Deep Learning Assignment - 3

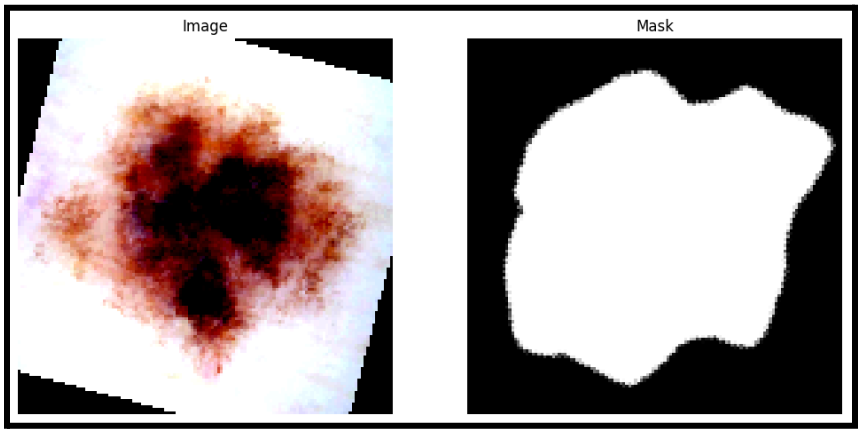
m23csa011.ipynb

Name : Kushal Agrawal **Roll No. : M23CSA011**

Dataset Understanding:

The dataset includes:

• train: 900 training images. (128X128)
• train masks: Segmented masks for training images.
• test: 379 test images.
• test masks: Segmented masks for test images.



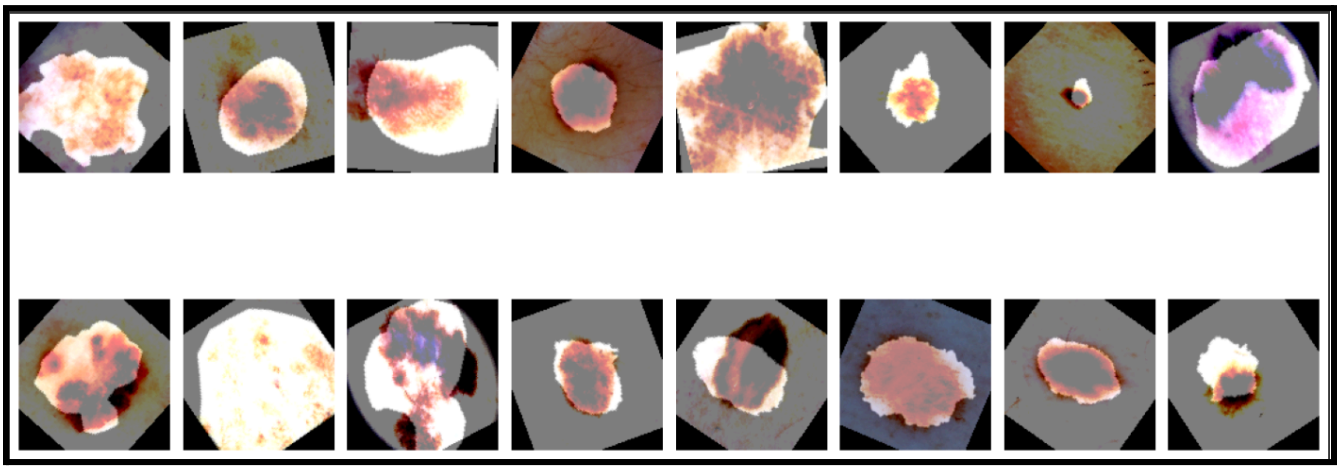
Pre-Processing :

Dataset Class Definition (ISICDataset):

- This class defines a custom PyTorch dataset for the ISIC (International Skin Imaging Collaboration) dataset, which typically consists of skin lesion images and corresponding segmentation masks.
- It takes input parameters such as img_dir (directory containing images), mask_dir (directory containing masks), image_transform (transforms to apply to images), and mask_transform (transforms to apply to masks).
- In the __init__ method, it initializes the directories, image and mask lists, and transformation functions.
- The __len__ method returns the total number of mask files in the dataset.
- The __getitem__ method loads an image and its corresponding mask, applies transformations if specified, and returns them as tensors.

Dataset Initialization:

- The code initializes directories for training and testing images and masks.
- It defines transformation pipelines for both images and masks, including resizing, random horizontal flipping, random rotation, converting to tensors, and normalization (only for images).



