ECE60146: Homework 7

Manish Kumar Krishne Gowda, 0033682812

(Spring 2024)

1 Introduction

In this homework we perform semantic segmentation of images with neural networks.This means we'll train a neural network to look at an image and label each pixel according to what it represents, like identifying where objects are in the picture. To do this, we're using a special neural network called a U-Net. It's designed with an encoder-decoder architecture with multiple convolutional layers, skip connections both along and across the two arms of the U-Net and transpose convolutions. This helps it recognize patterns and shapes within the image. We'll be training this network to understand different objects by associating class labels with each pixel.

We're going to evaluate how well our model performs using three different methods: Mean Squared Error (MSE) loss, Dice loss, and Mean Absolute Error (MAE) loss. While MSE and Dice Loss are part of the requirement for the HW as per task instructions, MAE Loss is used to train the U-Net for Coco Dataset segmentation. These metrics help us measure the accuracy of our segmentation.

For our experiments, we'll start by working with the PurdueShapes5MultiObject dataset from the DL Studio dataset archive - [1]. This dataset contains images with five different shapes: rectangle, triangle, disk, oval, and star. We'll train our model to recognize these shapes within the images using MSE and Dice Loss. Later on, we'll take on another challenge by working with the Coco dataset. This dataset focuses on three specific classes: cake, dog, and motorcycle. We'll be using binary masks to segment these objects from the background in the images using MAE and Dice Losses.

2 Methodology-PurdueShapes5MultiObject DataSet

For this report, we utilized the most recent version (2.3.6) of DLStudio. We took advantage of the code provided in semantic segmentation.py within DLStudio, which allowed us to make use of the SemanticSegmentation class defined in DLStudio.py. This class provided the necessary functionalities for our semantic segmentation tasks.

The goal of this SemanticSegmentation inner class is to utilize the DLStudio module for conducting experiments on semantic segmentation. Essentially, semantic segmentation involves correctly labeling different objects in a scene while also pinpointing their locations. This inner class primarily relies on the mUnet network, which is derived from the UNET network. UNET initially proposed by Ronneberger, Fischer, and Brox in their paper titled "U-Net: Convolutional Networks for Biomedical Image Segmentation." Their UNET is designed to extract binary masks for cell pixel blobs in biomedical images, effectively serving as a pixel-wise binary classifier at each pixel position. In contrast, the mUnet class is tailored for segmenting multiple objects simultaneously from an image. The original binary mask method of classifying is used for segmenting the coco image dataset classes. Unlike the original UNET class skip connections not only across the two arms of the "U" but also along the arms for both the PurdueShapes5MultiObject dataset as well as the Coco object dataset classification.

2.1 Source Code Description

To perform semantic segmentation on PurdueShapes5MultiObject, I utilized several components directly from the SemanticSegmentation class of the DLStudio. These include the mUnet network, PurdueShapes5MultiObjectDataset, and the run_code_for_testing_semantic_segmentation function. This allowed me to streamline the process and leverage existing functionalities for our specific task.

As per task instructions, following SegmentationLossDice class was included in the leveraged SemanticSegmentation class to compute the Dice loss for semantic segmentation tasks.

```
    class SegmentationLossDice(nn.Module):
        def __init__(self, batch_size):
            super(SemanticSegmentation.SegmentationLossDice, self).__init__()
            self.batch_size = batch_size
        def forward(self, output, mask_tensor):
            composite_loss = torch.zeros(1,self.batch_size)
            mask_based_loss = torch.zeros(1,5)
            for idx in range(self.batch_size):
                #outputh = output[idx,0,:,:]
                for mask_layer_idx in range(mask_tensor.shape[0]):
                    outputh = output[idx,mask_layer_idx,:,:]
                    mask = mask_tensor[idx,mask_layer_idx,:,:]
                    numerator = 2*(outputh*mask) + 1
                    denominator = outputh + mask + 1
                    dice_loss = 1-numerator/denominator
                    mask_based_loss[0,mask_layer_idx] = torch.mean(dice_loss)
                composite_loss[0,idx] = torch.sum(mask_based_loss)/mask_tensor.
                                                                    shape[0]
            return torch.sum(composite_loss) / self.batch_size
```

This method defines the forward pass computation of the module, which takes the predicted segmentation output and ground truth segmentation mask_tensor as inputs.It iterates over the batch dimension to compute the loss for each sample individually. Inside the loop, it iterates over the channels (or mask layers) in the output and mask tensors. For each channel, it computes the Dice loss:

1. It calculates the numerator and denominator terms of the Dice coefficient formula.
2. The numerator represents twice the intersection of the predicted and ground truth masks plus a small smoothing term to avoid division by zero.
3. The denominator represents the sum of predicted and ground truth masks plus the same smoothing term.
4. It computes the Dice loss for each pixel and takes the mean over all pixels in the mask.
5. The mask-based losses for each channel are stored in mask_based_loss variable.
6. Finally, it computes the composite loss for each sample by averaging the mask-based losses over all channels.

The overall loss across the batch is computed by averaging the composite losses over all samples in the batch.

For Mean Squared Error (MSE) loss, while torch.nn's MSELoss() module can be used for calculating the MSE loss, a SegmentationLossMSE was constructed to better analyse and understand the functioning of the loss type. The SegmentationLossMSE class calculates the MSE loss for semantic segmentation tasks. Similar to the SegmentationLossDice it iterates over the batch

dimension to compute the loss for each sample individually, calculating the squared difference between predicted and ground truth masks. Finally, it averages the losses over the batch and returns the total MSE loss.

```
        class SegmentationLossMSE(nn.Module):
        def __init__(self, batch_size):
            super(SemanticSegmentation.SegmentationLossMSE, self).__init__()
            self.batch_size = batch_size
        def forward(self, output, mask_tensor):
            composite_loss = torch.zeros(1,self.batch_size)
            mask_based_loss = torch.zeros(1,5)
            for idx in range(self.batch_size):
                for mask_layer_idx in range(mask_tensor.shape[0]):
                    outputh = output[idx,mask_layer_idx,:,:]
                    mask = mask_tensor[idx,mask_layer_idx,:,:]
                    element_wise = (outputh - mask)**2
                    mask_based_loss[0,mask_layer_idx] = torch.mean(element_wise)
                composite_loss[0,idx] = torch.sum(mask_based_loss)/mask_tensor.
                                                                    shape[0]
            return torch.sum(composite_loss) / self.batch_size
```

Using the above two SegmentationLossMSE and SegmentationLossDice classes a third class SegmentationLossComb is constructed. The SegmentationLossComb class combines both Dice loss and MSE loss for semantic segmentation tasks. During initialization, it accepts the batch size and a scaling factor for the Dice loss. In the forward pass, it calculates both the Dice loss and MSE loss using instances of SegmentationLossDice and SegmentationLossMSE classes respectively, and then combines them based on the provided scaling factor. This class enables a combination of both losses to be used in training the model.

```
class SegmentationLossComb(nn.Module):
    def __init__(self, batch_size,dice_scale):
        super(SemanticSegmentation.SegmentationLossComb, self).__init__()
        self.batch_size = batch_size
        self.loss_scaling_factor = dice_scale
    def forward(self, output, mask_tensor):
        dice_loss = SemanticSegmentation.SegmentationLossDice(self.batch_size)(
                                                output, mask_tensor)
        mse_loss = SemanticSegmentation.SegmentationLossMSE(self.batch_size)(
                                                output, mask_tensor)
```

A custom class custom_run_code_for_training_for_semantic_segmentation was constructed by leveraging the code from the original "run_code_for_training_for_semantic_segmentation" with arguments to intake loss type (MSE or Dice or Combined) and scale factor for the combined loss case of MSE and Dice.

2.2 Outputs Discussion

The UNET with about 7M (6999109) parameters and 300 (324) layers is trained and the training loss is evaluated for MSE loss, Dice loss and the combined loss of MSE and Dice Loss. For the combined loss is calculated using three scaling factors of 200, 300 and 400. The MSE loss is in the range of 300-400 in the 6 training epochs, while the Dice Loss is in the range of 0.8-1. Thus while combining the two losses (i.e. adding up the two losses) a scaling factor is used to bring the two losses in the same range. Post training, the model is evaluated with the test dataset (available in the same archive) and 4 test batches (each with 4 images) are plotted to evaluate the performance of the model.

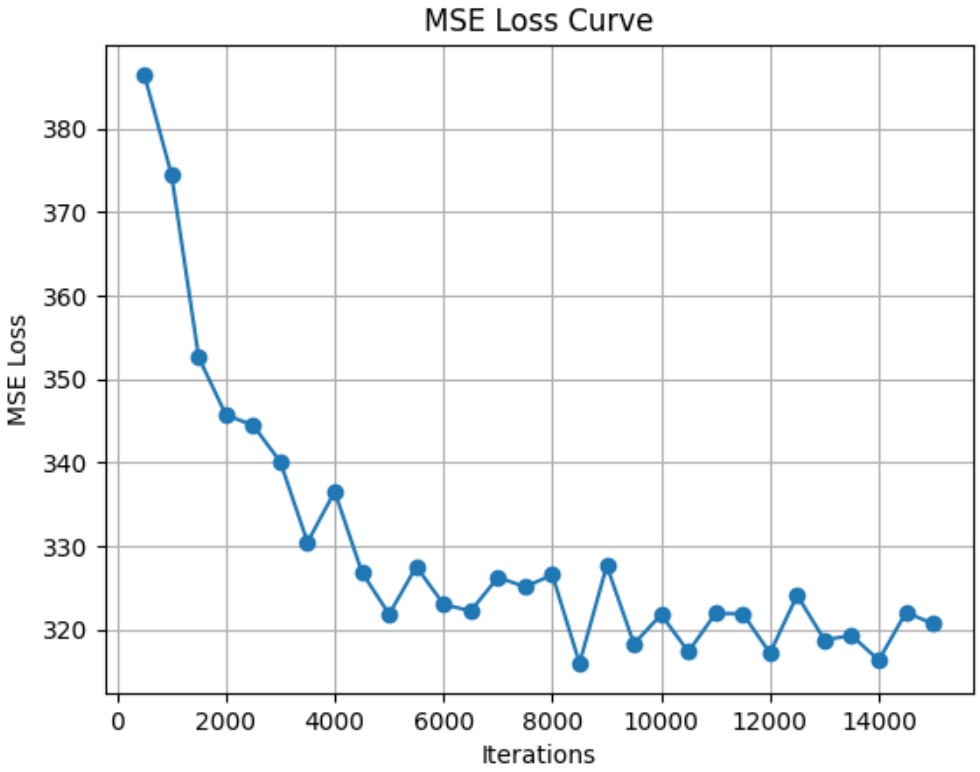

Figure 1: Training Loss MSE

2.2.1 MSE Loss

The training process spanned 6 epochs, each comprising approximately 2500 iterations. Throughout this training, the average loss was computed every 500 iterations, and these values were plotted to generate the training loss curve.The MSE training loss is shown in Figure 1. Overall, there seems to be a decreasing trend in the MSE loss values across epochs and iterations. This suggests that the model is progressively improving its performance in minimizing the difference between predicted and ground truth segmentation masks. Within each epoch, there are fluctuations in theaverage MSE loss values across iterations. This indicates that the training process encountered variations in the convergence rate or face challenges in fitting the data optimally. Initially, there are relatively higher MSE loss values in the early iterations of each epoch. However, as training progresses, these values tend to decrease, indicating that the model is converging towards a better solution. The fluctuations within epochs and the stabilization of loss values towards the end of each epoch suggest potential areas for optimization or exploration in improving the training process further. This issue can be addressed using Dice Loss, which will be explained in the next sections.

After running the test loop, the output of 5 test batches is depicted in Figure 2. While the test images demonstrate correct segmentation, there is room for improvement, particularly regarding mask brightness. Row 1 displays the final output of the segmented images alongside bounding boxes. Row 2 showcases the four RGB images used for testing in that batch. The subsequent 5 layers illustrate the results of the 5 shapes at the output of the last convolutional layer of UNET.

