DE-42 (EE) UMAIR, LARAIB, MAHAM, NOOR

AI Driven PET-like Synthesis from MRI Data

COLLEGE OF

ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
RAWALPINDI

2024

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-42 EE

PROJECT REPORT

"AI-Driven PET-like Synthesis from MRI Data"

Submitted to the Department of Electrical Engineering

in partial fulfillment of the requirements

for the degree of

Bachelor of Engineering

in

Electrical

2024

Sponsoring DS:

Submitted By:
UMAIR IRFAN
SYEDA MAHAM RAZA
LARAIB LAIQ
NOOR UL AIN ZAHRA

CERTIFICATE OF APPROVAL

It is to certify that the project **"AI Driven PET-Like Synthesis from MRI Data"** was done by **NS Umair Irfan**, **NS Laraib Laiq**, **NS Syeda Maham Raza**, and **NS Noor ul Ain Zahra** under the supervision of **Asst Prof Kamran Aziz Bhatti** and **Prof Dr Shahzad Amin Sheikh**.

Submission : This project was submitted to the College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan, as part of the requirement for the degree of Bachelor of Electrical Engineering.

Students:

1. **Umair Irfan**

NUST ID: 334988 Signature: _____

2. **Laraib Laiq**

NUST ID: 332473 Signature: _____

3. **Syeda Maham Raza**

NUST ID: 331709 Signature: _____

4. **Noor ul Ain Zahra**

NUST ID: 347157 Signature: _____

Approved By:

Project Supervisor: **Asst Prof Kamran Aziz Bhatti**

Signature: _____ Date: _____

Project Co-Supervisor: **Prof Dr Shahzad Amin Sheikh**

Signature: _____ Date: _____

Head of Department: **Assoc Prof Dr Qasim Umar Khan**

Signature: _____ Date: _____

DECLARATION

We declare that no part of this project thesis has been submitted in support of an application for another degree or qualification. We have not submitted this thesis to any other university or educational institution. We are totally liable for any disciplinary action taken against us based on the nature of the proved offence, including the revocation of our degree.

Students:

1. **Umair Irfan** _____
2. **Laraib Laiq** _____
3. **Syeda Maham Raza** _____
4. **Noor ul Ain Zahra** _____

ACKNOWLEDGMENTS

We would like to start by conveying our deepest gratitude for the blessings and guidance bestowed upon us by Allah Almighty. It was under His light and through his blessings that we found the will and motivation to complete this project. Truly there are none greater than He.

Furthermore, we wish to express great thanks to Mr. Kamran Aziz and Dr. Shahzad Amin. Without their patience, open heart and constant guidance none of this would have been possible.

We would also like to thank Ms. Zoha Fatima Syed, who took time out from her hectic schedule to help us connect and correspond with academics who were otherwise out of reach. It should be noted that partial credit for obtaining the data set and for communication with authors of previous research extends to her.

Finally, we would like to thank Dr. Iqtedar Ahmed Muazzam for his input and prompt feedback regarding our project and outputs as an oncologist.

ABSTRACT

Our project aims to synthesize Positron Emission Tomography (PET) - Like images from a MRI scan from artificial intelligence (AI) driven models. Dataset used in this regard is of 37 patients each having a T1w, FLAIR and PET image modality. These images which were in Neuroimaging Informatics Technology Initiative (NIFTI) format were pre-processed by converting into 2D tensors and extending them to 3D tensors by adding an extra dimension. The T1 and FLAIR images are concatenated and given as input to pix2pix model while PET images are set as the ground truth for our model. The synthesized output from the above model serves as the input to another machine learning model which is a modified super resolution convolutional neural network (SRCNN) called Fast Medical Image Super Resolution Method. This model maps a low resolution image to a super resolution image thus giving us better images. For hardware implementation, FPGA and DSP Kit are utilized for pattern recognition on the output PET-like image. Furthermore, the synthesis software model is uploaded on a Raspberry Pi to allow for localization and environment integration. This project will aid in bridging the healthcare gap by providing a non-invasive alternate for PET imaging by using easily accessible MRI data. It will also reduce the need of costly PET scanners which are limited in Pakistan.

SUSTAINABLE DEVELOPMENT GOALS

SDG 3 focuses on good health and wellbeing. This goal is well inclined with our project as it aims to provide an inexpensive and non-invasive alternative to patient serving as and advantage for them as well as for the doctors because it will assist them in taking decisions.

SDG 9 caters industry, innovation and infrastructure. Our project intends to bridge the healthcare gap and promote advancement in the medical field.

SDG 10 ensures equal opportunities to all. With the help of our project people who can't afford PET imaging can go towards MRI and then have a synthesized output.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Motivation

Medical image-to-image translation is where one image modality is converted to another. Image-to-image translation aids to create synthetic images for having greater insights from multiple imaging modalities. Mainly discussed modalities in this report would be Magnetic Resonance Imaging (MRI) and Positron Emission Tomography (PET).

MRI and PET are common but expensive diagnostic techniques used in cancer detection and cardiovascular and neurological disorders. Comparatively , PET machines are far less accessible due to their prohibitive costs, difficult logistics, and use of radioactive-labeled tracers to bind to specific molecular targets (e.g. glucose, oxygen, and amyloid-beta) [1].

In particular, Pakistan has in total 7 publically available PET machines which result in long waiting queues for patients requiring such scans [2].

Magnetic resonance imaging (MRI), on the other hand, use the magnetic properties of tissue or blood to create various contrasts (e.g. T1, T2, T2-weighted images) [1]. Thus, they are less expensive, cheaper to maintain and use, far more readily available, and do not require radioactive tracers [3].

To alleviate the issue of inaccessibility to PET scans, a method is proposed herein to synthesize PET-like scans from MRI data

1.2 Types of Imaging Modalities

1.2.1 Positron Emission Tomography (PET)

A PET scan uses a radioactive chemical called a radiotracer (or tracer) detected by a PET scan machine to produce images that help a doctor understand how tissues or organs are functioning. [4] PET scans are most often used to:

- Find cancer or track its progress,
- Assess brain damage or disorders such as tumors, seizures, or cognitive issues,
- Evaluate damage to the heart following a heart attack, or
- Assess the state of coronary artery disease

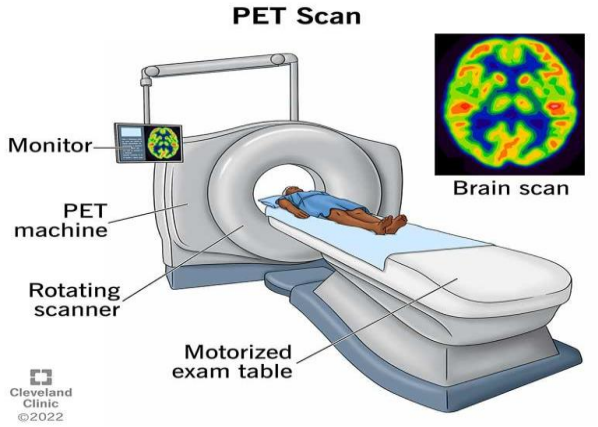


Figure 1.1 Scanner used for PET Imaging

1.2.2 Magnetic Resonance Imaging (MRI)

An MRI is an imaging technique that sends radio waves into the body, which are reflected by substances like water and fat. The waves are then captured and recorded by an MRI Scanner that turns this data into a detailed image of the area or organ. [4] It focuses on areas like:

- Joints
- Blood vessels
- Brain and spinal cord
- Abdominal organs

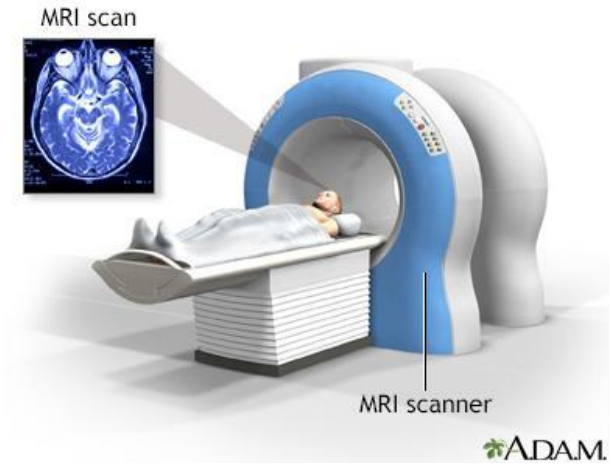


Figure 1.2 Scanner used for MRI Imaging

MRI can be further categorized into sequences such as:

1. **T1**

The contrast in T1-weighted sequences is mainly controlled by the Repetition Time (TR) and Echo Time (TE) values. To achieve this, a short TR, typically ranging from 400 to 700 ms, is chosen to ensure that tissues with different T1 relaxation times recover differently between successive RF pulses.[5] These parameters are fine-tuned to effectively highlight tissues with shorter T1 relaxation times, such as fat, making them appear brighter compared to other tissues.

2. **Fluid Attenuation Inversion Recovery (FLAIR)**

The technique involves an inversion recovery pulse to cancel the CSF signal, followed by a delay and then T2-weighted image acquisition. The result is an image where fluid appears dark, while pathological alterations in nearby tissue stand out with increased contrast. There are three parameters in FLAIR to determine contrast: Inversion Time (TI), Repetition Time (TR), and Echo Time (TE).

3. **T2**

This type of sequence tells about the differences in T2 relaxation times of various tissues. The T2 relaxation is the decay of transverse magnetization (Mxy) over time after an external radiofrequency (RF) pulse is applied. They are often used in clinical imaging to judge the water content and other tissue characteristics.

1.3 Risks and Limitations of MRI

High-quality shots require the ability to hold still perfectly while the images are being captured. If someone is experiencing extreme pain disorientation or anxiety, it may be difficult to lie motionless during imaging. Certain MRI machines have weight restrictions so larger patients might not be able to fit inside. Metallic objects like implants can interfere with the acquisition of clear images. Right now there is not enough data to draw the conclusion that non-contrast MRIs are harmful to a developing fetus. Still, if conditions are not life-threatening, physicians may decide to postpone it until after delivery. In order to search for findings that ultrasound is unable to fully assess, doctors may evaluate the fetus with non-contrast MRI after the first trimester. Gadolinium contrast agent application for MRIs should generally be avoided during pregnancy unless very specific circumstances demand it. Differentiating between cancerous tissue and fluid or edema may not always be possible with brain MRI imaging. An MRI could take longer than other imaging tests. Whether the radio waves and magnetic fields used in MRI scans could be harmful to human health has been the subject of much investigation. As no evidence of a risk has been discovered magnetic resonance imaging (MRI) is among the safest medical procedures currently available.

1.4 Risks and Limitations of PET

A patient's ability to absorb the sugar in the radiotracer may be compromised by diabetes potentially influencing scan outcomes. Before the test, physicians will make recommendations about how to change one's diet and medications. PET scans are generally risk-free and rarely result in issues. With the radioactive tracer there is very little radiation and it is not long-lasting in the body. After a PET scan, it is recommended to drink a lot of water to help the body rid itself of the radioactive drug. Typically, there are only a few circumstances in which PET scans are dangerous. PET scans should not be performed on pregnant nursing or chest-feeding individuals. Radiation can damage a fetus and be transferred to a baby through breast milk. Certain individuals experience an allergic response to radioactive tracers used in PET scans or contrast dyes used in CT scans. These allergic reactions are typically mild and very uncommon. If this reaction occurs, the medical team can promptly slow it down with medication

1.5 Prior Work

The following section will discuss the previous work that has been done regarding image synthesis from one modality to another via deep learning models.

1.3.1 Cross Modality Image-to-Image Translation

In medical imaging, cross-modality synthesis is very popular and beneficial for clinical decisions such as aid in diagnostic predictions. Several papers have been published regarding this task. For example, Avi Ben-Cohen et al. [6] provide a PET estimation from CT scans using Fully Convolutional Networks (FCN) and Generative Adversarial Networks (GAN). They suggest FCN and GAN for both training and testing purposes and finally outputs from both these models as their next step for image blending. The two evaluation metrics depict a TPR of 92.3% and an FPR of 0.25 per case. In [7] Rajagopal et al. employed deep learning techniques to predict PET sonograms from whole-body MRI images. A 3D residual UNet was trained on a dataset of 56 patients along with a balanced loss function to generate the synthetic PET. Quantitative results show a 7.6% error in mean-SUV compared to using real data. In [8] Jake et al. developed an automated segmentation method for prostate cancer based on the nn-UNet framework. This method achieves promising results by assessing metrics like dice similarity coefficient (DSC), positive predictive value (PPV), and sensitivity. Jeffrey P. Leal et al. have taken 100 PET/CT scans in [9], segmented into thresholds according to the software PERCIST v1.0. These classifications were then forward propagated to a CNN which then helped identify the breast cancer tissue. A sensitivity of 96-%, DICE score of 94-%, and Jaccard score of 89-% are demonstrated by this model. [10] Yang et al. developed a cross-modality MRI image generation method for multimodal registration and segmentation using conditional generative adversarial networks. This method achieved comparable results on five brain MRI datasets while using a single modality image as an input. Karim Armanious et al. describes in [11], a GAN-based framework in mix with the non-combinational losses was

developed. This addresses three different medical image-to-image translation problems: PET-to-CT translation, MRI motion artifacts correction, and PET image denoising. The quantitative evaluations for the three tasks exhibit the superiority of the proposed GAN architecture over the existing translation methods. Furthermore, Xie et al. [12] developed an advanced cascade neural network architecture that takes the contour information of the original input MRI images. Quantitative and qualitative assessments on a test set of 169 patients showed that there were no intensity differences observed in both tumor and non-tumor brain regions.