<u>Task 1:-</u>

Design a custom decoder atop this encoder for the segmentation task.

Convolutional Layers:
self.conv1:
Input channels: 1280 (Assuming this matches the output channels of the MobileNetV2 encoder).
Output channels: 512.
Kernel size: 3x3.
Padding: 1.

self.conv2, self.conv3, self.conv4, self.conv5:
Successive convolutional layers with decreasing output channels from 512 to 16.
Each layer has a kernel size of 3x3 and padding of 1.

Batch Normalization Layers:
Batch normalization layers (self.bn1, self.bn2, self.bn3, self.bn4, self.bn5) follow each convolutional layer.
Helps in stabilizing and accelerating the training process by normalizing the activations.

Activation Functions:
ReLU activation functions (self.relu1, self.relu2, self.relu3, self.relu4, self.relu5) are applied after each batch normalization layer. Introduces non-linearity to the network.

Upsampling Layers:
self.upsample1, self.upsample2, self.upsample3, self.upsample4, self.upsample5:
Upsampling layers using bilinear interpolation with a scale factor of 2.
Doubles the spatial dimensions of the feature maps.

Final Convolution Layer:
self.final_conv:
Input channels: 16 (Output channels of the last convolutional layer).
Output channels: num_classes (Number of segmentation classes, 1 in this case for binary segmentation).
Kernel size: 1x1.
No padding.
Followed by a sigmoid activation function (self.sigmoid).

Dropout Layer:
Applied after each upsampling operation (self.dropout).
Helps in preventing overfitting by randomly dropping a fraction of input units during training.

Decoder Class:

Initialization (__init__):
Initializes convolutional layers, batch normalization layers, activation functions, upsampling layers, dropout layer, and final convolution layer.

Parameters:
num_classes: Number of classes for segmentation output.

Forward Pass (forward):
Performs a series of convolutional, batch normalization, activation, and upsampling operations.
Features from the encoder are passed through the decoder layers.
The output is passed through a final convolution layer followed by a sigmoid activation function, which produces the segmentation mask.

SegmentationModel Class:

Initialization (__init__):
Initializes with an encoder (MobileNetV2) and a decoder.
Parameters:
encoder: Pre-trained MobileNetV2 encoder.

decoder: Custom decoder module.
Forward Pass (forward):
Takes input x.
Passes input through the encoder to extract features.
Passes the extracted features through the decoder to generate the segmentation mask.
Returns the segmentation mask.

Main Code:

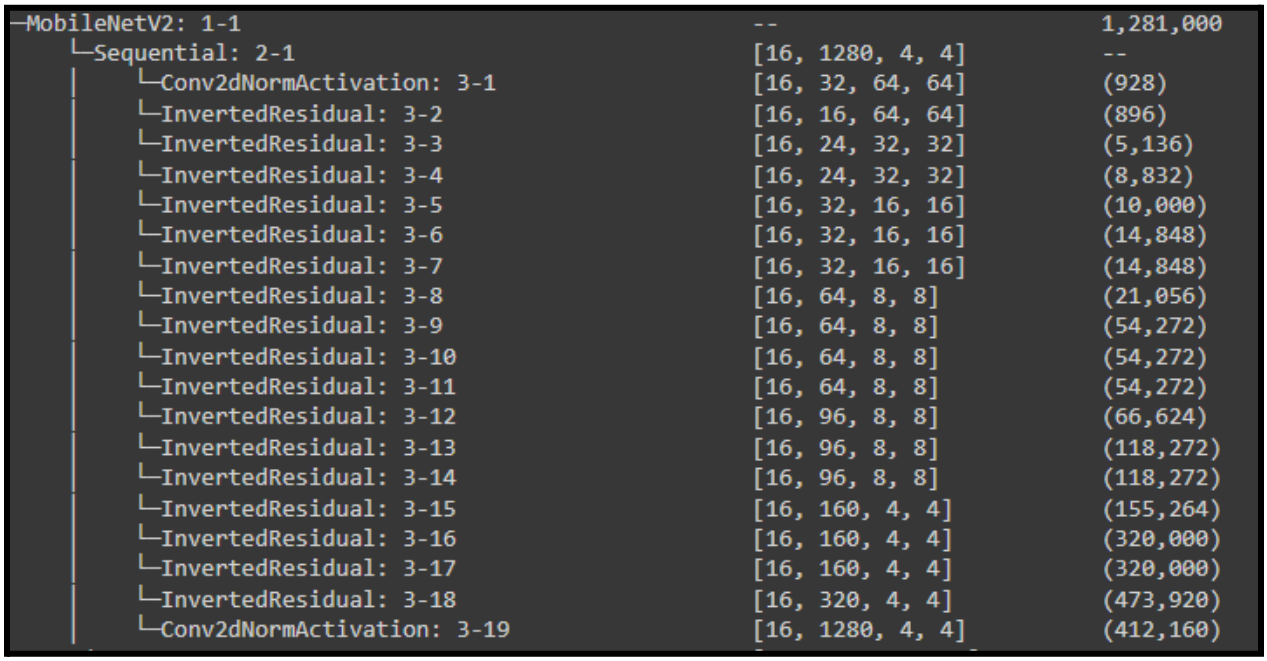
Encoder Initialization:
Initializes a MobileNetV2 encoder pretrained on ImageNet.

