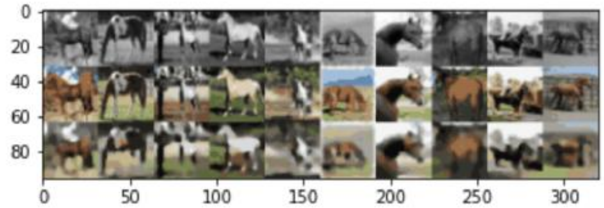
Programming Assignment 2

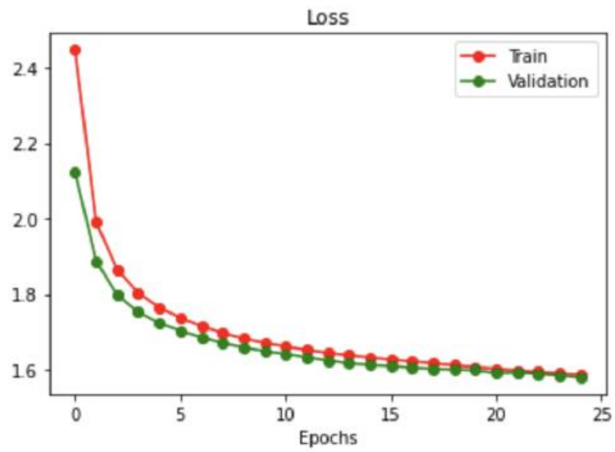
Part A: Pooling and Upsampling

1.

```
class PoolUpsampleNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        padding = kernel // 2

        ############### YOUR CODE GOES HERE ###############
        self.sequential1 = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.sequential2 = nn.Sequential(
            nn.Conv2d(num_filters, 2*num_filters, kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        self.sequential3 = nn.Sequential(
            nn.Conv2d(2*num_filters, num_filters, kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.sequential4 = nn.Sequential(
            nn.Conv2d(num_filters, num_colours, kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        self.conv1 = nn.Conv2d(num_colours, num_colours, kernel_size=kernel, padding=padding)
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        x = self.sequential1(x)
        x = self.sequential2(x)
        x = self.sequential3(x)
        x = self.sequential4(x)
        x = self.conv1(x)
        return x
        ###################################################
```

2.





The results do not show great prediction of the colour. This may be because samples of size 8 x 8 were upsampled to 32 x 32 which leads to low resolution.

3.

Input dimension: 32 x 32			
Layer	# of Outputs	# of Weights	# of Connections
Conv2d	$32^2 \cdot NF$	$3^2 \cdot NIC \cdot NF + NF$	$32^2(3^2 \cdot NIC \cdot NF + NF)$
MaxPool2d	$16^2 \cdot NF$	0	$32^2 \cdot NF$
BatchNorm2d	$16^2 \cdot NF$	$2NF$	16^4
Conv2d	$16^2 \cdot 2NF$	$3^2 \cdot NF \cdot 2NF + 2NF$	$16^2(3^2 \cdot NF \cdot 2NF + 2NF)$
MaxPool2d	$8^2 \cdot 2NF$	0	$16^2 \cdot 2NF$
BatchNorm2d	$8^2 \cdot 2NF$	$4NF$	8^4
Conv2d	$8^2 \cdot NF$	$3^2 \cdot 2NF \cdot NF + NF$	$8^2(3^2 \cdot 2NF \cdot NF + NF)$
Upsample	$16^2 \cdot NF$	0	$16^2 \cdot NF$
BatchNorm2d	$16^2 \cdot NF$	$2NF$	16^4
Conv2d	$16^2 \cdot NC$	$3^2 \cdot NF \cdot NC + NC$	$16^2(3^2 \cdot NF \cdot NC + NC)$
Upsample	$32^2 \cdot NC$	0	$32^2 \cdot NC$
BatchNorm2d	$32^2 \cdot NC$	$2NC$	32^4
Conv2d	$32^2 \cdot NC$	$3^2 \cdot NC \cdot NC + NC$	$32^2(3^2 \cdot NC \cdot NC + NC)$
Total	$2880NF$ $+3328NC$	$36NF^2 + 9NC^2 + 9NIC \cdot NF$ $+9NF \cdot NC + 12NF + 4NC$	$5760NF^2 + 9216NC^2$ $+9216NIC \cdot NF + 2304NF \cdot NC$ $+3392NF + 2304NC + 1183744$

