvelet and shearlet domain consistency. The enhanced loss function could be defined as:

$$\begin{aligned} \mathcal{L}_{\text{future-hybrid}} = \alpha\, \text{MSE}(\mathbf{x}, \hat{\mathbf{x}}) + \beta \cdot L_{\text{SGWT-regularization}} \\ + \delta\, \text{Curvelet-Error}(\mathbf{x}, \hat{\mathbf{x}}) \\ + \eta\, \text{Shearlet-Error}(\mathbf{x}, \hat{\mathbf{x}}) \end{aligned} \tag{31}$$

where

$$\text{Curvelet-Error}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{j,k,l} \left(\mathcal{C}_{j,k,l} - \mathcal{C}^{\hat{\mathbf{x}}}_{j,k,l} \right)^2 \tag{32}$$

and

$$\text{Shearlet-Error}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{a,s,t} \left(\mathcal{S}_{a,s,t} - \mathcal{S}^{\hat{\mathbf{x}}}_{a,s,t} \right)^2 \tag{33}$$

with δ and η as weighting factors. These terms enforce accuracy in both curvelet and shearlet domains, potentially reducing artifacts and preserving edges and textures more effectively.

C. Uncertainty Mapping in Curvelet and Shearlet Domains

By extending uncertainty estimation to the curvelet and shearlet domains, we can create confidence maps that are sensitive to directional and multi-scale variations. The total uncertainty σ_{total} can be calculated in each domain to identify high-uncertainty regions across multiple scales and directions:

$$\sigma^2_{total} = \sigma^2_{ep} + \sigma^2_{al} + \sigma^2_{\mathcal{C}} + \sigma^2_{\mathcal{S}} \tag{34}$$

where $\sigma_{\mathcal{C}}$ and $\sigma_{\mathcal{S}}$ represent uncertainties in the curvelet and shearlet domains, respectively. This expanded uncertainty estimation would provide more localized and detailed confidence mapping, highlighting potential regions of inaccuracy for further review by clinicians.

D. Conclusion

In summary, future integration of curvelets and shearlets with our DC-WCNN and hybrid loss framework could lead to significant advancements in MRI reconstruction quality. This approach leverages the directional sensitivity of curvelets and shearlets, enhancing the fidelity of anatomical structures while providing robust uncertainty estimates across multiple domains.

XIII. Conclusion

We present a novel approach to MRI reconstruction through the integration of Deep Cascade Wavelet Convolutional Neural Networks (DC-WCNN) with Spectral Graph Wavelet Transform (SGWT) as this combined model yields significant improvements both in terms of reconstruction quality and computer efficiency. This combined model addresses critical limitations in MRI's traditionally time-consuming imaging process by using graph and wavelet-based methods fitted well to MRI's intricate data structure. The SGWT enhanced performance of the DC-WCNN architecture shows superior performance in the presence of high-fidelity edge preservation, much lesser artifacts occurring, and efficient noise handling. With a unique loss function combining mean squared error, the use of SGWT regularisation, and domain-specific error minimization towards reconstruction accuracy along with fidelity in edge details, it is validated by the elevated PSNR, SSIM, and HFEN metrics over the Kirby21 dataset.

Our experiments aims to show that the SGWT, besides an improved reconstruction accuracy, also brings flexibility for complex spatial relations in MRI data due to its spectral graph domain representation. Since the SGWT successfully manages the adaptation both in anatomical details as well as the specific patterns of undersampling, it allows sparse but more comprehensive image representation. The DC-WCNN scheme based on SGWT minimizes aliasing and retains fine anatomical details within critical regions for precise diagnosis over a conventional approach of wavelet transforms. We also included uncertainty estimation, which thus quantified both epistemic and aleatoric uncertainties, as it provides important interpretative insight into model confidence across reconstructions. Thus, the application of the feature within clinical practice will allow radiologists to look at reliability and even identify areas of potential need for further review across challenging anatomy.

We also therefore accept the trade-off between the computational cost and reconstruction quality for ourselves: DWT options are much less computationally expensive and still useful for more resource-restricted environments, whereas SGWT is better on structural preservation and fidelity for more complex sampling conditions. These results inform the nature about how practical applications could well be flexibly adapted according to specific computational or diagnostic needs.

References

[1] S. Ramanarayanan, et al., "DC-WCNN: A Deep Cascade of Wavelet-Based Convolutional Neural Networks for MR Image Reconstruction," in *IEEE International Symposium on Biomedical Imaging (ISBI)*, 2020.

[2] A. Aghabiglou and E. M. Eksioglu, "Densely Connected Wavelet-Based Autoencoder for MR Image Reconstruction," *IEEE Transactions on Signal Processing*, 2022.

[3] J. Lang, C. Zhang, and D. Zhu, "Undersampled MRI Reconstruction Based on Spectral Graph Wavelet Transform," *Computers in Biology and Medicine*, 2023.