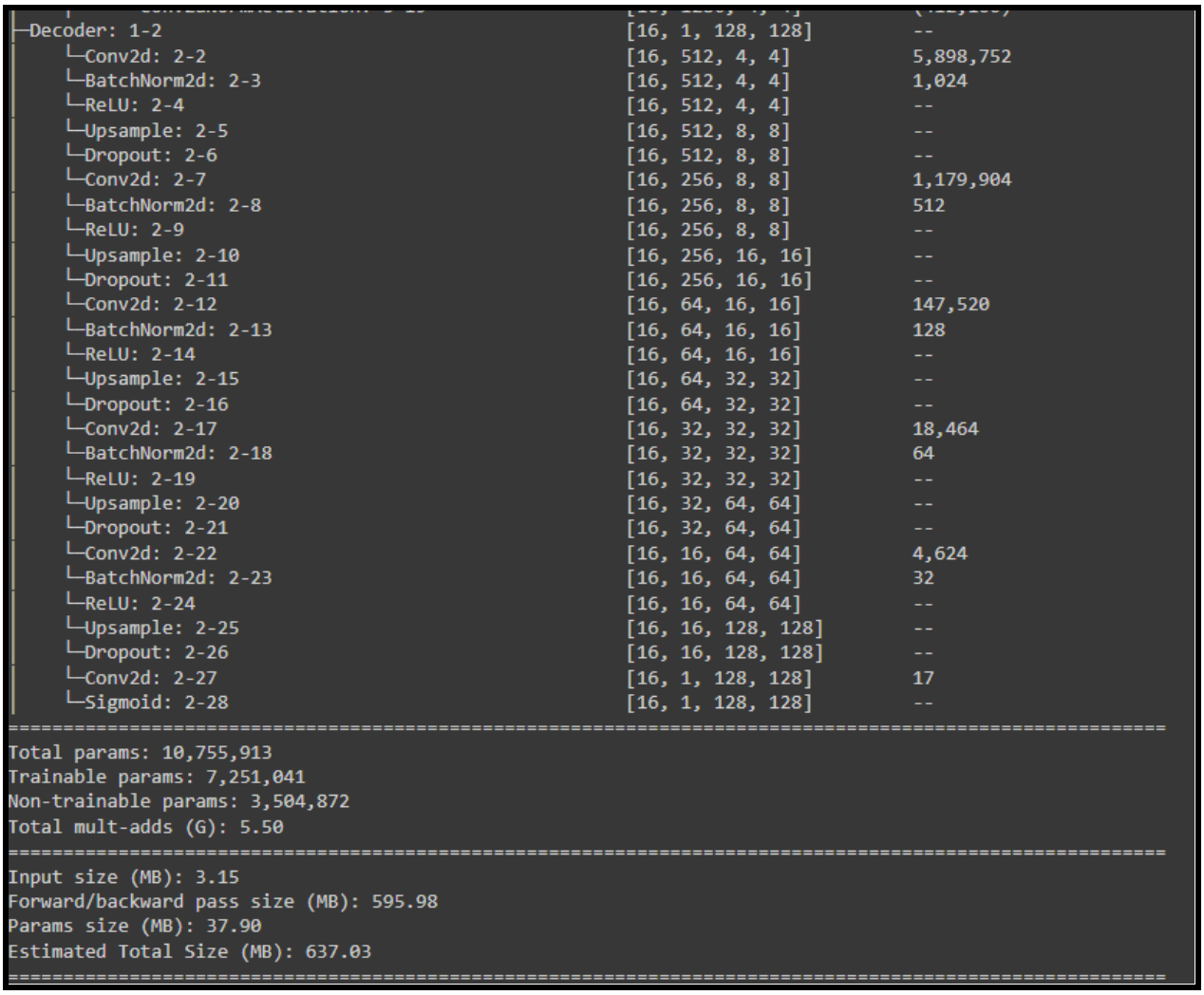
Freezing Encoder Parameters:
Freezes all parameters of the encoder to prevent them from being updated during training.