1.3.2 MRI to PET Translation

Image synthesis from MRI to PET has been explored before in research papers. For example, in [13] Se-In et al. have worked on synthesizing realistic tau-PET images from textual description and MRI images. Textual descriptions were encoded into this latent space, and the encoded information guided the diffusion process to synthesize the desired tau PET image. The preliminary results like MMSE values showed feasible realistic PET scans as output Taofeng Xie et al. [14] employed a joint diffusion attention model (JDAM) which convert high-field PET from an ultra-high field MRI.JDAM uses a joint probability distribution (JPD) and a diffusion process where Gaussian noise is gradually added to PET images, then employs a reverse diffusion process and Langevin dynamics to generate synthetic PET from MRI. In [15], Wei et al. has used an approach of Sketcher-Refined GANs where the sketcher portion tells about the physiological information while refiner works on the improvement of tissue myelin content. Sikka et al. [16] demonstrate GLA-GAN (Globally and Locally Aware GAN) to support the diagnosis of Alzheimer's via MRI to PET imaging. The model proposed above incorporates adversarial loss, voxel-level intensity, multi-scale structural similarity (MS-SSIM), and ROI-based loss. These metrics strengthen the enhancement of synthesized images and clinical utility. Jung et al. [3] addresses how a pix2pix GAN can estimate a PET-like scan from a MRI image. Two experiments have been done where one includes the usage of data sliced in one dimension and the other experiment includes data sliced in all three dimensions (axial, sagittal, and coronal). MSE, PSNR and SSIM were the evaluation metrics used on the middle slices to check the model authenticity. Similarly in [17] Jia et al. developed a deep convolutional neural network to train a combination of single and multi-delay ASL (arterial spin labeling) so that they can predict PET images. This was done in order to enhance the MRI based CBF measurements. It also demonstrated reduced estimation error for mean CBF and a stronger correlation with PET measurements.

2. BACKGROUND AND LITERATURE REVIEW

2.1 Dataset

Imaging databases play an integral part in research areas and projects related to medical image-to-image translation. Alzheimer's disease Neuroimaging Initiative (ADNI) is a database where researchers study the developments of Alzheimer's disease and provides publicly accessible MRI and PET images previously used before in research papers. In prior years, an increasing number of articles and journals related to medical imaging have used datasets that are either acquired through ADNI and similar public databases or via private data sources. The following sections focus on such datasets.

2.1.1 MRI or PET Only Data

In [18], experiments for assessing performance of an E-GAN was done on ADNI dataset. MRI and corresponding pairs of 256 healthy individuals were taken for use. The scans used were ensured to have a delay of a year so that the differences can be attenuated between the acquisition times. Sikka et al. [16] collected 402 samples from ADNI-1 and ADNI-2 which are the phases of images taken from 2004 to 2010 and 2011 to 2017 respectively. These samples contain both preprocessed MRI and PET modalities. Furthermore, another 179 samples from ADNI-1 and ADNI-2 were taken which had only MRI images. Similarly in [14], 13560 pairs of images from 452 subjects were used to conduct the research for the joint diffusion model.

2.1.2 Simultaneous MRI and PET Data

Imaging technique where MRI and PET scanning technology combines simultaneously results in a more detailed picture. ADNI obtains MRI and PET images which are divided in visits depending upon the patient. The visits are after every 6 months and yearly.

2.1.2.1 Privately Collected Data

[1] and [17] acquired datasets privately via GE Healthcare where they did scans of 131 and 32 patients respectively to obtain MRI and PET scans at the same time-of-flight. In another instance [19] takes an institutional dataset from a university hospital where patients underwent both PET and MRI scan.

2.1.2.2 Publicly Available Data

Mostly image translation related research papers have used publicly accessible data from ADNI database as they perform imaging in a specific window which is of ranges in between months and years. [14] and [16] make use of ADNI database to obtain subject scans.

2.1.3 Data Dimensionality

In medical imaging, slices are the 2D cross-sections in which an image of a patient's concerned area is divided into during the imaging procedure. These 2D slices combine to form a whole 3D picture. Each of this slice provides anatomical features of that specific body part whose scan has been performed. Therefore slices and their different views (axial, coronal, sagittal) aid doctors to identify where and what the problem is with the patient. The databases found online like on ADNI or other open sources are stored in image formats, Digital Imaging and Communications in Medicine (DICOM) and Neuroimaging Informatics Technology Initiative (NIFTI). The NIFTI format saves medical images in volumetric form which isn't directly compatible with machine learning (ML) models. Therefore they need to be converted to tensor form so that one can use it as input for the model. Mathematically, tensors are representations of scalar, vector and matrices in higher dimensions.

In the context of ML, tensors are multi-dimensional arrays which use matrices to represent. Similarly, in medical imaging these arrays store voxel intensities which are actually a cumulation of three-dimensional pixels within an image volume. Converting image format into tensors help in pre-processing steps like resizing and normalization as they improve model performance. Also tensor conversions allow seamless integration of the ML model and input pipeline.

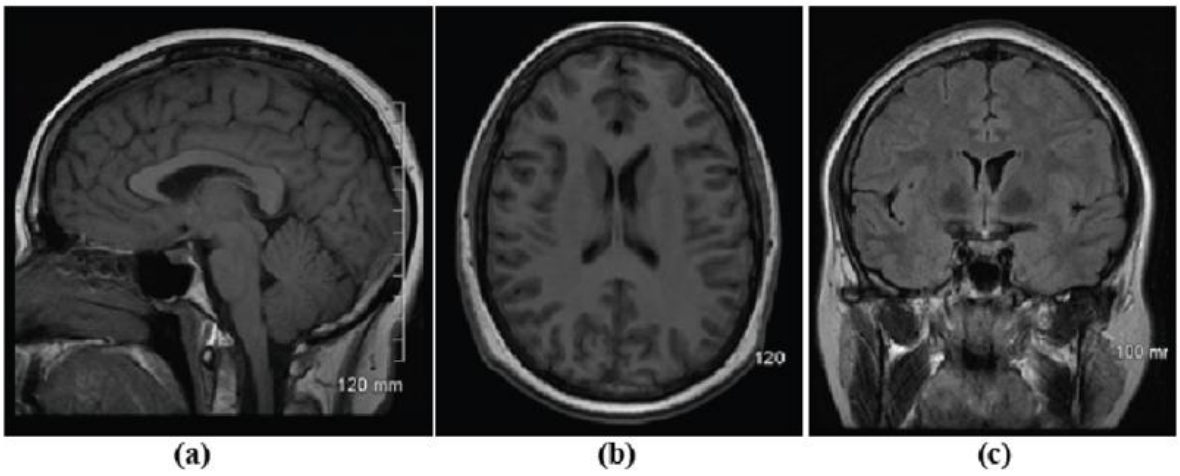


Figure 2.1: Axial View (a), Sagittal View (b), Coronal View (c)

2.1.3.1 3D Slice Synthesis

H. Ramy. et. al [1] employs a 3D CNN with attention mechanisms where the input to the network is a combination

of three-dimensional (3D) scans from T1w, T2w and other perfusion MRI scans. Similarly in [7], a dataset containing 3D whole body MRI/PET images have been taken for training a 3D residual UNet.

2.1.3.2 2D Slice Synthesis

[20] takes 680 subjects of MRI and PET images from ADNI and picks the 40th slice of corresponding MRI and PET pairs. This is followed by a pre-processing step which is resizing.

2.2 Pre-processing and Data Management

Input data needs to be pre-processed or ready to train before feeding the ML model. Therefore preprocessing techniques like normalization and resizing have to be used.

2.2.1 Normalization

Images are normalized by using methods like intensity normalization, in-max and spatial normalization. In the context of medical imaging, different image modalities result in scans having various intensity levels. Thus there is a need to scale them up or down by setting a range [0, 1] or [-1, 1] to have a balance intensity level. Spatial normalization is mapping of MRI and PET to a reference space that involves warping of gray matter into a stereotaxic space. [21]

2.2.2 Resizing

Resizing is changing the dimensions that are width and height of an image in order to train the machine learning model. For example in [20] each pre-processed image was resized into a 128x128 image and in [19] the images are resized to 96x96x48.

2.3 Types of Networks Used in Synthesis

2.3.1 Convolutional Neural Network (CNN)

CNN consists mainly of three layers: convolutional layer, pooling layer and a fully connected layer. In the convolutional layer a filter is usually applied to input images to get a feature map. Pooling layers reduce the volume size by down sampling the feature maps, therefore lowering further computations. Finally a fully connected layer propagates the input from the previous layer and performs classification..

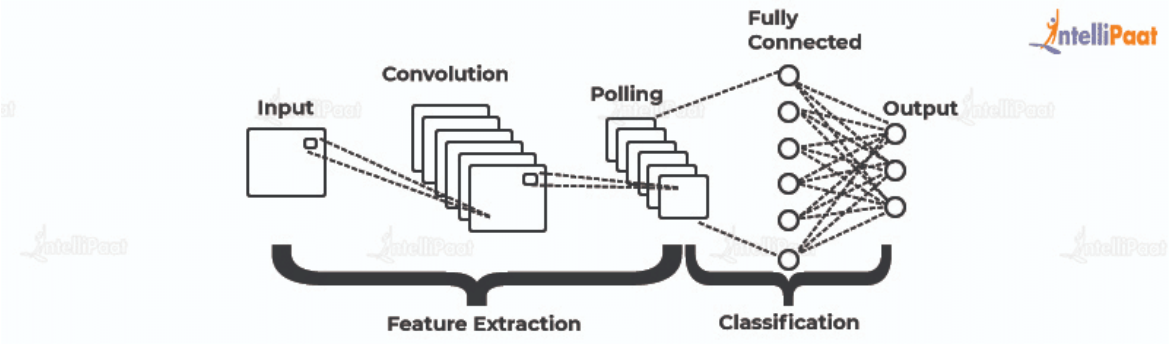


Figure 2.2: CNN Architecture

2.3.2 UNET

UNET is a U-shaped network architecture that is basically designed to work on fewer training examples. It consists of an encoder block, decoder block, and skip connections. The encoder block includes a convolution function followed by an activation layer and then a pooling layer which reduces the processing time. The decoder is the complete opposite of an encoder block because it up sample the feature maps and combines all the details from the previous layers of the neural network to give a refined image. Skip connections work on the improvement of the output image by comparing it with the input.

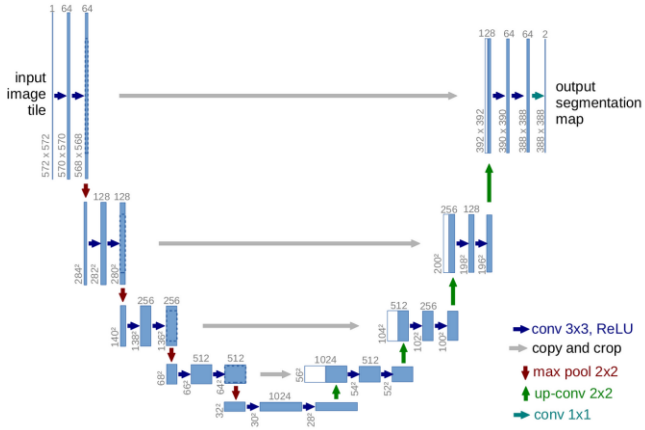


Figure 2.3 UNet Architecture

2.3.3 Generative Adversarial Network (GAN)

This concept was proposed first in 2014 by Goodfellow [22] where he described them as deep neural networks generating real samples. It consists of two main components: a generator and a discriminator. The generator generates fake samples based on the real input samples while the discriminator differentiates between those fake and real ones produced by the generator. The generator wants to fool the discriminator into thinking a fake sample is real. This is done using backpropagation, where the generator maps a random distribution into one that matches the distribution of real data, and the discriminator evaluates the differences between the fake and the real data distributions, with the error being back propagated to the generator if the discriminator finds that the sample is fake. After some time, the two models will reach a balance, with the generator producing images that look extremely realistic.

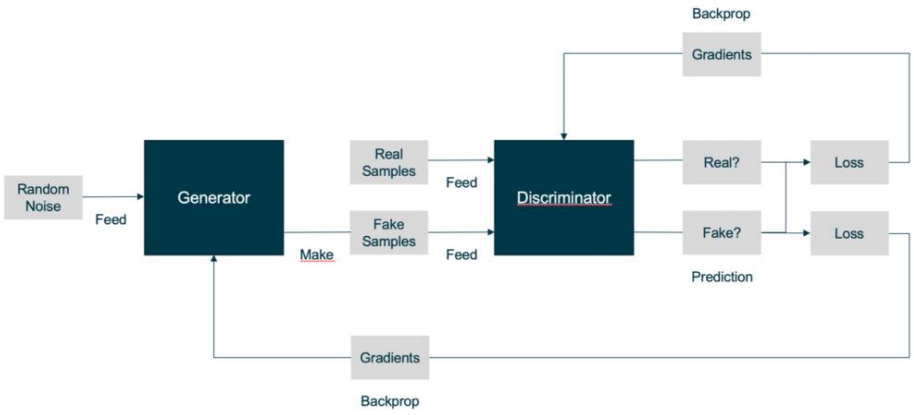


Figure 2.4 GAN Architecture

2.3.4 Super Resolution Convolution Neural Network (SRCNN)

SRCNN is an innovative approach developed by Dong et al. [23] which deals with the single image super-resolution problem. It is a three-layer convolutional neural network that functions by mapping an input low-resolution image into the target high-resolution image. The input image is up scaled into the desired size through bicubic interpolation, which again is considered the initial low-resolution input (Y). The network deals with three main operations: patch extraction and representation, non-linear mapping, and reconstruction. Patch extraction is the first layer of convolutional processing where overlapping patches are extracted from the input image and represented as high-dimensional vectors. Every high-dimensional vector in the second layer of the convolutional operation is mapped into another such vector, representing high-resolution patch vectors. The last convolution layer pools in high-resolution patch representations to derive a final high-resolution image. The output of the non-linear mapping is pooled together to reconstruct a coherent high-resolution image that should be as close as possible to the high-resolution ground truth image, denoted as X.

3. METHODOLOGY

The aim of our project 'AI-Driven PET-like Synthesis from MRI Data' is to synthesize PET-like images from MRI input (T1 MRI and Flair MRI). For this, we have used two AI models – a custom GAN called Pix2Pix and a modified SRCNN called Fast Medical Image Super Resolution (FMISR) Network. Furthermore, to lay the groundwork for future work regarding pattern recognition for objects such as tumors, feature verification etc. a DSP Kit and FPGA have been employed. Both hardware units employ similar computer vision (CV) techniques to perform this task. Finally, the code synthesizing images from MRI scans has been implemented on a Raspberry Pi unit as a physical embodiment.

