

Training a Diffusion-GAN With Modified Loss Functions to Improve the Head-and-Neck  
Intensity Modulated Radiation Therapy Fluence Generator

by

Scott William Reid

The Graduate Program in Medical Physics  
Duke University

Defense Date: April 3rd, 2023

Approved:

Jackie Wu, Supervisor

Yang Sheng, Supervisor

Chu Wang

Thesis submitted in partial fulfillment of the requirements for the degree of  
Master of Science in The Graduate Program in Medical Physics  
in the Graduate School of Duke University  
2024

ABSTRACT

Training a Diffusion-GAN With Modified Loss Functions to Improve the Head-and-Neck  
Intensity Modulated Radiation Therapy Fluence Generator

by

Scott William Reid

The Graduate Program in Medical Physics  
Duke University

Defense Date: April 3rd, 2023

Approved:

Jackie Wu, Supervisor

Yang Sheng, Supervisor

Chu Wang

An abstract of a thesis submitted in partial fulfillment of the requirements for the degree  
of  
Master of Science in The Graduate Program in Medical Physics  
in the Graduate School of Duke University  
2024

Copyright by  
Scott William Reid  
2024  
All rights reserved

## Abstract

**Introduction:** The current head-and-neck (HN) fluence map generator tends to produce highly modulated fluence maps and therefore high monitor units (MUs) for each beam, which leads to more delivery uncertainty and leakage dose. This project implements diffusion into the training process and modifies the loss functions to mitigate this effect.

**Methods:** The dataset consists of 200 head-and-neck (HN) patients receiving intensity modulated radiation therapy (IMRT) for training, 16 for validation, and 15 for testing. Two models were trained, one with diffusion and one without. The original model was a conditional generative adversarial network (GAN) written in TensorFlow, the model without diffusion was written to be the PyTorch equivalent of the original model. After confirming the model was properly converted to PyTorch by comparing outputs, both new models were modified to use binary cross entropy for the GAN loss and mean absolute error as a third loss function for the generator. Hyperparameters were carefully selected based on the training script for the original model, and further tuned with trial and error. The diffusion was implemented based on Diffusion-GAN and the associated GitHub repository. The two new models were compared by plotting training loss vs epoch over 500 epochs. The two models were compared to the original model by comparing the output fluence maps to the ground truth using similarity index and comparing DVH statistics among the three models.

**Results:** The with-diffusion model and no-diffusion model achieved similar training loss. The diffusion model and no-diffusion model consistently delivered better parotid sparing than the original model and delivered less dose to four of the six tested OAR. The with-diffusion model delivered less dose to five of the six tested OAR. The diffusion model had the least MUs: 23% less than the original model and 3% less than the no-diffusion model. The diffusion model had lower D2cc: 4% less than the original model and 1% less than the no-diffusion model on average. All three plans deliver 95% of the prescription dose to nearly the same percentage of PTV volume.

**Conclusion:** Implementing diffusion does not provide a significant impact on training time and training loss. However, it does enable comparable dose performance to both the no-diffusion and original models, while significantly reducing the total MU's and 3D max 2cc relative to the original model and slightly reducing these metrics relative to the no-diffusion model, indicating smoother fluence modulation. In addition, both new models reduced the dose to the right and left parotids relative to the original model, and to four of six tested OAR total, while the with-diffusion model consistently delivered less dose to OAR than the no-diffusion model. This indicates that both the new loss functions and diffusion reduce the overall dose to the OARs while preserving dose conformity around the target.

## **Dedication**

To my wife Breanna and our daughter Lyla.

# Contents

Abstract . . . . .	iv
List of Tables . . . . .	x
List of Figures . . . . .	xi
Acknowledgements . . . . .	xii
1 Introduction . . . . .	1
1.1 Head and Neck Cancer . . . . .	1
1.2 Intensity-Modulated Radiotherapy . . . . .	1
1.3 The Motivation for AI in IMRT Planning . . . . .	3
1.4 Convolutional Neural Networks and Generative Adversarial Networks . . . . .	4
1.5 The Current In-House HN AI . . . . .	5
1.6 The Motivation for Diffusion and New Loss Functions . . . . .	6
1.7 The Aim of This Study . . . . .	7
2 Methods . . . . .	9
2.1 Data Collection . . . . .	9
2.2 Converting to PyTorch . . . . .	9
2.2.1 Modifying the Model Structure . . . . .	9
2.2.2 Structuring the Tensors . . . . .	10
2.2.3 Converting the Wavelet Loss . . . . .	10
2.2.4 Model Initialization and Data Loading . . . . .	11
2.2.5 The Training and Validation Loops . . . . .	11
2.3 Adding in Diffusion . . . . .	12
2.3.1 Modification 1: Setting the Diffusion Hyperparameters . . . . .	13
2.3.2 Modification 2: Diffusion Plug-in . . . . .	14
2.4 Hyperparameter Modification . . . . .	14
2.5 Features for Easy Starting and Stopping . . . . .	15

2.6	Running Training and Using the Linux Command Line . . . . .	16
2.7	A New Loss Function: Binary Cross Entropy (BCE) . . . . .	17
2.8	A New Loss Function: Mean Absolute Error (MAE) . . . . .	18
2.9	Comparing Performance Between the Models . . . . .	19
2.9.1	Similarity Metrics . . . . .	19
2.9.2	Training Performance . . . . .	20
2.9.3	Plan Quality . . . . .	20
2.9.4	Visualization . . . . .	21
3	Results . . . . .	22
3.1	Fluence Map Comparisons . . . . .	22
3.2	Similarity Metrics . . . . .	25
3.3	Validation Losses . . . . .	25
3.4	Plan Quality Metrics . . . . .	26
3.4.1	Total Dose . . . . .	26
3.4.2	PTV Coverage . . . . .	28
3.4.3	OAR Sparing . . . . .	29
4	Discussion . . . . .	35
4.1	Fluence Map Similarity . . . . .	35
4.2	Validation Loss Analysis . . . . .	35
4.3	Plan Quality . . . . .	35
4.3.1	Total Dose . . . . .	35
4.3.2	PTV Coverage . . . . .	36
4.3.3	OAR Sparing . . . . .	36
4.4	Clinical Use . . . . .	37
4.5	Limitations and Future Research . . . . .	38
5	Conclusion . . . . .	39

Bibliography . . . . . 40

## List of Tables

2.1	Diffusion hyperparameters . . . . .	14
2.2	Learning Rate Ranges . . . . .	15
2.3	Commonly used Linux and Git commands. . . . .	17
3.1	Comparison of SSIM and MAE for Different Models . . . . .	25
3.2	Difference and Percent Differences Compared to the original model for the dose statistics. . . . .	27
3.3	Difference and percent differences compared to the original model for the OAR. . . . .	31

## List of Figures

1.1	Movement of MLCs to create a modulated dose distribution. . . . .	3
1.2	Generator architecture. [6] . . . . .	5
3.1	Fluence Map Comparison for Patient 217 at the 9th beam angle. . . . .	22
3.2	Fluence Map Comparison for Patient 218 at the 5th beam angle. . . . .	23
3.3	Fluence Map Comparison for Patient 229 at the 4th beam angle. . . . .	24
3.4	Validation loss vs epoch number between the diffusion and no-diffusion models. 25	
3.5	Maximum dose delivered to a 2 centimeter cubed region. . . . .	26
3.6	Total MUs delivered by the plan. . . . .	27
3.7	Volume receiving 95% of the prescription dose normalized to 44Gy (PTV44 D95%). . . . .	28
3.8	Median dose to the left parotid. . . . .	29
3.9	Median dose to the right parotid. . . . .	30
3.10	Dose volume histogram comparing the with-diffusion and no-diffusion models. 32	
3.11	Superimposed isodose lines for two patients. Column 1: original model, Column 2: no-diffusion model, Column 3: with-diffusion model. . . . .	33
3.12	Superimposed isodose lines for two patients (frontal and saggital). . . . .	34

## **Acknowledgements**

Thank you to my family and friends for supporting me throughout my education, to my professors current and past for their desire to help me learn and grow, to Dr. Jackie Wu and Dr. Yang Sheng for giving me the opportunity to do research in their lab, and to Dr. Xinyi Li for her constant guidance through this project. Thank you to the Watt family for supporting my education by awarding me a scholarship in honor of their son Kelly, who was an exceptional student and athlete. A special thank you to my late professor Dr. Robert Fersch, who invited me into his office one day to tell me I “have what it takes” and encouraged me to pursue graduate school. He was the first professor and physicist to believe in me and was vital to my success in the field. May he rest in peace.

# 1. Introduction

## 1.1 *Head and Neck Cancer*

Head-and-neck cancers (HN) typically begin in the squamous cells that line the mucosal surfaces of the head and neck and can form in the oral cavity, pharynx, larynx, sinuses, and parotids [1]. The annual rate and the annual number of new HN cancers have been increasing on average since 1999 [4]. Further, HN cancers account for nearly 4% of all cancers in the United States [7] and are the seventh most common cancer in the world [3]. In 2020, 277,597 people worldwide died as a result of HN cancer [2]. While there are many carcinogens linked to HN cancer, a few common ones are alcohol and tobacco use, human papillomavirus (HPV), betel quid, ancestry, genetic disorders, radiation exposure, and certain occupational exposures [1].

## 1.2 *Intensity-Modulated Radiotherapy*

Intensity-modulated radiotherapy (IMRT) is one of many external beam radiation therapy techniques designed to deliver high amounts of dose to the planning target volume (PTV) while sparing the surrounding organs at risk (OAR). IMRT is easily implemented thanks to innovation in computer technologies, which is necessary due to the computationally taxing inverse planning process.

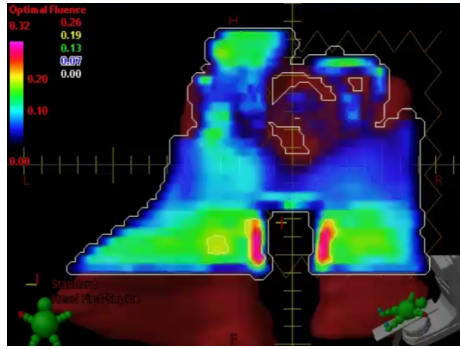
Forward planning involves getting the dose distribution by manually setting the plan parameters. This requires planners to specify parameters like jaw size, multileaf collimator (MLC) shape, beam number, beam weighting, beam angles, beam modifiers, field size, and more. The planner puts the settings in the treatment planning system and runs simulations until the desired dose prescription to the PTV is achieved within the required OAR dose restraints. While effective, this process can take significant amounts of time to optimize the setting. Inverse planning with IMRT allows the planner to input the dose prescription and constraints, as well as the desired amount of beams and their angles. The computer can then utilize gradient descent, with the input prescription and constraints as the objective function, to automatically optimize the amount of MU's delivered per beam angle, jaw

size, MLC shape, and more. Inverse planning has greatly improved both plan quality and planning time in many applications, including in HN cancer treatment planning.