Decoder Initialization:
Initializes the custom decoder with the specified number of classes (1 in this case).

SegmentationModel Initialization:
Initializes the segmentation model with the MobileNetV2 encoder and custom decoder.

Model Summary:
Uses torchinfo.summary to print a summary of the model, including input size and the number of parameters.





```
├─Decoder: 1-2                                  [16, 1, 128, 128]         --
│    └─Conv2d: 2-2                              [16, 512, 4, 4]           5,898,752
│    └─BatchNorm2d: 2-3                         [16, 512, 4, 4]           1,024
│    └─ReLU: 2-4                                [16, 512, 4, 4]           --
│    └─Upsample: 2-5                            [16, 512, 8, 8]           --
│    └─Dropout: 2-6                             [16, 512, 8, 8]           --
│    └─Conv2d: 2-7                              [16, 256, 8, 8]           1,179,904
│    └─BatchNorm2d: 2-8                         [16, 256, 8, 8]           512
│    └─ReLU: 2-9                                [16, 256, 8, 8]           --
│    └─Upsample: 2-10                           [16, 256, 16, 16]         --
│    └─Dropout: 2-11                            [16, 256, 16, 16]         --
│    └─Conv2d: 2-12                             [16, 64, 16, 16]          147,520
│    └─BatchNorm2d: 2-13                        [16, 64, 16, 16]          128
│    └─ReLU: 2-14                               [16, 64, 16, 16]          --
│    └─Upsample: 2-15                           [16, 64, 32, 32]          --
│    └─Dropout: 2-16                            [16, 64, 32, 32]          --
│    └─Conv2d: 2-17                             [16, 32, 32, 32]          18,464
│    └─BatchNorm2d: 2-18                        [16, 32, 32, 32]          64
│    └─ReLU: 2-19                               [16, 32, 32, 32]          --
│    └─Upsample: 2-20                           [16, 32, 64, 64]          --
│    └─Dropout: 2-21                            [16, 32, 64, 64]          --
│    └─Conv2d: 2-22                             [16, 16, 64, 64]          4,624
│    └─BatchNorm2d: 2-23                        [16, 16, 64, 64]          32
│    └─ReLU: 2-24                               [16, 16, 64, 64]          --
│    └─Upsample: 2-25                           [16, 16, 128, 128]        --
│    └─Dropout: 2-26                            [16, 16, 128, 128]        --
│    └─Conv2d: 2-27                             [16, 1, 128, 128]         17
│    └─Sigmoid: 2-28                            [16, 1, 128, 128]         --
==========================================================================================
Total params: 10,755,913
Trainable params: 7,251,041
Non-trainable params: 3,504,872
Total mult-adds (G): 5.50
==========================================================================================
Input size (MB): 3.15
Forward/backward pass size (MB): 595.98
Params size (MB): 37.90
Estimated Total Size (MB): 637.03
==========================================================================================
```

The parameters of the Segmentation Model is :
Total params: 10,755,913
Trainable params: 7,251,041 (decoder weights)
Non-trainable params: 3,504,872 (encoder weights)

SegmentationMetrics Class:

- __init__: Initializes the segmentation metrics class with a threshold value for converting probabilities to binary predictions.
- _convert_to_binary: Converts the probabilities to binary predictions based on the specified threshold.
- intersection_over_union: Calculates the Intersection over Union (IoU) metric between the predicted masks and the ground truth masks.
- dice_score: Calculates the Dice Score metric between the predicted masks and the ground truth masks.