3.1 Dataset

For our project, we are using the dataset called 'CERMEP-IDB-MRXFDG: a database of 37 normal adult human brain [18F]FDG PET, T1 and FLAIR MRI, and CT images available for research[23].

3.3.1 Data Acquisition

Our project deals with medical images that are obtained from specific machines – The MRI machine and PET scanner. We required a unique dataset, comprising of T1 MRI, Flair MRI, and PET scan of a group of patients, all done in a short time – within our of each other. This kind of data is very hard to come by, as it requires the use of expensive resources and machinery.

After extensive research and exploring various avenues, we found an article online 'CERMEP-IDB-MRXFDG: a database of 37 normal adult human brain [18F] FDG PET, T1 and FLAIR MRI, and CT images available for research. This article describes a database of 37 patients whose T1 MRI, Flair MRI, and PET scans were taken on the same day. The authors of this article include medical experts, and they collected this data specifically for research purposes. We contacted Mr. Inés Mérida, one of the authors, and he kindly provided us with the dataset.

The data was already passed through a process of anonymization and pre-processing. The pre-processing techniques applied were coregistration, spatial normalization, and intensity normalization. Once, we successfully obtained the dataset, we were able to view the scans in various orientations, like sagittal, coronal, and axial, using the 3D slicer. We were also able to view individual slices of each scan.

3.1.2 Data Pre-Processing

For our project, we have converted the MRI and PET scans from NIfTI format to 2D tensors.

3.1.2.1 NIfTI to Tensor Conversion

The MRI scans and the PET scans in the acquired dataset are in NIfTI format. The **NIfTI (Neuroimaging Informatics Technology Initiative)** format is a file format that is globally used for storing medical images. It is used for medical image analysis. It can be easily used in various software as many software have libraries for it [24], but it is not suitable for training and testing our AI models. Therefore, we need to convert the images from NIfTI format to Tensors to make the training and testing pipelines for the two AI models.

Mathematically, tensors are representations of scalar, vector, and matrices in higher dimensions. In the context of ML, tensors are multi-dimensional arrays that use matrices to represent complex data . Similarly, in medical imaging, these arrays store voxel intensities which are a cumulation of three-dimensional pixels within an image volume. Converting image format into tensors helps in pre-processing steps like resizing and normalization as they improve model performance. Also, tensor conversions allow seamless integration of the ML model and input pipeline.

3.1.2.2 Slice Extraction and Normalization

The data we acquired was already pre-processed to some extent. We received the data in NIfTI (Neuroimaging Informatics Technology Initiative) format. To process the images, we extracted the center slices of all images and converted them into 2D tensors. We then add a third dimension to convert the data to 3D tensors. We extract the center slice of all the T1 MRI, Flair MRI, and PET scans and normalize the tensors to the range [0, 1]. Next, we concatenate the T1 MRI tensors and the Flair MRI tensors of the same patients to form a single input tensor for all the patients. The PET tensors are saved to be fed to the models as ground truth.

This way, we have made the training and testing pipelines after batching and shuffling the data for our models, mainly using the **Nitabel**, **NumPy,** and **Tensorflow** libraries. We use this method for both AI models used in our project.

3.1.3 Training and Testing Sets

Our dataset consists of 37 patients in total and we have 3 scans of each patient. We divide our dataset into training and testing sets, keeping a 70 - 30 ratio. This means that there are 26 patients in our training set and 11 patients in our testing set.

3.2 Project Software

After the dividing the data into training and test sets and applying the required pre-processing techniques, our data is ready to be used in the AI models. We use two AI models in our project: a conditional Generative Adversarial Network (cGAN) called Pix2Pix and a modified Super Resolution Convolutional Neural Network (SRCNN) called Fast Medical Image Super Resolution (FMISR) Network. These models are concatenated with each other and they serve different purposes in the project that allow us to successfully achieve our goal of enhanced synthesis of PET-like images.

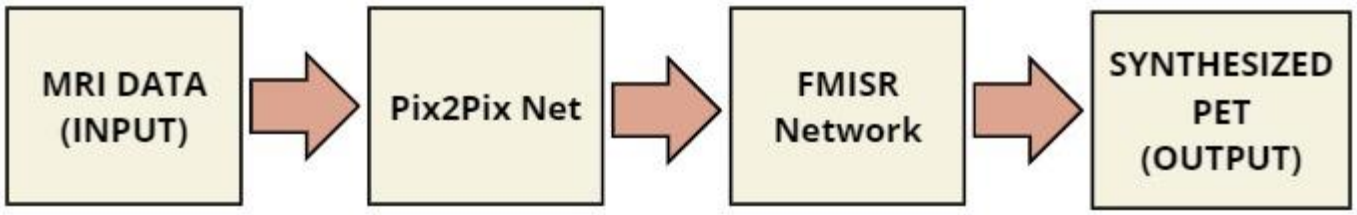


Figure 3.1 Block Diagram of Proposed Model

The input MRI data is passed through the Pix2Pix Net and the output is then fed to the FMISR, from which we receive the final output of improved synthesized PET-like images. Both AI models have been explained below in Chapter 4.

3.3 Hardware

Due to various laws worldwide regarding medical images and patient privacy a need arises to keep and process as much data as possible locally or in house. On the other hand sophisticated AI driven models require dedicated hardware and constitute large numbers of computation that cannot be easily sustained on the average workplace system. However, such models deliver immense benefits in the form of medical imaging and diagnostic aids. Such systems and setup provide tools such as Image synthesis, pattern/feature detections, image post processing and predictions based on symptoms. All in all, AI based medical tools allow access to patterns and information that would have been overlooked or not even found by the average medical practitioner. Doctors have and will continue to require such systems to more accurately diagnose and treat patients. Thus, a problem arises, how to integrate such systems and models into already established medical ecosystems

As a solution, three (3) hardware units are presented herein, each of which embody a processing step within the workflow using entirely localized systems.

Of the three (3) hardware units, 2 provide a post-processed pattern detection software, Both the FPGA and DSP Kit based systems have the same application running on them with only some minor hardware based differences in the implementation.

4. AI MODELS

We use two AI models for the software part of our project– a conditional Generative Adversarial Network (cGAN) called Pix2Pix and a Super Resolution Convolutional Neural Network (SRCNN), based on a deep learning network. We have mainly used Google Colab to write our code in Python, using **Tensorflow** and **Keras** libraries.

The training and testing pipelines are designed similarly for both models, to maintain uniformity, as explained in Section 3.2. However, these two models have different functionalities that help us attain the goal of our project.

4.1 Conditional Generative Adversarial Network (cGAN): Pix2Pix

Pix2Pix is a conditional Generative Adversarial Network that uses real data, noise, and labels to generate images. It is meant for image-to-image translation tasks in which it takes input data as a guide to synthesize images to achieve an output close to the target images. The Pix2Pix Net has two main components: the generator and the discriminator.

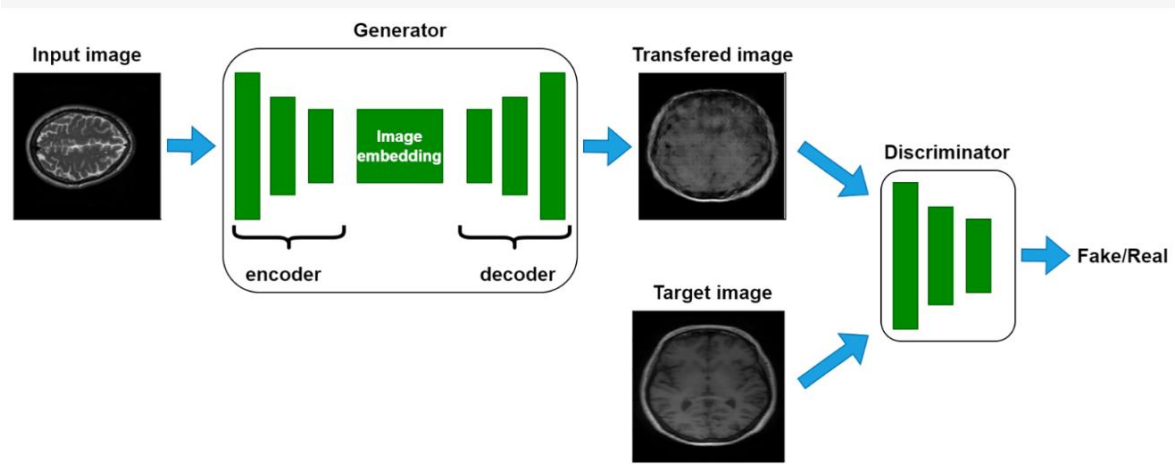


Figure 4.1 Generator Architecture

4.1.1 Generator

The generator of the Pix2Pix is a modified UNet, which is responsible for generating the images by learning from the real data and noise. The generator mainly has three parts, the encoder (down sampler) and decoder (up sampler) and skip connection.

Encoder (down sampler): The encoder (down sampler) consists of convolutional layers that take out critical features from input data. They convolve the input feature maps with filters whose purpose is to identify different patterns, textures, and features in the input data. There is a non-linear activation function (ReLU - Rectified Linear

Unit), after each convolutional layer. The ReLU activation function is used because it brings non-linearity to our model which is good for model optimization. The down sampling layer is also trivial for the spatial reduction of the feature maps. To reduce the dimensions, it uses convolutional layers with large strides and max pooling techniques. The down sampler also increases the number of channels which allows our model to be able to extract more diverse and abstract features and map complex relationships between the input data and the features.

Decoder (Up sampler): The decoder is the complete opposite of the encoder. It consists of up sampling layers which increases the spatial dimensions so that the output image is built from the encoder features. Batch Normalization Layers used in this block help in making the computations faster. Here also, the same activation function (ReLU) is used. Convolutional Layers aid in improving the image features and give a good-resolution image.

Skip connections: Skip connections are essential in linking the encoder and decoder as they have to concatenate the input features from the encoder to the decoder. They retain the fine details of input images, resulting in a better-looking generated output.

4.1.2 Discriminator

It eventually aims to differentiate between real and fake data by picking up features from the concatenated input images a producing a single-channel image. The discriminator takes input data and target data as its input with equal dimensions. They are further concatenated resulting in such a tensor shape that the input data and target data are two channels of the output.

Encoder (Down sampler): The concatenated inputs are fed to three down sampling layers having an ascending order i.e. 64,128 and 256. They each have a convolutional layer with an activation function and batch normalization.

Zero Padding: Zero padding is like adding extra dimensions to the input image. Here this technique is applied two times before and after convolutional layers so that the dimensions of the convolution layer match with that of the next layer.

Output Layer: The Final layer represents that a convolution has been applied on the previous zero padded layer with a filter and stride of 1 to get the discriminator's output.

4.2 Loss Functions

We have defined the loss function of the generator and discriminator separately.

Generator Loss: The generator's loss function has three components. The Binary Cross-Entropy Loss (GAN Loss) expects raw input and it works by calculating the difference between the predicted probability distribution and the target distribution. The Mean Absolute Loss (L1 Loss) subtracts the predicted output from the target image and then takes the mean of this difference. The third component is the total loss, which is the sum of the other two losses, multiplied by lambda, which controls the importance of the GAN and L1 loss.

Discriminator Loss: The discriminator's loss has three components as well. The real loss calculates the binary cross-entropy loss for the discriminator's real output images and the target distribution. The generated loss is the difference between the discriminator's fake output images and the target distribution. The total loss is the sum of the real loss and the generator loss.

4.3 Model Training

Finally, we train our Pix2Pix model. We feed the concatenated MRI images as input and the real PET scans as ground truth. The model is trained over 30000 steps and the resultant predicted PET-like images closely resemble the target data. The training process is long and iterative. In the beginning of the training process, the output looks nothing like the ground truth. The image constructs slowly as the model learns from the testing set and finally, we get an output image that closely resembles the target image.

4.3 Model Testing

After the training process has been completed, we save the trained model in our google drive. It is now ready for testing. We use our test set of 11 patients to test the trained Pix2Pix model.

4.4 Evaluation Metrics

The evaluation metric we are using for the Pix2Pix Net is Structural Similarity Index Measure (SSIM) which shows how similar two images are in terms of structure and contrast.

4.5 Modified Super Resolution Convolutional Neural Network (SRCNN): Fast Medical Image Super Resolution (FMISR) Network

Fast Medical Image Super Resolution (FMISR) Network is modified version of Super Resolution Convolutional Neural Network (SRCNN). This means that the hidden layers incorporated in the FMISR work on the same principle as an SRCNN. These hidden layers perform feature extraction. The FMISR gives better performance in image reconstruction as compared to the traditional SRCNN. We have chosen to implement the FMISR model in order to enhance the images synthesized by the pix2pix.

The goal of using FMISR is to improve the synthesized PET-like scans. This will make the images resemble the real PET scans as much as possible, aiding doctors in making timely diagnosis. Following is the general architecture of FMISR:

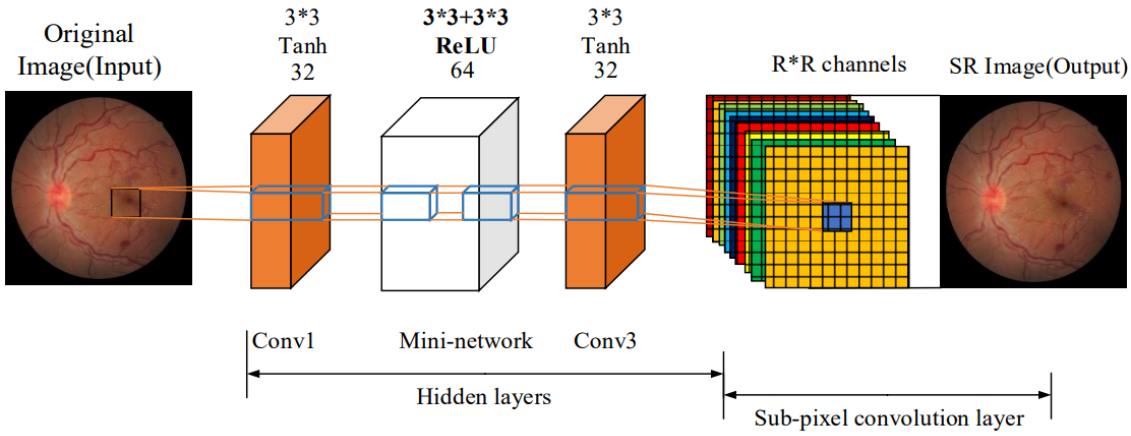


Figure 4.2 FMISR Architecture

4.5.1 Hidden Layers

The FMISR model has 3 hidden layers which are responsible for extracting features from the input data. The hidden layers involve 2 convolutional layer, each followed by batch normalization. The third layer is the mini network that is incorporated in the hidden layers. The two convolutional layers conv1 and conv2 both have 64 filters with kernel size 3x3. The middle layer is the mini network, which is explained in section 4.5.2. The architecture in the figure (above) is derived from the paper [25]. We have modified our model to have the Parametric Rectified Linear Unit (PReLU) to introduce non-linearity.