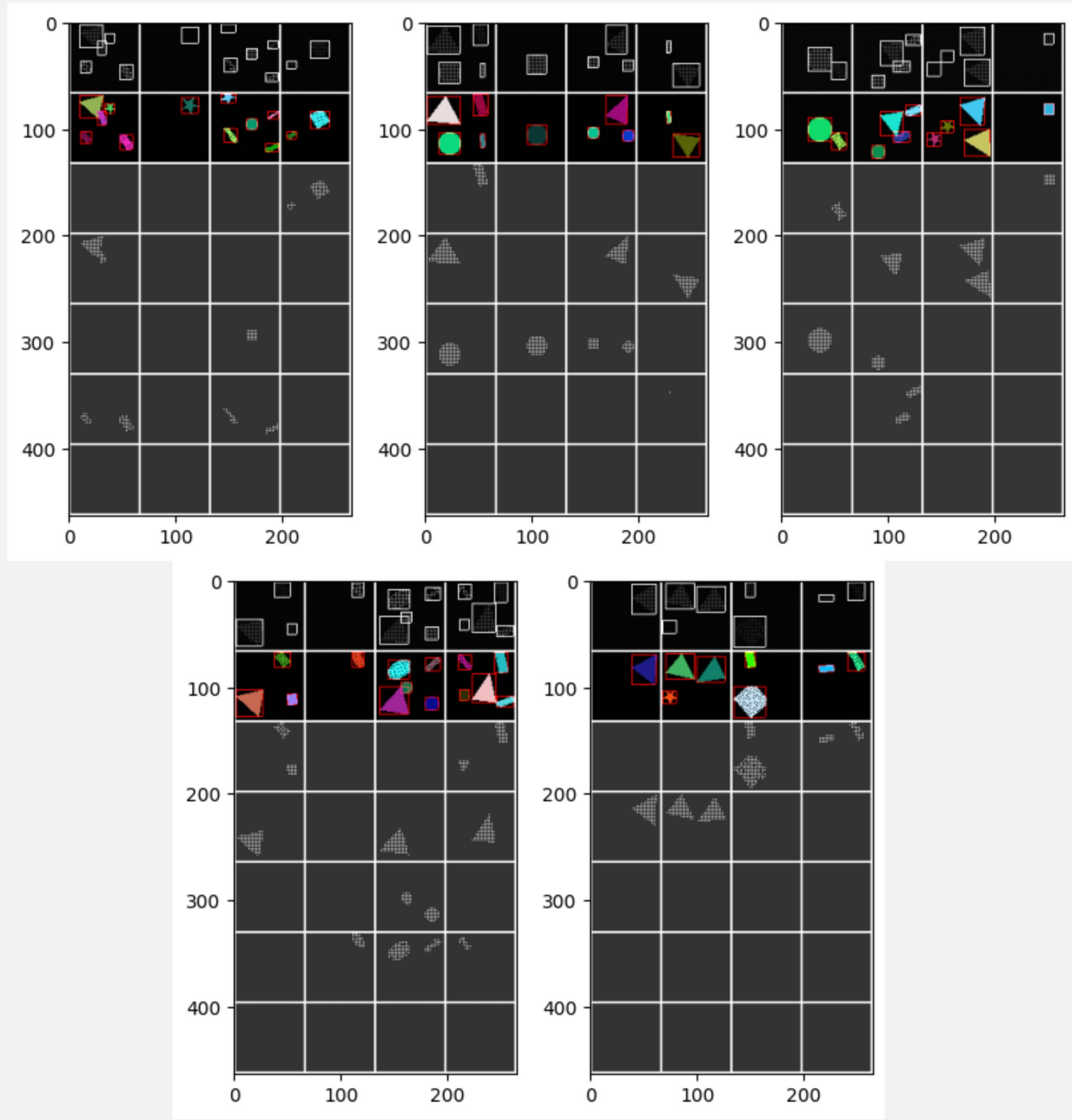

Figure 2: Output of 5 Test Batches for MSE Loss

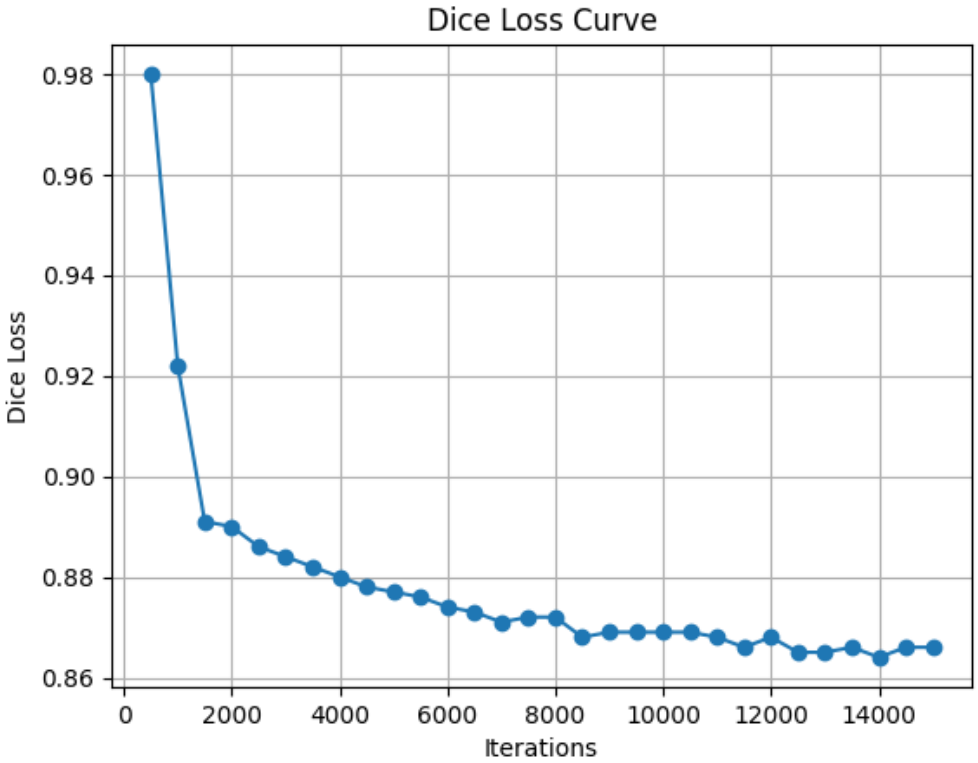

Figure 3: Training Loss Dice

2.2.2 Dice Loss

The training loss curve for Dice Loss is shown in Figure 3. Similar to the MSE loss curve Dice loss values exhibit a decreasing trend throughout the training process. In fact, as opposed to the MSE loss curve the Dice Loss curve consistently decreases as training progresses, indicating that the model is learning and improving its ability to predict segmentation masks accurately.The loss values stabilize over time, particularly after the first few epochs. This stabilization suggests that the model has converged to a stable solution and is no longer experiencing significant fluctuations in performance. While both MSE loss and Dice Loss values exhibit a decreasing trend over the course of training, the rate of decrease in the Dice loss values appears to be more gradual and consistent, while the MSE values showed larger fluctuations and slower convergence. Also teh Dice loss values seem to stabilize more quickly and consistently across epochs compared to the MSE Loss. (The absolute value of the losses is disregarded, and only the trends are compared, as it is not meaningful to directly compare the absolute values of losses calculated using different methods)

After running the test loop, the output of 5 test batches is depicted in Figure 4. Similar to the MSE test images, correct segmentation is observed, but there is potential for enhancement. Interestingly, despite the improved loss curve in the second Dice loss case, this doesn't result in visibly superior segmentation. This can be attributed to various factors such as the complexity of the dataset, limitations in the model architecture, or insufficient training data. Additionally, while the loss function provides an objective measure of model performance, it may not capture all nuances of image quality or segmentation accuracy that are perceptible to human observers.

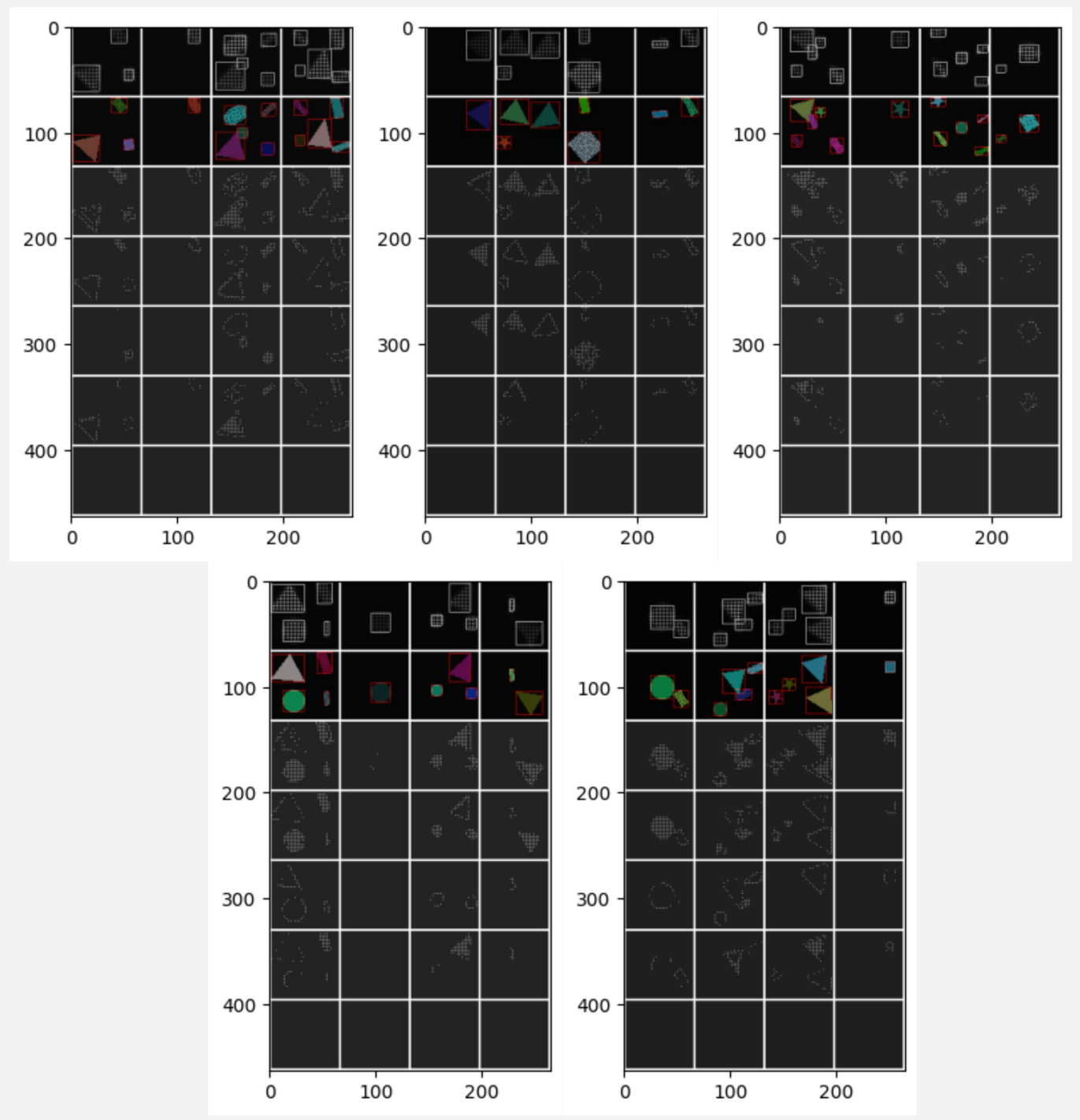

Figure 4: Output of 5 Test Batches for Dice Loss

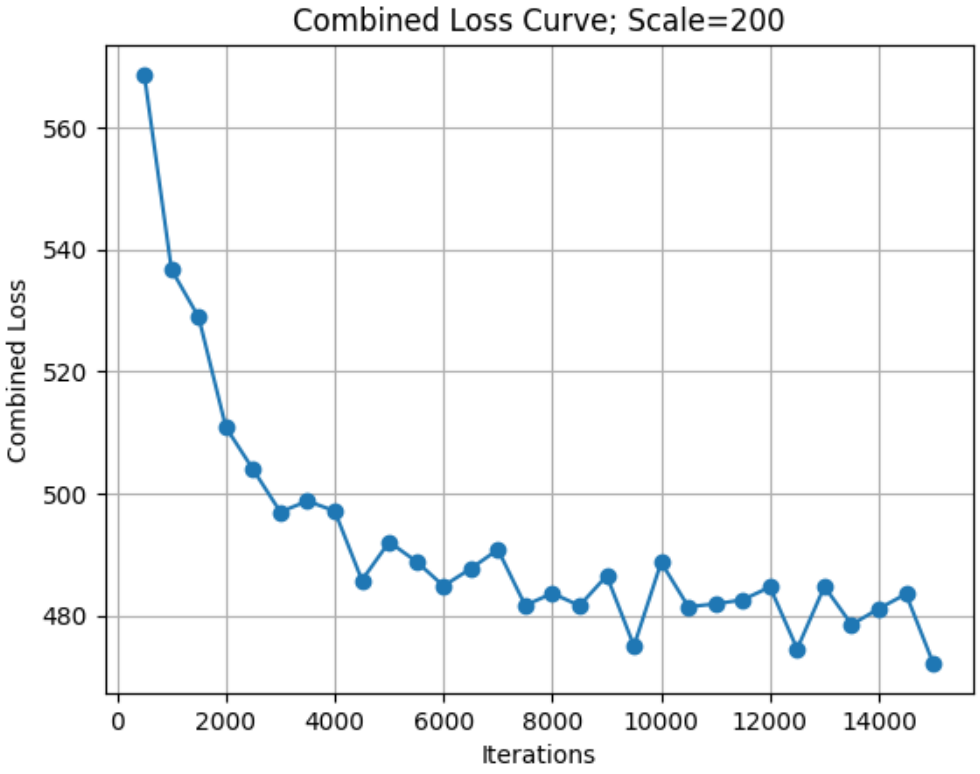

Figure 5: Training Loss Combined, Scale=200

2.2.3 Combined Loss

Three Different Scaling factors of 200, 300 and 400 were used to combine the two (MSE and Dice) losses