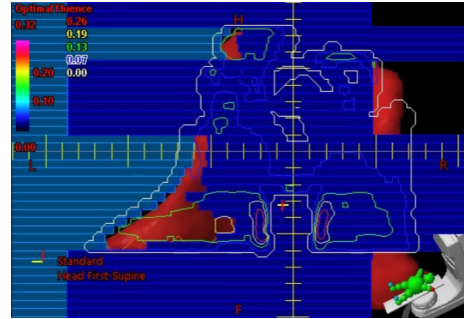
Typically, the first step in IMRT planning is using three-dimensional (3D) scans to define the PTV and surrounding OARs. Since IMRT utilizes inverse optimization, these must be accurately contoured as they define the objective function during the optimization process. Once these are set, the planner then selects various parameters, including the treatment machine, photon energy, and patient position. The planner then sets the number of beams and their angles. With HN cases, nine equally separated beams are commonly used. Once the general setup is complete, the planner then inputs the dose objectives and dose limits that are prescribed by the radiation oncologist and starts the optimization.

This optimization process is where the true benefit of IMRT lies. This is where the weight or radiation intensity of each beamlet at the size of 2.5mm x 2.5mm is determined. All the beamlets from the same gantry angle form the radiation intensity map or fluence map (Figure 1.1a). To deliver such a fluence map, the motion of the MLC is determined, and this is what allows for modulated intensity at each beamlet across the PTV.

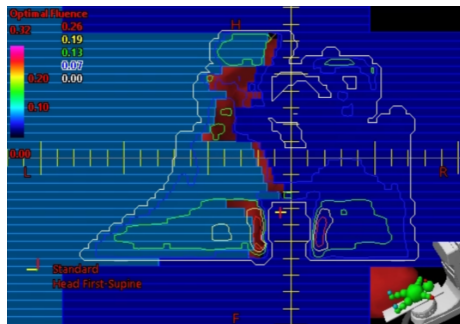
The blue rectangles in Figure 1.1 represent the MLC leaves. Each leaf can move forward or backward, leading to the ability to create asymmetric dose patterns. Additionally, the leaves can move whilst the beam is on and the dose is being delivered. This leaf motion is what causes the effect of the intensity being non-uniform in Figure 1.1. Additionally, the MLCs are positioned in such a way that radiation is obstructed beyond the edge of the structure. The ability for IMRT to modulate the dose across the PTV enables much better dose conformity than traditional radiotherapy, as well as lower OAR dose.



(a) Fluence from a beam.



(b) Snapshot of MLC movement near the beginning.



(c) Snapshot of MLC movement near the middle.



(d) Snapshot of MLC movement near the end.

FIGURE 1.1: Movement of MLCs to create a modulated dose distribution.

### 1.3 The Motivation for AI in IMRT Planning

If all goes well, the first optimization run meets all the requirements and the plan is ready to be verified. But this is almost always not the case. Oftentimes, parameters need to be adjusted leading to more runs of the optimizer - which typically takes several minutes to complete. Even in non-outlier cases, the process of creating plans often takes more than an hour to complete. Creating plans for these common cases can be time-consuming, as the steps taken to make the plans will be very similar to other cases. Implementation of artificial intelligence (AI) to generate optimal fluence maps would streamline the treatment planning pipeline by bypassing many of the steps taken between contouring and plan verification. This would enable treatment planners to spend less time working on standard cases, and more time working on outlier cases.

## **1.4 Convolutional Neural Networks and Generative Adversarial Networks**

Convolutional neural networks (CNN) are a powerful deep learning tool commonly used with image processing and recognition. A standard convolutional layer works by sliding a kernel (a matrix) over an image and taking the dot product at each step. This is placed on a feature map, which can then be used in a subsequent operation.

Subsequent operations using these feature maps can involve further convolutional layers but also can pull from other methods like linear layers, max pooling, dropout, and more. A key part of the convolutional neural network is the gradient descent process. After the input data has passed through the layers, the loss is calculated using the defined objective function, and this is used to propagate backward through the model. Using the chain rule, the derivatives are taken at each step in the model to calculate the total derivative. The optimizer then looks at these gradients and updates the kernel weights in an attempt to reduce the loss. Over many batches of data and iterations through these batches, the model continues updating its weights.

Despite their efficacy, CNNs have several drawbacks worth noting. Primarily, they are susceptible to overfitting. During the training process, the model adjusts itself based on a subset of the data, known as the training set. If the model learns patterns that are only present in the training set to minimize training loss, it may struggle to perform adequately on data lacking these patterns. Avoiding overfitting on the training data is crucial for ensuring the model's effectiveness on unseen data. Another thing to consider is the issue of exploding or vanishing gradients. This is when the gradient used to update kernel weights is too large or too small to make meaningful adjustments. CNN in general performs poorly when the dataset it's learning from is small, which can render them obsolete in scenarios where a large batch of data is not available. These issues must be taken into consideration when constructing the networks.

Another emerging deep learning architecture is generative adversarial networks (GANs). It comprises two primary models: a generator and a discriminator. The generator's objec-

tive is to fabricate synthetic data capable of deceiving the discriminator, while the discriminator aims to differentiate between real and fake images. Both models iteratively refine their parameters by leveraging the outputs of the other, fostering an adversarial dynamic. Despite the potency of GAN training, attaining stable convergence poses a challenge. GANs are susceptible to issues such as vanishing gradients, like CNNs, and frequently encounter mode collapse. This phenomenon arises when the generator produces a limited range of outputs, impeding the discriminator’s ability to effectively learn. Subtle alterations within this confined range allow the generator to easily outsmart the discriminator. Furthermore, GANs exhibit sensitivity where even minor adjustments to input can trigger significant variations in output, leading to training instability.

### 1.5 The Current In-House HN AI

Currently, at Duke, we have an AI that can generate fluence maps from patient anatomy with decent performance. This generator is trained using a GAN, with the architecture shown below.

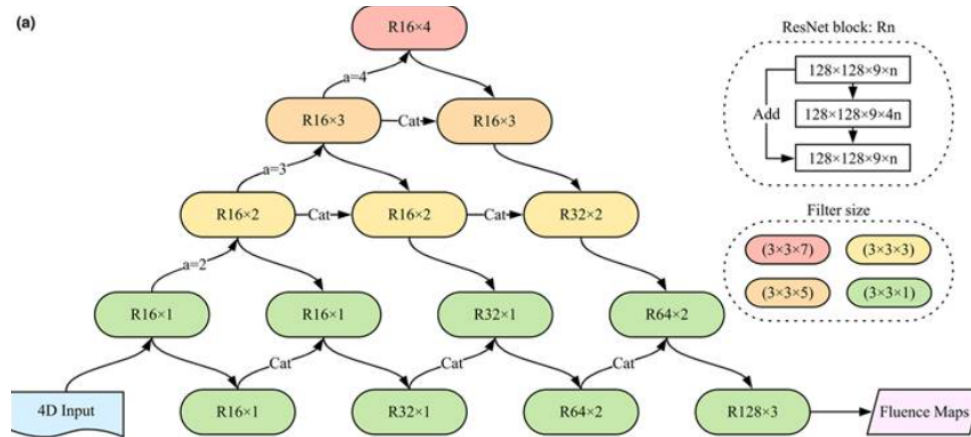


FIGURE 1.2: Generator architecture. [6]

There are a few key features to this design that make it powerful. First, it uses varying kernel sizes and varying strides. This allows for the model to capture information on a broader scale both within a plane and also between planes. The architecture utilizes

residual blocks, which is the technique designed to provide a “shortcut” during the gradient descent process. This helps mitigate the vanishing gradient issue that is common in GANs. Additionally, the middle layer in the blocks expands the feature output before contracting again in the third layer, allowing for extra information to be captured without having an excessive amount of data. The complex series of concatenations allows for the gradients to flow in many directions, but most importantly allows for a much shorter path from output to input. This allows for the model to maintain a deep architecture with many layers while mitigating the vanishing gradient issue that is common within deep models.

A discriminator of much smaller size takes both the input data and a fluence map. The output of the discriminator is a value between 0 and 1 for each pixel in the fluence map, 0 being the discriminator thinks the pixel is fake and 1 being it thinks the pixel is real. This discriminator is then used to evaluate how good the generator is at producing fake data that seems real. Additionally, the generator is also optimized to reduce the difference between its output and the ground truth fluence map. This adversarial training is run and the best generator is selected, which then creates treatment plans that can be used to treat patients.

## ***1.6 The Motivation for Diffusion and New Loss Functions***

While effective, this GAN process has issues. The training is extremely sensitive to small changes in hyperparameters. Finding the correct hyperparameters is not an exact science, and requires constant starting and stopping to optimize. This becomes more difficult due to the model being deep and the input images being large. Even with a high-end machine with four GPUs, training can take over five days. This makes the hyperparameter optimization process take a long time. Additionally, neural networks work better with more data, and the current HN dataset is not as large as desired. These issues contribute to issues in the stability of training. Optimizers are used to regulate some of the hyperparameters the models use, but without a good starting point and enough epochs of stable training, they are not able to learn enough to be effective.

Stabilizing the training would allow for the optimizer to be more effective, and would mitigate the need to closely monitor the progress during training. It would also reduce the amount of starting and stopping required to train the model, and would also reduce the training time.

Further, the existing model generates fluence maps characterized by significant modulation, resulting in elevated leakage dose and delivery uncertainty. By integrating a diffusion process, we aim to enhance our comprehension of the true data distribution, specifically, the set of training images representing patient anatomy and the corresponding ground truth fluence maps, to diminish modulation in the outputs and alleviate these concerns.

The idea for a Diffusion-GAN comes from the 2022 paper "Diffusion-GAN: Training GANs with Diffusion" [8]. The idea is to expand the amount of data the generator and discriminator have to learn from. By injecting Gaussian-mixture distributed noise into real and fake data at time-dependent intervals, the models can learn from new data as time progresses [8]. Not only does this have the potential to prevent over-fitting, but the varying noise-to-data ratio can help prevent the discriminator from becoming too powerful, and also help the generator learn the true data distribution [8]. Further, the paper shows that Diffusion-GAN can boost the performance of the GAN, meaning in our application implementing this has the potential to improve the quality of generated treatment plans.

