# ECE695DL Homework 7

Utsav A. Negi

March 16, 2024

## Aim

The aim of this report is to demonstrate the approach used to implement multi-instance semantic segmentation on Purdue Shapes Dataset Objects and single-instance semantic segmentation on COCO Dataset Objects by training model based on Unet Architecture.

### Source Code Detail

For this assignment, the details about the source code are as follows:

* *hw7\_extra.py*: Source code consisting of all the essential classes, functions & main ML pipeline for single instance semantic segmentation on COCO Dataset.
* *hw7\_main.py:* Source code consisting of all the essential classes, functions & main ML pipeline for multi-instance semantic segmentation on PurdueShapes Dataset.

### Unet Model Architecture

The Unet Model is a commonly used model for semantic segmentation. The model has two phases: encoder phase and decoder phase. In the encoder phase, the size of the input image gets smaller which results in high levels of feature abstraction. In the decode phase, the determined feature abstractions are mapped to each pixel in the image. The code uses SkipBlockDN to encode the image and SkipBlockUP to decode the image. Furthermore, both phases are inter-connected using SkipConnections at each stage of respective encoding and decoding. The result produced by the model is a mask which segregates the object pixels from the background pixels. This model is being used to implement multi-instance semantic segmentation on Purdue Shapes Dataset Objects and single-instance semantic segmentation on COCO Dataset Objects. The architecture of the Unet model used is provided in Table 1

### Semantic Segmentation on Purdue Shapes Dataset Objects

### Background

As provided in the assignment requirements, the model for semantic segmentation on the Purdue Shape Dataset Objects is trained on three different loss criteria: Mean Square Error Loss, Dice Loss, and a combination of scaled Dice Loss and Mean Square Error Loss. The model parameters are optimized using SGD optimizer with a learning rate of 1e-4 and 0.9 as momentum hyperparameter. The training epoch is set to 6 with a batch size of 6. At the end of training with different loss functions, the training function returns a list of average loss incurred during training.