Scale Factor 200 : Here again we can observe fluctuations in the Combined loss values throughout the training process. During the early iterations of the first epoch, the loss values are relatively high, ranging from approximately 568 to 504. This could be due to the model's random initialization and the initial lack of meaningful patterns learned from the data. As training progresses, there is a general decreasing trend in the MSE loss values. The losses gradually decrease over the epochs, indicating that the model is improving in its ability to minimize the error between predicted and ground truth segmentation masks. Within each epoch, there are fluctuations in the MSE loss values across iterations. For example, in the second epoch, the loss values fluctuate between approximately 496 and 492. These fluctuations could be due to various factors such as unstable backpropagation due to combined losses of two different kinds,the complexity of the dataset, stochasticity in the optimization process, or changes in the training data distribution. Towards the later epochs, the MSE loss values appear to stabilize or exhibit slower rates of decrease. This suggests that the model may be approaching a point of convergence, where further improvements become more incremental.

The output of the test images with the combined loss and scale=200 is shown in Figure 6

Scale Factor 300 : Due to the higher scaling factor, i n this trend, the initial loss values are notably higher, ranging from approximately 654 to 590 in the first epoch. Throughout the epochs,

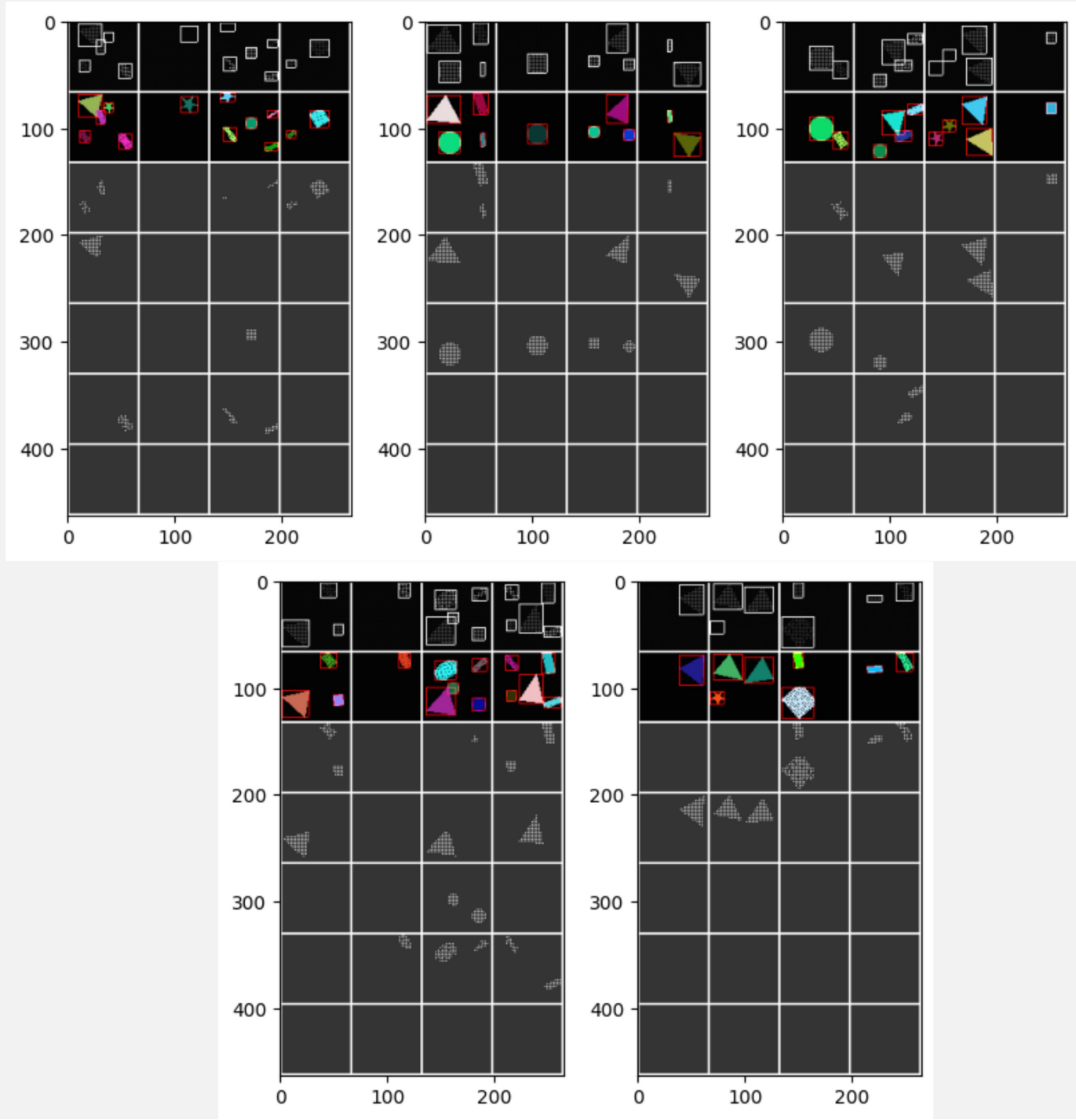

Figure 6: Output of 5 Test Batches for Combined Loss, Scale=200

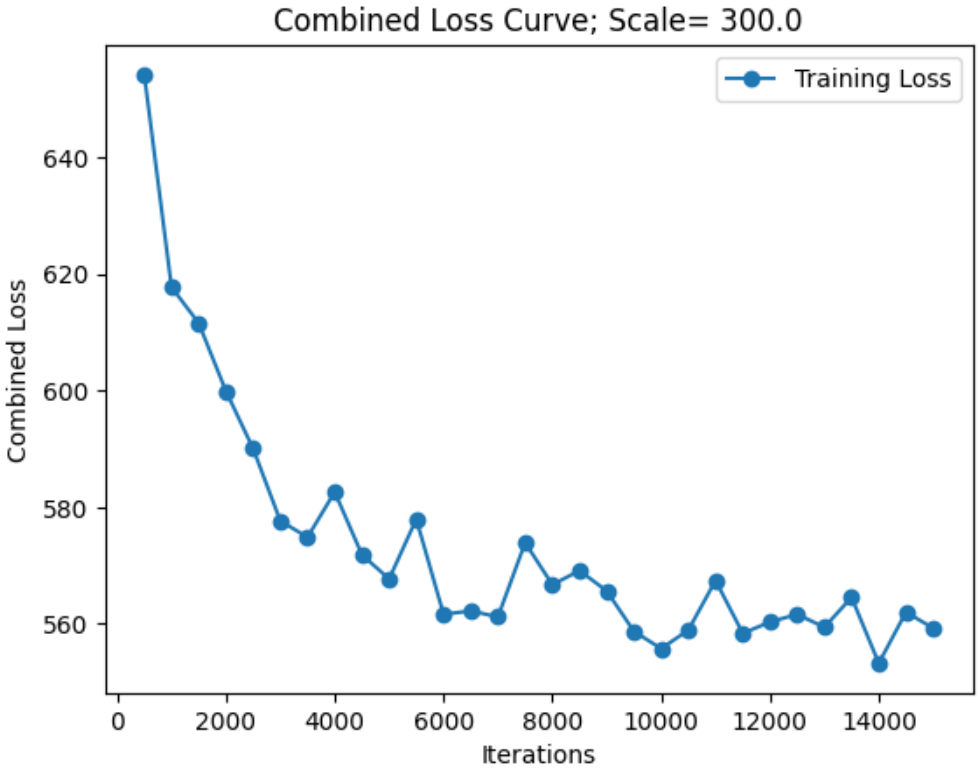

Figure 7: Training Loss Combined, Scale=300

the MSE loss values fluctuate and oscillate around a certain range. There are instances where the loss decreases, followed by an increase in subsequent iterations or epochs. This indicates that the training process encounters challenges in consistently reducing the error. Unlike the previous trend, where the loss values stabilized or exhibited a more consistent decreasing trend towards the later epochs, this trend does not show clear signs of convergence. The loss values continue to fluctuate even in the later epochs, suggesting that the model may struggle to converge to a stable solution.

Scale Factor 400 : The loss values with scale=400 demonstrate a consistent decreasing trend across epochs and iterations. This trend contrasts with the fluctuations and oscillations observed in the previous two trends. The more consistent and steady decrease in loss values suggests a smoother convergence of the model during training. This is indicative of a more stable optimization process and a more effective learning trajectory for the model.

Compared to the previous two scales, where fluctuations and lack of convergence were observed, this trend exhibits a more desirable behavior for training neural networks. The consistent decrease in loss values indicates that the model is progressively improving its ability to minimize the error between predicted and ground truth segmentation masks. This is a result of having both MSE and Dice loss in the same range (around 300). This shows the importance of choosing the right scaling factor while combining the losses.