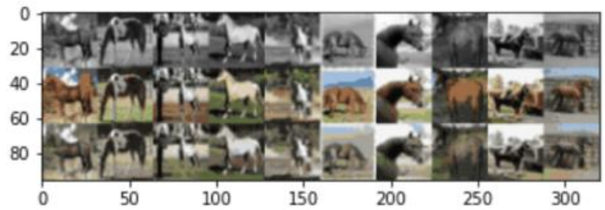
Input dimension: 64 x 64			
Layer	# of Outputs	# of Weights	# of Connections
Conv2d	$64^2 \cdot NF$	$3^2 \cdot NIC \cdot NF + NF$	$64^2(3^2 \cdot NIC \cdot NF + NF)$
MaxPool2d	$32^2 \cdot NF$	0	$64^2 \cdot NF$
BatchNorm2d	$32^2 \cdot NF$	$2NF$	32^4
Conv2d	$32^2 \cdot 2NF$	$3^2 \cdot NF \cdot 2NF + 2NF$	$32^2(3^2 \cdot NF \cdot 2NF + 2NF)$
MaxPool2d	$16^2 \cdot 2NF$	0	$32^2 \cdot 2NF$
BatchNorm2d	$16^2 \cdot 2NF$	$4NF$	16^4
Conv2d	$16^2 \cdot NF$	$3^2 \cdot 2NF \cdot NF + NF$	$16^2(3^2 \cdot 2NF \cdot NF + NF)$
Upsample	$32^2 \cdot NF$	0	$32^2 \cdot NF$
BatchNorm2d	$32^2 \cdot NF$	$2NF$	32^4
Conv2d	$32^2 \cdot NC$	$3^2 \cdot NF \cdot NC + NC$	$32^2(3^2 \cdot NF \cdot NC + NC)$
Upsample	$64^2 \cdot NC$	0	$64^2 \cdot NC$
BatchNorm2d	$64^2 \cdot NC$	$2NC$	64^4
Conv2d	$64^2 \cdot NC$	$3^2 \cdot NC \cdot NC + NC$	$64^2(3^2 \cdot NC \cdot NC + NC)$
Total	$11520NF$ $+13312NC$	$36NF^2 + 9NC^2 + 9NIC \cdot NF$ $+9NF \cdot NC + 12NF + 4NC$	$23040NF^2 + 36864NC^2$ $+36864NIC \cdot NF + 9216NF \cdot NC$ $+13568NF + 9216NC + 18939904$

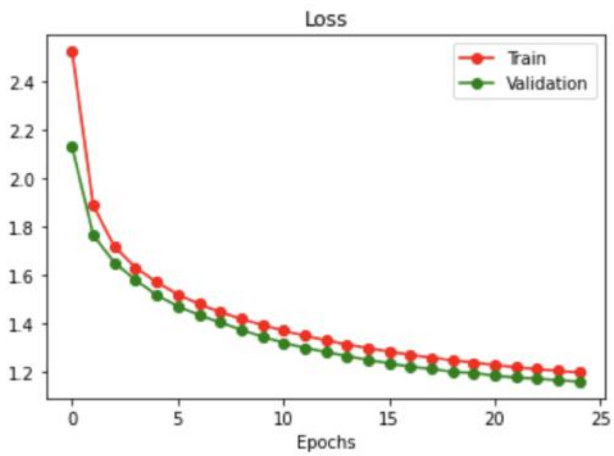
Part B: Strided and Transposed Convolutions

1.

```
class ConvTransposeNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        stride = 2
        padding = kernel // 2
        output_padding = 1

        ############### YOUR CODE GOES HERE ###############
        self.sequential1 = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.sequential2 = nn.Sequential(
            nn.Conv2d(num_filters, 2*num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        self.sequential3 = nn.Sequential(
            nn.ConvTranspose2d(2*num_filters, num_filters, kernel_size=kernel,
                               stride=stride, padding=padding, output_padding=output_padding),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.sequential4 = nn.Sequential(
            nn.ConvTranspose2d(num_filters, num_colours, kernel_size=kernel,
                               stride=stride, padding=padding, output_padding=output_padding),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        self.conv1 = nn.Conv2d(num_colours, num_colours, kernel_size=kernel, padding=padding)

        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        x = self.sequential1(x)
        x = self.sequential2(x)
        x = self.sequential3(x)
        x = self.sequential4(x)
        x = self.conv1(x)
        return x
        ###################################################
```

2.