Additionally, the loss functions used to optimize the models have a large impact on the weights and thus the output prediction. To better emphasize the classification nature of GAN loss and better predict low-dose regions, this project proposes a binary cross entropy loss function for the GAN loss and the addition of a mean absolute error loss function to the generator fluence loss.

## ***1.7 The Aim of This Study***

This study aims to improve the current HN AI in two main ways. First, by reducing the training time and increasing the training stability. We currently have one very high-end computer capable of running the training in a reasonable amount of time. If the training

needs to be run many times and the parameters need to continuously be adjusted, the computer will be occupied by model training for extended periods. Implementing diffusion has the potential to reduce the time needed for training and the optimization process. Second, to reduce the modulation in the fluence map outputs. If the generator can better understand the true data distribution, the output fluence maps will reduce leakage dose, total dose, and OAR dose.

## **2. Methods**

### ***2.1 Data Collection***

The data used in this study is the same data used to train the current HN AI. It was collected from 231 patients and was split into 200 training cases, 16 validation cases, and 15 test cases. All 231 plans were done using a clinically validated script, and set to the parameters of 44 Gy, 2 Gy/fx, and 9 static beams varied at 40 degrees of separation using a 6MV beam energy [6]. A more in-depth description of how the data was collected can be found in the original paper [6].

### ***2.2 Converting to PyTorch***

This study utilizes the forward diffusion process from the GitHub associated with the Diffusion-GAN paper [8]. This process is constructed within the PyTorch framework, while the HN AI was constructed using Tensorflow. To make the two compatible, they must work within the same framework. Thus, the decision was made to convert the HN AI to PyTorch. Once again, more in-depth information on the structure and workflow of the AI can be found in the original paper [6]. This project does not change the GAN from the original paper but rather adds a new feature, and thus this report will focus on the diffusion feature.

#### **2.2.1 Modifying the Model Structure**

The HN AI was written as a defined function, which is an optimal way to write them in Tensorflow as it has a method to easily compile the model from defined functions. However, PyTorch leverages a feature called the module, which does not work with defined functions but instead with a class-based structure. The module has several features, including parameter loading and management, as well as making nesting modules intuitive. The fundamental difference here is when using the module, the layers get defined in the initialization, and data gets passed through the forward section.

## 2.2.2 Structuring the Tensors

Another key difference between Tensorflow and PyTorch is the tensor order. The data has three dimensions: rows, columns, and angle. The angle represents the 9 beam angles used when constructing the plans. The fourth dimension is the features. For RGB data, this would be the color channels. In this context, the features represent the contours of the selected anatomy. In tensorflow, the proper tensor order would be [batch, image rows, image columns, angles, features]. This tensor order would be immediately compatible with the different operations in Keras and Tensorflow, like the Conv3D layers. However, PyTorch used a different convention: [batch, features, image rows, image columns, angles]. Therefore, it was critical to permute the dimensions before using the PyTorch functions. Additionally, Tensorflow is compatible with using numpy arrays as input, while PyTorch requires it to be converted to a tensor first. Additionally, if the model is loaded onto the GPU, the tensor must specifically be moved to the GPU in the code, unlike in Tensorflow. Therefore, to perform the numpy operations that exist in the original GAN, the tensors need to later be both moved from the GPU to the CPU and also converted to a numpy array. While the process of converting tensor format is tedious, it is a necessary step for PyTorch compatibility.

## 2.2.3 Converting the Wavelet Loss

The wavelet loss is the loss function selected to compare the ground truth fluence map to the generated fluence maps. They leverage a Haar filter and require four depthwise convolutional layers. In Tensorflow, this is straightforward, as there is a built-in depthwise 2d convolutional layer. However, PyTorch has a much different syntax. First, the filters needed to be converted to torch tensors and permuted to match the Pytorch tensor order. Next, the same needed to be done with the input, as addressed in the previous section. The final consideration is PyTorch requires the group's parameter to match the length of the third dimension (the depth) for the 2D convolution to be done depth-wise. These structural changes, combined with syntactical differences, make the functionality the same between

frameworks.

## **2.2.4 Model Initialization and Data Loading**

To properly determine if the conversion from Tensorflow to PyTorch was correct, model initialization and data loading methods were kept the same. However, the model must leverage the four available GPUs, otherwise a full training loop would take over two weeks. The major difference here is PyTorch requires wrapping the models in the DataParallel module. Additionally, the optimizers and loss functions selected were the same ones used in the original model, the only difference being they were loaded through PyTorch. The loading loop is kept constant between frameworks, but PyTorch is optimized to use its TensorDataset module to better organize the data for the training loop. Therefore, these were used, but parameters like shuffle data were carefully selected to ensure the looping process would be constant between the Tensorflow and PyTorch models. While these are differences in the code, it does not change the actual data itself, or the way the data is used in model training.

## **2.2.5 The Training and Validation Loops**

The most significant difference between these frameworks is how the training loops are used. Tensorflow makes model training much simpler, requiring significantly fewer lines of code to accomplish training and optimization. The need for more lines of code in PyTorch does make the model training process more customizable, however, it does allow for more opportunities for user error. Therefore, careful considerations need to be taken into account when designing these loops.

The first consideration is PyTorch requires the user to manually define the device the data goes to. Since the models were moved to GPU to begin, the tensors must also be moved. Additionally, the tensor datatype must be converted to float32 to be compatible with the convolutional layers.

The first model to get trained is the discriminator. It needs to be trained on both real and fake data. The generator and discriminator must be trained separately, and this must

be specified. Discriminator training involves the output of the generator when training on the fake data, and thus we begin by doing a forward pass of the ground truth data through the generator. We then use `.detach()` on the output to disassociate this tensor from the generator model. Now, when we pass this into the discriminator, gradients will not flow through the generator during back-propagation.

The next step is to clear the previous gradients. When doing back-propagation, it is important to avoid combining the calculated gradients with previously saved ones from earlier batches. Therefore, `.zero_grad()` is called to clear these gradients.

PyTorch then requires each step of the training process (the forward pass, loss calculation, and backward pass) to be called separately. This gets done for both the real and fake passes for the discriminator. Finally, the optimizer is “stepped” to update the model parameters. In Tensorflow once the model gets compiled, all that needs to be done is a `train_on_batch` call to do these steps. PyTorch adds much more complexity to this area of the code.

For generator training, the same initial step is taken with clearing the gradients. Using the same forward pass through the generator as before, but without the detach method, a forward pass through the discriminator using this output is done to compute the generator GAN loss. This discriminator forward pass must also be detached so that gradients do not flow through it and model training is kept separate. The next steps are to mimic the behavior of the original Tensorflow model, with the loss calculations being done, then back-propagation, and then the optimization being done subsequently.

Finally, the validation phase mimics the Tensorflow validation, just using the required PyTorch steps described above. We are not updating the models in this step, and thus there are no backward or step calls.

### ***2.3 Adding in Diffusion***

Once the model was converted to PyTorch, adding in diffusion was straightforward. Using the diffusion code from the Diffusion-GAN paper [8], small modifications were made

to `diffusion.py` to support the extra dimension. The code was designed to work on standard color images, while the data in this project is four-dimensional (width, height, angle, feature). Width and height are the dimensions of the 2D grayscale images of patient anatomy, angle represents the angle at which the 2D image is shown (matches the angle the beam is delivered at), and feature is the patient’s anatomical feature that is shown. Once this was completed, the modifications to the PyTorch model began.

### **2.3.1 Modification 1: Setting the Diffusion Hyperparameters**

Four hyperparameters needed to be added for the diffusion process to work. While more modifications can be done for the full version, the authors noted that for the simple data augmentation method there were no significant empirical differences from the full version. Therefore, this project leveraged the simple plug-in method. The added hyperparameters are `diffusion_p`, `ada_target`, `ada_interval`, and `ada_kimg`.

The first parameter, `diffusion_p`, is simply the initial value of the diffusion level. The higher this is, the noisier the data becomes. We set this at zero to allow the model to first learn from the raw images. The next parameter, `ada_target`, is the target for the adaptive diffusion. The higher it goes, the noisier the target. This parameter can be adjusted throughout training to introduce the model to different levels of noise, and typically it was set between 0.6 and 0.8. The `ada_interval` tells the model how many batches of data occur before the adjustment to the diffusion level is made. This was set to be 4, so every four batches the diffusion would step. Finally, `ada_kimg` tells us how many thousands of images it takes for the diffusion level to increase or decrease by one unit. In the original code, this was set to 25000. However, this project has a significantly smaller dataset, and therefore the number selected was between 1000 and 5000 for runs.

Table 2.1: Diffusion hyperparameters

Hyperparameter	Range
Diffusion Level (diffusion_p)	[0]
ADA Target (ada_target)	0.6 - 0.8
ADA Interval (ada_interval)	4
ADA Speed (ada_kimg)	1000-5000

### 2.3.2 Modification 2: Diffusion Plug-in

The next step uses the simple plug-in method from the Diffusion-GAN GitHub [8]. First, we check to see if the batch index meets the required threshold set by `ada_interval`. After this, we perform the adjustments using the other hyperparameters.

$$C = \frac{\text{batch\_size} \times \text{ada\_interval}}{\text{ada\_kimg} \times 1000} \quad (2.1)$$

$$\text{diffusion\_p\_adjust} = \text{sign}(\text{sign}(\text{Discriminator}(\text{real\_images}) - \text{ada\_target}) \cdot C) \quad (2.2)$$

$$\text{diffusion.p} = (\text{diffusion.p} + \text{diffusion\_p\_adjust}).\text{clip}(\text{min} = 0., \text{max} = 1.) \quad (2.3)$$

This `diffusion.p` value is used to determine the degree of diffusion during the next call to the diffusion class. This method allows for the continuous augmentation of diffusion level so the model is constantly being introduced to different data.

## 2.4 Hyperparameter Modification

Due to the unstable nature of GAN training, starting and stopping is necessary. A good way to regulate stability is to manually change hyperparameters after each training run. While a variety of hyperparameters were experimented with, like batch size and betas, the one that ended up being modified throughout training (alongside two of the diffusion parameters) was the learning rate. To begin, the learning rate for the discriminator was set to be extremely low compared to the generator. A frequent GAN failure mode comes from the discriminator becoming too powerful, and making it difficult for the generator to learn. By making the learning rate for the generator comparatively higher, the generator

gets a head start and allows it to detect specific trends that commonly occur in fluence maps quicker without getting penalized too heavily by the discriminator’s determination of how real the data looks. After the loss for the generator stops dropping at a high rate, the hyperparameters can be re-initialized to give the discriminator a learning rate high enough for it to update faster.