The test image outputs for the three scales is shown in Figures **??**, **??** and **??**. It is difficult to discern significant differences in the test image outputs for the three cases. This is perhaps because, the differences in the loss trends may not necessarily translate to easily noticeable changes in the test image outputs. Even small variations in loss values during training may not manifest

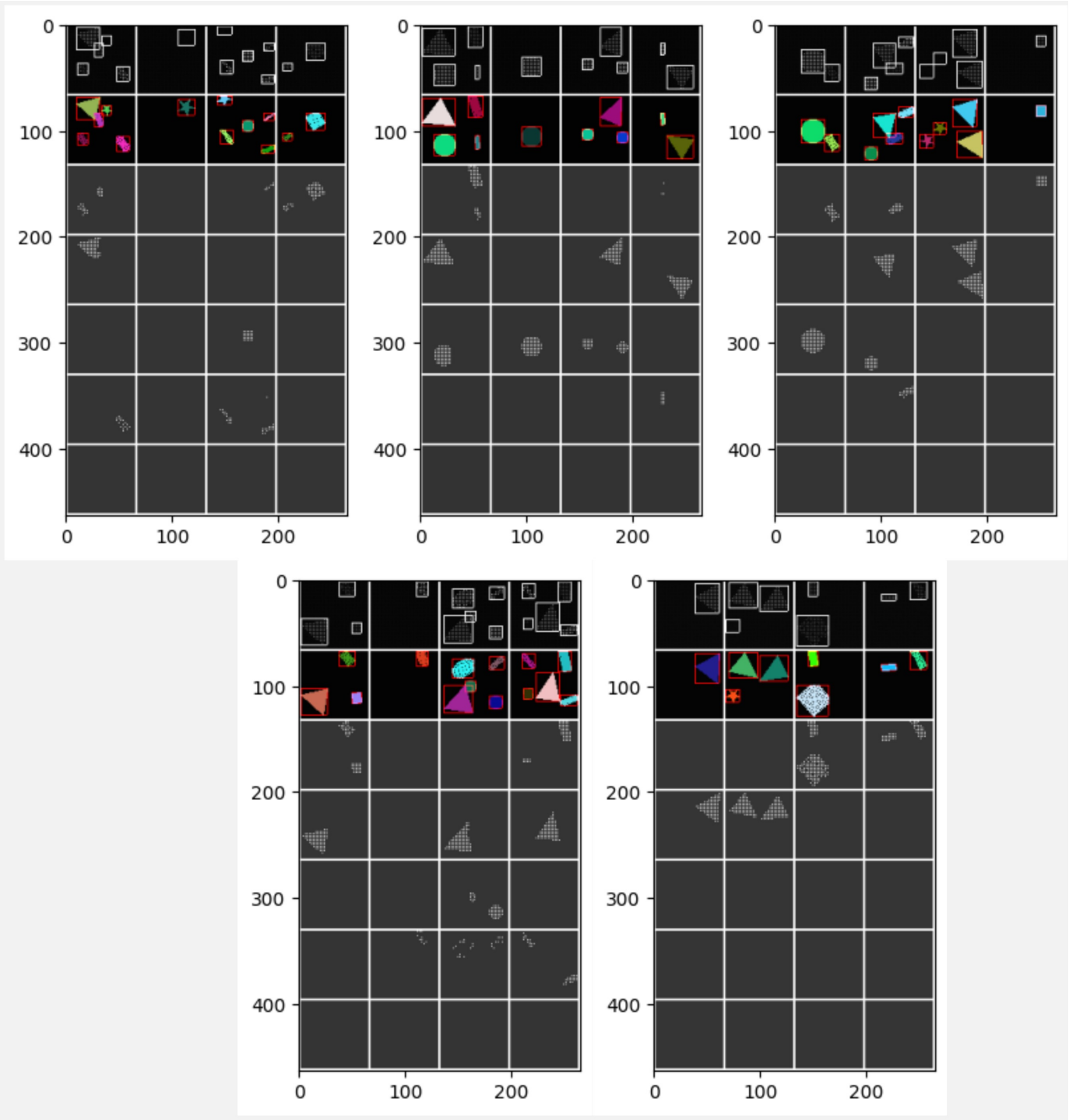

Figure 8: Output of 5 Test Batches for Combined Loss, Scale=300

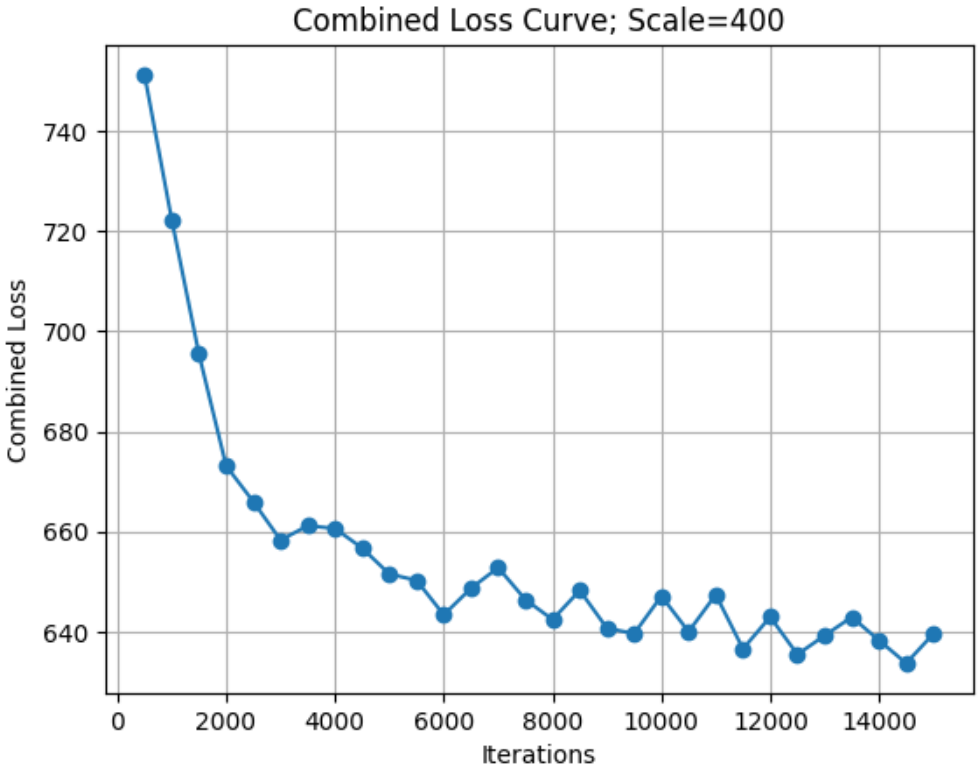

Figure 9: Training Loss Combined, Scale=400

as noticeable differences in the visual quality of the segmented images. Despite the difficulty in discerning differences in the test image outputs, the importance of choosing the right scaling factor was realized and applied to segment the Coco dataset for the bonus task. This suggests that even if the differences in loss trends may not be visibly apparent in the test images, they can still have practical implications for improving the segmentation performance on more complex datasets like Coco.

3 Methodology-Coco DataSet; The Bonus Task

Images from the COCO dataset classes 'cake', 'dog', and 'motorcycle' were selected. Specifically, images containing only a single object instance with a bounding box of at least 200 × 200 pixels were chosen. The segmentation masks corresponding to these objects were extracted using the annToMask function and converted into binary masks.

For the training dataset, a total of 891 images labeled as 'cake', 1201 images labeled as 'dog', and 1193 images labeled as 'motorcycle' were downloaded. For the testing dataset, a smaller subset of images was used for evaluation purposes. Specifically, 30 images labeled as 'cake', 65 images labeled as 'dog', and 57 images labeled as 'motorcycle' were downloaded. Sample downloaded images in the train dataset along with their corresponding masks is shown in Figure 11

However, instead of resizing the masks to 256 × 256 pixels as per the task instructions, a decision was made to resize them to 64 × 64 pixels. This decision was made to optimize the training process and reduce computational complexity. Resizing to 256 × 256 pixels would require upsampling to 512 × 512 pixels when using a modified version of mUNET class (about 300M