*Table 1: Unet Model Summary with input size of (25, 3, 256, 256)*

Layer (type) Output Shape Param #

========================================

Conv2d-1 [25, 64, 256, 256] 1,792

Conv2d-2 [25, 64, 128, 128] 36,928

BatchNorm2d-3 [25, 64, 128, 128] 128

Conv2d-4 [25, 64, 128, 128] 36,928

BatchNorm2d-5 [25, 64, 128, 128] 128

SkipBlockDN-6 [25, 64, 128, 128] 0

Conv2d-7 [25, 64, 128, 128] 36,928

BatchNorm2d-8 [25, 64, 128, 128] 128

Conv2d-9 [25, 64, 128, 128] 36,928

BatchNorm2d-10 [25, 64, 128, 128] 128

SkipBlockDN-11 [25, 64, 128, 128] 0

Conv2d-12 [25, 64, 128, 128] 36,928

BatchNorm2d-13 [25, 64, 128, 128] 128

Conv2d-14 [25, 64, 128, 128] 36,928

BatchNorm2d-15 [25, 64, 128, 128] 128

Conv2d-16 [25, 64, 64, 64] 4,160

Conv2d-17 [25, 64, 64, 64] 4,160

SkipBlockDN-18 [25, 64, 64, 64] 0

Conv2d-19 [25, 64, 64, 64] 36,928

BatchNorm2d-20 [25, 64, 64, 64] 128

Conv2d-21 [25, 64, 64, 64] 36,928

BatchNorm2d-22 [25, 64, 64, 64] 128

SkipBlockDN-23 [25, 64, 64, 64] 0

Conv2d-24 [25, 64, 64, 64] 36,928

BatchNorm2d-25 [25, 64, 64, 64] 128

Conv2d-26 [25, 64, 64, 64] 36,928

BatchNorm2d-27 [25, 64, 64, 64] 128

SkipBlockDN-28 [25, 64, 64, 64] 0

Conv2d-29 [25, 64, 64, 64] 36,928

BatchNorm2d-30 [25, 64, 64, 64] 128

Conv2d-31 [25, 64, 64, 64] 36,928

BatchNorm2d-32 [25, 64, 64, 64] 128

SkipBlockDN-33 [25, 64, 64, 64] 0

Conv2d-34 [25, 64, 64, 64] 36,928

BatchNorm2d-35 [25, 64, 64, 64] 128

Conv2d-36 [25, 64, 64, 64] 36,928

BatchNorm2d-37 [25, 64, 64, 64] 128

SkipBlockDN-38 [25, 64, 64, 64] 0

Conv2d-39 [25, 64, 64, 64] 36,928

BatchNorm2d-40 [25, 64, 64, 64] 128

Conv2d-41 [25, 64, 64, 64] 36,928

BatchNorm2d-42 [25, 64, 64, 64] 128

SkipBlockDN-43 [25, 64, 64, 64] 0

Conv2d-44 [25, 64, 64, 64] 36,928

BatchNorm2d-45 [25, 64, 64, 64] 128

Conv2d-46 [25, 64, 64, 64] 36,928

BatchNorm2d-47 [25, 64, 64, 64] 128

SkipBlockDN-48 [25, 64, 64, 64] 0

BatchNorm2d-49 [25, 64, 64, 64] 128

Conv2d-50 [25, 128, 64, 64] 73,856

BatchNorm2d-51 [25, 128, 64, 64] 256

SkipBlockDN-52 [25, 128, 64, 64] 0

Conv2d-53 [25, 128, 64, 64] 147,584

BatchNorm2d-54 [25, 128, 64, 64] 256

Conv2d-55 [25, 128, 64, 64] 147,584

BatchNorm2d-56 [25, 128, 64, 64] 256

SkipBlockDN-57 [25, 128, 64, 64] 0

Conv2d-58 [25, 128, 64, 64] 147,584

BatchNorm2d-59 [25, 128, 64, 64] 256

Conv2d-60 [25, 128, 64, 64] 147,584

BatchNorm2d-61 [25, 128, 64, 64] 256

SkipBlockDN-62 [25, 128, 64, 64] 0

BatchNorm2d-63 [25, 128, 64, 64] 256

Conv2d-64 [25, 128, 64, 64] 147,584

BatchNorm2d-65 [25, 128, 64, 64] 256

Conv2d-66 [25, 128, 64, 64] 147,584

BatchNorm2d-67 [25, 128, 64, 64] 256

SkipBlockDN-68 [25, 128, 64, 64] 0

Conv2d-69 [25, 128, 64, 64] 147,584

BatchNorm2d-70 [25, 128, 64, 64] 256

Conv2d-71 [25, 128, 64, 64] 147,584

BatchNorm2d-72 [25, 128, 64, 64] 256

SkipBlockDN-73 [25, 128, 64, 64] 0

Conv2d-74 [25, 128, 64, 64] 147,584

BatchNorm2d-75 [25, 128, 64, 64] 256

Conv2d-76 [25, 128, 64, 64] 147,584

BatchNorm2d-77 [25, 128, 64, 64] 256

SkipBlockDN-78 [25, 128, 64, 64] 0