The losses for every batch and every epoch get printed and saved. This way it is possible to monitor the GANs behavior before a failure mode occurs. For example, a common failure comes with an exploding gradient in the generator’s wavelet loss. Once this gradient gets extremely high, the discriminator becomes too good at distinguishing real and fake data for the model to continue training stably. In this scenario, the user can analyze this and re-initialize the training at the most recent validation minimum with a lower learning rate in the generator. That way, the generator does not make the radical changes that led to the exploding gradient in the first place. Another failure occurs when the discriminator starts to get too powerful. The user could re-initialize at validation minimum again, but instead lower the discriminator learning rate and increase the generator’s. This way, the discriminator is limited in its ability to quickly respond to the generator’s changes, and the generator can change faster to fool the discriminator.

Table 2.2: Learning Rate Ranges

<b>Model</b>	<b>Learning Rate</b>
Generator	1e-6 to 1e-3
Discriminator	1e-10 to 1e-4

While the starting and stopping is tedious, it is necessary to allow for stable GAN training. Several features of the program allow for ease of starting and stopping, which are discussed in the subsequent section.

## ***2.5 Features for Easy Starting and Stopping***

When the user decides it is time to stop the program in the middle of a run, they must be easily able to restart it. The best place to restart the program is from the validation

minimum. The validation minimum occurs when the validation loss is the lowest, and is a good measure of model performance as it is calculated based on data the model has not yet seen. Additionally, the user must be able to see the behavior before stopping to determine which adjustments need to be made.

First, the model's state dictionary, which contains its parameters, is saved when validation loss is at its lowest point. To do this, the validation loss is saved at the end of every epoch and saved to a numpy file. This way, the model can track validation loss between separate runs by loading in the data from previous runs. At the end of each epoch, if the validation loss is less than the minimum value in the entire array of validation minimums, the previously saved model state dictionary is overwritten with the current. Now, when the user wants to restart from the validation minimum, the program automatically detects and loads it.

For tracking and plotting purposes, the training loss stats are also saved, but these are not used to update or create checkpoints. However, by saving these, the user can decide how to adjust the learning rate. To do this, it simply requires a modification to the two lines of code that contain the learning rate for the generator and discriminator.

## ***2.6 Running Training and Using the Linux Command Line***

The computer used to run training, named lambda1, is accessed via the command line and runs on Linux. The method used to update the code and retrieve the data was using git. On GitHub or a local computer, updates were made, committed, and pushed. On lambda1, it would get pulled and ran, and then changes could again be committed and pushed. This way, code can be developed using an integrated development environment (IDE) and with the ability to reference file structure using an intuitive file viewing user interface like the Windows file explorer. These tools make development much easier and quicker than doing so within the command line. Using the command line requires several commands, here is a list of the frequently used commands for working with the code on lambda1.

Table 2.3: Commonly used Linux and Git commands.

Command	Description
<code>cd folder_path</code>	Switches the current working directory to the one specified in the path.
<code>ls</code>	Lists the files in the current working directory.
<code>rm file_path</code>	Removes the file corresponding with the file path.
<code>nvidia-smi</code>	Lists the current NVIDIA processes. Makes it easy to find the PID to terminate the process.
<code>kill PID_name</code>	Ends the process corresponding to the input PID.
<code>nohup python file_name &amp;</code>	Runs the Python program corresponding to the file name in the background and saves all prints to <code>nohup.out</code> .
<code>git pull</code>	Pulls repository updates from GitHub.
<code>git commit -am "message"</code>	Commits all local changes with the commit message.
<code>git push</code>	Pushes local commits to the GitHub repository.

## 2.7 A New Loss Function: Binary Cross Entropy (BCE)

Another variable introduced in this study is a new loss function. Mean squared error (MSE) is the loss function used in the original model, based on the setup from the original GAN paper [5]. In this experiment, binary cross entropy (BCE) is substituted for MSE. The formula for BCE is given by:

$$H(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)] \quad (2.4)$$

Where:

$y$  : The true binary labels.

$\hat{y}$  : The predicted probabilities.

$N$  : The number of samples.

$y_i$  : The true label of the  $i$ -th sample.

$\hat{y}_i$  : The predicted probability of the  $i$ -th sample being in the positive class.

BCE is well suited for classification based tasks, while MSE is best suited for regression. This is due to how the loss functions are derived. MSE assumes the error follows a normal

distribution, while BCE assumes the error follows the binomial distribution. The goal of the discriminator is to predict a binomial outcome: real or fake. Each output feature from the discriminator represents the probability that it is real or fake. Therefore, the discriminator is attempting to classify each entry on the feature map. By using BCE instead of MSE, the model is calculating the GAN loss as a classification task rather than a regression task, which better matches the task at hand.

Additionally, BCE comes with several advantages. One of which is it is more sensitive to outliers. Fluence maps can have specific areas with much higher dose relative to its surroundings to deliver locally focused dose. We want to capture these regions as well as possible and therefore want the model to be sensitive to them. Additionally, BCE can come with reduced training and convergence times, as the gradient between predicted values and true values is larger in BCE than in MSE. Additionally, this can enable the model to learn more once the predicted values become closer to true values, as the larger gradient will give the optimizer more information to update the model parameters with.

## ***2.8 A New Loss Function: Mean Absolute Error (MAE)***

Another pivotal aspect introduced in this investigation is the utilization of MAE loss. While the wavelet loss works well alone, MAE was introduced as a way to improve conformity in the low-dose regions. The formula for MAE is represented as:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \tag{2.5}$$

Where:

$y$  : The true labels.

$\hat{y}$  : The predicted values.

$N$  : The total number of samples.

$y_i$  : The true value of the  $i$ -th sample.

$\hat{y}_i$  : The predicted value of the  $i$ -th sample.

Employing MAE offers several advantages. Firstly, MAE does not apply a strong penalty to outliers. Modulated fluence maps can have outliers, and thus it is important to account for this. Secondly, the use of MAE can contribute to faster convergence and training times, as it tends to exhibit less oscillation during optimization compared to other loss functions. Thirdly, using wavelet alone led to elevated background fluence, and MAE can enable the model to better capture the low-dose regions. To summarize, MAE enables the model to converge efficiently and learn from the training data more effectively, ultimately improving the overall quality of generated outputs.

## **2.9 Comparing Performance Between the Models**

### **2.9.1 Similarity Metrics**

This study uses two metrics to compare the models in terms of similarity to the ground truth. The Structural Similarity Index (SSIM) is the first metric used, which is given by:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (2.6)$$

Where:

$x, y$  : The two images being compared.

$\mu_x, \mu_y$  : The average pixel values of  $x$  and  $y$ , respectively.

$\sigma_x, \sigma_y$  : The standard deviations of pixel values in  $x$  and  $y$ , respectively.

$\sigma_{xy}$  : The covariance of pixel values between  $x$  and  $y$ .

$C_1, C_2$  : Constants.

The SSIM is a common benchmark used to check image processing algorithms by comparing the processed image to the original. In this case, we use it to compare the predicted image to the ground truth.

The next metric is the normalized Mean Absolute Error (MAE). The normalized Mean

Absolute Error (MAE) is given by:

$$\text{MAE}_{\text{normalized}}(x, y) = \frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N \frac{|x_{ij} - y_{ij}|}{\max(y)} \quad (2.7)$$

Where:

$x$  : The predicted image.

$y$  : The reference image.

$M$  : Number of rows.

$N$  : Number of columns.

$x_{ij}$  : The pixel value at position  $(i, j)$  in the predicted image.

$y_{ij}$  : The pixel value at position  $(i, j)$  in the reference image.

While SSIM is designed to prioritize luminance, contrast, and structure, MAE, directly concentrates on the absolute difference between pixel values. This approach provides a straightforward insight into the difference between fluence map pixel intensities.

## 2.9.2 Training Performance

We use the validation loss vs epoch number to compare training performance. The model aims to decrease the loss function, and thus tracking the validation loss with respect to the epoch number gives insight into both how well the model is performing, and how quickly it is training.

## 2.9.3 Plan Quality

There are many metrics to choose from when it comes to assessing plan quality. It is vital to understand how diffusion impacts total dose, PTV dose coverage, and OAR dose. To measure total dose, two important metrics are 3D max dose 2cc (D2cc) and total MUs. D2cc is the maximum dose delivered to a 2 centimeter cubed region, and total MUs being the total amount of MUs delivered throughout treatment. D2cc is effective in measuring how well the plan reduces high doses delivered to the patient, and total MUs

is effective in measuring delivery uncertainty, as more MUs lead to more scattering in the linear accelerator head as well as leakage dose. Next, for PTV dose coverage, we measure the volume receiving 95% of the prescription dose normalized to 44 Gy (PTV44 D95%). This metric shows how well the plans cover the PTV with enough dose to effectively treat the patient. Finally, in HN cases, the parotids are considered to be of high importance. This study uses the median left and right parotid dose as the OAR dose metric. Additionally, the larynx, pharynx, mandible, and oral cavity are also tested, as they are considered important in HN cases as well.

### **2.9.4 Visualization**

To view the outputs of the models and compare them to the ground truth, the four fluence maps (script plan, original GAN model, with-diffusion, no-diffusion) are plotted side-by-side for three randomly selected patients at three randomly selected beam angles. To provide a consistent comparison between the four fluence maps, the color bar is set up to go up to the maximum value from all four maps, and all four maps use the same coloring scale.

SSIM and MAE are visualized using tables containing their mean value. These give insight into the differences in performance between the models when it comes to producing outputs that are similar to the ground truth.

The visualization of validation loss involves creating a line plot that depicts the validation loss against the epoch number. The goal is to assess whether the implementation of diffusion can lead to a reduction in training time and training loss. The original model, which uses a different GAN loss function (MSE), is omitted from this visualization to maintain a focused comparison between the with diffusion and no-diffusion models.

The plan quality metrics are visualized using box plots, as well as a table listing their mean value. This gives insight into the plan quality, as the ultimate goal is to produce high-quality treatment plans. A DVH plot and two isodose plots are generated to further show the dose distribution to the OAR and PTV.

### 3. Results

#### 3.1 Fluence Map Comparisons

Of three randomly selected fluence maps, all three models appear visually similar to the ground truth.