4.5.2 Mini Network

The mini network constitutes of two convolution filters with 64 filters of kernel size 3x3, after which both outputs are passed through normalization layers and the activation function PReLU. The result is low level features extracted from the input data.

4.5.3 Sub-Pixel Convolutional Layer

After the hidden layers and mini network, the sub-pixel convolutional layer is applied. The purpose of this layer is to scale up the low resolution feature map of the input data. Finally, the PReLU activation function is applied again. Overall, we used 128 filters to capture details.

4.5.4 Down sampling and Up sampling

We are also using the maxpooling of stride 2 to reduce the spatial dimensions of the feature maps. Then, we are up sampling them using subpixel convolution to increase the resolution of the output images.

4.5.5 Loss Functions

We are using a custom loss function caled perceptual loss function. This is typically used to calculate the difference between the feature maps of images. In this case, it is used to calculate the difference between the features of the enhanced output and the real PET images.

4.5.6 Model Training

We are training the model by feeding it the low resolution synthesized PET-like images in our training set after passing the through Pix2Pix Net as input and the real PET scans as target. Since the FMISR learns quickly, it only requires 80 epochs.

4.5.7 Model Testing

We are testing the model by feed it the images in our test set after passing them through the Pix2Pix Net as input and applying some post-processing. The outputs are enhanced PET-like images that closely resemble the real PET scans.

4.7.8 Evaluation Metrics

To evaluate the model performance of FMISR, we are using Peak Signal-to-Noise (PSNR). It measures the quality of the reconstructed image in comparison to the original.

5. HARDWARE

5.1 General Architecture of Pattern Search

The general workflow or flowchart design of the application describes a system to find the number of times a given pattern occurs in a provided image.

The steps are detailed in the provided figure and below:

1. Laplacian Layer: An input image of size 128x128 is convolved with a Laplacian Filter to get a matrix of 126X126 dimensions. Similarly, an input pattern of size 8x8 is convolved with the Laplacian Filter to get a matrix of 6X6.This layer enhances edges and appoints values to regions depending on their change of intensity.

2. Convolution Stage 2: The outputs of the Laplacian Layer are convolved (the 126x126 matrix and the 6x6 matrix) together to provide a 121x121.This layer compares the patterns or the features extracted from the two images.

3. Max Pooling: The output matrix from the second convolution layer is subjected to max pooling with a factor of 2. Here we reduce the dimensionality of the matrix while retaining the most significant features.

4. Thresholding and Counting: Finally, the elements of the pooled matrix are thresholded. Each element is compared with a threshold value, which is determined as a percentage (25%) of the maximum value obtained from convolving the test pattern with itself.

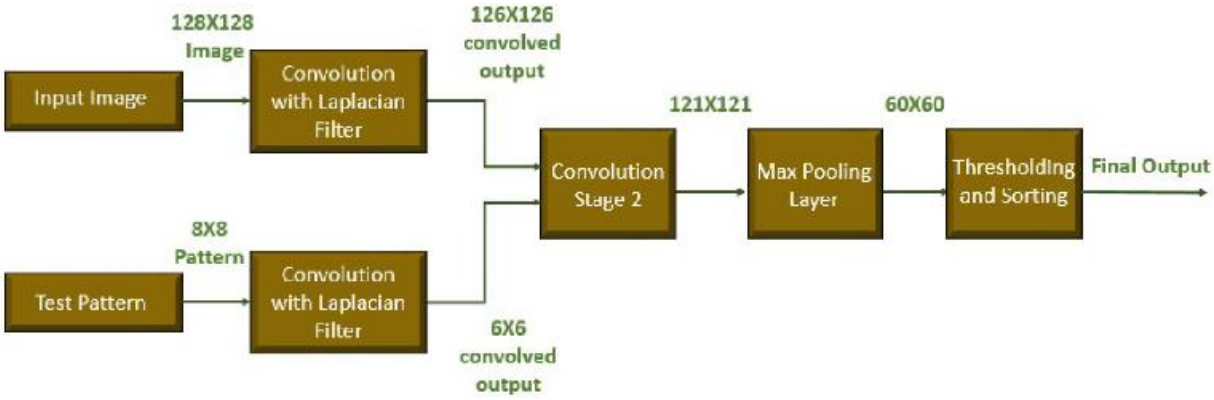


Figure 5.1 Flowchart

5.2 FPGA

Field Programmable Gate Arrays are developer tools that allow for the designing and modeling of custom digital circuits on a hardware level. Unlike other development tools FPGAs offer great flexibility and adaptation to changing requirements. Thus, they are a perfect hardware unit to mimic an IC that could be integrated into any user workflow.

Furthermore, by isolating the entire computational system to an external device we avoid any risks of the user system/personal computer not being up to specifications.

Specific to our application, we have implemented the CNN based pattern detection application which uses general Computer Vision based design choices to extract and count the occurrence of a provided pattern. In our testing, the FPGA based CNN gave similar accuracies to generic CPU and GPU based setups and slightly better performance and computational times due to its dedicated nature. The design layout largely follows the provided flowchart in figure. An important point to note is that both input images have been collapsed to a singular dimension in binary.

5.2.1 Circuit Layout

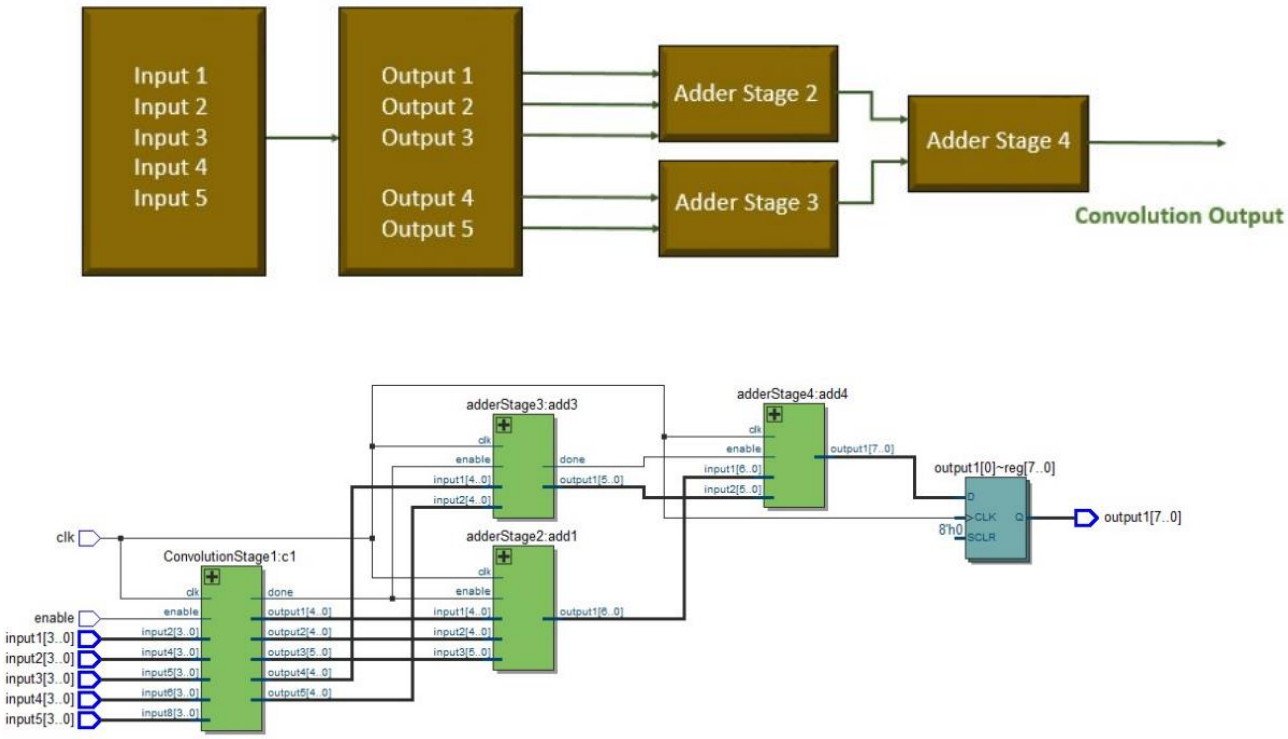


Figure 5.2 Convolutional Circuit

Above block and circuit diagrams represent the pipelined implementation of the first stage of convolution of input image and test pattern with Laplacian filter. Each convolution of a 3X3 image/pattern pixel grid with the Laplacian filter is completed in 4 clock cycles. The operation of convolution has been divided into 4 stages, which helps in pipelined implementation of the convolution operation. Because of pipelining, an output is obtained at each clock cycle.

In the pipelined implementation, same numbered rows are given as input to the multiplication block, which gives 6 output and addition is also performed in a pipelined manner. Overall, a 6X6 convolution operation takes 5 clock cycles, because of use of pipelining and hardware parallelism. Once an output is obtained, each clock cycle produces one new output.

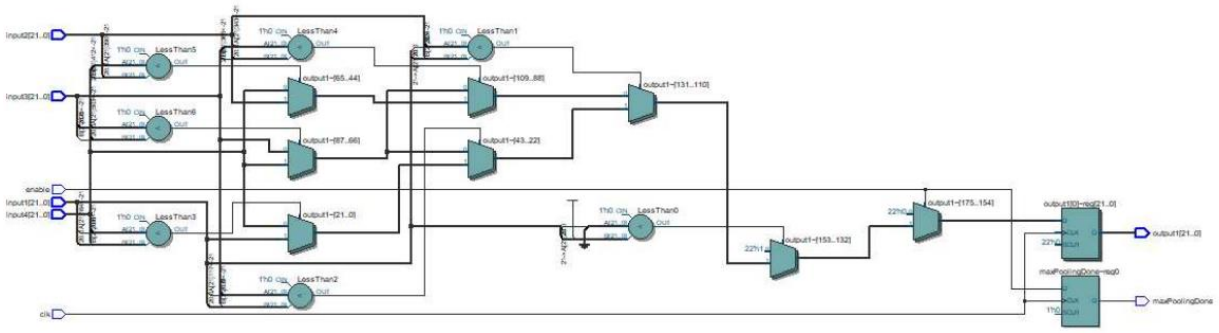


Figure 5.3 Max-Pooling Circuit

A conventional Max Pool algorithm has been employed into Hardware Design Language (HDL), in particular, in Verilog. In essence, four (4) twenty-two (22) bit inputs are considered and the output is the input with the maximum value.

In the final layer, instead of sorting and then comparing the values for finding the number of times a pattern appears in the image, direct comparison is done to avoid the time-consuming activity of sorting.

5.2.2 Resources and Performance

Using an Intel/Altera Cyclone V as our baseline hardware unit, a synthesis of the design was performed which yielded total block memory bits of 86,000. ModelSim was used to successfully validate the encoded design and Quartus was used to synthesize the circuit and its post mappings.

In terms of performance, the table below details the clock cycles per process.

For all stages	37722
For Convolution of Image (128X128) with Laplacian Filter (3X3)	15880
For Convolution of pattern (8X8) with Laplacian Filter (3X3)	400
For Convolution of convolved image (126X126) with convolved pattern (6X6)	14647
For Max Pooling Layer (121X121)	3602
For Thresholding (direct layer)	3600

The synthesis suggested a maximum clock speed of **155.67 MHz**

5.3 DSP Kit

DSP Kits provide for a higher level design option over FPGAs. While they sacrifice customizability and access to how specific the implementation is on a hardware level. However, this allows for a more accessible coding environment and easier access to future updates and changes. As a further proof of concept, the same application implemented on the FPGA has been translated and implemented and tested on a Texas Instruments DsK6713 DSP Kit.

The C code follows the same flowchart as the FPGA implementation except here the images are retained as 2 dimensional vectors. A custom library was imported and used to convert the images from their native png format to code readable binary.

The code will be provided in the appendix and not much further elaboration will be provided here as all core workings are in the same vein as the Verilog implementation.

5.4 Raspberry Pi 3 B+:

In the spirit of localizing as much of our system as possible a Raspberry Pi 3 has been used to store and run the MRI to PET synthesis model. We used a Linux OS in conjunction with a Python interpreter. All necessary libraries were imported using the pip command. Given that the model had already been trained, all that was left was to set up a pipeline and deploy it. The code for this embodiment is also available in the appendix.

With regards to performance, no major changes were noticed and the model behaved in the same manner as on the testing platform. The only drawback of this embodiment was the loss of cloud computing resulting in a longer processing time. Overall the results remained the same while allowing for entirely localized processing.

6. RESULTS

6.1 Project Results

Our project includes both a software setup and a hardware setup, and the results of both setups show good performance. In this chapter, we will discuss the qualitative and quantitative results we have derived from both the software and the hardware parts of the project. The overall performance of our two AI models is satisfactory and the hardware is showing expected results.

6.1.1 Software

In this section, we discuss the results of the software part of the project. This consists of two AI models: a conditional Generative Adversarial Network (cGAN) called Pix2Pix and a modified Super Resolution Convolutional Neural Network (SRCNN) called Fast Medical Image Super Resolution (FMISR) Network. Their results are discussed below.