Conv2d-79 [25, 128, 64, 64] 147,584

BatchNorm2d-80 [25, 128, 64, 64] 256

Conv2d-81 [25, 128, 64, 64] 147,584

BatchNorm2d-82 [25, 128, 64, 64] 256

SkipBlockDN-83 [25, 128, 64, 64] 0

Conv2d-84 [25, 128, 64, 64] 147,584

BatchNorm2d-85 [25, 128, 64, 64] 256

Conv2d-86 [25, 128, 64, 64] 147,584

BatchNorm2d-87 [25, 128, 64, 64] 256

SkipBlockDN-88 [25, 128, 64, 64] 0

Conv2d-89 [25, 128, 64, 64] 147,584

BatchNorm2d-90 [25, 128, 64, 64] 256

Conv2d-91 [25, 128, 64, 64] 147,584

BatchNorm2d-92 [25, 128, 64, 64] 256

SkipBlockDN-93 [25, 128, 64, 64] 0

Conv2d-94 [25, 128, 64, 64] 147,584

BatchNorm2d-95 [25, 128, 64, 64] 256

Conv2d-96 [25, 128, 64, 64] 147,584

BatchNorm2d-97 [25, 128, 64, 64] 256

Conv2d-98 [25, 128, 32, 32] 16,512

Conv2d-99 [25, 128, 32, 32] 16,512

SkipBlockDN-100 [25, 128, 32, 32] 0

ConvTranspose2d-101 [25, 128, 32, 32] 147,584

BatchNorm2d-102 [25, 128, 32, 32] 256

ConvTranspose2d-103 [25, 128, 32, 32] 147,584

BatchNorm2d-104 [25, 128, 32, 32] 256

ConvTranspose2d-105 [25, 128, 64, 64] 16,512

ConvTranspose2d-106 [25, 128, 64, 64] 16,512

SkipBlockUP-107 [25, 128, 64, 64] 0

ConvTranspose2d-108 [25, 128, 64, 64] 147,584

BatchNorm2d-109 [25, 128, 64, 64] 256

ConvTranspose2d-110 [25, 128, 64, 64] 147,584

BatchNorm2d-111 [25, 128, 64, 64] 256

SkipBlockUP-112 [25, 128, 64, 64] 0

ConvTranspose2d-113 [25, 128, 64, 64] 147,584

BatchNorm2d-114 [25, 128, 64, 64] 256

ConvTranspose2d-115 [25, 128, 64, 64] 147,584

BatchNorm2d-116 [25, 128, 64, 64] 256

SkipBlockUP-117 [25, 128, 64, 64] 0

BatchNorm2d-118 [25, 128, 64, 64] 256

ConvTranspose2d-119 [25, 128, 64, 64] 147,584

BatchNorm2d-120 [25, 128, 64, 64] 256

ConvTranspose2d-121 [25, 128, 64, 64] 147,584

BatchNorm2d-122 [25, 128, 64, 64] 256

SkipBlockUP-123 [25, 128, 64, 64] 0

ConvTranspose2d-124 [25, 128, 64, 64] 147,584

BatchNorm2d-125 [25, 128, 64, 64] 256

ConvTranspose2d-126 [25, 128, 64, 64] 147,584

BatchNorm2d-127 [25, 128, 64, 64] 256

SkipBlockUP-128 [25, 128, 64, 64] 0

ConvTranspose2d-129 [25, 64, 64, 64] 73,792

BatchNorm2d-130 [25, 64, 64, 64] 128

SkipBlockUP-131 [25, 64, 64, 64] 0

ConvTranspose2d-132 [25, 64, 64, 64] 36,928

BatchNorm2d-133 [25, 64, 64, 64] 128

ConvTranspose2d-134 [25, 64, 64, 64] 36,928

BatchNorm2d-135 [25, 64, 64, 64] 128

SkipBlockUP-136 [25, 64, 64, 64] 0

ConvTranspose2d-137 [25, 64, 64, 64] 36,928

BatchNorm2d-138 [25, 64, 64, 64] 128

ConvTranspose2d-139 [25, 64, 64, 64] 36,928

BatchNorm2d-140 [25, 64, 64, 64] 128

SkipBlockUP-141 [25, 64, 64, 64] 0

ConvTranspose2d-142 [25, 64, 64, 64] 36,928

BatchNorm2d-143 [25, 64, 64, 64] 128

ConvTranspose2d-144 [25, 64, 64, 64] 36,928

BatchNorm2d-145 [25, 64, 64, 64] 128

SkipBlockUP-146 [25, 64, 64, 64] 0

ConvTranspose2d-147 [25, 64, 64, 64] 36,928

BatchNorm2d-148 [25, 64, 64, 64] 128

ConvTranspose2d-149 [25, 64, 64, 64] 36,928

BatchNorm2d-150 [25, 64, 64, 64] 128

SkipBlockUP-151 [25, 64, 64, 64] 0

ConvTranspose2d-152 [25, 64, 64, 64] 36,928

BatchNorm2d-153 [25, 64, 64, 64] 128