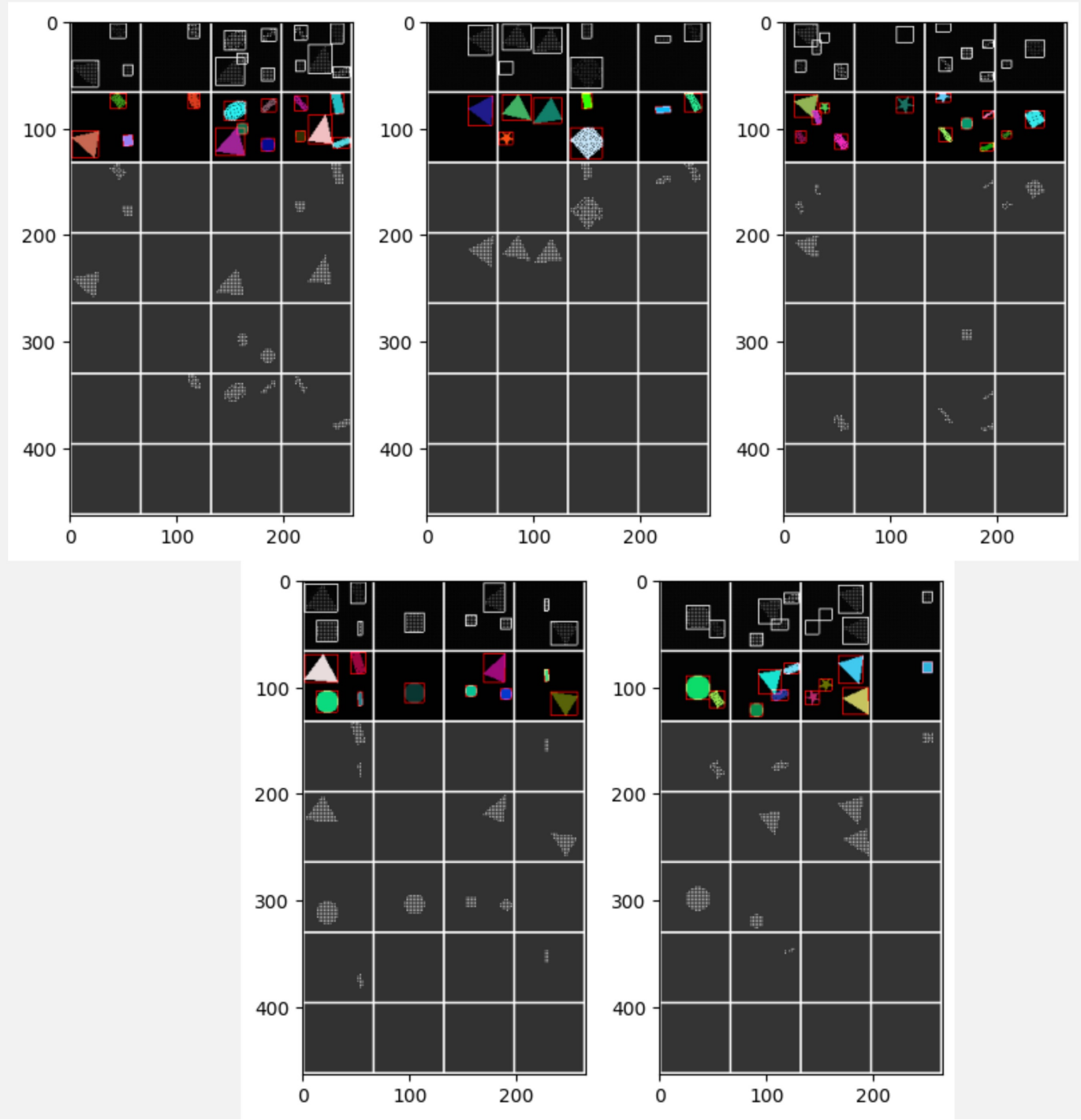

Figure 10: Output of 5 Test Batches for Combined Loss, Scale=400

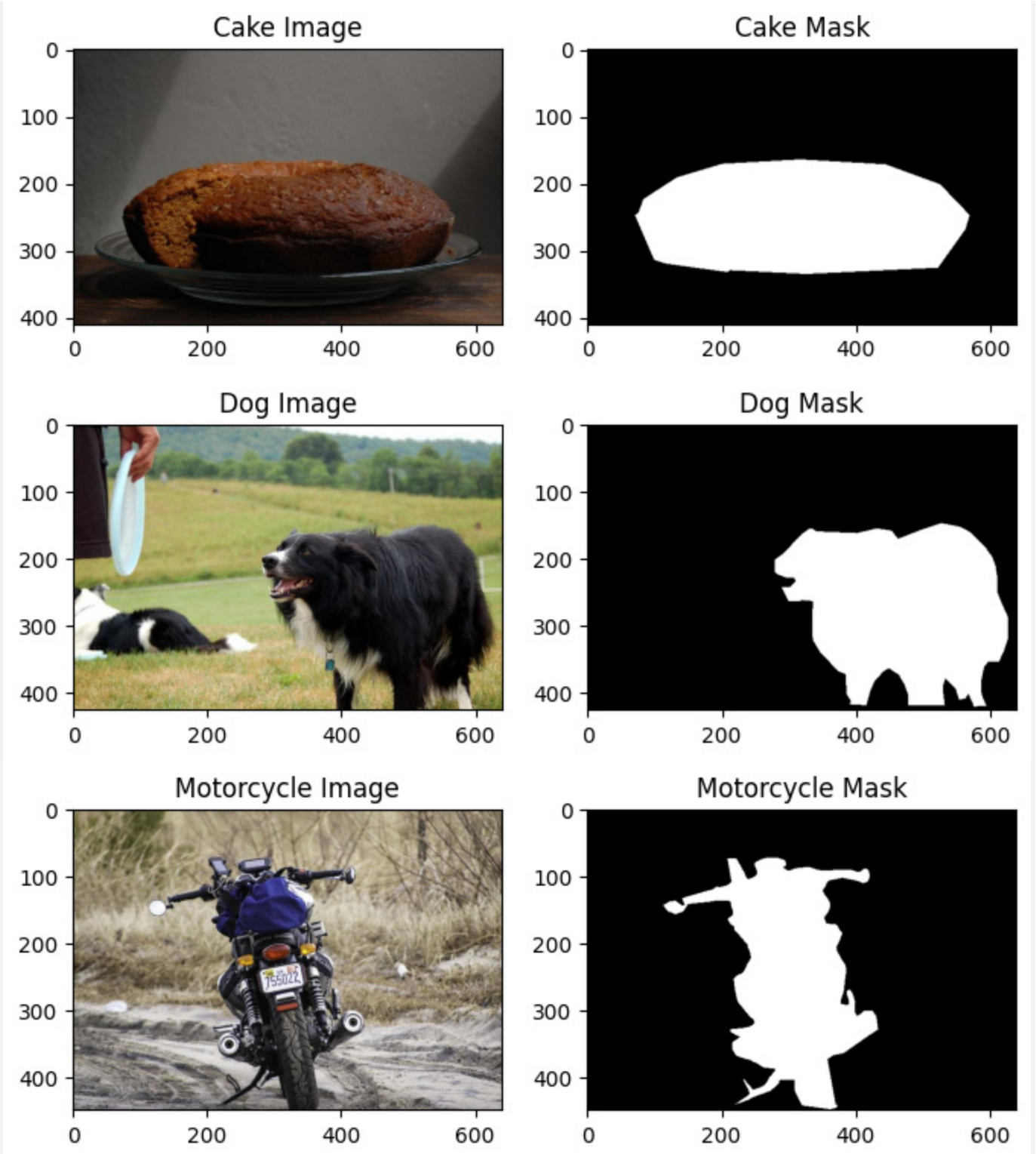

Figure 11: Sample Train images and their masks

parameters). This upsampling and convolutions on larger image sizes would significantly increase the number of parameters in the model, resulting in a substantial increase in training time. By resizing to 64 × 64 pixels, the computational burden was reduced while still allowing for effective training of the model. A satisfactory output was achieved even with the downsized 64x64 image size. Despite the smaller dimensions, the segmentation model was able to effectively capture and delineate the objects of interest from the images

It's important to note that in the context of this task, the actual label associated with each image class is not important. This is because the UNet model is trained solely to generate binary segmented masks, disregarding the specific class labels. Additionally, unlike the mUNET variant, which also predicts bounding boxes, the UNet model for this task does not undergo training to produce any bounding box predictions. While it's feasible to incorporate both the class label and bounding box predictions if necessary, it's omitted here as it's not a requirement for bonus the task at hand.

Furthermore, a modification was made to the UNet architecture wherein the final layer was altered to have only one output channel. Specifically, the final convolutional layer was adjusted to output a single-channel mask (self.conv_out = nn.ConvTranspose2d(64, 1, 3, stride=2, dilation=2, output_padding=1, padding=2)), as opposed to the five channels used in the PurdueShapes5MultiObject scenario. This adjustment simplifies the model's output to produce a binary mask, aligning with the objective of semantic segmentation in this task.

3.1 Outputs Discussion

3.1.1 MAE Loss

Instead of MSE Loss, MAE Loss is used for Coco Dataset segmentation. This was because the MSE losses were leading to exploding gradient issue, resulting in "nan" outputs. Exploding gradient losses happen when the gradients in the neural network become extremely large during training, causing the network's weights to update drastically and leading to unstable training. This instability can result in "nan" outputs, meaning the network's predictions become undefined or not a number.

The main reason for this issue with MSE loss is that it squares the errors between predicted and actual values during training. When there are large errors, squaring them amplifies their magnitude, leading to even larger gradients during backpropagation. This is especially true for a complicated dataset like Coco. These large gradients can accumulate and eventually cause the model's weights to update excessively, resulting in unstable training and ultimately "nan" outputs.

By using MAE loss instead, which calculates the average of the absolute differences between predicted and actual values, the issue of exploding gradient losses is mitigated. MAE loss does not amplify errors as drastically as MSE loss does, leading to more stable training and preventing the occurrence of "nan" outputs. Therefore, switching to MAE loss was a necessary adjustment to ensure smoother and more reliable training for the Coco Dataset segmentation task.

The MAE training loss is shown in Figure 12. The trend observed in the MAE loss values suggests that the loss gradually decreases over the course of training epochs and iterations. Initially, during the first epoch, the loss exhibits some fluctuations but generally decreases from around 81.688 to 76.785. This trend continues into the subsequent epochs, with the loss values generally decreasing, albeit with some fluctuations, indicating the model's learning progress. Towards the later epochs, particularly in the fifth and sixth epochs, the loss stabilizes at lower values, around 72-75, indicating that the model has converged to a relatively stable state.

Three output images along with their ground truth masks and the predicted masks is shown in Figures 13, 14 and 15.

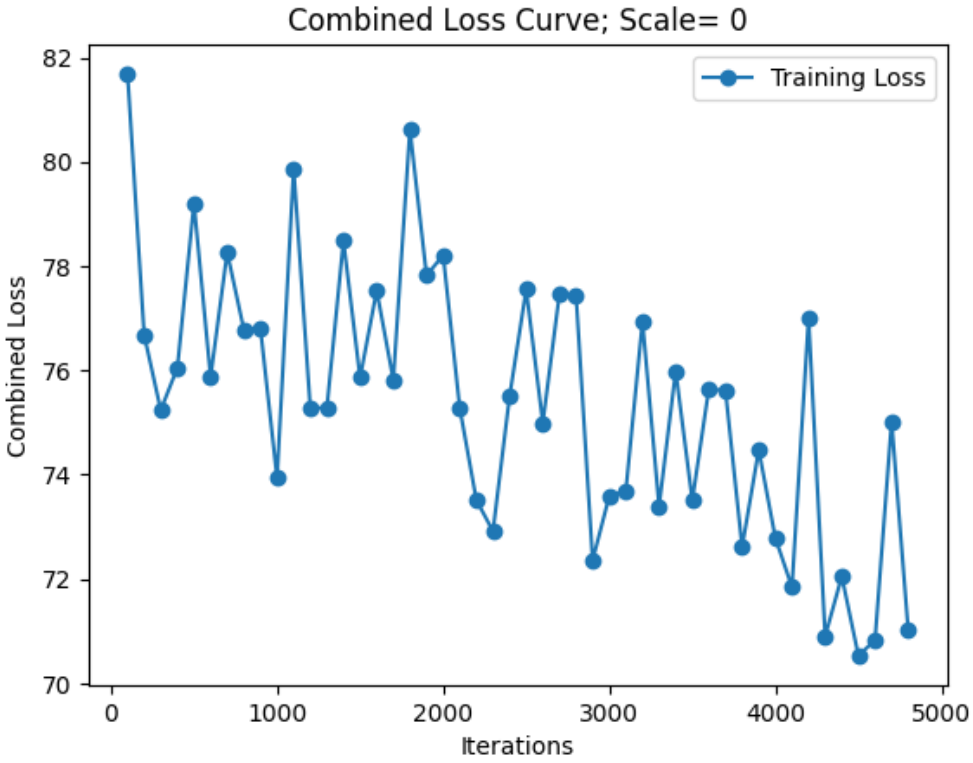

Figure 12: Training Loss MAE - Coco Dataset

3.1.2 Dice Loss

The Dice training loss is shown in Figure 16.In this trend, the loss decreases gradually over epochs and iterations, indicating that the model is consistently improving its performance. The loss values stabilize around a certain range after a few epochs, suggesting that the model has converged to a stable solution.

This trend contrasts with the previous loss trend, where the MAE loss fluctuated more significantly from iteration to iteration. The fluctuations in the previous trend could indicate instability in training, potentially caused by exploding gradient issues or other factors leading to erratic behavior during optimization. The stability and smooth decrease in Dice loss in this trend suggest that the model training is more reliable and robust. The absence of drastic fluctuations in loss values indicates that the model is not getting stuck in local minima and is able to navigate the optimization landscape effectively. This can be seen in the comparatively better (compared to MAE loss output masks) output predicted masks as seen in Figures 17, 18 and 19. More patterns are captured compared to the MAE loss, but cant be decicively said the predicted masks are better than the MAE. THe magic happens in the combined loss case

3.1.3 Combined Loss - Scale 100

The Dice training loss is shown in Figure 20. In this trend, the combined loss starts relatively high at the beginning of training, around 170, and gradually decreases as the training progresses. We observe a consistent downward trend in loss values over both epochs and iterations.

During the initial epochs and iterations, the model's performance improves rapidly, with the loss decreasing significantly from around 170 to approximately 140. As training continues, the rate