6.1.1.1 Conditional Generative Adversarial Network (cGAN): Pix2Pix

Here is one patient output from the Pix2Pix Net that we obtained after testing. These are the scans of Patient 13.

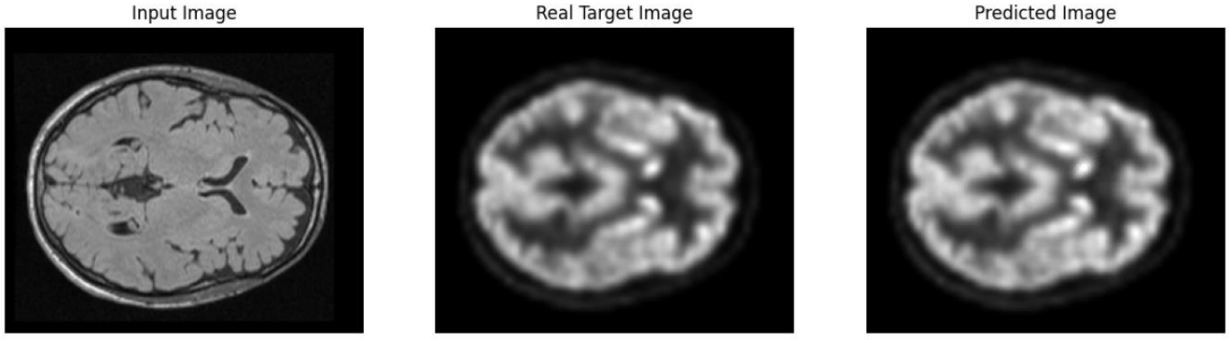


Figure 6.1 Pix2Pix Output

The Structural Similarity Index Measure (SSIM) we are getting is 0.98. This shows the while the synthesized PET-like scan is quite close to the real PEGT scan, there are still some dissimilarities in the contrast and structure. Visual inspection also validates our conclusion.

6.1.1.2 Modified Super Resolution Convolutional Neural Network (SRCNN): Fast Medical Image Super Resolution (FMISR) Network

Here is the one patient output from the FMISR that we have obtained after testing the model. These are the scans of patient 13.

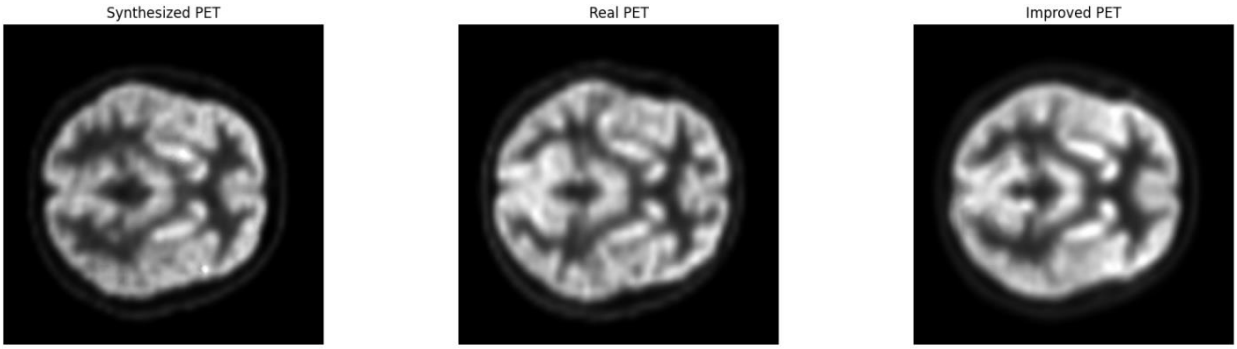


Figure 6.2 SRCNN Output

The Peak Signal - to - Noise Ratio (PSNR) we are getting is 22.24 dB. The PSNR of brain scans in [25] is 25.5dB. The little difference between these results, which leads us to believe that our results are satisfactory. Visual inspection also validates our conclusion. Therefore, we have been able to enhance the synthesized PET-like scans.

6.1.2 Hardware

In this section, we'll discuss the results achieved in the hardware part of our project. Our hardware includes FPGA, DSP Kit DsK6713, and Raspberry Pi B+. The results are discussed below.

6.1.2.1 FPGA

We successfully synthesized and validated post-timing/post-mapping of the Verilog code for the pattern detecting CNN on Quartus. Furthermore, all results were further validated using ModelSim.

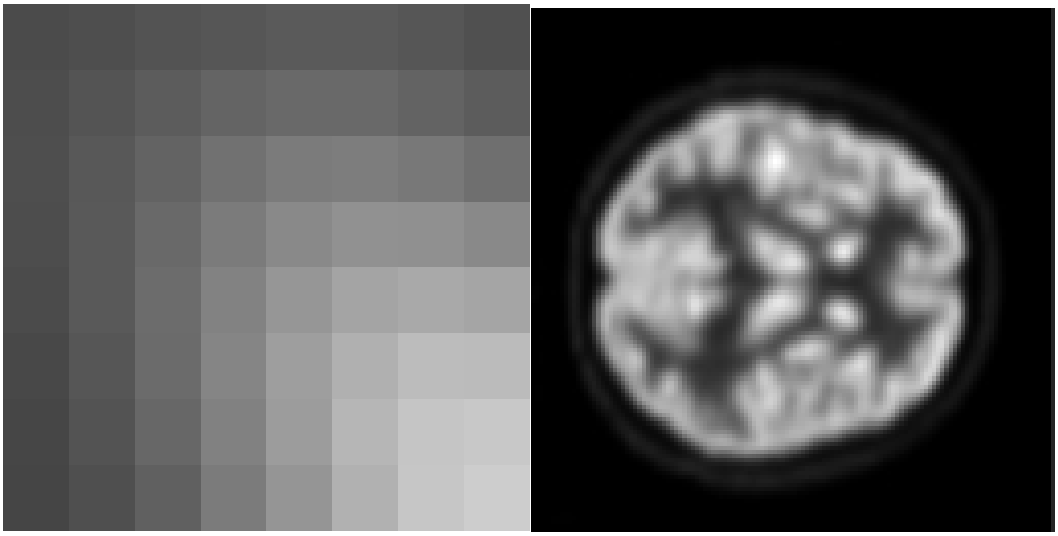
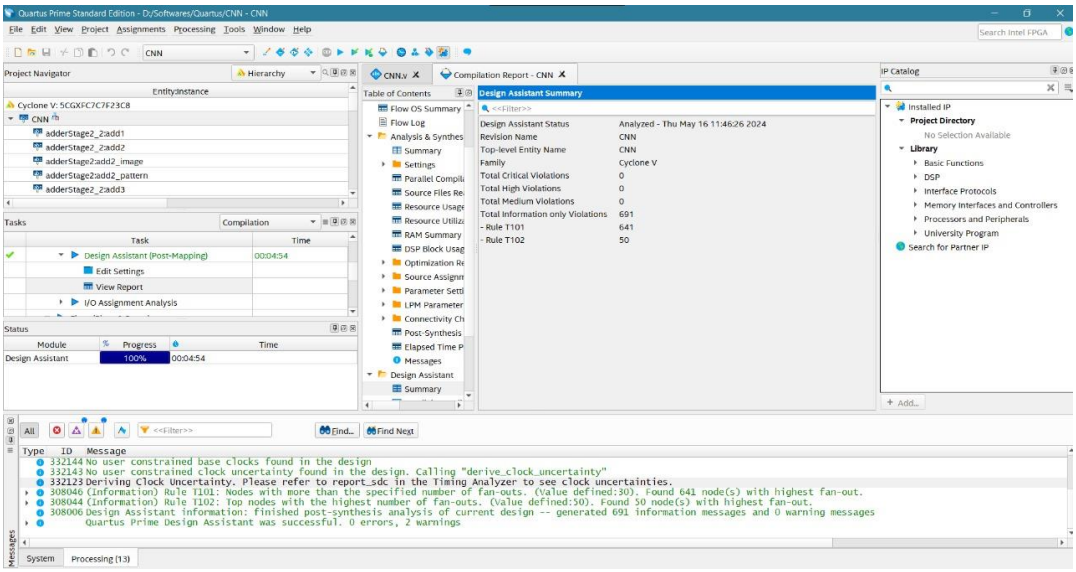


Figure 6.3: Pattern (a), Test Image (b)



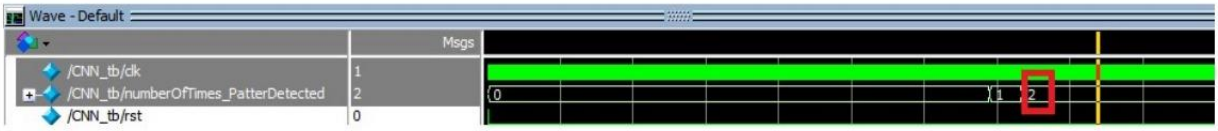


Figure 6.4 FPGA Synthesis and Verification

6.1.2.2 DSP KIT

Similar to the FPGA, the code was realized and validated through hardware implementation and emulations using Code Composer Studio and a DsK6713 kit.

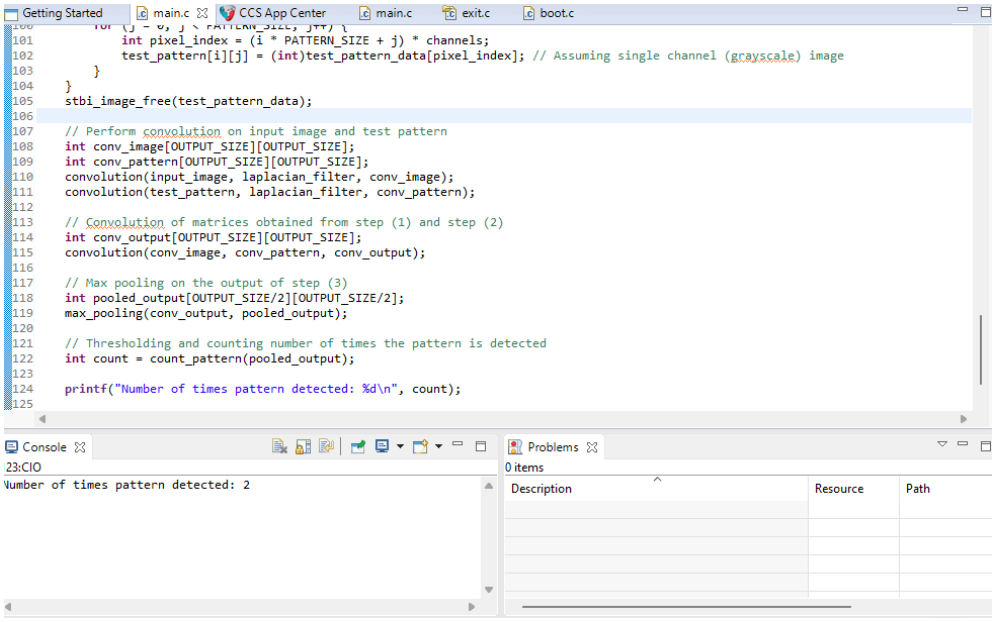


Figure 6.5 DSP KIT Output

6.1.2.3 Raspberry Pi

The code was hosted on the Raspberry Pi running a Linux OS and performed similar to the cloud-based server.

7. CONCLUSIONS, FUTURE WORK, EXPERT EVALUATION

7.1 Conclusions

Both the software and hardware aspects of our project are working as expected and we are receiving satisfactory results. The Pix2Pix Net is successfully in synthesizing PET - like images, which leads us to conclude that it is in fact possible to generate PET-like scans from MRI data alone. This is a good solution for patients who do not have the time to get on PET scan waiting lists, or for patients who simply cannot afford PET scans. These PET-like images will allow doctors to make quicker diagnoses and to get a better idea of what ailments their patients are suffering from. All this said, the PET-like scans cannot replace actual PET scans.

The FMISR is giving good results in enhancing the images synthesized by the Pix2Pix Net. Using these two AI models in concatenation is a novel technique that can bring significant innovation in the biomedical industry.

The results obtained from the hardware part of our project represent that we have achieved all the prerequisite goals required to successfully implement pattern detection in human brain scans. This is proof that future work can lead to exemplary results in this domain.

7.2 Future Work

In future work, some key points can be addressed to expand our project to discover further advancements. The following suggestions represent areas where improvement can take place:

1. In this report 37 subjects' MRI and PET scans are used as input and target data respectfully. Consider using a larger dataset for training and testing the models so that a much better efficiency is achieved. With a larger dataset, more complex models can be trained and overfitting can also be prevented.

2. Implementation of AI system setups to produce efficient results. Our project includes the output of pix2pix being fed to a modified SRCNN: FMISR which further works on image enhancement. Usage of other models in the future would result in much better-looking scans for PET synthesis.

3. Currently, the hardware portion of our project recognizes a pattern from a healthy brain scan. In the future, researchers can consider using pattern recognition techniques on tumor images so that they turn out to be beneficial to the doctors doing research.

7.3 Expert Evaluation

Our work deal with medical imaging, which is a very sensitive data. Therefore, it was our moral responsibility to get expert evaluation of our work. We are grateful to have gotten the opportunity to connect with Dr. Iqtedar Ahmed Muazzam, a medical oncologist working Queens Cancer Centre at Hull University Teaching Hospital's NHS Trust, East Yorkshire, England.

7.3.1 Mid-Progress Evaluation

Dr. Muazzam evaluated our project results while it was still in progress and gave the following remarks.

"I am a medical oncologist and work in Queens Cancer Centre at Hull University Teaching Hospital's NHS Trust. My job is to treat cancer patients with anticancer medicines and hence I use different imaging modalities for staging and response assessment. I had the opportunity to review the research project ----- which is designed to improve sensitivity of MRI scans to replace PET scans. It is a novel idea which will not only expand use of MRI scan but also provide more widespread use of high-quality imaging in developing countries where PET scan is not readily available. Considering how much medical decisions depend on the accurate imaging findings, this project has its challenges as well. It means that results need a robust criterion to validate the results. I had the privilege to review and discuss the blueprints of this project which provides me the confident on the material-and-methods parts as well the ethical background of the data collection and schema. It appears that the investigators are making serious effort to validate results using well recognized criteria. I am hopeful that the results will be reproducible if repeated in different population. I congratulate group of these young investigators who have thought of this innovative project and worked very hard to make it applicable for wider use. Like all research ideas, I would recommend that once we have final results and conclusions should be reviewed with caution and consider this research as an exploratory to test in bigger sample before its application in bigger patient population."