3. The results from the `ConvTransposeNet` looks similar to the results in Part A. The validation loss is lower than the `PoolUpsampleNet` (1.6 vs 1.2). This may be because, unlike pooling and upsampling which are fixed functions, the weights and biases associated with strided and transposed convolutions can be learned increasing the expressive power of the neural network.

4. For a kernel size of 4, the `padding` parameter passed to the first two `Conv2d` layers can still be set to 1. For a kernal size of 5, `padding` should be set to 2 for the `Conv2d` layers.
For a kernel size of 4, the `padding` parameter passed to the `ConvTranspose2d` layers should be set to 1 and `output_padding` should be set to 0. However, if the kernel size is increased to 5, `padding` will need to be set to 2 and `output_padding` will need to be set to 1 for the `ConvTranspose2d` layers.

5.

Batch Size	Training Loss	Validation Loss
25	1.2520	1.1129
50	1.1850	1.1461
100	1.1980	1.1639
200	1.2577	1.2286
500	1.4204	1.4184

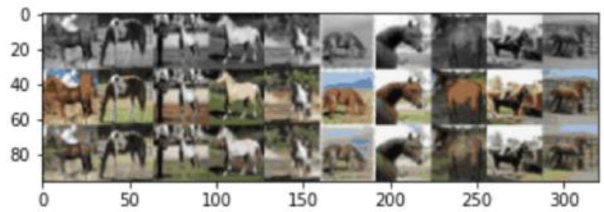
The smaller the batch size, the lower the training and validation loss and better the output image quality. However, too small of a batch size (e.g. 25) also results in higher losses and worse output image quality. The lowest losses and best image quality was obtained with a batch size of 50.

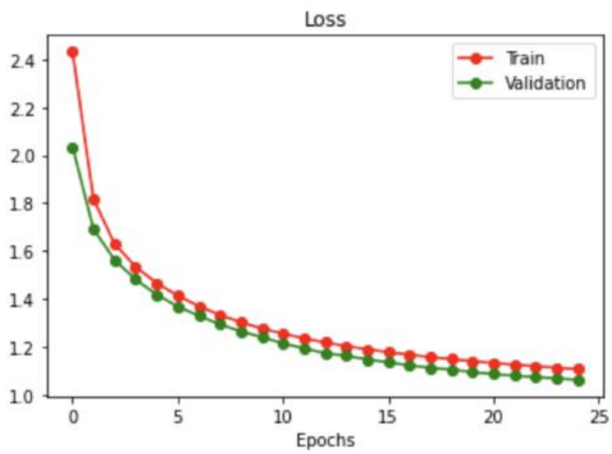
Part C: Skip Connections

1.

```
class UNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        stride = 2
        padding = kernel // 2
        output_padding = 1

        ############### YOUR CODE GOES HERE ###############
        self.sequential1 = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.sequential2 = nn.Sequential(
            nn.Conv2d(num_filters, 2*num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        self.sequential3 = nn.Sequential(
            nn.ConvTranspose2d(2*num_filters, num_filters, kernel_size=kernel,
                               stride=stride, padding=padding, output_padding=output_padding),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.sequential4 = nn.Sequential(
            nn.ConvTranspose2d(2*num_filters, num_colours, kernel_size=kernel,
                               stride=stride, padding=padding, output_padding=output_padding),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        self.conv1 = nn.Conv2d(num_in_channels+num_colours, num_colours, kernel_size=kernel, padding=padding)
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        x1 = self.sequential1(x)
        x2 = self.sequential2(x1)
        x3 = self.sequential3(x2)
        x4 = torch.cat((x1,x3), dim=1)
        x5 = self.sequential4(x4)
        x6 = torch.cat((x,x5), dim=1)
        x7 = self.conv1(x6)
        return x7
```

2.