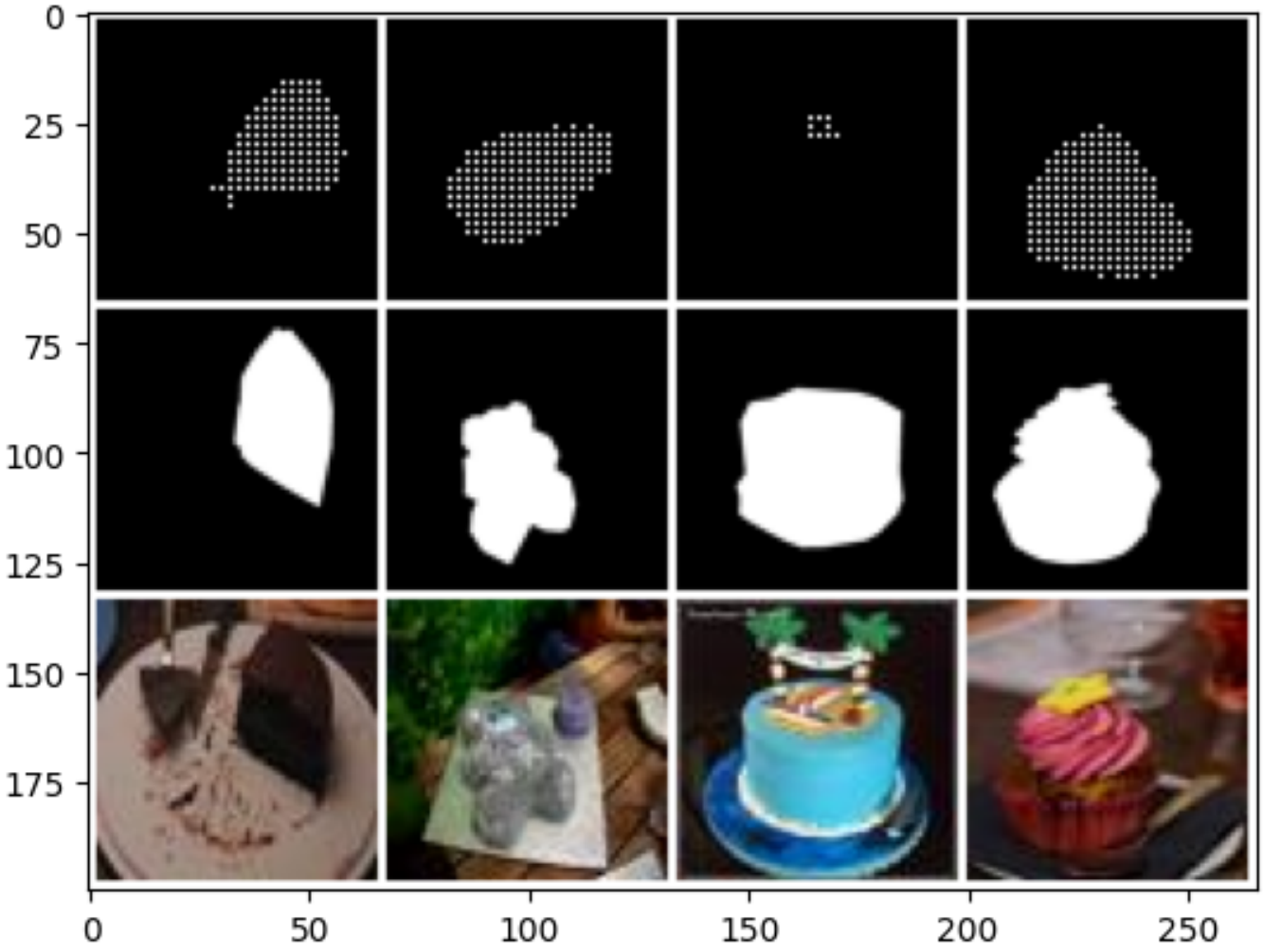

Figure 13: Sampled Cake Test images and their predicted masks for MAE Loss

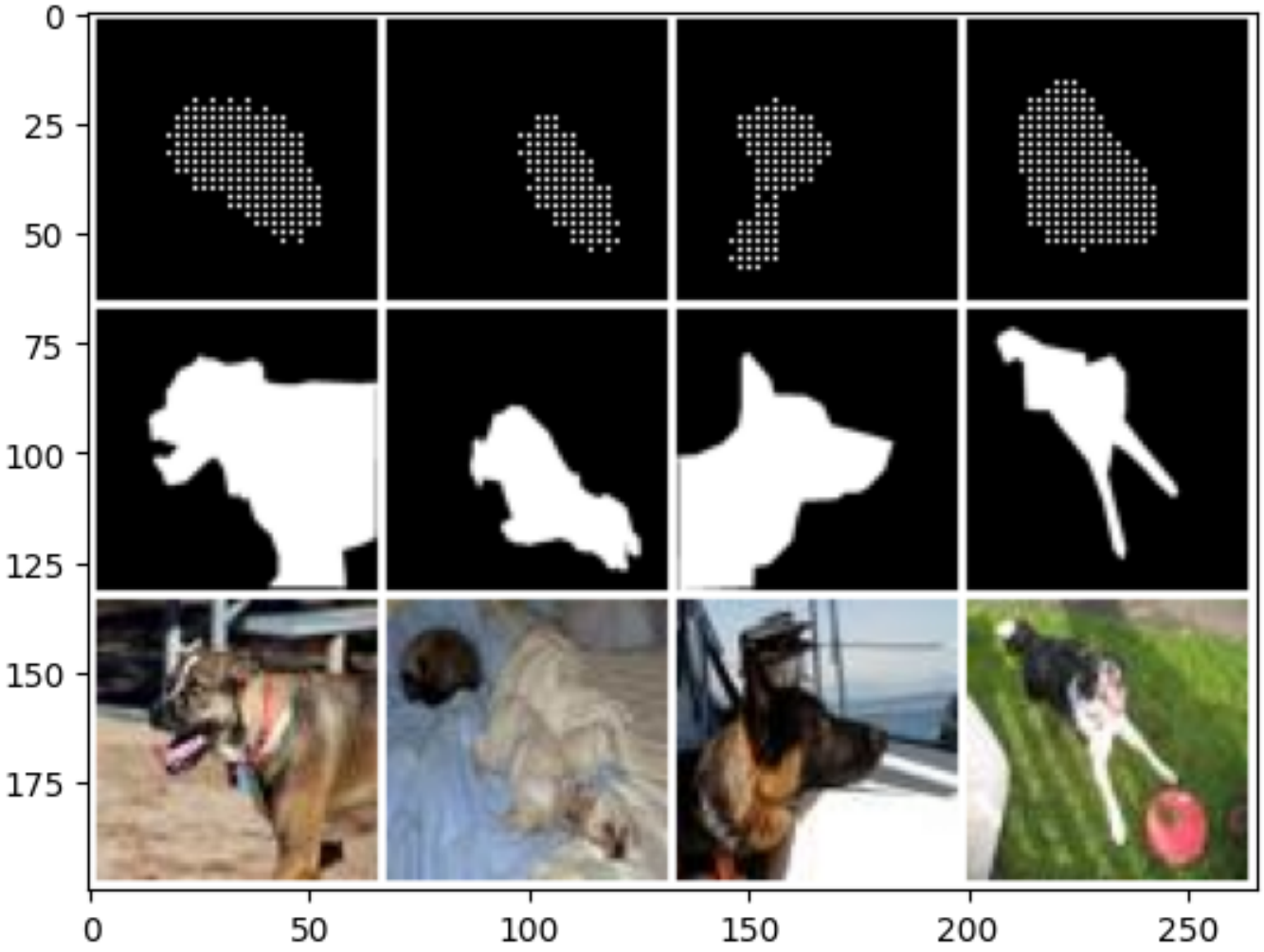

Figure 14: Sampled Dog Test images and their predicted masks for MAE Loss

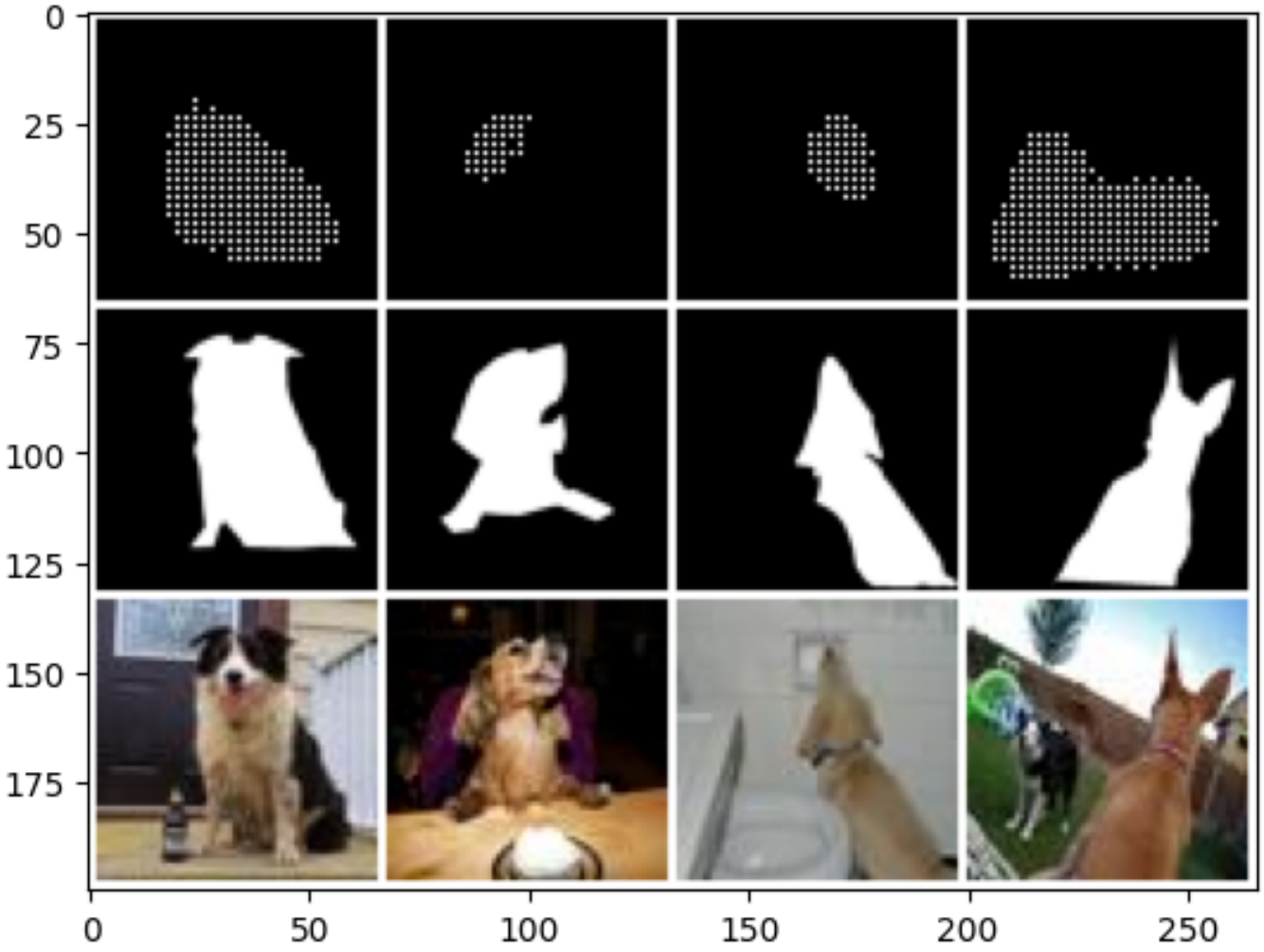

Figure 15: Sampled Motorcycle Test images and their predicted masks for MAE Loss

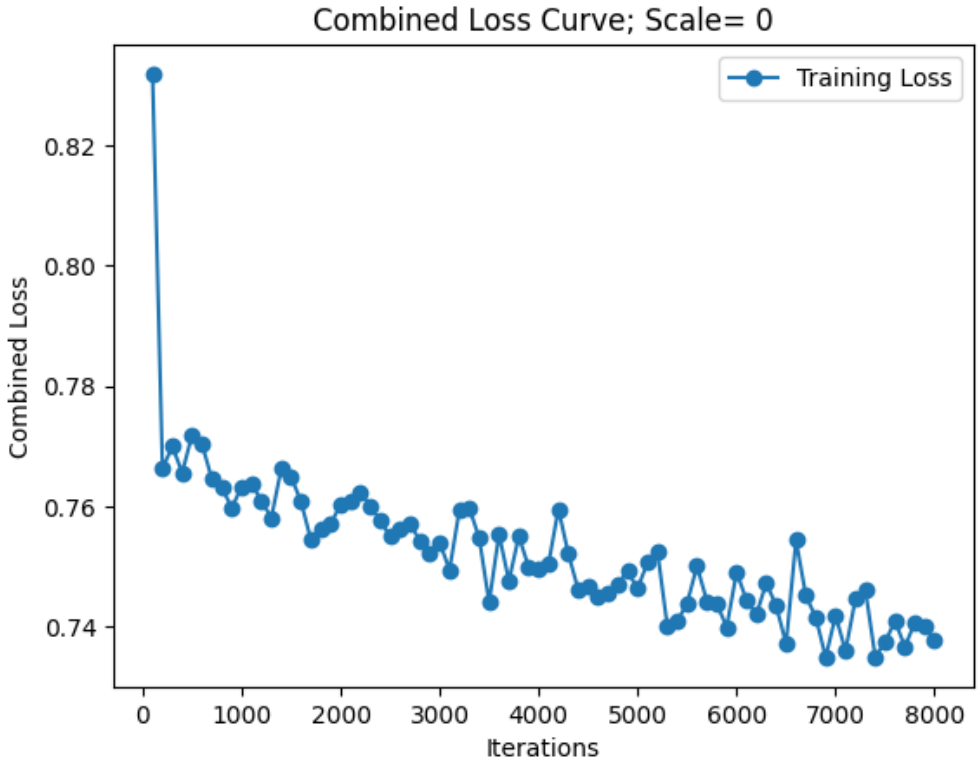

Figure 16: Training Loss Dice - Coco Dataset

of decrease in MSE loss slows down, indicating that the model is approaching convergence.

Three output images along with their ground truth masks and the predicted masks is shown in Figures 21, 22 and 23.

Initially, it may seem that the predicted mask is quite distant from the ground truth mask. However, upon closer examination of the original image, we can discern that the predicted mask is indeed capturing the underlying patterns present in the input images. This suggests that the model is beginning to grasp the essential features of the data.

With access to more training resources, additional training epochs, and more sophisticated network architectures, it is likely that higher levels of accuracy can be attained. These resources would allow the model to further refine its understanding of the data and improve its ability to accurately predict the segmentation masks.

4 Conclusion

By the end of this homework, we have developed a good understanding of how neural networks with Encoder-Decoder architecture can be used for semantic segmentation tasks, and we have gained hands-on experience with two different datasets.