7.3.2 Results Evaluation

He later evaluated the final result of our AI models, and gave the following comments: "*I have the opportunity to review images 27 to 37. I can confirm that all the images are visible, clear and of good quality. The predicted images reflect anatomical and functional features presented in the corresponding real and target scans. In conclusion, predicted images represent the correct match and fulfil the study outcomes."*

After validating the results through evaluation metrics and expert evaluation, we can conclude that we have achieved what we set out to achieve.

REFERENCES

[1] R. Hussein *et al.*, "Brain MRI-to-PET Synthesis using 3D Convolutional Attention Networks," *arXiv.org*, Nov. 22, 2022. https://arxiv.org/abs/2211.12082

[2] Tribune, "Karachi's 20m population has access to only four PET scanners," *The Express Tribune*, Nov. 10, 2017. [Online]. Available: https://tribune.com.pk/story/1554360/karachis-20m-population-access-four-pet-scanners

[3] M. M. Jung, B. Van Den Berg, E. Postma, and W. Huijbers, "Inferring PET from MRI with pix2pix," *Tilburg University Research Portal*, 2018. https://research.tilburguniversity.edu/en/publications/inferring-pet-from-mri-with-pix2pix

[4] "PET scan vs. MRI: What's the difference - Baptist Health," *Baptist Health*. https://www.baptisthealth.com/blog/health-and-wellness/pet-scan-vs-mri-what-s-the-difference

[5] Mrimaster, "T1 vs T2 vs PD vs FLAIR MRI | T1 vs T2 vs PD vs FLAIR MRI image comparison," *Mrimaster*, Oct. 18, 2023. https://mrimaster.com/t1-vs-t2-vs-pd-vs-flair-mri/

[6] A. Ben-Cohen, E. Klang, S. P. Raskin, M. M. Amitai, and H. Greenspan, "Virtual PET Images from CT Data Using Deep Convolutional Networks: Initial Results," in *Lecture notes in computer science*, 2017, pp. 49–57. doi: 10.1007/978-3-319-68127-6_6.

[7] A. Rajagopal *et al.*, "Synthetic PET via domain translation of 3D MRI," *arXiv.org*, Jun. 11, 2022. https://arxiv.org/abs/2206.05618

[8] J. Kendrick, R. J. Francis, G. M. Hassan, P. Rowshanfarzad, J. S. L. Ong, and M. A. Ebert, "Fully automatic prognostic biomarker extraction from metastatic prostate lesion segmentations in whole-body [68Ga]Ga-PSMA-11 PET/CT images," *European Journal of Nuclear Medicine and Molecular Imaging*, vol. 50, no. 1, pp. 67–79, Aug. 2022, doi: 10.1007/s00259-022-05927-1.

[9] J. P. Leal *et al.*, "Automated lesion detection of breast cancer in [18F] FDG PET/CT using a novel AI-Based workflow," *Frontiers in Oncology*, vol. 12, Nov. 2022, doi: 10.3389/fonc.2022.1007874.

[10] Q. Yang, N. Li, Z. Zhao, X. Fan, E. I.-C. Chang, and Y. Xu, "MRI Cross-Modality Image-to-Image Translation," *Scientific Reports*, vol. 10, no. 1, Feb. 2020, doi: 10.1038/s41598-020-60520-6.

[11] K. Armanious *et al.*, "MedGAN: Medical image translation using GANs," *Computerized Medical Imaging and Graphics*, vol. 79, p. 101684, Jan. 2020, doi: 10.1016/j.compmedimag.2019.101684.

[12] H. Xie *et al.*, "Magnetic resonance imaging contrast enhancement synthesis using cascade networks with local supervision," *Medical Physics on CD-ROM/Medical Physics*, vol. 49, no. 5, pp. 3278–3287, Mar. 2022, doi: 10.1002/mp.15578.

[13] S.-I. Jang *et al.*, "TAUPETGeN: Text-Conditional TAU PET Image Synthesis based on Latent diffusion models," *arXiv.org*, Jun. 21, 2023. https://arxiv.org/abs/2306.11984

[14] T. Xie *et al.*, "Synthesizing PET images from High-field and Ultra-high-field MR images Using Joint Diffusion Attention Model," *arXiv.org*, May 06, 2023. https://arxiv.org/abs/2305.03901

[15] W. Wei *et al.*, "Predicting PET-derived demyelination from multimodal MRI using sketcher-refiner adversarial training for multiple sclerosis," *Medical Image Analysis*, vol. 58, p. 101546, Dec. 2019, doi: 10.1016/j.media.2019.101546.

[16] A. Sikka, Skand, J. S. Virk, and D. R. Bathula, "MRI to PET Cross-Modality Translation using Globally and Locally Aware GAN (GLA-GAN) for Multi-Modal Diagnosis of Alzheimer's Disease," 2021. https://semanticscholar.org/paper/MRI-to-PET-Cross-Modality-Translation-using-and-GAN-Sikka-Skand/7dfa3f6eab0f1c953a19712534b2bfa9d28c47b8

[17] J. Guo, E. Gong, A. P. Fan, M. Goubran, M. M. Khalighi, and G. Zaharchuk, "Predicting 15O-Water PET cerebral blood flow maps from multi-contrast MRI using a deep convolutional neural network with evaluation of training cohort bias," *Journal of Cerebral Blood Flow and Metabolism*, vol. 40, no. 11, pp. 2240–2253, Nov. 2019, doi: 10.1177/0271678x19888123.

[18] F. Bazangani, F. J. P. Richard, B. Ghattas, and E. Guedj, "FDG-PET to T1 Weighted MRI Translation with 3D Elicit Generative Adversarial Network (E-GAN)," *Sensors*, vol. 22, no. 12, p. 4640, Jun. 2022, doi: 10.3390/s22124640.

[19] H. Takita *et al.*, "AI-based Virtual Synthesis of Methionine PET from Contrast-enhanced MRI: Development and External Validation Study," *Radiology*, vol. 308, no. 2, Aug. 2023, doi: 10.1148/radiol.223016.

[20] "Cross-modality Synthesis from MRI to PET Using Adversarial U-Net with Different Normalization," *IEEE Conference Publication | IEEE Xplore*, Nov. 01, 2019. https://ieeexplore.ieee.org/document/9098219

[21] C. Davatzikos, "Computational neuroanatomy using shape transformations," in *Elsevier eBooks*, 2000, pp. 249–260. doi: 10.1016/b978-012077790-7/50021-7.

[22] N. Barla, "PiX2PiX: Key Model architecture Decisions," *neptune.ai*, Aug. 08, 2023. https://neptune.ai/blog/pix2pix-key-model-architecture-decisions

[23] "A fast medical image super resolution method based on deep learning network," *IEEE Journals & Magazine | IEEE Xplore*, 2019. https://ieeexplore.ieee.org/document/8471089

[24] I. Mérida *et al.*, "CERMEP-IDB-MRXFDG: a database of 37 normal adult human brain [18F]FDG PET, T1 and FLAIR MRI, and CT images available for research," *EJNMMI Research*, vol. 11, no. 1, Sep. 2021, doi: 10.1186/s13550-021-

APPENDIX

Code:

Pix2Pix Architecture

Here is the Pix2Pix model architecture:

Generator

```
import tensorflow as tf
from tensorflow import keras
from keras.layers import Input, Concatenate, Conv2DTranspose, Conv2D

def Generator():
    inputs = Input(shape=[207, 243, 1])  # Adjusted the input shape to
match your data

    down_stack = [
      downsample(64, 4, apply_batchnorm=False),  # (batch_size, 104, 122,
64)
      downsample(128, 4),  # (batch_size, 52, 61, 128)
      downsample(256, 4),  # (batch_size, 26, 31, 256)
      downsample(512, 4),  # (batch_size, 13, 16, 512)
      downsample(512, 4),  # (batch_size, 7, 8, 512)
      downsample(512, 4),  # (batch_size, 4, 4, 512)
      downsample(512, 4),  # (batch_size, 2, 2, 512)
    ]

    up_stack = [
      upsample(512, 4, apply_dropout=True),  # (batch_size, 4, 4, 1024)
      upsample(512, 4, apply_dropout=True),  # (batch_size, 8, 8, 1024)
      upsample(512, 4, apply_dropout=True),  # (batch_size, 16, 16, 1024)
      upsample(512, 4),  # (batch_size, 32, 32, 1024)
      upsample(256, 4),  # (batch_size, 64, 64, 512)
      upsample(128, 4),  # (batch_size, 128, 128, 256)
      upsample(64, 4),   # (batch_size, 256, 256, 128)
    ]

    initializer = tf.random_normal_initializer(0., 0.02)
    last = Conv2DTranspose(1, 4, strides=2, padding='same',  # Changed the
number of output channels to 1
                           kernel_initializer=initializer,
                           activation='tanh')  # (batch_size, 512, 512, 1)

    x = inputs

    # Downsampling through the model

    skips = []
```

```
            for down in down_stack:
                x = down(x)

        skips.append(x)

    skips = reversed(skips[:-1])

    # Upsampling and establishing the skip connections
    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = x[:, :skip.shape[1], :skip.shape[2], :]  # Crop x to the shape of
skip
        x = Concatenate()([x, skip])

    x = last(x)



    # Add a final Conv2D layer to adjust the dimensions
    final = Conv2D(1, 1, strides=1, padding='same',
                   kernel_initializer=initializer,
                   activation='tanh')  # (batch_size, 208, 244, 1)

    x = final(x)

    # Resize the output tensor to the desired shape
    x = tf.image.resize(x, [207, 243])

    return tf.keras.Model(inputs=inputs, outputs=x)
```

Discriminator

```
def Discriminator():
  initializer = tf.random_normal_initializer(0., 0.02)

  inp = tf.keras.layers.Input(shape=[207, 243, 1], name='input_image')
  tar = tf.keras.layers.Input(shape=[207, 243, 1], name='target_image')

  x = tf.keras.layers.concatenate([inp, tar])  # (batch_size, 207, 243,
channels*2)

  down1 = downsample(64, 4, False)(x)  # (batch_size, 104, 122, 64)
  down2 = downsample(128, 4)(down1)  # (batch_size, 52, 61, 128)
  down3 = downsample(256, 4)(down2)  # (batch_size, 26, 31, 256)

  zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)  # (batch_size, 28, 33, 256)
  conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                kernel_initializer=initializer,
                                use_bias=False)(zero_pad1)  # (batch_size, 25,
30, 512)
```

```
batchnorm1 = tf.keras.layers.BatchNormalization()(conv)


leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)
```

```
  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)  # (batch_size,
27, 32, 512)

  last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                kernel_initializer=initializer)(zero_pad2)
# (batch_size, 24, 28, 1)

  return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

FMISR Architecture

Here is the FMISR Architecture

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Activation, UpSampling2D,
MaxPooling2D, BatchNormalization, PReLU

def subpixel_conv_layer(x, scale=2):
    # Sub-Pixel Convolution Layer (ESPCN-inspired)
    x = Conv2D(filters=128, kernel_size=3, padding='same')(x)  # Increase
filters to capture more details
    x = PReLU()(x)  # Use PReLU activation
    x = Conv2D(filters=128, kernel_size=3, padding='same')(x)  # Increase
filters to capture more details
    x = tf.nn.depth_to_space(x, scale)  # Rearrange feature maps
    return x

def mini_network(x):
    # Mini-Network
    x = Conv2D(filters=64, kernel_size=3, padding='same')(x)  # Increase
filters to capture more details
    x = BatchNormalization()(x)
    x = PReLU()(x)  # Use PReLU activation
    x = Conv2D(filters=64, kernel_size=3, padding='same')(x)  # Increase
filters to capture more details
    x = BatchNormalization()(x)
    x = PReLU()(x)  # Use PReLU activation
    return x

def srcnn_model(input_shape):
    inputs = tf.keras.Input(shape=input_shape)
    hidden_layers = mini_network(inputs)
    sr_output = Conv2D(filters=32, kernel_size=3,
padding='same')(hidden_layers)
    sr_output = BatchNormalization()(sr_output)
    sr_output = PReLU()(sr_output)  # Use PReLU activation
```

```
        sr_output = Conv2D(filters=16, kernel_size=3,
    padding='same')(sr_output)

        sr_output = BatchNormalization()(sr_output)
        sr_output = PReLU()(sr_output)  # Use PReLU activation
        downsampling_layer = MaxPooling2D(pool_size=(2, 2))(sr_output)



        upsampling_layer = subpixel_conv_layer(downsampling_layer)  # Apply
    subpixel convolution to increase resolution
        final_output = Conv2D(filters=1, kernel_size=3,
    padding='same')(upsampling_layer)  # Remove activation here
        final_output = PReLU()(final_output)  # Use PReLU activation
        model = tf.keras.Model(inputs, final_output)
    return model

# Example usage for grayscale images:
input_shape = (256, 256, 1)  # Adjust based on your image size and channels
model = srcnn_model(input_shape)
model.summary()
```