3. The `UNet` performs better than the two previous models. It had the lowest losses and highest accuracy out of all the models. Qualitatively, the output images for the `UNet` are a little more accurate than the output images for the `ConvTransposeNet`. Skip connections might improve the performance of CNN models because:

1) The model is able to recover features that may have have been captured in the earlier layers but lost during downsampling.

2) Previous research by Li et al. has shown that deep neural networks with skip connections have a much smoother loss function near the vicinity of the global minimum than deep neural networks without skip connections [1]. Thus, skip connections can help with model convergence.

Part D: Image Segmentation as Classification

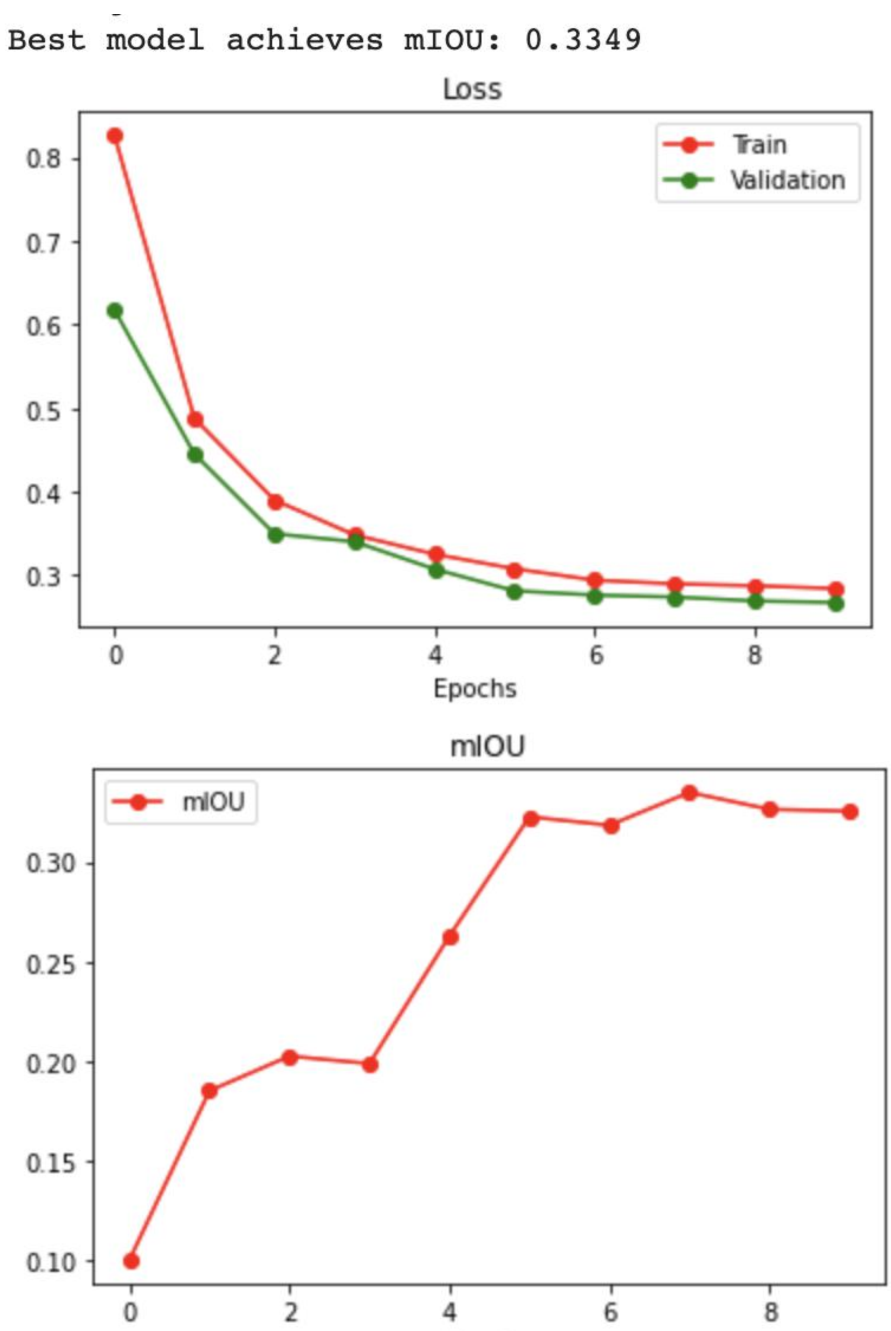
Part D.1

1.

```
def train(args, model):

    # Set the maximum number of threads to prevent crash in Teaching Labs
    torch.set_num_threads(5)
    # Numpy random seed
    np.random.seed(args.seed)

    # Save directory
    # Create the outputs folder if not created already
    save_dir = "outputs/" + args.experiment_name
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)

    learned_parameters = []
    # We only learn the last layer and freeze all the other weights
    ################ Code goes here ######################
    for name, param in model.named_parameters():
      if name.startswith("classifier.4"):