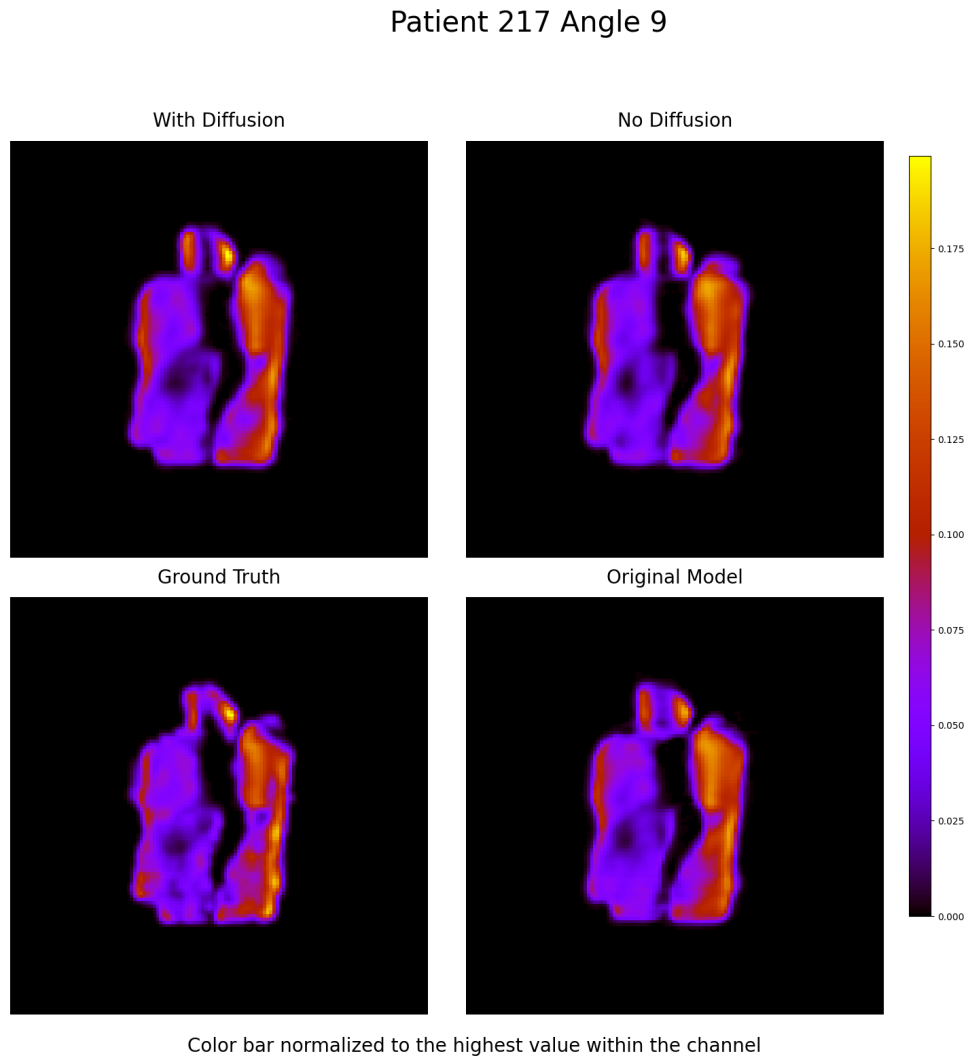


FIGURE 3.1: Fluence Map Comparison for Patient 217 at the 9th beam angle.

### Patient 218 Angle 5

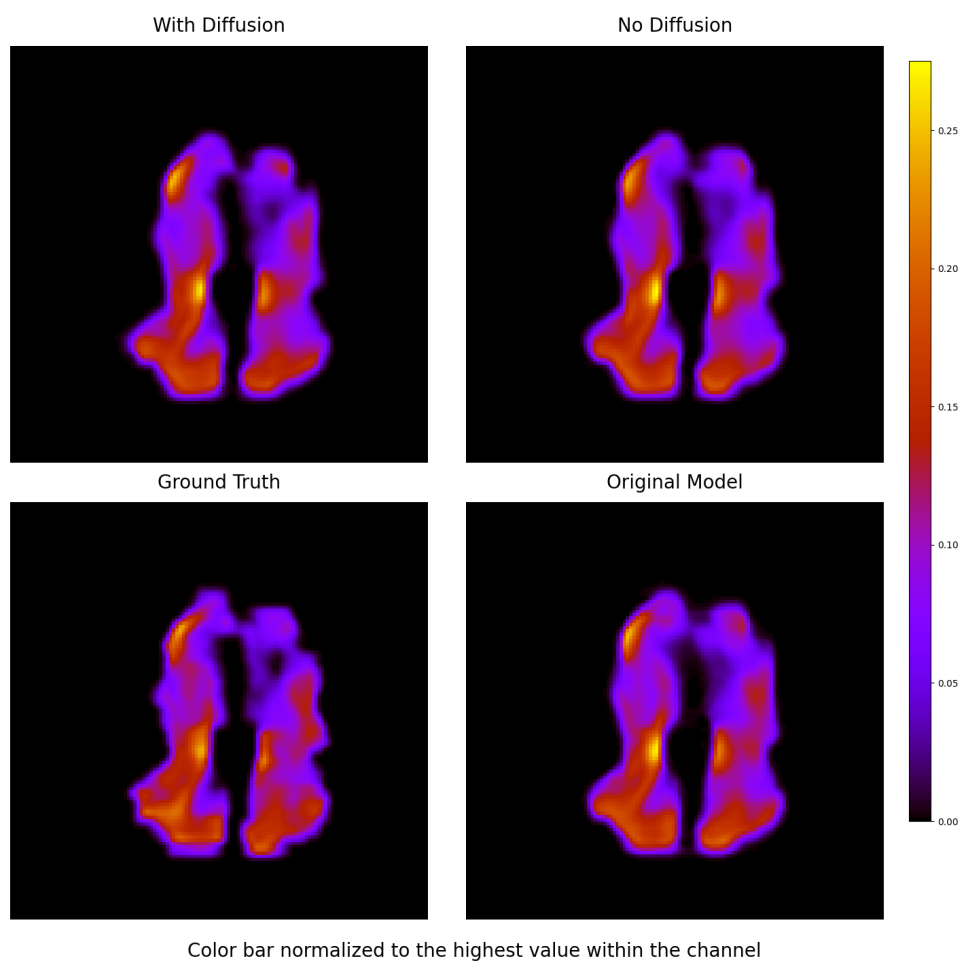


FIGURE 3.2: Fluence Map Comparison for Patient 218 at the 5th beam angle.

Patient 229 Angle 4

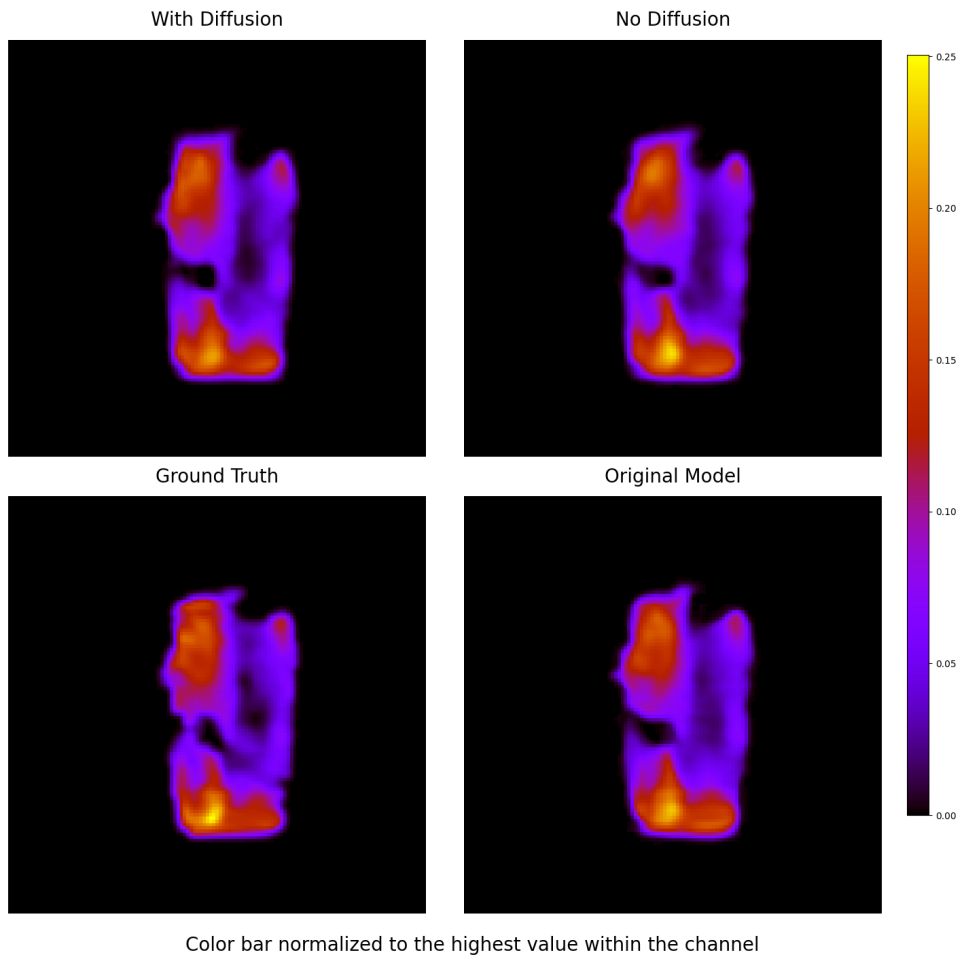


FIGURE 3.3: Fluence Map Comparison for Patient 229 at the 4th beam angle.

### 3.2 Similarity Metrics

The mean SSIM and MAE are comparable between the three models.

Table 3.1: Comparison of SSIM and MAE for Different Models

Model	SSIM	MAE
Original Model	0.978	0.011
No-Diffusion Model	0.978	0.011
With-Diffusion Model	0.978	0.011

### 3.3 Validation Losses

The validation loss was similar between the two new models, the with-diffusion model achieved a slightly lower validation loss.