ConvTranspose2d-154 [25, 64, 64, 64] 36,928

BatchNorm2d-155 [25, 64, 64, 64] 128

SkipBlockUP-156 [25, 64, 64, 64] 0

ConvTranspose2d-157 [25, 64, 64, 64] 36,928

BatchNorm2d-158 [25, 64, 64, 64] 128

ConvTranspose2d-159 [25, 64, 64, 64] 36,928

BatchNorm2d-160 [25, 64, 64, 64] 128

SkipBlockUP-161 [25, 64, 64, 64] 0

BatchNorm2d-162 [25, 64, 64, 64] 128

ConvTranspose2d-163 [25, 64, 64, 64] 36,928

BatchNorm2d-164 [25, 64, 64, 64] 128

ConvTranspose2d-165 [25, 64, 64, 64] 36,928

BatchNorm2d-166 [25, 64, 64, 64] 128

ConvTranspose2d-167 [25, 64, 128, 128] 4,160

ConvTranspose2d-168 [25, 64, 128, 128] 4,160

SkipBlockUP-169 [25, 64, 128, 128] 0

ConvTranspose2d-170 [25, 64, 128, 128] 36,928

BatchNorm2d-171 [25, 64, 128, 128] 128

ConvTranspose2d-172 [25, 64, 128, 128] 36,928

BatchNorm2d-173 [25, 64, 128, 128] 128

SkipBlockUP-174 [25, 64, 128, 128] 0

ConvTranspose2d-175 [25, 64, 128, 128] 36,928

BatchNorm2d-176 [25, 64, 128, 128] 128

ConvTranspose2d-177 [25, 64, 128, 128] 36,928

BatchNorm2d-178 [25, 64, 128, 128] 128

SkipBlockUP-179 [25, 64, 128, 128] 0

ConvTranspose2d-180 [25, 1, 256, 256] 577

========================================

Total params: 5,707,393

Trainable params: 5,707,393

Non-trainable params: 0

---------------------------------------------------------------

Input size (MB): 18.75

Forward/backward pass size (MB): 17187.50

Params size (MB): 21.77

Estimated Total Size (MB): 17228.02

### Observations and Testing

From Figure 1, it can be observed that the Dice Loss performed better than MSE Loss, while the combination of MSE and scaled Dice Loss performed the worst out of the three. But this is not completely true. This is because Dice Loss uses IOU to determine the intersection and union between the predicted mask and the ground truth mask which results in value less than 1. Furthermore, the MSE Loss focuses more on pixel co-ordinates which results in higher values.

A graph of a graph of a graph

Description automatically generated with medium confidenceA graph of loss and loss

Description automatically generated

*Figure 1: Plot showing the training loss incurred when loss function is set to MSE, Dice and a combination of scaled Dice and MSE Loss while training model on PurdueShapes Dataset*

To verify the performance of each model, the models trained with MSE Loss, Dice Loss and the combination of MSE Loss and scaled Dice Loss are evaluated using the images in the validation dataset. According to results shown in Figure 2, Figure 3, and Figure 4, it can be observed that the Dice Loss performed the worst compared to MSE Loss and the combination of MSE Loss and scaled Dice Loss. Furthermore, the result yielded by the combination of MSE Loss, and scaled Dice Loss were missing some shapes whereas the MSE Loss has all the shapes while it lacks accurate masking. This shows that the training loss is just a relative number and showing low loss doesn’t mean that a model is completely accurate.