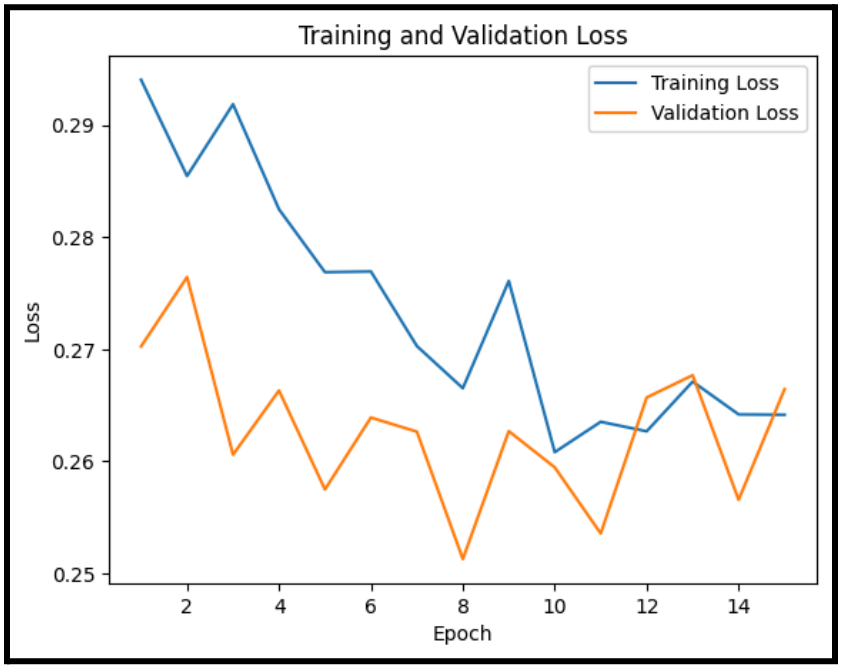
Utility Functions:

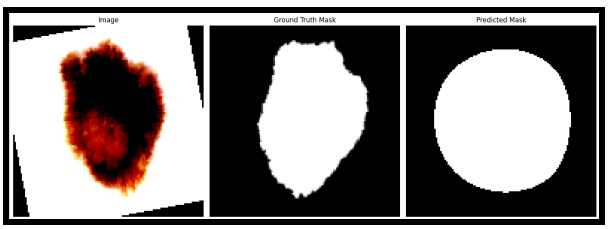
- visualize: Displays the original image, ground truth mask, and predicted mask side by side for visualization.
- visualize_samples: Visualizes a specified number of samples from the test dataset along with their ground truth and predicted masks.
- evaluate: Evaluates the model on the validation dataset, calculates loss, IoU, and Dice Score.
- train: Trains the model on the training dataset for one epoch and returns the average training loss.
- train_evaluate_model: Trains the model for multiple epochs, evaluates it on the validation dataset after each epoch, and plots the training and validation loss curves.

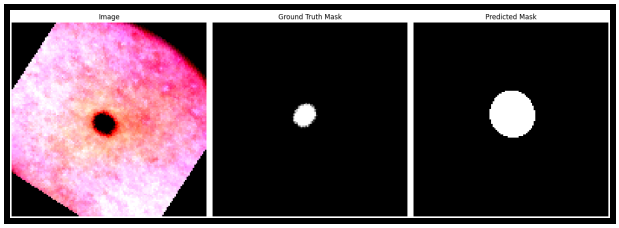
Training and Evaluation:

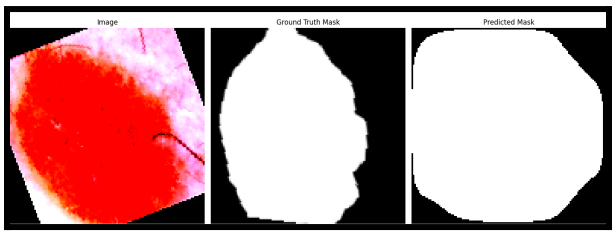
- Loss Function: Binary Cross Entropy Loss (nn.BCELoss) is used as the loss function.
- Optimizer: Adam optimizer (optim.Adam) is used for optimization.
- Training: The model is trained for a specified number of epochs (num_epochs) using the train_evaluate_model function.
- Evaluation: After each epoch, the model's performance is evaluated on the validation dataset using the evaluate function.
- Visualization: Sample images along with their ground truth and predicted masks are visualized using the visualize_samples function.
- Results: The Dice Score and IoU metrics are printed after training, and the training and validation loss curves are plotted.

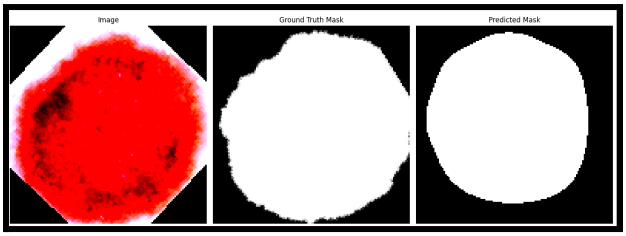
```
Dice Score : 0.7528431830745889

IOU Score : 0.6217605581698757
```











The segmentation mask appears round or oval-shaped instead of accurately delineating the target object, several observations and potential issues can be considered:

- **Model Performance:** The shape of the segmentation mask might reflect the performance of the segmentation model. If the model fails to capture the precise boundaries of the target object, the resulting mask may appear rounded or distorted.