References

[1] URL `https://engineering.purdue.edu/kak/distDLS/#113`

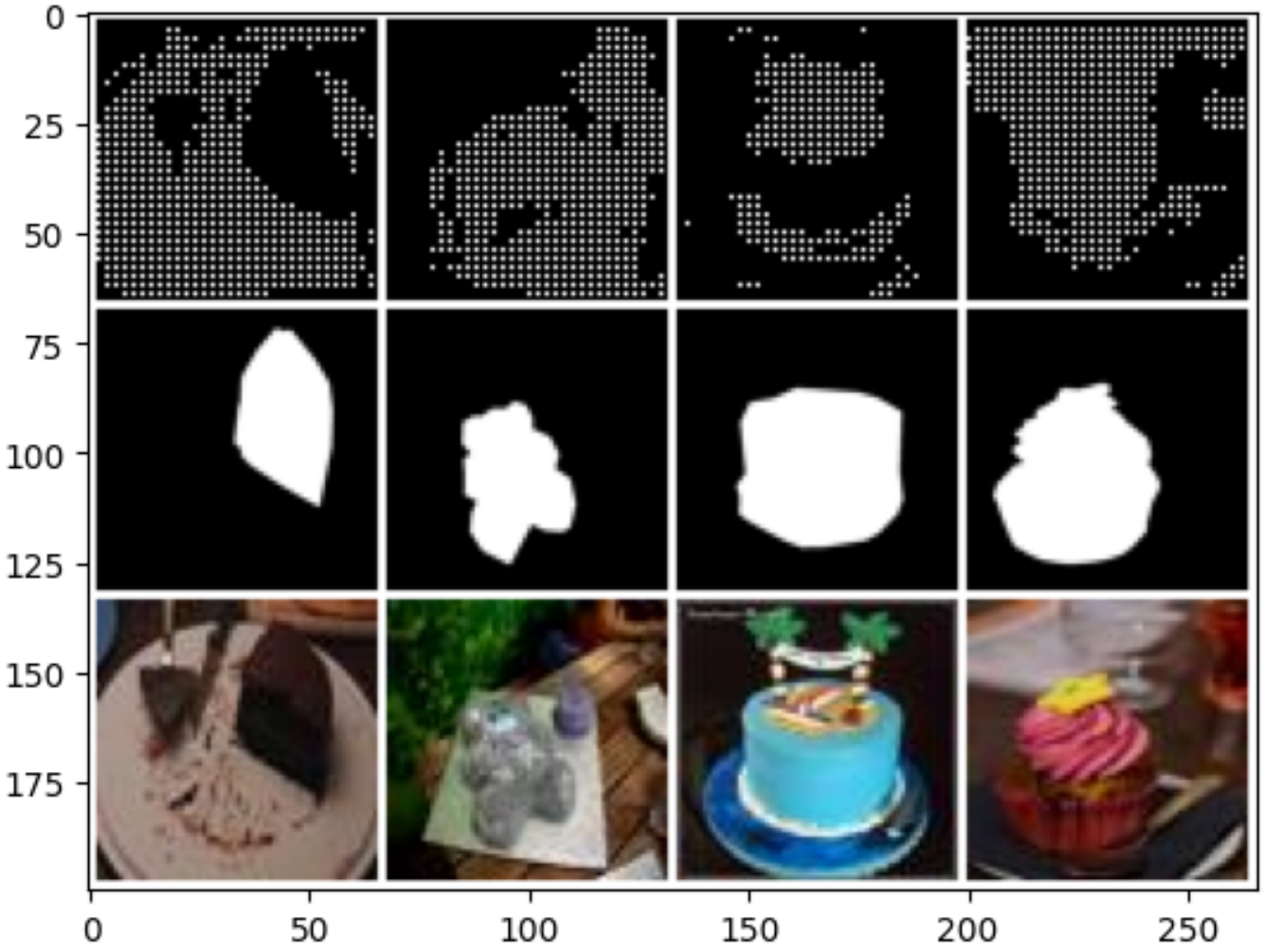

Figure 17: Sampled Cake Test images and their predicted masks for Dice Loss

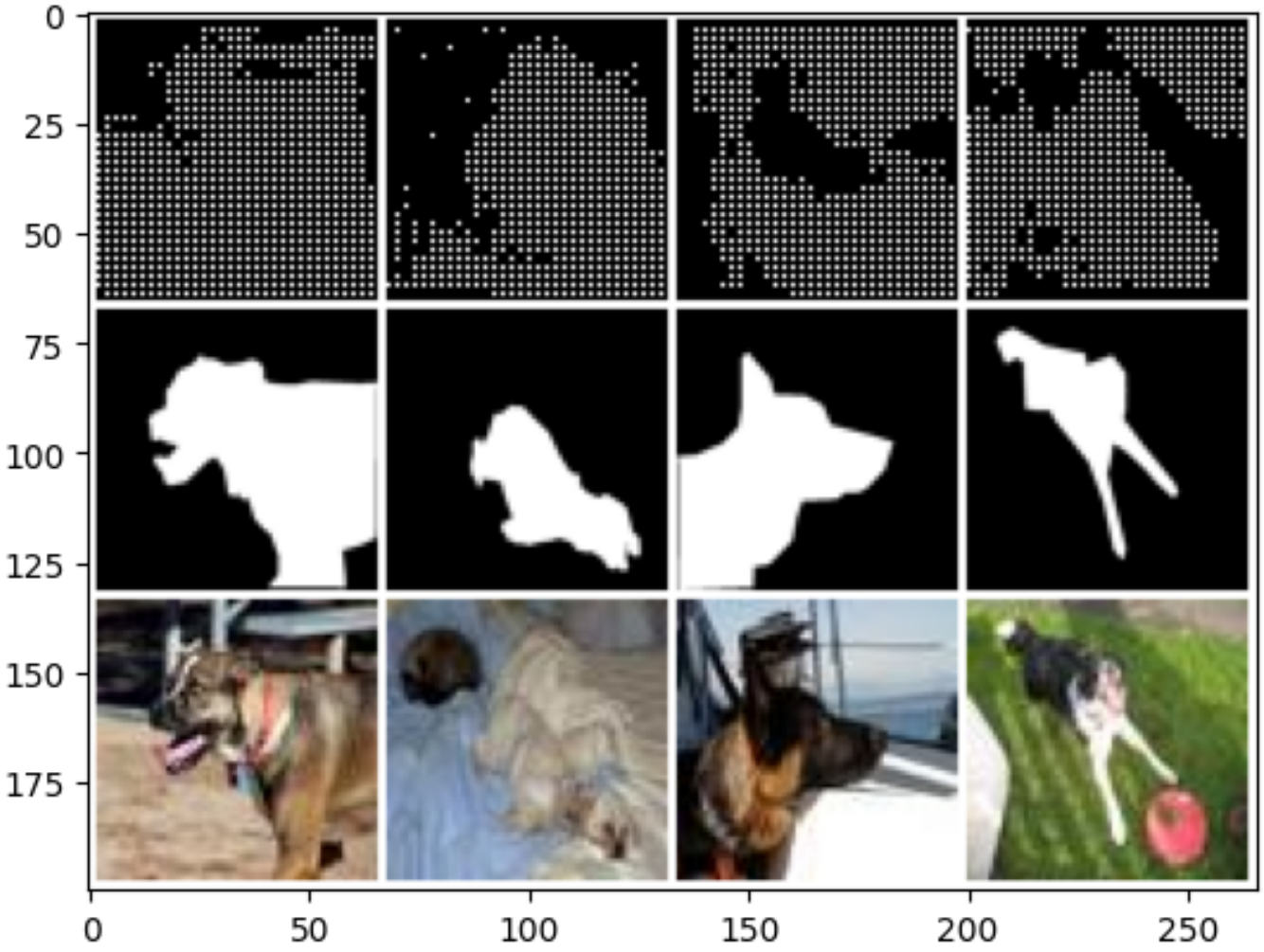

Figure 18: Sampled Dog Test images and their predicted masks for Dice Loss

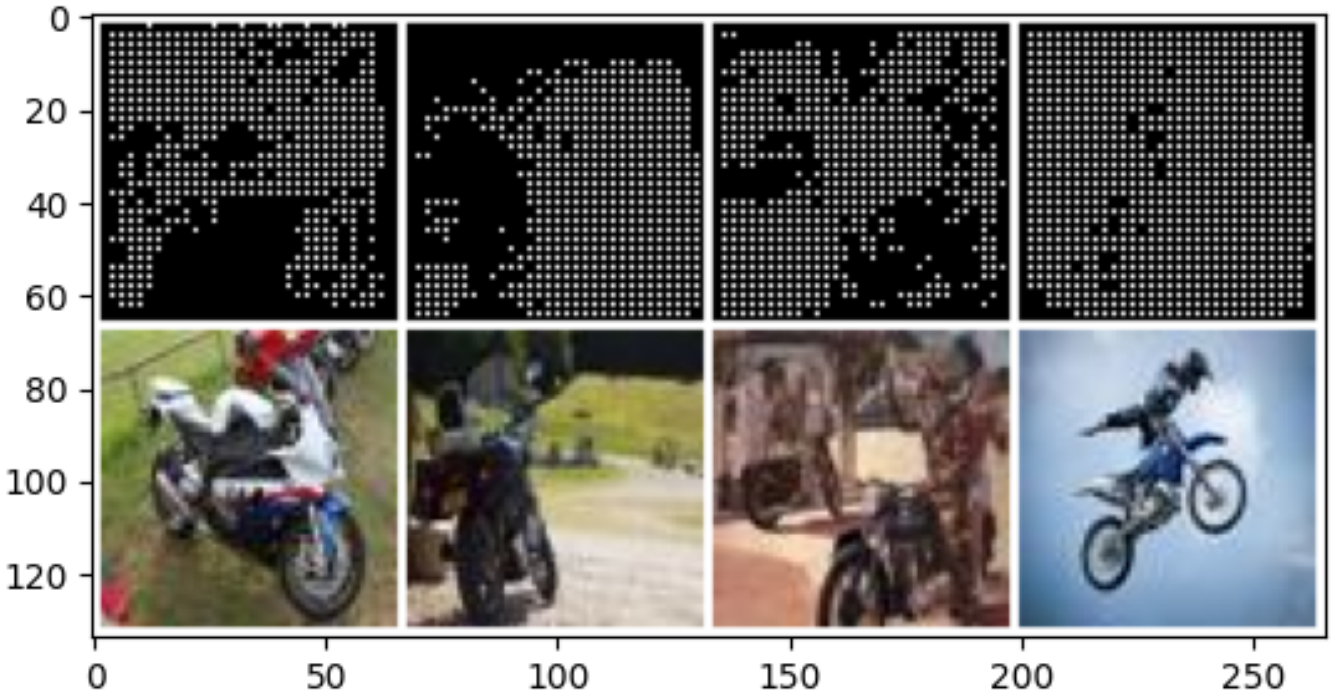

Figure 19: Sampled Motorcycle Test images and their predicted masks for Dice Loss