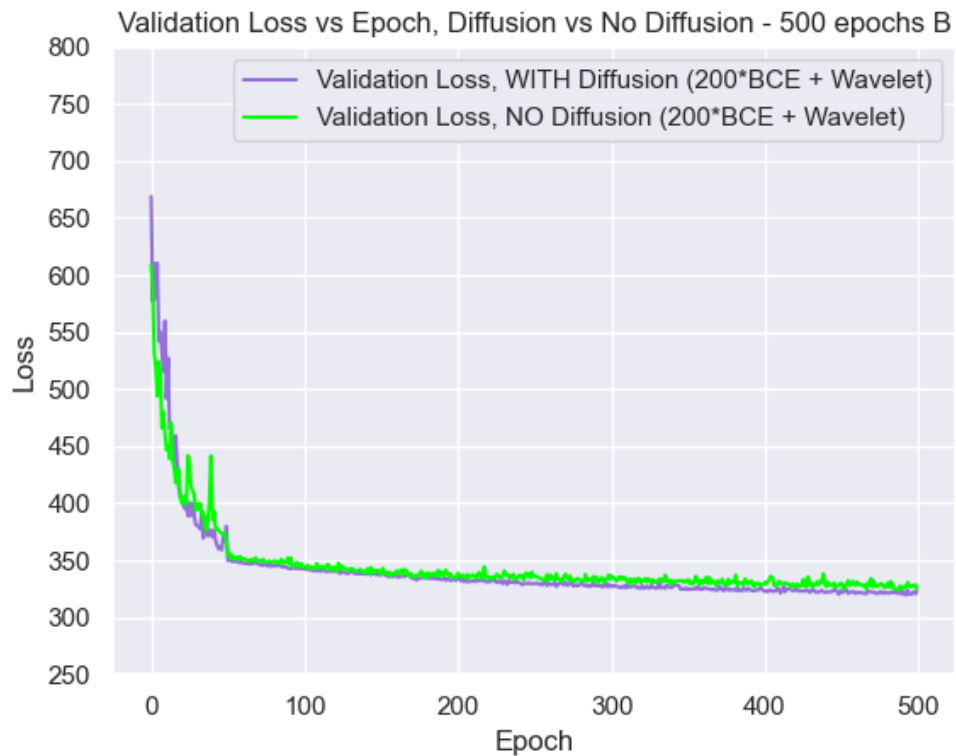


FIGURE 3.4: Validation loss vs epoch number between the diffusion and no-diffusion models.

### 3.4 Plan Quality Metrics

#### 3.4.1 Total Dose

On average, the with-diffusion model achieved the lowest D2cc (4% less than the original model and 1% less than the no-diffusion model) and lowest total MUs (23% less than the original model and 3% less than the no-diffusion model).

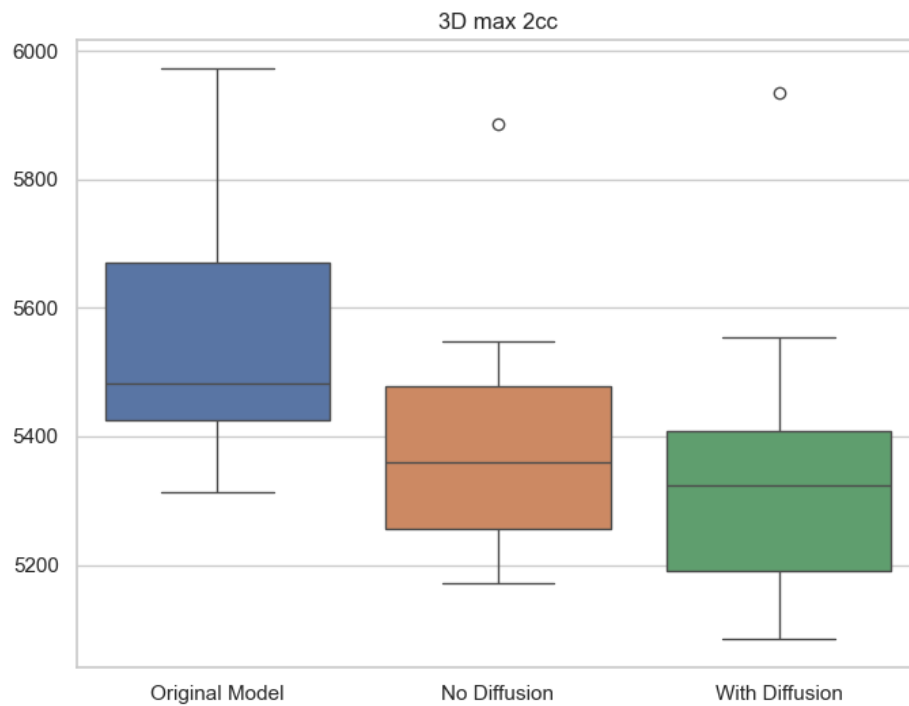


FIGURE 3.5: Maximum dose delivered to a 2 centimeter cubed region.

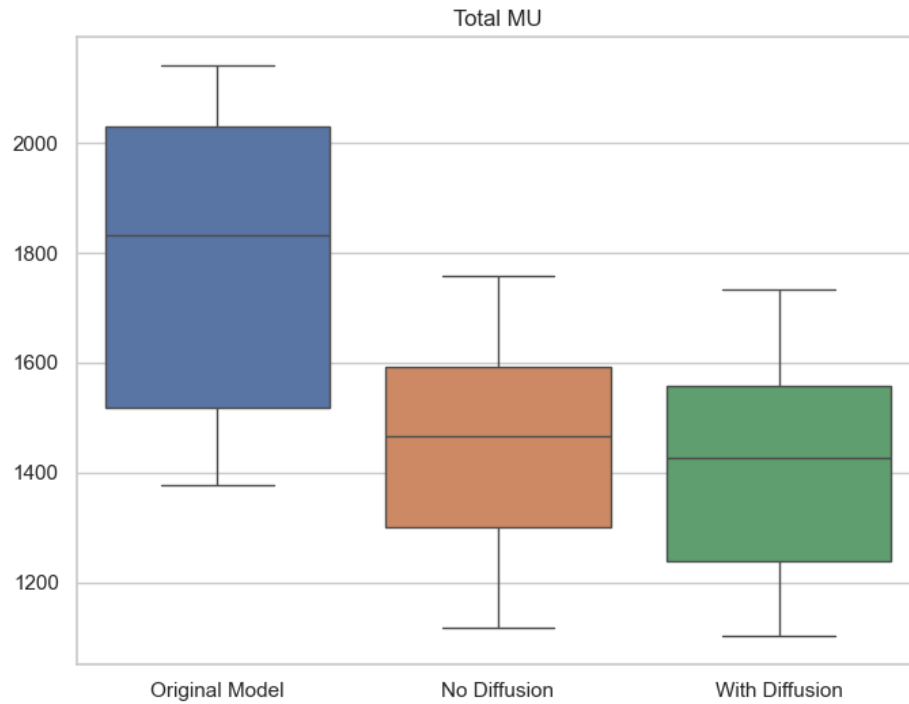


FIGURE 3.6: Total MUs delivered by the plan.

Table 3.2: Difference and Percent Differences Compared to the original model for the dose statistics.

	<b>Difference (cGy)</b>	<b>Percent Difference (%)</b>
3D Max 2cc		
No-Diffusion	-153 cGy	-2.80%
With-Diffusion	-206 cGy	-3.78%
Total MU		
No-Diffusion	-329 cGy	-20.31%
With-Diffusion	-371 cGy	-23.21%

### 3.4.2 PTV Coverage

All three plans had comparable PTV coverage normalized to 44Gy, with the two new models slightly outperforming the original model.

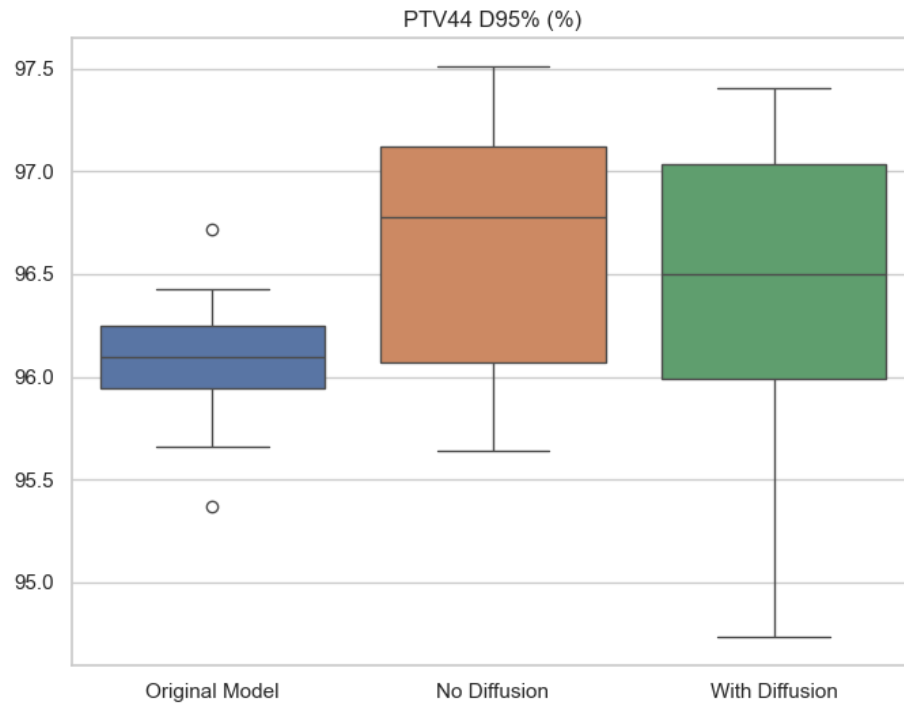


FIGURE 3.7: Volume receiving 95% of the prescription dose normalized to 44Gy (PTV44 D95%).

### 3.4.3 OAR Sparing

The new models deliver lower median dose to the left and right parotids.

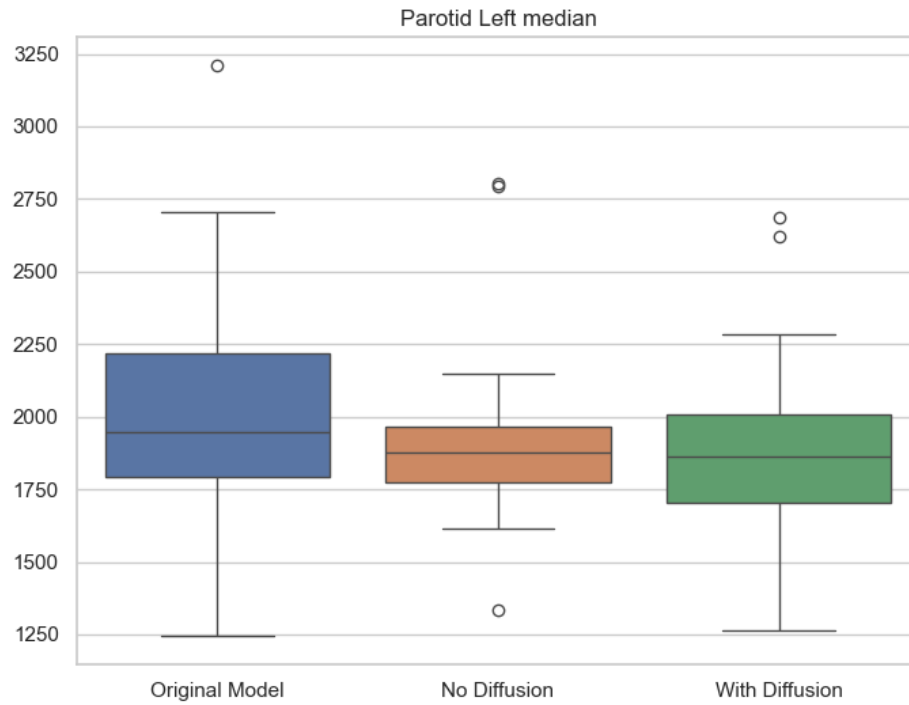


FIGURE 3.8: Median dose to the left parotid.