        print(name)
        learned_parameters.append(param)
    ######################################################
                                ........
```

2.

```
class AttrDict(dict):
    def __init__(self, *args, **kwargs):
        super(AttrDict, self).__init__(*args, **kwargs)
        self.__dict__ = self

args = AttrDict()
# You can play with the hyperparameters here, but to finish the assignment,
# there is no need to tune the hyperparameters here.
args_dict = {
    "gpu": True,
    "checkpoint_name": "finetune-segmentation",
    "learn_rate": 0.05,
    "train_batch_size": 128,
    "val_batch_size": 256,
    "epochs": 10,
    "loss": 'cross-entropy',
    "seed": 0,
    "plot": True,
    "experiment_name": "finetune-segmentation",
}
args.update(args_dict)

# Truncate the last layer and replace it with the new one.
# To avoid `CUDA out of memory` error, you might find it useful (sometimes required)
#   to set the `requires_grad`=False for some layers
################ YOUR CODE GOES HERE ######################
for name, param in model.named_parameters():
  if (not name.startswith("classifier.4")):
    param.requires_grad = False
model._modules["classifier"][4] = nn.Conv2d(256, 2, kernel_size=(1, 1), stride=(1, 1))
#######################################################


# Clear the cache in GPU
torch.cuda.empty_cache()
train(args, model)
```

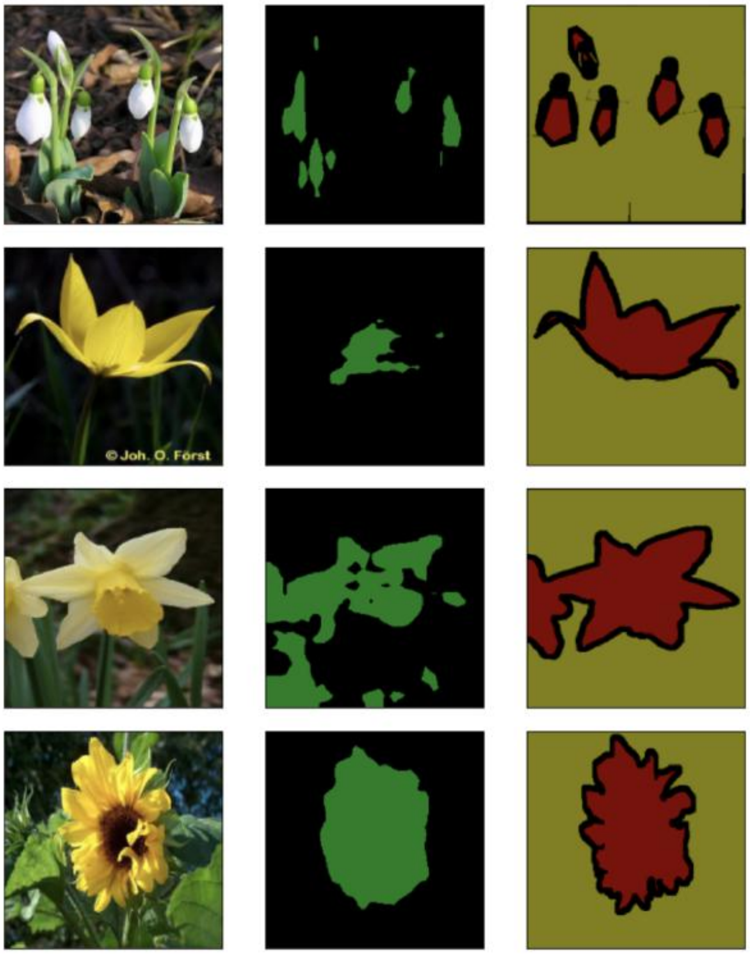


Best validation mIOU is 0.3349.