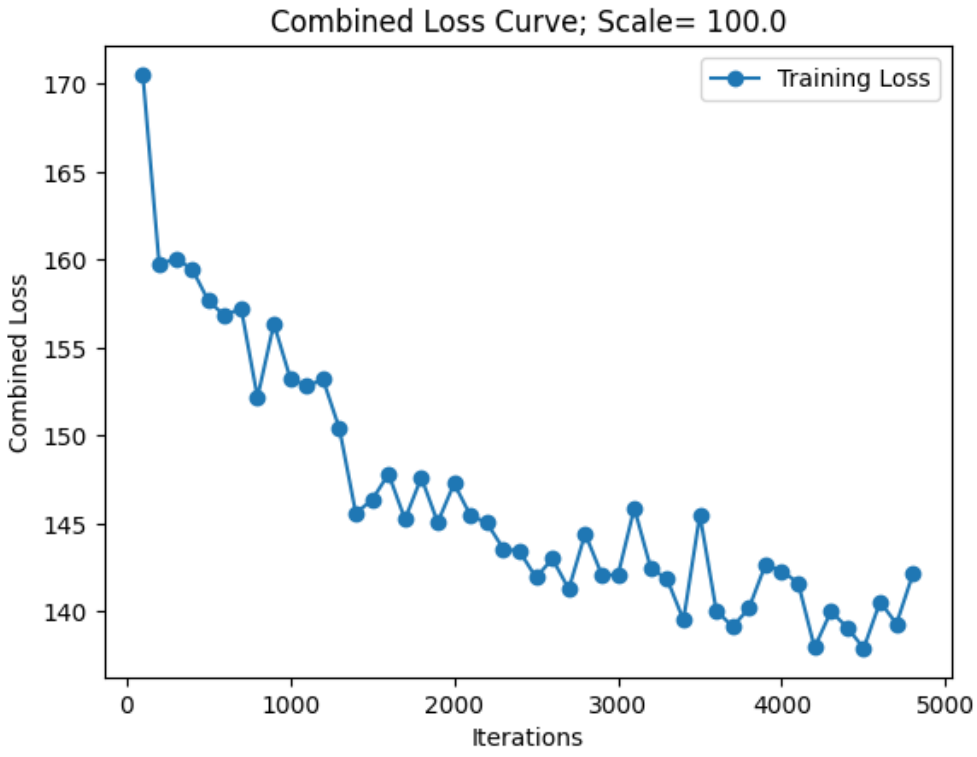

Figure 20: Training Loss Combined - Coco Dataset

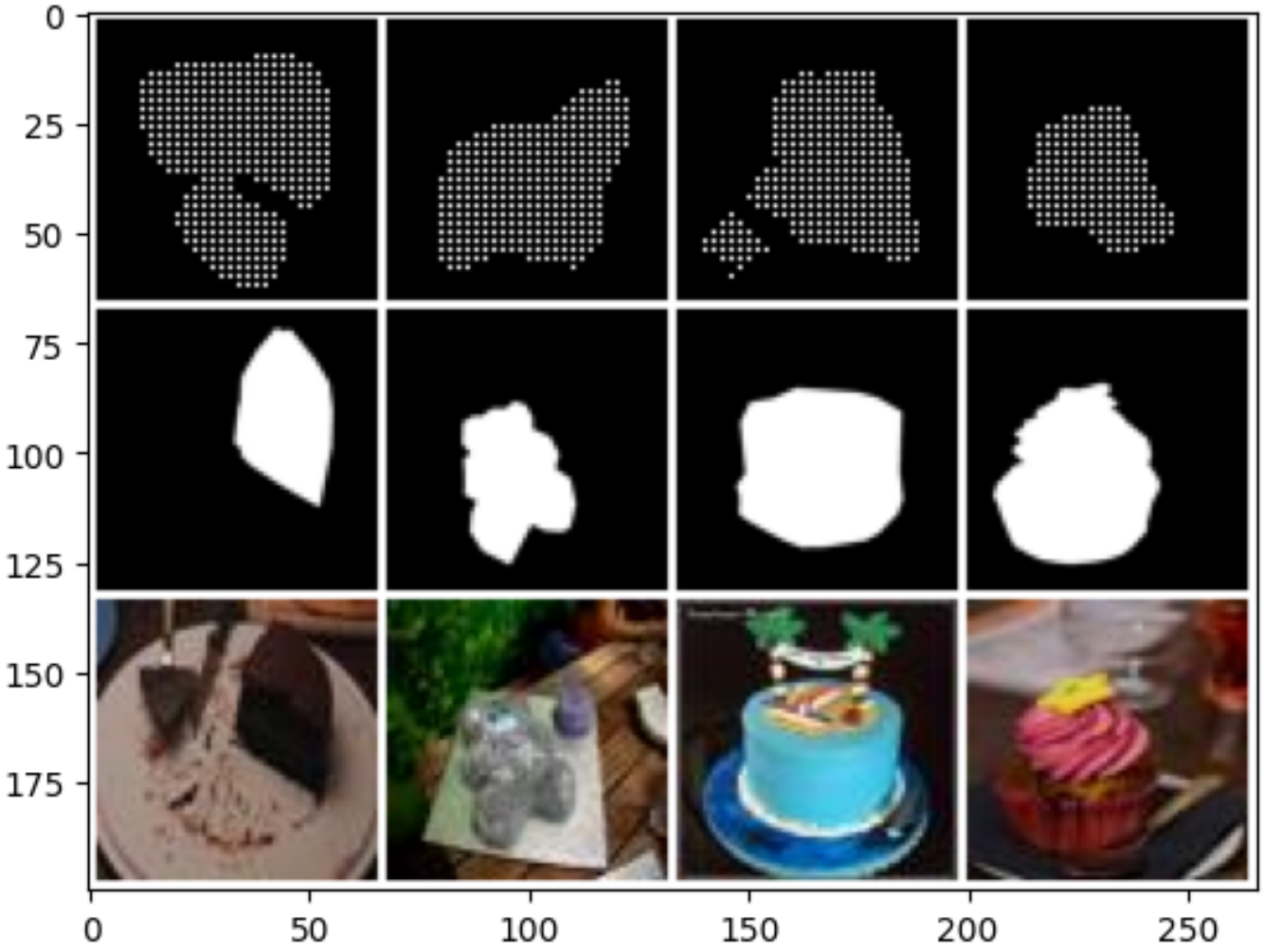

Figure 21: Sampled Cake Test images and their predicted masks for Combined Loss

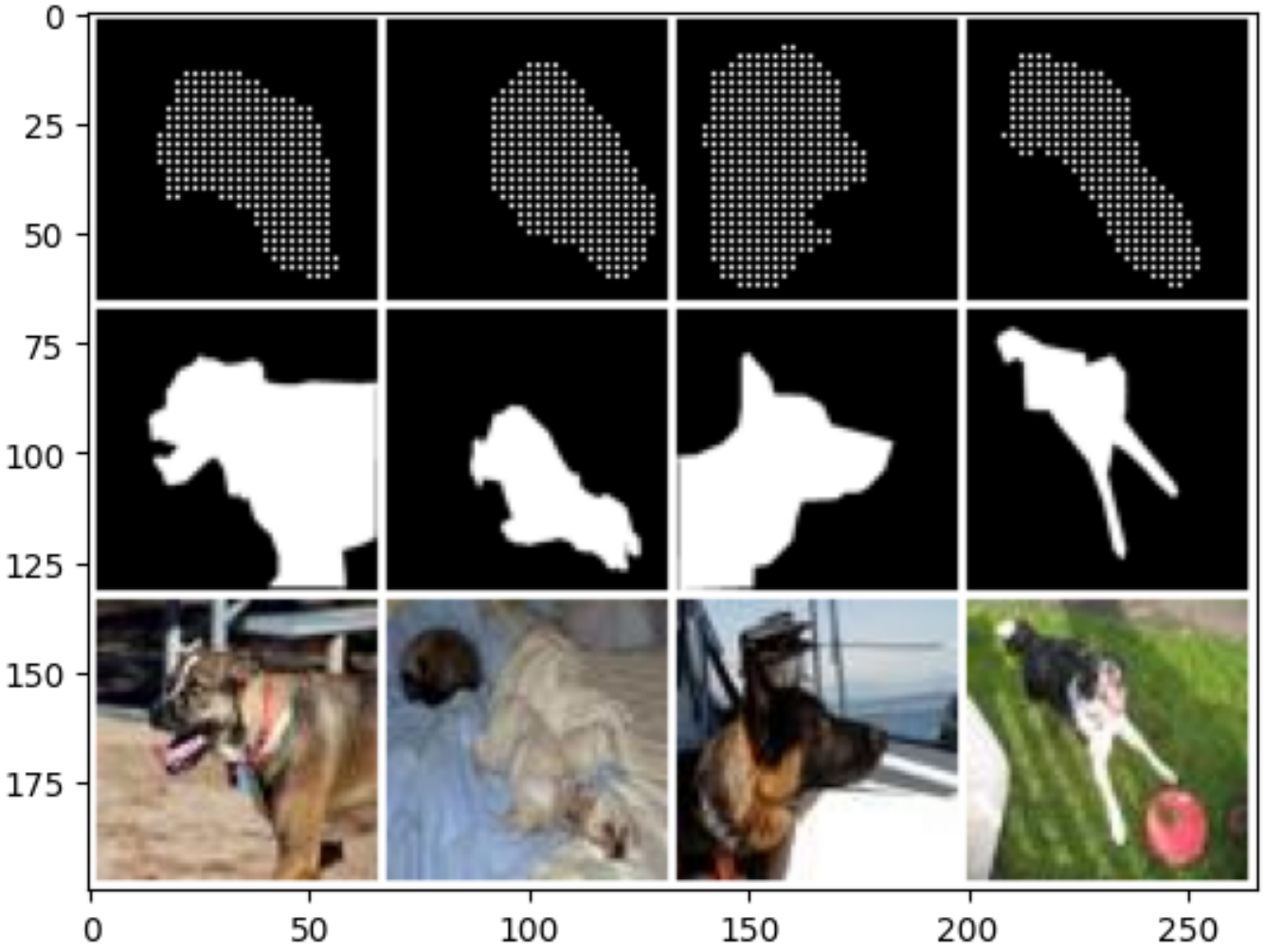

Figure 22: Sampled Dog Test images and their predicted masks for Combined Loss

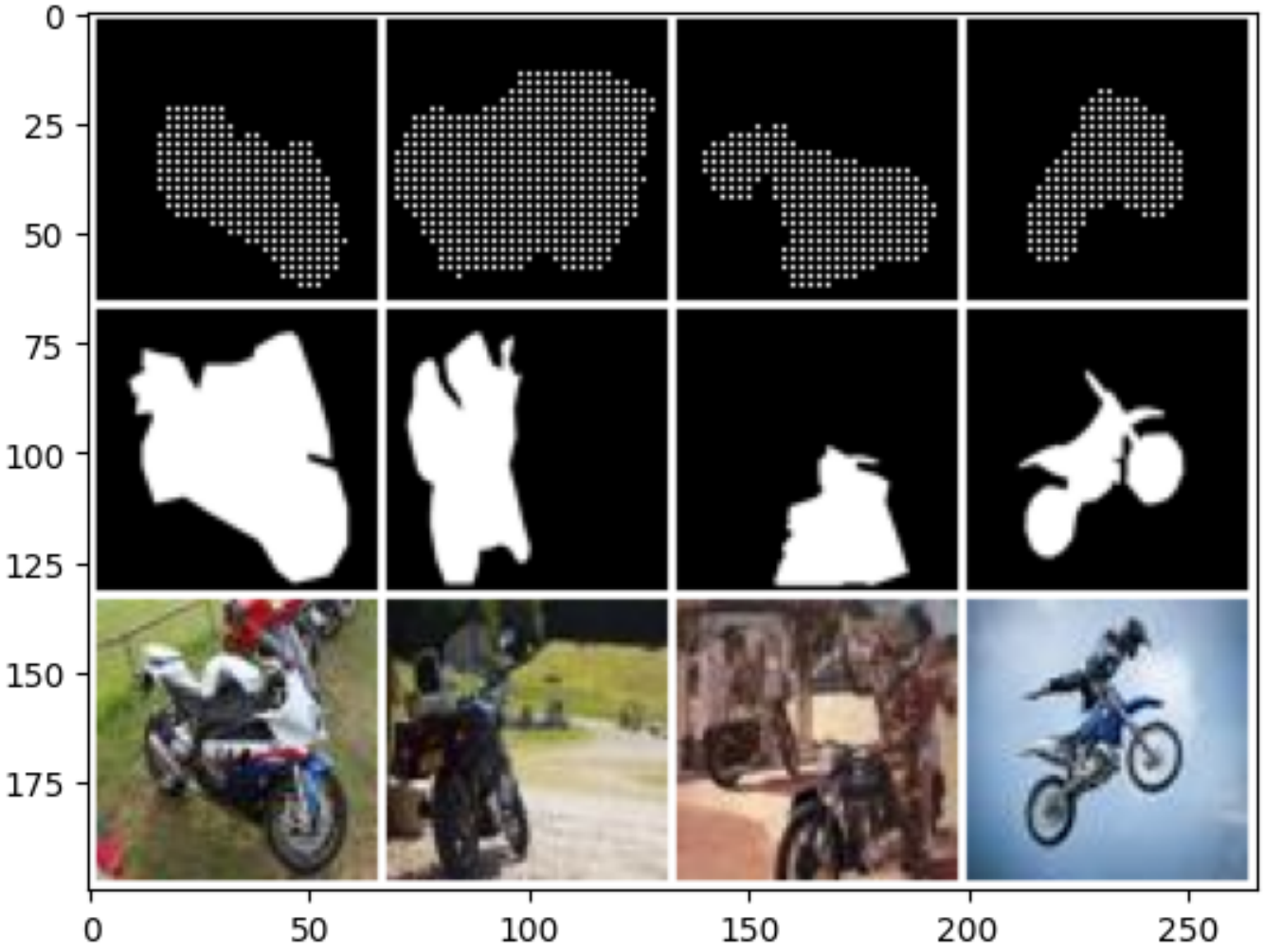

Figure 23: Sampled Motorcycle Test images and their predicted masks for Combined Loss