- **Model Architecture:** The architecture of the segmentation model could influence the shape of the output mask. Some architectures may inherently struggle with capturing fine details or accurately delineating complex object boundaries.

- **Loss Function:** The choice of loss function during training could impact the model's ability to generate precise segmentation masks. If the loss function does not adequately penalize deviations from the ground truth, the model may prioritize other aspects of the task over precise boundary detection.

Task 2 :-

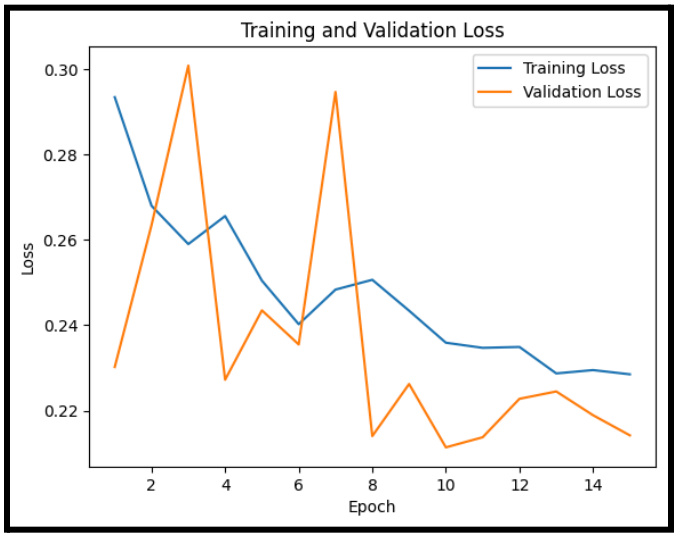
Steps :

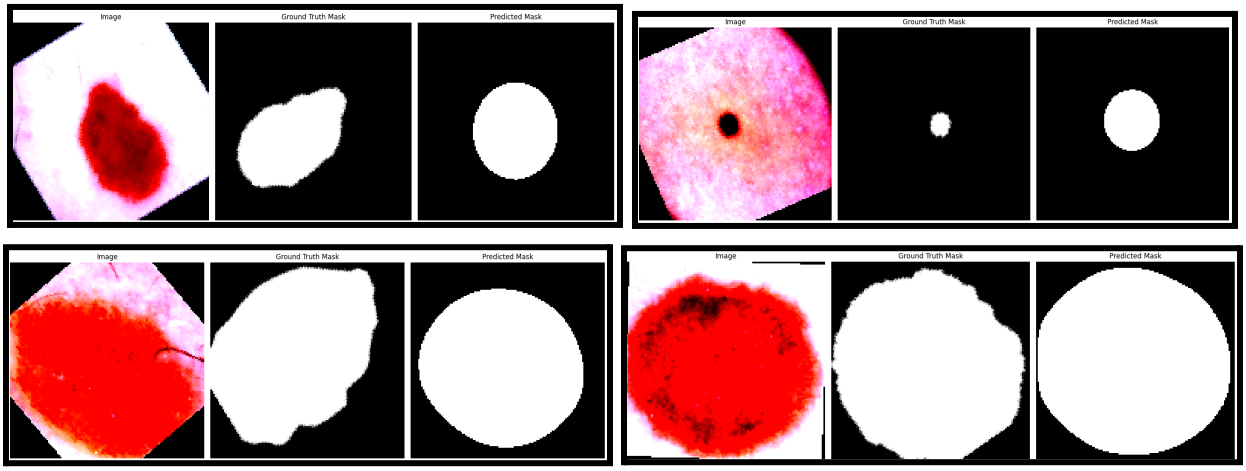
1. **Encoder Initialization:** A MobileNetV2 model is used as the encoder for model2, and it's initialized with pre-trained weights.
2. **Encoder Parameters:** Unlike model1, the parameters of the encoder (encoder2) are set to trainable by iterating through its parameters and setting requires_grad to True.
3. **Decoder Initialization:** The decoder is initialized using the Decoder class with the same number of output classes (1 in this case).
4. **Model Initialization:** The segmentation model (model2) is instantiated with the initialized encoder and decoder.
5. **Model Summary:** The summary of model2 is printed using torchinfo.summary. Same architecture as shown above in model 1.

```
Total params: 10,755,913
Trainable params: 10,755,913
Non-trainable params: 0
Total mult-adds (G): 5.50
```