3.

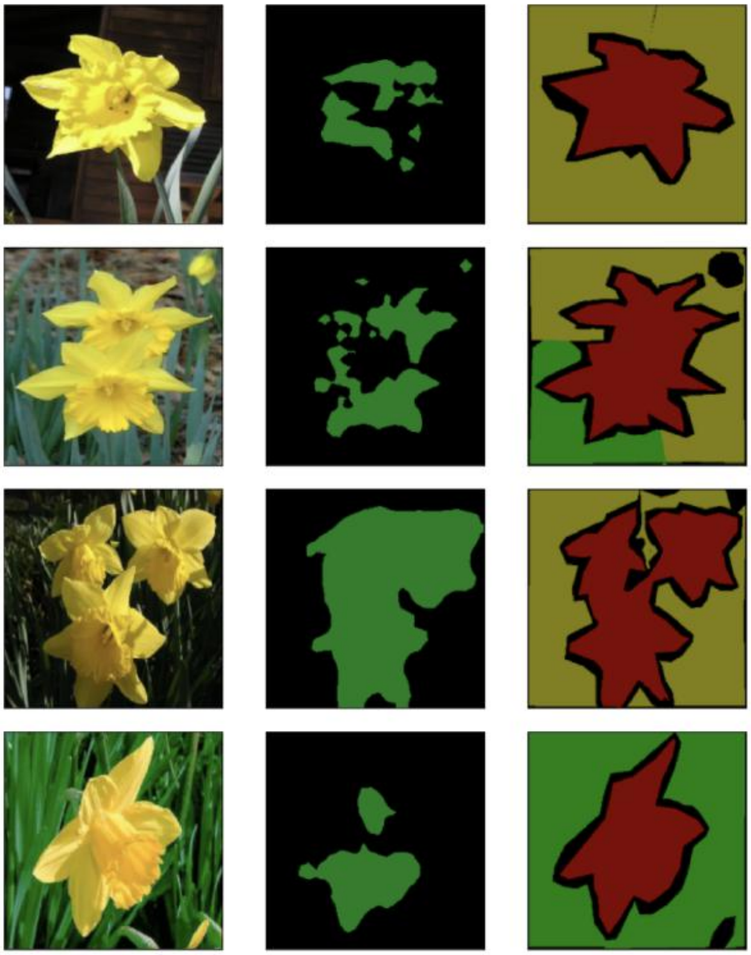
```
plot_prediction(args, model, is_train=True, index_list=[0, 1, 2, 3])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



```
plot_prediction(args, model, is_train=False, index_list=[0, 1, 2, 3])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