Project Pipleline:
Here is how we are making the training and testing pipelines and feeding them to the trained models.

```
import tensorflow as tf

BUFFER_SIZE = 52
BATCH_SIZE = 1

# Get lists of T1, FLAIR, and PET scan file paths
input_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/MNIold2/model_train'
, '*_input.png')))
target_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/MNIold2/model_train'
, '*_target.png')))

# Make sure the lists of T1, FLAIR, and PET scan file paths have the same length
assert len(input_files) == len(target_files)

# Create a dataset of tuples, where each tuple is (T1 path, FLAIR path, PET path)
for a single patient
file_paths_dataset_train = tf.data.Dataset.from_tensor_slices((input_files,
target_files))

# Define a function to load and preprocess the images
def load_and_preprocess_images(input_path, target_path):
    # Load the images
    input_image = tf.io.read_file(input_path)
    target_image = tf.io.read_file(target_path)
```

```
    # Decode the images
    input_image = tf.image.decode_png(input_image, channels=1)
    target_image = tf.image.decode_png(target_image, channels=1)


    # Convert the images to float32 and normalize to [0, 1]
    input_image = tf.image.convert_image_dtype(input_image, tf.float32)
    target_image = tf.image.convert_image_dtype(target_image, tf.float32)



    return input_image, target_image

# Apply the function to the dataset


file_paths_dataset_train =
file_paths_dataset_train.map(load_and_preprocess_images)

# Shuffle the dataset
file_paths_dataset_train = file_paths_dataset_train.shuffle(BUFFER_SIZE)

# Batch the dataset
file_paths_dataset_train = file_paths_dataset_train.batch(BATCH_SIZE)

# Prefetch the dataset
file_paths_dataset_train =
file_paths_dataset_train.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
import tensorflow as tf
import os

# Get lists of T1, FLAIR, and PET scan file paths for testing data
input_files_test =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/MNIold2/model_test',
'*_input.png')))
target_files_test =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/MNIold2/model_test',
'*_target.png')))

# Make sure the lists of T1, FLAIR, and PET scan file paths have the same length
assert len(input_files_test) == len(target_files_test)

# Create a dataset of tuples, where each tuple is (T1 path, FLAIR path, PET path)
for a single patient
file_paths_dataset_test = tf.data.Dataset.from_tensor_slices((input_files_test,
target_files_test))

# Apply the function to the dataset
file_paths_dataset_test = file_paths_dataset_test.map(load_and_preprocess_images)

# Batch the dataset
file_paths_dataset_test = file_paths_dataset_test.batch(BATCH_SIZE)
```

```
# Prefetch the dataset
file_paths_dataset_test =
file_paths_dataset_test.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
# Import necessary libraries
from keras.models import load_model
import tensorflow as tf
import matplotlib.pyplot as plt

# Load the model from the file in Google Drive
trained_generator =
load_model('/content/drive/MyDrive/saved_pix2pix_slices/trainedpix2pix.h5')

# Print a success message
print("The trained generator model was successfully loaded from
'trained_generator.h5' in your Google Drive")

# Initialize the error metrics
mse_total = 0
mae_total = 0
num_samples = 0

# Assuming that 'file_paths_dataset_test' is your test dataset
for input_image, target in file_paths_dataset_test:
    # Generate output image from the input image
    prediction = trained_generator(input_image, training=False)

    # Calculate the error metrics
    mse = tf.keras.losses.MeanSquaredError()
    mae = tf.keras.losses.MeanAbsoluteError()
    mse_total += mse(target, prediction).numpy()
    mae_total += mae(target, prediction).numpy()
    num_samples += 1

# Calculate the average error metrics
mse_average = mse_total / num_samples
mae_average = mae_total / num_samples

print("Average Mean Squared Error: ", mse_average)
print("Average Mean Absolute Error: ", mae_average)
```

```
import tensorflow as tf

BUFFER_SIZE = 52
BATCH_SIZE = 1

# Get lists of T1, FLAIR, and PET scan file paths
input_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
dcnn_train', '*_input.png')))
target_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
dcnn_train', '*_target.png')))
```

```
# Make sure the lists of T1, FLAIR, and PET scan file paths have the same length
assert len(input_files) == len(target_files)



# Create a dataset of tuples, where each tuple is (T1 path, FLAIR path, PET path)
for a single patient
file_paths_dataset_train = tf.data.Dataset.from_tensor_slices((input_files,
target_files))

# Define a function to load and preprocess the images

def load_and_preprocess_images(input_path, target_path):
    # Load the images
    input_image = tf.io.read_file(input_path)
    target_image = tf.io.read_file(target_path)

    # Decode the images
    input_image = tf.image.decode_png(input_image, channels=1)

    target_image = tf.image.decode_png(target_image, channels=1)

    # Convert the images to float32 and normalize to [0, 1]
    input_image = tf.image.convert_image_dtype(input_image, tf.float32)
    target_image = tf.image.convert_image_dtype(target_image, tf.float32)

    return input_image, target_image

# Apply the function to the dataset
file_paths_dataset_train =
file_paths_dataset_train.map(load_and_preprocess_images)

# Shuffle the dataset
file_paths_dataset_train = file_paths_dataset_train.shuffle(BUFFER_SIZE)

# Batch the dataset
file_paths_dataset_train = file_paths_dataset_train.batch(BATCH_SIZE)

# Prefetch the dataset
file_paths_dataset_train =
file_paths_dataset_train.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
# Import necessary libraries
from keras.models import load_model
import tensorflow as tf
import matplotlib.pyplot as plt

# Load the model from the file in Google Drive
trained_generator =
load_model('/content/drive/MyDrive/saved_pix2pix_slices/trainedpix2pix.h5')

# Print a success message
```

```
print("The trained generator model was successfully loaded from
'trained_generator.h5' in your Google Drive")

# Initialize the error metrics
mse_total = 0
mae_total = 0

nrmse_total = 0
psnr_total = 0
ssim_total = 0
num_samples = 0

# Assuming that 'file_paths_dataset_test' is your test dataset
for input_image, target in file_paths_dataset_train:

    # Generate output image from the input image
    prediction = trained_generator(input_image, training=False)

    # Calculate the error metrics
    mse = tf.keras.losses.MeanSquaredError()
    mae = tf.keras.losses.MeanAbsoluteError()
    mse_total += mse(target, prediction).numpy()
    mae_total += mae(target, prediction).numpy()

    #Calculate NRMSE
    max_target = tf.reduce_max(target)
    min_target = tf.reduce_min(target)
    nrmse = tf.sqrt(mse(target, prediction)) / (max_target - min_target)
    nrmse_total += nrmse.numpy()

    # Calculate PSNR
    psnr = 20 * tf.math.log(max_target / tf.sqrt(mse(target, prediction))) /
tf.math.log(10.0)
    psnr_total += psnr.numpy()

    #Calculate SSIM
    ssim = tf.image.ssim(target, prediction, max_val=max_target)
    ssim_total += ssim.numpy()

    num_samples += 1

# Calculate the average error metrics
mse_average = mse_total / num_samples
mae_average = mae_total / num_samples
nrmse_average = nrmse_total / num_samples
psnr_average = psnr_total / num_samples
ssim_average = ssim_total / num_samples

print("Average Mean Squared Error: ", mse_average)
print("Average Mean Absolute Error: ", mae_average)
print("Average NRMSE: ", nrmse_average)
print("Average PSNR: ", psnr_average)
print("Average SSIM: ", ssim_average)
```

```
import os
import matplotlib.pyplot as plt
import tensorflow as tf

# Define the path to the new folder where the images will be stored
new_output_folder = "/content/drive/MyDrive/trained_dcnn_slices/results_p2p"


# Create the new folder if it doesn't exist
if not os.path.exists(new_output_folder):
    os.makedirs(new_output_folder)

# Iterate through the test dataset and save images for each patient
for idx, (input_image, target) in enumerate(file_paths_dataset_train):
    # Generate output image from the input image

    prediction = trained_generator(input_image, training=False)

    # Resize the prediction to (256, 256)
    prediction_resized = tf.image.resize(prediction, [256, 256])

    # Create a unique folder name for each patient (starting from 1)
    patient_folder = f"sub_{idx + 1}"  # Adjust indexing here

    # Save the resized image

    plt.imsave(os.path.join(new_output_folder,
f"{patient_folder}_predicted.png"), prediction_resized[0, :, :, 0].numpy(),
cmap='gray')

print(f"Saved images for {len(file_paths_dataset_train)} patients to
{new_output_folder}")
```

```
import os
import tensorflow as tf
import numpy as np
from PIL import Image

# Path to the folder containing your images
folder_path = "/content/drive/MyDrive/trained_dcnn_slices/results_p2p"

# Get all image filenames in the folder with PNG extension
image_filenames = [f for f in os.listdir(folder_path) if f.endswith('.png')]

# List to store the image tensors
image_tensors = []

# Read and convert each image into a tensor
for filename in image_filenames:
    # Construct the full path to the image
    image_path = os.path.join(folder_path, filename)
```

```
    # Open the image
    image = Image.open(image_path)

    # Convert the image to grayscale
    image = image.convert('L')

    # Convert the image to a TensorFlow tensor

    image_tensor = tf.convert_to_tensor(np.array(image), dtype=tf.float32)

    # Add the tensor to the list
    image_tensors.append(image_tensor)

print(f"Converted {len(image_tensors)} PET images to tensors.")

# Define the output folder

output_folder =
"/content/drive/MyDrive/trained_dcnn_slices/input_tensorsUNETtraining" #input of
dcnn in form of tensors saved to resultsofp2p

# Make sure the output folder exists
os.makedirs(output_folder, exist_ok=True)

# Iterate through the test dataset and save images for each patient
for idx, image_tensor in enumerate(image_tensors):

    # Create a unique folder name for each patient

    patient_folder = f"sub_{idx + 1}"  # Assuming patient names are sequential
starting at sub_0027

    # Save the input image, the real target image, and the predicted image
    plt.imsave(os.path.join(output_folder, f"{patient_folder}_input_unet.png"),
image_tensor, cmap='gray')

print(f"Saved images for {len(file_paths_dataset_train)} patients to
{new_output_folder}")
```

```
import os
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import resize  # Add this line to import the resize
function

# Define the path to the main folder containing the patient folders
main_folder = "/content/drive/MyDrive/MNIold2/train"  # Replace this with your
actual folder path

# Define the path to the output folder where the 2D tensors will be saved
```

```
output_folder_train =
"/content/drive/MyDrive/trained_dcnn_slices/targets_tensorsofUNET"  # Replace
this with your actual output folder path

# Define a function to convert a NIFTI image to a 2D tensor
def nifti_to_tensor(image_path):
    # Load the NIFTI image
    image = nib.load(image_path).get_fdata()

    # Calculate the center slice index

    center_slice_index = image.shape[2] // 2

    # Extract the 5th slice above the center slice
    target_slice_index = center_slice_index

    # Extract the desired slice along the z-axis
    tensor = image[:, :, target_slice_index]

    # Normalize the tensor to the range [0, 1]

    tensor = (tensor - np.min(tensor)) / (np.max(tensor) - np.min(tensor))

    # Add an extra dimension to make it a 3D tensor
    tensor = np.expand_dims(tensor[..., np.newaxis], axis=2)

    return tensor

# Iterate over each patient folder
for patient_folder in os.listdir(main_folder):
    # Get a list of all files in the current patient's folder
    files = os.listdir(os.path.join(main_folder, patient_folder))

    # Initialize an empty list to store resized PET tensors

    resized_pet_tensors = []

    # Find the PET scan based on its filename
    for file in files:
        if "pet" in file:
            pet_path = os.path.join(main_folder, patient_folder, file)
            # Convert the PET image to a 2D tensor
            pet_tensor = nifti_to_tensor(pet_path)
            # Resize the tensor to (256, 256) using skimage.transform.resize
            resized_pet_tensor = resize(pet_tensor.squeeze(), (256, 256),
mode='reflect', anti_aliasing=True)
            resized_pet_tensors.append(resized_pet_tensor)

    # Save all resized PET tensors as images to the output folder
    for i, resized_pet_tensor in enumerate(resized_pet_tensors):
        output_path = os.path.join(output_folder_train,
f"{patient_folder}_resized_pet_{i}.png")
        plt.imsave(output_path, resized_pet_tensor.squeeze(), cmap='gray')
```

```
print("All resized PET tensors have been successfully saved to the output
folder.")
```

```
# Get lists of T1, FLAIR, and PET scan file paths for testing data
input_files_test =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
dcnn_test', '*_input.png')))
target_files_test =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
dcnn_test', '*_target.png')))

# Make sure the lists of T1, FLAIR, and PET scan file paths have the same length
assert len(input_files_test) == len(target_files_test)

# Create a dataset of tuples, where each tuple is (T1 path, FLAIR path, PET path)
for a single patient
file_paths_dataset_test = tf.data.Dataset.from_tensor_slices((input_files_test,
target_files_test))

# Apply the function to the dataset

file_paths_dataset_test = file_paths_dataset_test.map(load_and_preprocess_images)

# Batch the dataset
file_paths_dataset_test = file_paths_dataset_test.batch(BATCH_SIZE)

# Prefetch the dataset
file_paths_dataset_test =
file_paths_dataset_test.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
import os
import matplotlib.pyplot as plt
import tensorflow as tf

# Define the path to the new folder where the images will be stored
new_output_folder =
"/content/drive/MyDrive/trained_dcnn_slices/results_p2p_testset"

# Create the new folder if it doesn't exist
if not os.path.exists(new_output_folder):
    os.makedirs(new_output_folder)

# Iterate through the test dataset and save images for each patient
for idx, (input_image, target) in enumerate(file_paths_dataset_test):
    # Generate output image from the input image
    prediction = trained_generator(input_image, training=False)

    # Resize the prediction to (256, 256)
    prediction_resized = tf.image.resize(prediction, [256, 256])

    # Create a unique folder name for each patient (starting from 1)
    patient_folder = f"sub_{idx + 27}"  # Adjust indexing here
```

```
    # Save the resized image
    plt.imsave(os.path.join(new_output_folder,
f"{patient_folder}_predicted.png"), prediction_resized[0, :, :, 0].numpy(),
cmap='gray')

print(f"Saved images for {len(file_paths_dataset_test)} patients to
{new_output_folder}")
```

```
import os
import tensorflow as tf
import numpy as np
from PIL import Image

# Path to the folder containing your images
folder_path = "/content/drive/MyDrive/trained_dcnn_slices/results_p2p_testset"

# Get all image filenames in the folder with PNG extension
image_filenames = [f for f in os.listdir(folder_path) if f.endswith('.png')]


# List to store the image tensors
image_tensors = []

# Read and convert each image into a tensor
for filename in image_filenames:
    # Construct the full path to the image
    image_path = os.path.join(folder_path, filename)

    # Open the image
    image = Image.open(image_path)

    # Convert the image to grayscale
    image = image.convert('L')


    # Convert the image to a TensorFlow tensor
    image_tensor = tf.convert_to_tensor(np.array(image), dtype=tf.float32)

    # Add the tensor to the list
    image_tensors.append(image_tensor)

print(f"Converted {len(image_tensors)} PET images to tensors.")

# Define the output folder
output_folder =
"/content/drive/MyDrive/trained_dcnn_slices/input_tensorsUNETtesting" #input of
dcnn in form of tensors saved to resultsofp2p

# Make sure the output folder exists
os.makedirs(output_folder, exist_ok=True)
```

```
# Iterate through the test dataset and save images for each patient
for idx, image_tensor in enumerate(image_tensors):

    # Create a unique folder name for each patient
    patient_folder = f"sub_{idx + 27}"  # Assuming patient names are sequential
starting at sub_0027

    # Save the input image, the real target image, and the predicted image
    plt.imsave(os.path.join(output_folder, f"{patient_folder}_input_unet.png"),
image_tensor, cmap='gray')


print(f"Saved images for {len(file_paths_dataset_test)} patients to
{new_output_folder}")
```

```
import tensorflow as tf

BUFFER_SIZE = 52
BATCH_SIZE = 1

# Get lists of input and target image file paths



input_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
input_tensorsUNETtraining', '*.png'))) #input
target_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
targets_tensorsofUNET', '*.png')))

# Make sure the lists of input and target image file paths have the same length
assert len(input_files) == len(target_files)

# Create a dataset of tuples, where each tuple is (input path, target path) for a
single patient
file_paths_dataset_train = tf.data.Dataset.from_tensor_slices((input_files,
target_files))

# Define a function to load and preprocess the images
def load_and_preprocess_images(input_path, target_path):

    # Load the images
    input_image = tf.io.read_file(input_path)
    target_image = tf.io.read_file(target_path)

    # Decode the images
    input_image = tf.image.decode_png(input_image, channels=1)
    target_image = tf.image.decode_png(target_image, channels=1)

    # # Resize the images to dimensions divisible by 8
    # input_image = tf.image.resize(input_image, [256, 256])
    # target_image = tf.image.resize(target_image, [208, 240])
```

```
    # Convert the images to float32 and normalize to [0, 1]
    input_image = tf.image.convert_image_dtype(input_image, tf.float32)
    target_image = tf.image.convert_image_dtype(target_image, tf.float32)

    return input_image, target_image


# Get the first pair of input and target paths
input_path, target_path = next(iter(file_paths_dataset_train))

# Call the function load_and_preprocess_images with the first pair of paths

input_image, target_image = load_and_preprocess_images(input_path, target_path)

# Print the shape of input_image and target_image
print(f"The shape of input_image is {input_image.shape}")
print(f"The shape of target_image is {target_image.shape}")

# Apply the function to the dataset
file_paths_dataset_train =
file_paths_dataset_train.map(load_and_preprocess_images)

# Shuffle the dataset
file_paths_dataset_train = file_paths_dataset_train.shuffle(BUFFER_SIZE)


# Batch the dataset
file_paths_dataset_train = file_paths_dataset_train.batch(BATCH_SIZE)

# Prefetch the dataset
file_paths_dataset_train =
file_paths_dataset_train.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)



import os
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import resize  # Add this line to import the resize
function

# Define the path to the main folder containing the patient folders


main_folder = "/content/drive/MyDrive/MNIold2/test"  # Replace this with your
actual folder path

# Define the path to the output folder where the 2D tensors will be saved
output_folder_test =
"/content/drive/MyDrive/trained_dcnn_slices/targets_tensorsofUNET_testing"  #
Replace this with your actual output folder path

# Define a function to convert a NIFTI image to a 2D tensor
```

```
def nifti_to_tensor(image_path):
    # Load the NIFTI image
    image = nib.load(image_path).get_fdata()

    # Calculate the center slice index
    center_slice_index = image.shape[2] // 2

    # Extract the 5th slice above the center slice
    target_slice_index = center_slice_index

    # Extract the desired slice along the z-axis
    tensor = image[:, :, target_slice_index]

    # Normalize the tensor to the range [0, 1]
    tensor = (tensor - np.min(tensor)) / (np.max(tensor) - np.min(tensor))

    # Add an extra dimension to make it a 3D tensor
    tensor = np.expand_dims(tensor[..., np.newaxis], axis=2)

    return tensor

# Iterate over each patient folder
for patient_folder in os.listdir(main_folder):
    # Get a list of all files in the current patient's folder
    files = os.listdir(os.path.join(main_folder, patient_folder))


    # Initialize an empty list to store resized PET tensors
    resized_pet_tensors = []

    # Find the PET scan based on its filename
    for file in files:
        if "pet" in file:
            pet_path = os.path.join(main_folder, patient_folder, file)
            # Convert the PET image to a 2D tensor
            pet_tensor = nifti_to_tensor(pet_path)
            # Resize the tensor to (256, 256) using skimage.transform.resize
            resized_pet_tensor = resize(pet_tensor.squeeze(), (256, 256),
mode='reflect', anti_aliasing=True)
            resized_pet_tensors.append(resized_pet_tensor)

    # Save all resized PET tensors as images to the output folder
    for i, resized_pet_tensor in enumerate(resized_pet_tensors):
        output_path = os.path.join(output_folder_test,
f"{patient_folder}_resized_pet_{i}.png")
        plt.imsave(output_path, resized_pet_tensor.squeeze(), cmap='gray')

print("All resized PET tensors have been successfully saved to the output
folder.")
```

```
import tensorflow as tf

BUFFER_SIZE = 22
```

```
BATCH_SIZE = 1

# Get lists of input and target image file paths
input_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
input_tensorsUNETtesting', '*.png'))) #input
target_files =
sorted(tf.io.gfile.glob(os.path.join('/content/drive/MyDrive/trained_dcnn_slices/
targets_tensorsofUNET_testing', '*.png')))

# Make sure the lists of input and target image file paths have the same length
assert len(input_files) == len(target_files)


# Create a dataset of tuples, where each tuple is (input path, target path) for a
single patient
file_paths_dataset_test = tf.data.Dataset.from_tensor_slices((input_files,
target_files))

# Define a function to load and preprocess the images
def load_and_preprocess_images(input_path, target_path):
    # Load the images
    input_image = tf.io.read_file(input_path)
    target_image = tf.io.read_file(target_path)

    # Decode the images
    input_image = tf.image.decode_png(input_image, channels=1)
    target_image = tf.image.decode_png(target_image, channels=1)

    # # Resize the images to dimensions divisible by 8
    # input_image = tf.image.resize(input_image, [256, 256])
    # target_image = tf.image.resize(target_image, [208, 240])

    # Convert the images to float32 and normalize to [0, 1]
    input_image = tf.image.convert_image_dtype(input_image, tf.float32)
    target_image = tf.image.convert_image_dtype(target_image, tf.float32)

    return input_image, target_image


# Get the first pair of input and target paths
input_path, target_path = next(iter(file_paths_dataset_test))

# Call the function load_and_preprocess_images with the first pair of paths
input_image, target_image = load_and_preprocess_images(input_path, target_path)

# Print the shape of input_image and target_image
print(f"The shape of input_image is {input_image.shape}")

print(f"The shape of target_image is {target_image.shape}")

# Apply the function to the dataset
file_paths_dataset_test = file_paths_dataset_test.map(load_and_preprocess_images)
```

```
# Shuffle the dataset
file_paths_dataset_test = file_paths_dataset_test.shuffle(BUFFER_SIZE)

# Batch the dataset
file_paths_dataset_test = file_paths_dataset_test.batch(BATCH_SIZE)

# Prefetch the dataset
file_paths_dataset_test =
file_paths_dataset_test.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
import tensorflow as tf
from tensorflow.keras import backend as K
import cv2

def perceptual_loss(y_true, y_pred):
    # Extract features from earlier layers of the model for both true and
predicted images
    features_true = y_true  # Extract features from earlier layers of y_true
    features_pred = y_pred  # Extract features from earlier layers of y_pred

    # Compute the perceptual loss based on feature differences
    loss = K.mean(K.square(features_true - features_pred))

# Load your trained U-Net-like model
model_path = '/content/drive/MyDrive/trained_dcnn_slices/new_model.h5'
trained_model = tf.keras.models.load_model(model_path,
custom_objects={'perceptual_loss': perceptual_loss})

# Print a success message
print(f"The trained model was successfully loaded from '{model_path}' in your
Google Drive")

import tensorflow as tf
import matplotlib.pyplot as plt

def increase_contrast(image, alpha=1.0, clip=True):
    # Convert the image to float32 format
    image = image.astype(np.float32)

    # Compute the minimum and maximum pixel values
    min_val = np.min(image)
    max_val = np.max(image)

    # Normalize the image to the range [0, 1]
    image_normalized = (image - min_val) / (max_val - min_val)

    # Apply contrast adjustment

    increased_contrast_image = (image_normalized ** alpha) * (max_val - min_val)
+ min_val
```

```
    # Clip pixel values if specified
    if clip:

        increased_contrast_image = np.clip(increased_contrast_image, 0, 1)

    return increased_contrast_image.astype(np.float32)


def remove_noise(image, kernel_size=3):
    # Apply Gaussian blur to remove noise
    denoised_image = cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)
    return denoised_image

# Assuming that 'file_paths_dataset_test' is your test dataset

mse_total = 0
psnr_total = 0
num_samples = 0

for input_image, target_image in file_paths_dataset_test:
    # Generate output image from the input image
    prediction = trained_model.predict(input_image)

    input_image = np.squeeze(input_image)
    target_image = np.squeeze(target_image)
    prediction = np.squeeze(prediction)

    # Apply the same post-processing steps as in training (e.g., increase
contrast and remove noise)
    increased_contrast_prediction = increase_contrast(prediction)
    denoised_prediction = remove_noise(increased_contrast_prediction)

    mse = tf.keras.losses.MeanSquaredError()

    mse_value = mse(tf.convert_to_tensor(target_image),
tf.convert_to_tensor(denoised_prediction))
    mse_total += mse_value.numpy()

    max_target = tf.reduce_max(target_image)
    psnr = 20 * tf.math.log(max_target / tf.sqrt(mse_value)) / tf.math.log(10.0)
    psnr_total += psnr.numpy()


    num_samples += 1

    # Display the input image, the real target image, the original predicted
image, and the post-processed predicted image
    plt.figure(figsize=(20, 5))

    plt.subplot(1, 3, 1)
    plt.title('Synthesized PET')
    plt.imshow(input_image, cmap='gray')
    plt.axis('off')
```

```
    plt.subplot(1, 3, 2)

    plt.title('Real PET')
    plt.imshow(target_image, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.title('Improved PET')
    plt.imshow(denoised_prediction, cmap='gray')
    plt.axis('off')

    plt.show()


# Calculate average error metrics

mse_average = mse_total / num_samples
psnr_average = psnr_total / num_samples

print("Average Mean Squared Error: ", mse_average)
print("Average PSNR: ", psnr_average)
```