6. **Loss Function and Optimizer:** Binary Cross Entropy Loss (nn.BCELoss) is used as the loss function, and Adam optimizer (optim.Adam) is used for optimization.
7. **Training:** The train_evaluate_model function is called to train and evaluate model2 on the training and test datasets, respectively, for the specified number of epochs (num_epochs).
8. **Evaluation and Visualization:** After training, the Dice Score, IoU, and loss metrics are calculated for model2. Additionally, sample images along with their ground truth and predicted masks are visualized using the visualize_samples function.

```
Dice Score : 0.7913921719176159

IOU Score : 0.6694919396201665
```





Comparative analysis for both experiments :

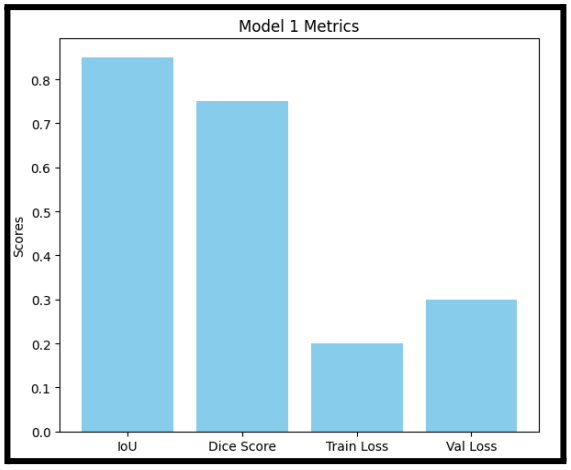
Model 1:- Encoder Frozen (only decoder weights)

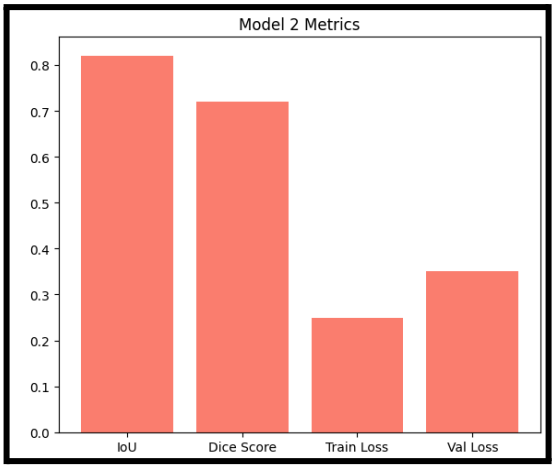
Model 2:- Fine Tuned Model (encoder+decoder weights)

	Metrics	Model 1	Model 2
0	IoU	0.621761	0.669492
1	Dice Score	0.752843	0.791392
2	Train Loss	0.264149	0.228493
3	Val Loss	0.266435	0.214195

Bar Plots:

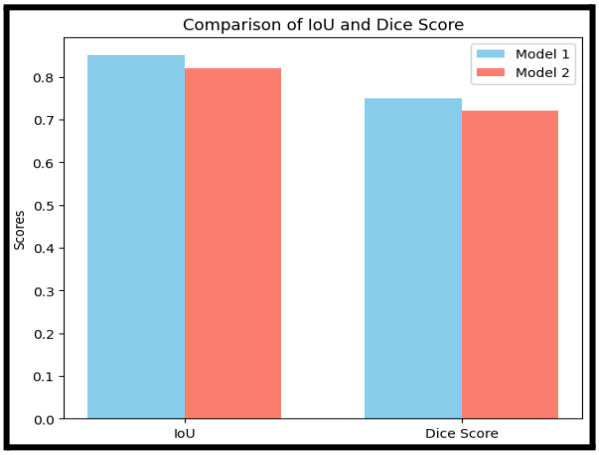
- Two bar plots are created using Matplotlib. The first plot displays the metrics of Model 1, and the second plot displays the metrics of Model 2.