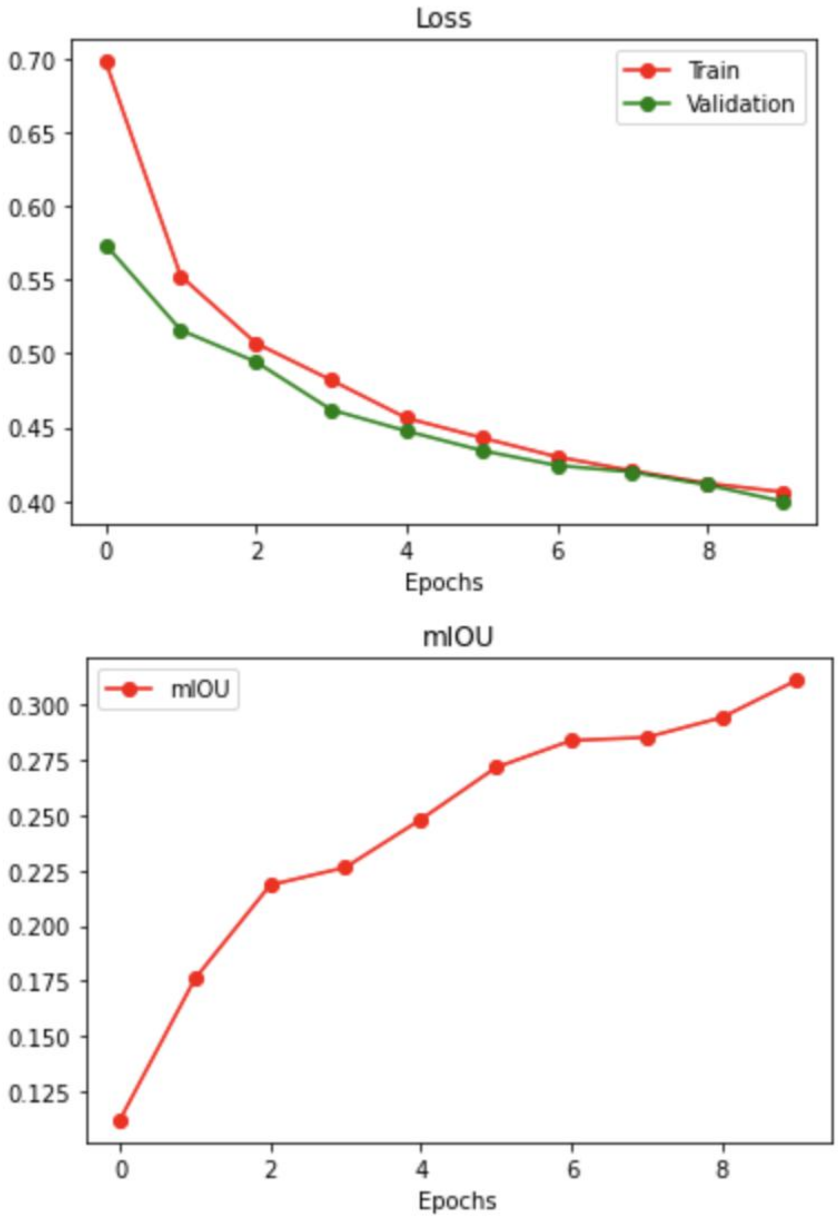


Part D.2

```
def compute_iou_loss(pred, gt, SMOOTH=1e-6):
    # Compute the IoU between the pred and the gt (ground truth)
    ################ YOUR CODE GOES HERE ######################
    m = nn.Softmax(dim=1)
    sm_pred = m(pred)
    foreground = sm_pred[:, 1, :, :]
    I_fgt = (foreground * gt).sum()
    U_fgt = (foreground + gt - foreground * gt).sum()
    loss = 1 - I_fgt/U_fgt
    ######################################################

    return loss
```

```
args = AttrDict()
# You can play with the hyperparameters here, but to finish the assignment,
# there is no need to tune the hyperparameters here.
args_dict = {
    "gpu": True,
    "checkpoint_name": "finetune-segmentation",
    "learn_rate": 0.05,
    "train_batch_size": 128,
    "val_batch_size": 256,
    "epochs": 10,
    "loss": 'iou',
    "seed": 0,
    "plot": True,
    "experiment_name": "finetune-segmentation",
}
args.update(args_dict)

# Truncate the last layer and replace it with the new one.
# To avoid `CUDA out of memory` error, you might find it useful (sometimes required)
#   to set the `requires_grad`=False for some layers
################ YOUR CODE GOES HERE ######################
for name, param in model.named_parameters():
  if (not name.startswith("classifier.4")):
    param.requires_grad = False
model._modules["classifier"][4] = nn.Conv2d(256, 2, kernel_size=(1, 1), stride=(1, 1))
######################################################


# Clear the cache in GPU
torch.cuda.empty_cache()
train(args, model)
```