A graph of geometric shapes

Description automatically generatedA black grid with different shapes

Description automatically generated A graph of geometric shapes

Description automatically generatedA black grid with many different shapes

Description automatically generatedA graph of a graph

Description automatically generated with medium confidence

*Figure 2: Plot showing mask results generated by model trained with MSE Loss*

*A black grid with different colored squares

Description automatically generatedA black grid with different shapes

Description automatically generatedA black grid with many squares

Description automatically generatedA black grid with many squares

Description automatically generatedA black grid with many squares

Description automatically generated*

*Figure 3: Plot showing mask results generated by model trained with Dice Loss*

A graph of geometric shapes

Description automatically generatedA black grid with different shapes

Description automatically generatedA graph of geometric shapes

Description automatically generatedA black grid with many different colored shapes

Description automatically generated with medium confidenceA graph of a graph

Description automatically generated with medium confidence

*Figure 4: Plot showing mask results generated by model trained with MSE+Dice Loss*

### Semantic Segmentation on COCO Dataset Objects

### Background

Like HW6, semantic segmentation is conducted on Motorcycle, Dog and Cake categories of COCO Dataset Objects. At the start of the pipeline, the image data along with its binary mask are extracted using the COCO API which are resized to 256 x 256 and are stored in two dictionaries: one for training data and another for validation data. These dictionaries are used to create the ImageDataset class which is a child class of torch.utils.data.Dataset class. For this purpose, the same Unet model is used with some minor tuning at the output layer to match the output dimensions with dimensions of the ground truth binary mask. The model is trained using MSE Loss, Dice Loss and a combination of MSE Loss and scaled Dice Loss to compare its performance on COCO Dataset Objects. The model parameters are optimized using SGD optimizer with a learning rate of 5e-4 and 0.95 as momentum hyperparameter. The training epoch is set to 15 with a batch size of 25. At the end of training with different loss functions, the training function returns a list of average loss incurred during training.

### Observations and Testing

A graph of a graph

Description automatically generatedA graph with a line

Description automatically generatedA graph of a graph

Description automatically generated

*Figure 5: Plot showing the training loss incurred when loss function is set to MSE, Dice and a combination of scaled Dice and MSE Loss while training model on COCO Dataset*

From Figure 5, it can be observed that the loss gradually decreases when scaled Dice Loss and MSE Loss, and Dice Loss are used for determining the loss, while the training loss rapidly decreases when MSE is used for loss calculation. This infers that with more iterations, models using the combination of scaled Dice Loss and MSE Loss, and Dice Loss can further be trained to a better level of training loss.

To verify the performance of each model, the models trained with MSE Loss, Dice Loss and the combination of MSE Loss and scaled Dice Loss are evaluated using the images in the validation dataset. From Figure 6, Figure 7, and Figure 8, it can be observed that the model trained using the MSE Loss and the model trained using the Dice Loss is able to mask the object well; however, it is susceptible to the background objects and the object’s environment. The model trained using the combination of MSE Loss and scaled Dice Loss looks more confident at masking the object, but for some images, it either misses some pixels to be considered as a part of the mask or considers background objects as the part of the mask.

A collage of images of two people

Description automatically generated A collage of images of dogs

Description automatically generated A collage of images of cakes

Description automatically generated

*Figure 6: Table showing mask results generated by model trained with MSE Loss*

*A collage of images of a person riding a scooter

Description automatically generatedA collage of images of dogs

Description automatically generated A collage of images of food

Description automatically generated*

*Figure 7: Table showing mask results generated by model trained with Dice Loss*

A collage of images of motorcycles and bikes

Description automatically generated A collage of images of dogs

Description automatically generated A collage of images of a cake

Description automatically generated

*Figure 8: Table showing mask results generated by model trained with MSE+Dice Loss*

### Verdict

The Unet Model does a great work in semantic segmentation as the encoder-decoder phase allows to relate features of the image with each of its pixels. Based on the experience multi-instance semantic segmentation on Purdue Shapes Dataset Objects and single-instance semantic segmentation on COCO Dataset Objects, it can be inferred that model trained using the MSE Loss function generates the best results, followed by the model trained using a combination of MSE Loss and scaled Dice Loss. However, it is expected that the model trained using a combination of MSE Loss and scaled Dice Loss should have a better mask result, which looks certainly possible from the results, by more training epochs and better hyperparameters.