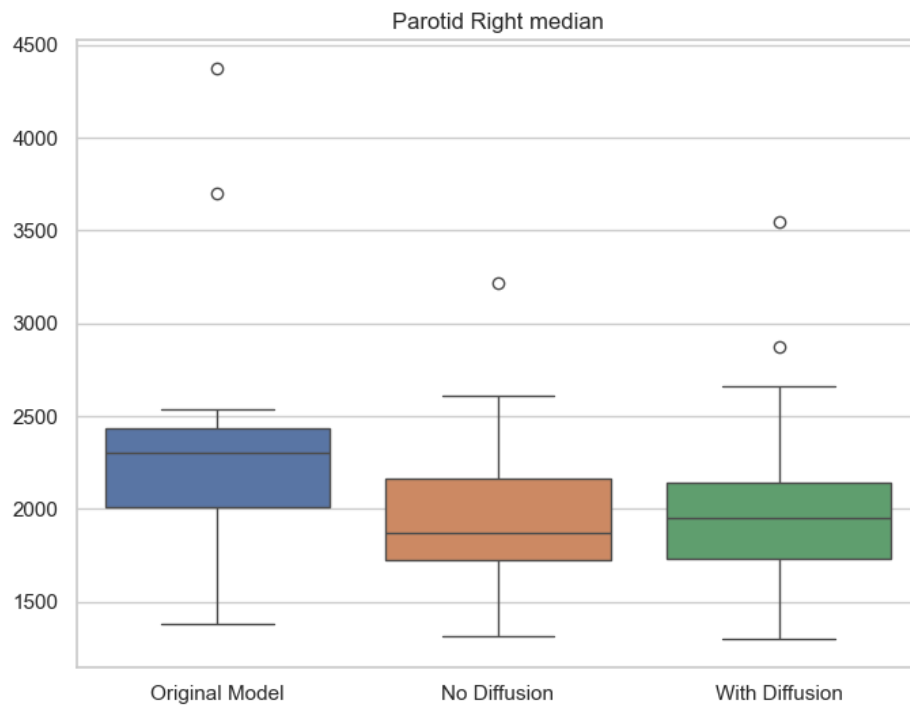


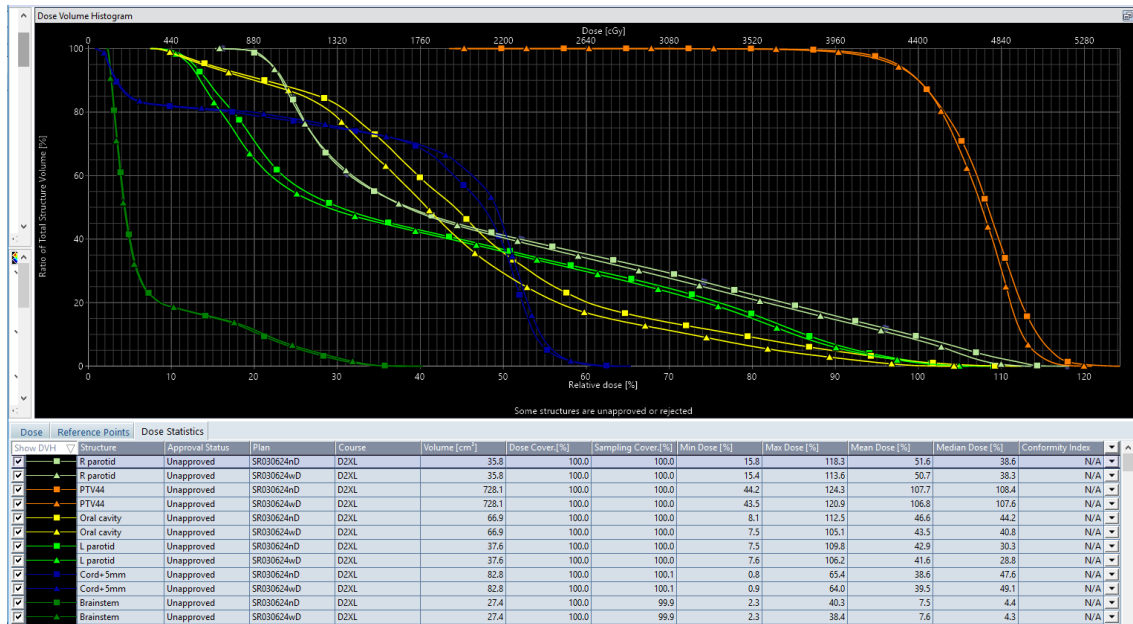
FIGURE 3.9: Median dose to the right parotid.

Of the six tested OAR, the new models deliver lower dose to four of the six (67%). Compared to the no-diffusion model, the with-diffusion model delivers a lower dose to five of the six tested OAR (83%).

Table 3.3: Difference and percent differences compared to the original model for the OAR.

	<b>Difference (cGy)</b>	<b>Percent Difference (%)</b>
<b>Right Parotid median</b>		
No-Diffusion	-383 cGy	-16.24%
With-Diffusion	-313 cGy	-13.43%
<b>Left Parotid median</b>		
No-Diffusion	-129 cGy	-6.23%
With-Diffusion	-171 cGy	-8.48%
<b>Larynx median</b>		
No-Diffusion	153 cGy	7.44%
With-Diffusion	162 cGy	7.43%
<b>Pharynx median</b>		
No-Diffusion	158 cGy	4.32%
With-Diffusion	151 cGy	4.10%
<b>Oral Cavity median</b>		
No-Diffusion	-142 cGy	-3.57%
With-Diffusion	-189 cGy	-7.64%
<b>Mandible 1cc</b>		
No-Diffusion	-124 cGy	-2.52%
With-Diffusion	-176 cGy	-3.66%

The dose volume histogram plot demonstrates the performance gain diffusion provides relative to no-diffusion, with consistently comparable or lower dose for the PTV and OAR.





 - No-Diffusion model  
 - With-Diffusion model

FIGURE 3.10: Dose volume histogram comparing the with-diffusion and no-diffusion models.

The isodose lines provide a further visualization of the smoother fluence modulation the newer models deliver, and the reduction in dose that diffusion provides.

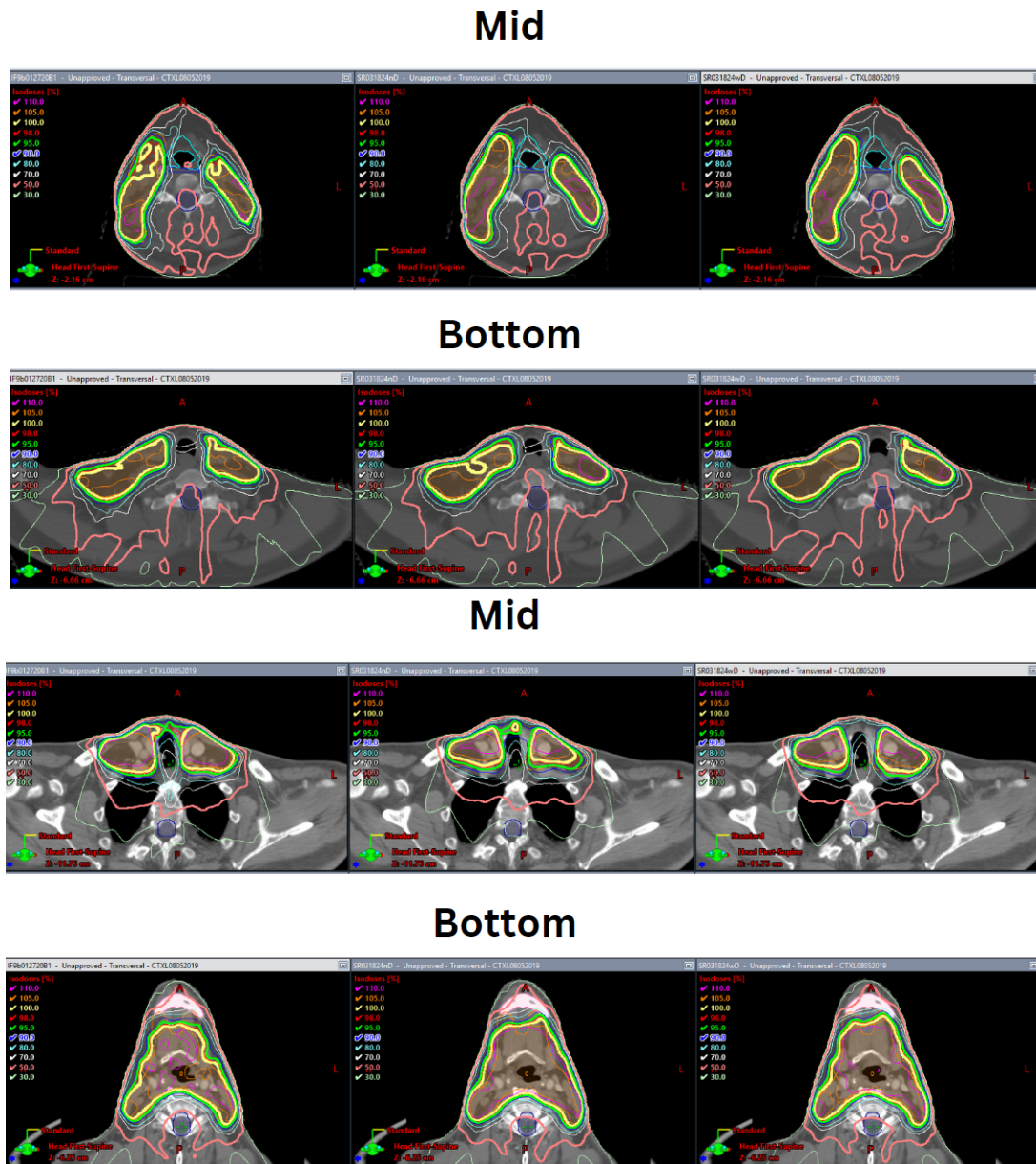


FIGURE 3.11: Superimposed isodose lines for two patients. Column 1: original model, Column 2: no-diffusion model, Column 3: with-diffusion model.