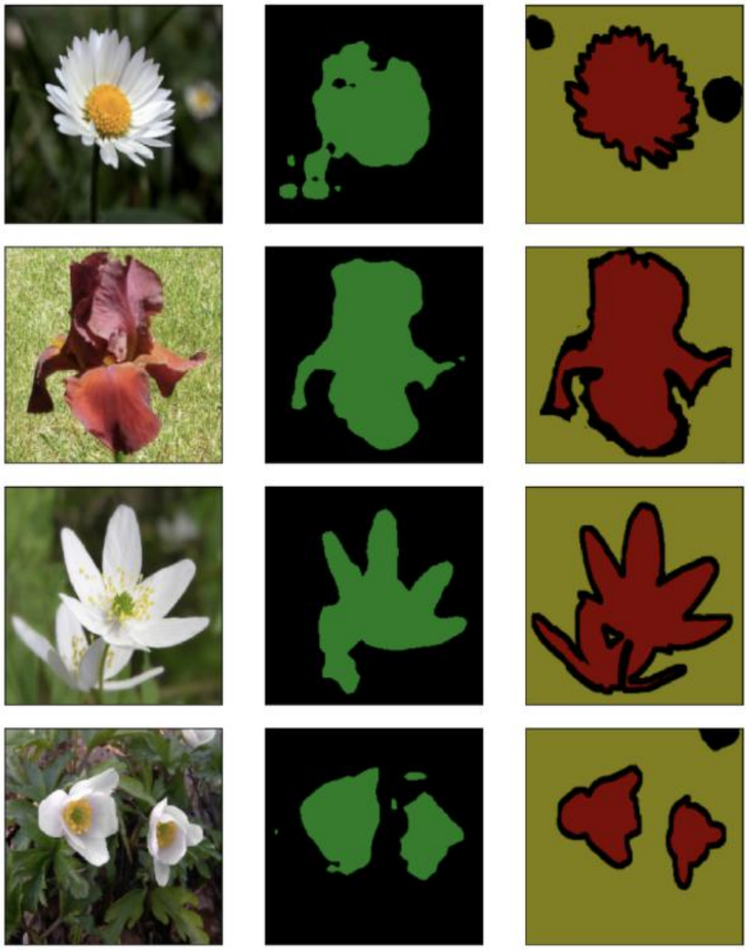
```
Best model achieves mIOU: 0.3113
```



Best validation mIOU is 0.3113. This value is smaller than the loss calculated in Part D.2.

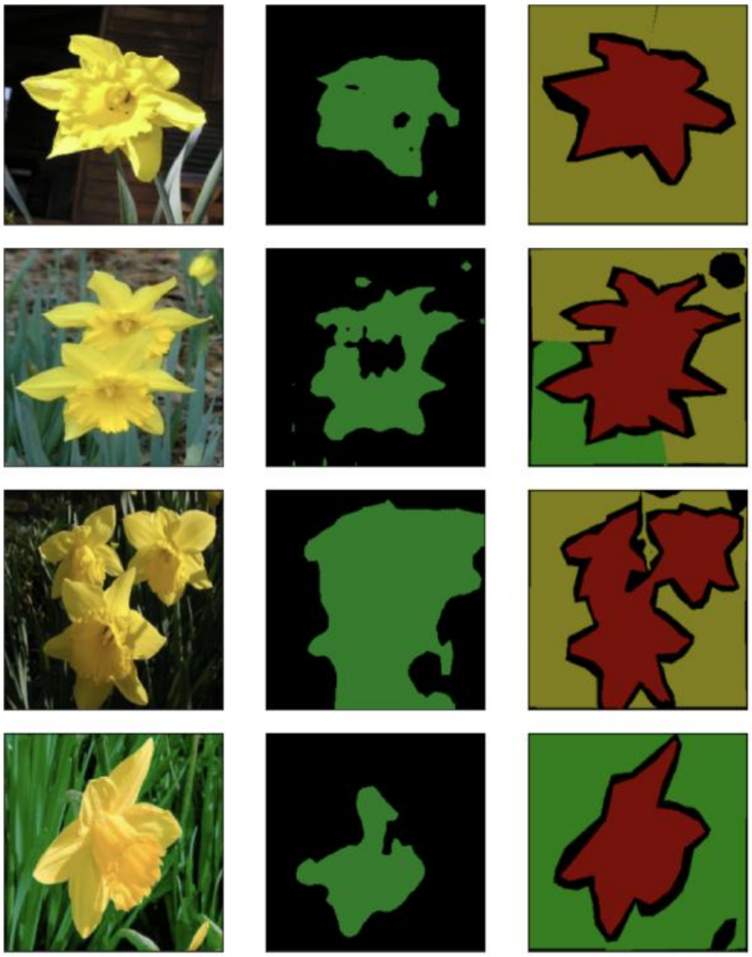
```
plot_prediction(args, model, is_train=True, index_list=[0, 1, 2, 3])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



```
plot_prediction(args, model, is_train=False, index_list=[0, 1, 2, 3])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



References

[1] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, *Visualizing the loss landscape of neural nets*, 2018. arXiv: `1712.09913 [cs.LG]`.