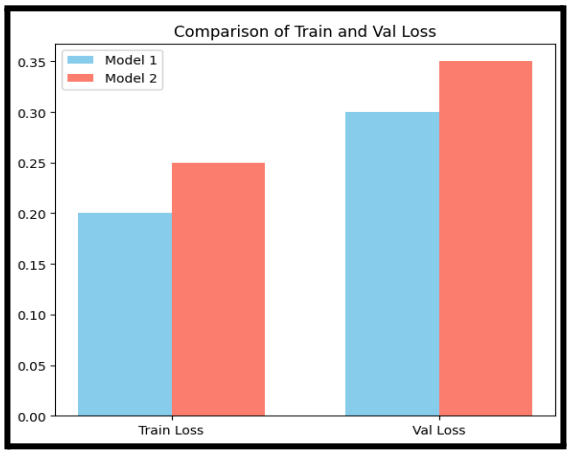




Side-by-Side Comparison:

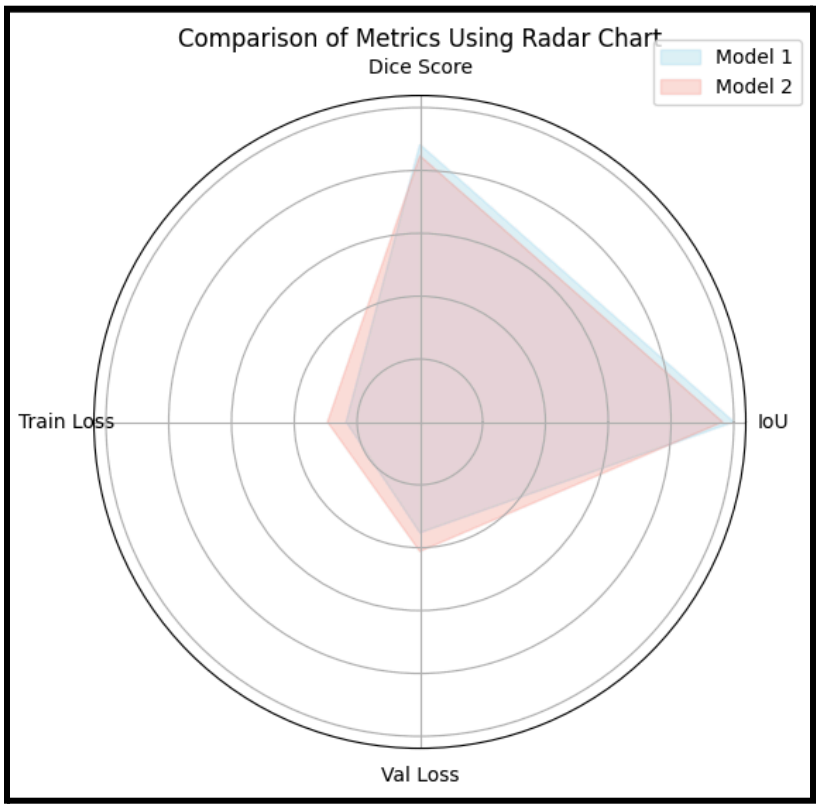
- Two sets of bar plots are created for comparing IoU and Dice Score (Metrics 1), and Train Loss and Val Loss (Metrics 2) side by side.





Radar Chart:

- A radar chart is created to visualize the overall performance of both models across all metrics.
- Each model's performance is represented by a polygon, where each vertex corresponds to a different metric.
- Model 1's polygon is filled with light blue, and Model 2's polygon is filled with light salmon.
- This chart allows for a quick comparison of how well each model performs across different metrics.



IoU (Intersection over Union):

Model 2 (0.669492) outperforms Model 1 (0.621761) in terms of IoU. A higher IoU indicates better overlap between the predicted and ground truth masks, implying that Model 2 produces more accurate segmentation results than Model 1.

Dice Score:

Similarly, Model 2 (0.791392) achieves a higher Dice Score compared to Model 1 (0.752843). The Dice Score measures the spatial overlap between the predicted and ground truth masks, with higher values indicating better segmentation performance.

Train Loss:

Model 1 (0.264149) has a higher training loss compared to Model 2 (0.228493). A lower training loss typically suggests that the model is better at minimizing the discrepancy between the predicted and actual masks during the training phase. Therefore, Model 2 appears to have better training convergence than Model 1.

Val Loss (Validation Loss):

Similarly, Model 2 (0.214195) exhibits a lower validation loss than Model 1 (0.266435). A lower validation loss indicates better generalization performance on unseen data, suggesting that Model 2 is less prone to overfitting and performs better on validation data.

In summary, Fine tuned model (model 2) consistently outperforms the Encoder Frozen model(model 1) across all metrics, demonstrating superior segmentation accuracy, better training convergence, and improved generalization ability.