COMPLEX ENGINEERING PROBLEM
AI DRIVEN PET LIKE SYNTHESIS FROM MRI DATA

ABSTRACT

Our project aims to synthesize Positron Emission Tomography (PET) - like image from a MRI scan from artificial intelligence (AI) driven models. Dataset used in this regard is of 37 patients each having a T1w, FLAIR and PET image modality. These images which were in Neuroimaging Informatics Technology Initiative (NIFTI) format were pre-processed by converting into 2D tensors and extending them to 3D tensors by adding an extra dimension. The T1 and FLAIR images are concatenated and given as input to pix2pix model while PET images are set as the ground truth for our model. The synthesized output from the above model serves as the input to another machine learning model which is a modified super resolution convolutional neural network (SRCNN) called Fast Medical Image Super Resolution Method. This model maps a low resolution image to a super resolution image thus giving us better images. For hardware implementation, FPGA and DSP Kit are utilized for pattern recognition on the output PET-like image. Furthermore, the synthesis software model is uploaded on a Raspberry Pi to allow for localization and environment integration. This project will aid in bridging the healthcare gap by providing a non-invasive alternate for PET imaging by using easily accessible MRI data. It will also reduce the need of costly PET scanners which are limited in Pakistan.

DEPTH OF KNOWLEDGE REQUIRED (WP1)

Project requires an understanding of engineering like image processing and machine learning (WK3). It also requires engineering specialist knowledge of machine learning models like pix2pix and deep CNN to synthesize PET-like images (WK4). The engineering design is supported by the utilization of FPGA for pattern recognition (WK5). The role of engineering in society via this project can be providing an inexpensive and a non-invasive alternate to patients who have to perform PET imaging and also assist doctors in deciding whether the patient needs a further scan or not (WK7).

Depth of Analysis Required (WP3)

To incorporate AI models for image synthesis strong understanding of neural network architecture is required to implement pix2pix and deep CNN.

Familiarity of issues (WP4)

The project involves infrequent issue of patient files not properly loading in the input and target folders.

	WP1						WP2	WP3	WP4	WP5	WP6	WP7
	WK3	WK4	WK5	WK6	WK7	WK8						
PLO1 (WA1)	X											
PLO2 (WA2)		X						X				
PLO3 (WA3)			X									
PLO4 (WA4)									X			
PLO5 (WA5)									X			
PLO6 (WA6)					X							
PLO7 (WA7)												
PLO8 (WA8)												

PLAGIARISM REPORT

ORIGINALITY REPORT

17%	10%	10%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

SUSTAINABLE DEVELOPMENT GOALS

SDG 3 focuses on good health and wellbeing. This goal is well inclined with our project as it aims to provide an inexpensive and non-invasive alternative to patient serving as and advantage for them as well as for the doctors because it will assist them in taking decisions.

SDG 9 caters industry, innovation and infrastructure. Our project intends to bridge the healthcare gap and promote advancement in the medical field.

SDG 10 ensures equal opportunities to all. With the help of our project people who can't afford PET imaging can go towards MRI and then have a synthesized output.