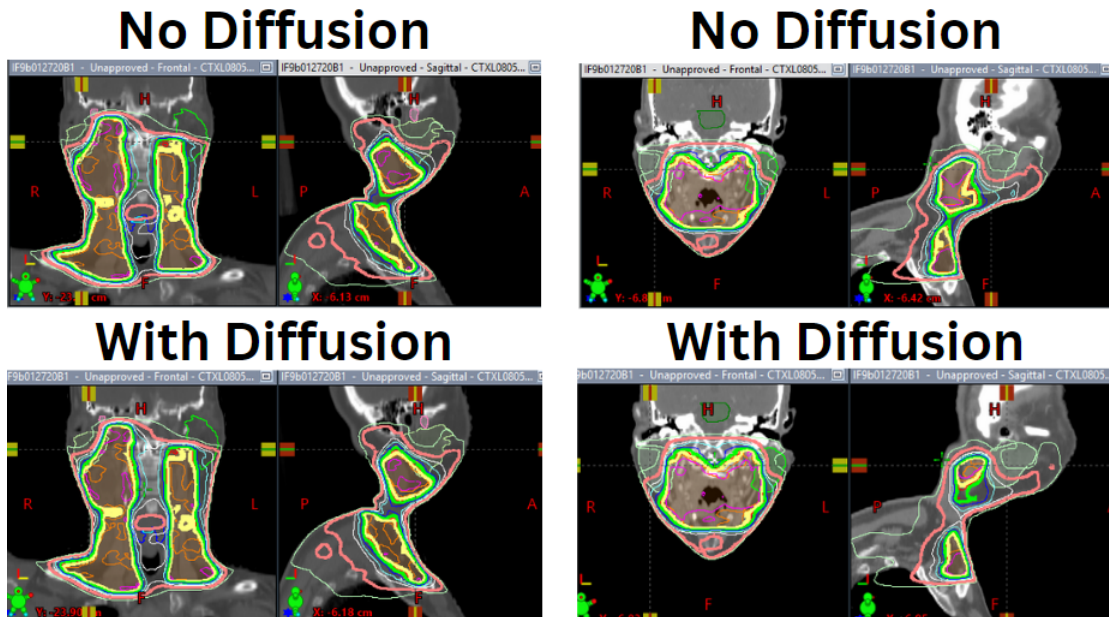


FIGURE 3.12: Superimposed isodose lines for two patients (frontal and sagittal).

## **4. Discussion**

### ***4.1 Fluence Map Similarity***

From the eye test alone, it is hard to tell which of the three models is producing higher quality plans in section 3.1. All three appear to conform well to the ground truth and appear similar.

This gets confirmed using the similarity metrics in Table 3.1. All three models score the same in SSIM and MAE. This demonstrates how pixel by pixel, there is not much difference in terms of the maps appear.

### ***4.2 Validation Loss Analysis***

Figure 3.4 shows very little difference between the training tendencies between no-diffusion and with-diffusion. The generator uses two loss functions: wavelet and BCE. The diffusion process prevents the discriminator from getting strong, and therefore it is easier for the generator GAN loss to be lower in the diffusion model. The small difference between the losses from the no-diffusion and diffusion models comes from the difference in the BCE loss, the wavelet losses are comparable. Therefore, this small difference in the validation losses can be attributed to the diffusion making the discriminator weaker, not to the model producing better outputs.

### ***4.3 Plan Quality***

#### **4.3.1 Total Dose**

The new models deliver fantastic results in both D2cc and total MU's (Figures 3.5 and 3.6). The with-diffusion slightly outperforms the no-diffusion model, and outperforms the original model by a large margin, on average delivering 206 cGy less dose, or roughly 4% less. This margin becomes even wider in the total MU's, where the with-diffusion model delivers 371 fewer MU's than the original model on average, or 23% less. The with-diffusion model also slightly outperforms the no-diffusion model in this metric as well, delivering 3% less MU on average.

The reduction in D2cc is significant so long as the PTV dose coverage is suitable. Less dose to the patient means less recurrence, a lower risk of radiation skin burns, and a lower risk of acute radiation toxicity. Similarly, for total MU's, this reduction means less radiation scatter in the head of the linear accelerator, which will reduce treatment uncertainty and dose to the patient. These metrics show great promise that diffusion combined with the new loss functions is generating better treatment plans that are safer for the patients.

### **4.3.2 PTV Coverage**

All three models have comparable PTV44 D95% scores (Figure 3.7), indicating they all deliver similar dose coverage to the PTV. Both of the new models appear to slightly outperform the original model in this metric, however, it does not seem diffusion leads to an improvement in this metric. This indicates that the new loss functions have enabled better PTV dose conformity, which once again indicates the new models are creating higher-quality treatment plans.

### **4.3.3 OAR Sparing**

The new models deliver significantly less dose to the left and right parotids compared to the original model (Table 3.3). The parotids are very radiosensitive. In HN cases, high dose to these frequently leads to patients having issues with salivation. This effect is extremely uncomfortable and can cause patients to have dry mouth and even require a feeding tube. Thus, sparing the parotids is vital to preserving the quality of life for HN patients. There is no indication that diffusion improved the dose to parotids, however, the data shows the new loss functions enable much better parotid sparing. As OAR sparing is a key aspect of plan quality, this indicates the new models are producing better treatment plans.

Of the remaining four OAR investigated, the new models delivered lower dose to two of them. Why the larynx and pharynx receive higher dose on average while the rest of the OAR receive lower dose is unknown. While this would be a good question to investigate, and hopefully reduce in the future, the lower dose to four of six tested OAR (and to the two most important OAR) shows the new loss functions work well at reducing OAR

dose. The isodose plots in Figure 3.11 show a clear improvement in the smoothness fluence modulation from the original model to the no-diffusion model, specifically in the bottom slice of the second patient and the mid slice of the first patient. There is also a noticeable improvement in dose sparing in the with-diffusion model over the no-diffusion model, most notably observed in the mid slice of the second patient. Additionally, it should be noted that, while in this figure it appears both the larynx and pharynx are spared more in the new models, they are spared better on average with the original model (as seen in Table 3.2). Further work should be done to investigate why the original model does better with these two organs, and how the new models can be modified to improve dose sparing to them.

Comparing the two new models, the with-diffusion model delivered a lower OAR dose to five of six tested OAR. The DVH plot in Figure 3.10 makes this clear, as the with-diffusion model's dose for the OAR and PTV stay below the no-diffusion model's throughout a majority of the volume. This is a strong indication that diffusion works to reduce OAR and PTV dose, and thus improves overall plan quality.

#### ***4.4 Clinical Use***

The current in-house method for generating treatment plans uses the original model base, with some included pre and post processing steps. This is used frequently in the clinic and saves hours of planning time by bypassing many of the normal planning steps. Using the with diffusion model from this study in place of the original model would improve the quality of the output plans by reducing OAR, leakage, and total dose, as well as allowing for smoother fluence modulation. Simply swapping the models and keeping all else constant will allow for planners to continue using the tool as usual, but with better output treatment plans. This does not need FDA approval, and since the original model was already clinically approved, there is no reason to believe this model would not get approved for clinical use.

## ***4.5 Limitations and Future Research***

The biggest limitation of this study is the sample size. 216 patients for training and validation is a good start, but neural networks get better as more data is added. Expanding the dataset is likely to cause an improvement in plan generation without the need to change anything with the models.

Future research should also be done to assess training time between the no-diffusion model and the original model. An advantage of PyTorch is how much more customizable the training loops are. Instead of training a combined model and a discriminator model, which requires two generator forward passes like in the Tensorflow version, the PyTorch approach does only one forward pass through the generator. By leveraging PyTorch's detach function, we can pass the synthetically generated data through the discriminator during its training phase without gradients flowing back through the generator, and then use the undetached generator output to train the generator separately. In theory, this should provide a significant cut to training time each epoch, however, it is difficult to perform a fair assessment as to how much due to the differences in the packages. Investigating this would allow for additional proof that the new models are an upgrade.

Additionally, an investigation should be done into weighting the generator loss functions. This study uses a 200 to 1 to 1 weighting for BCE, wavelet, and MAE respectively. It is possible that optimizing these could allow for better stability in training, faster training time, lower validation loss, and better outputs.

## 5. Conclusion

Diffusion does not lead to a significant improvement in training time and training stability. While providing comparable fluence map similarity and PTV dose coverage, both the no-diffusion and with-diffusion models deliver less total dose and less OAR dose. The with-diffusion model outperforms the no-diffusion model in OAR dose, indicating diffusion also works to reduce OAR dose. The with-diffusion model and the no-diffusion model outperform the original model in the total dose metrics, and the with-diffusion model slightly outperforms the no-diffusion model in these metrics as well. These results indicate the with-diffusion model (the new loss functions in conjunction with-diffusion) results in the highest plan quality of the tested models. The reduction in total dose, as well as dose to the parotids, oral cavity, and mandible with this model, means a lower risk of adverse side effects, a lower risk of recurrence, and a better quality of life for patients. Therefore, the results of this study show a noticeable improvement in treatment plan quality as a result of the variables tested.

## Bibliography

- [1] en. cgvArticle. June 2021. URL: <https://www.cancer.gov/types/head-and-neck/head-neck-fact-sheet>.
- [2] en. 2023. URL: <https://www.cancer.net/cancer-types/head-and-neck-cancer/statistics>.
- [3] Freddie Bray et al. "Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries". In: *CA: A Cancer Journal for Clinicians* 68.6 (2018), pp. 394–424. DOI: <https://doi.org/10.3322/caac.21492>. eprint: <https://acsjournals.onlinelibrary.wiley.com/doi/pdf/10.3322/caac.21492>. URL: <https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.3322/caac.21492>.
- [4] CDC. *Changes Over Time: Oral Cavity and Pharynx*. URL: <https://gis.cdc.gov/Cancer/USCS/#/Trends/>.
- [5] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].
- [6] Xinyi Li et al. "An artificial intelligence-driven agent for real-time head-and-neck IMRT plan generation using conditional generative adversarial network (cGAN)". en. In: *Med. Phys.* 48.6 (June 2021), pp. 2714–2723.
- [7] Rebecca L Siegel et al. "Cancer statistics, 2021". en. In: *CA Cancer J. Clin.* 71.1 (Jan. 2021), pp. 7–33.
- [8] Zhendong Wang et al. "Diffusion-GAN: Training GANs with Diffusion". In: *arXiv preprint arXiv:2206.02262* (2022). URL: <https://arxiv.org/abs/2206.02262>.