CSC2516: Programming Assignment 2: Convolutional Neural Networks

February 2021

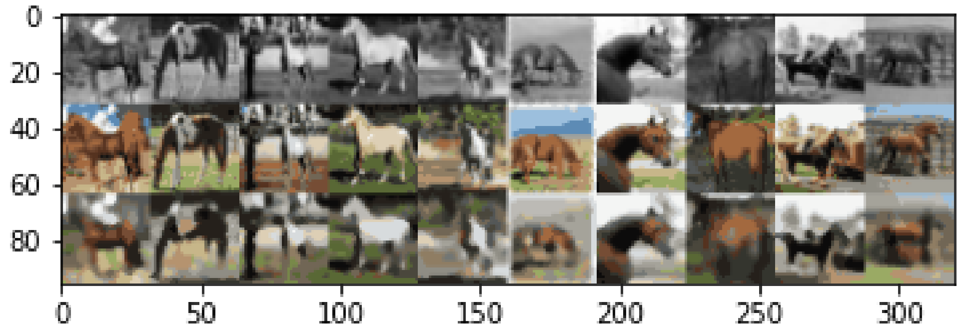
Part A: Pooling and Upsampling

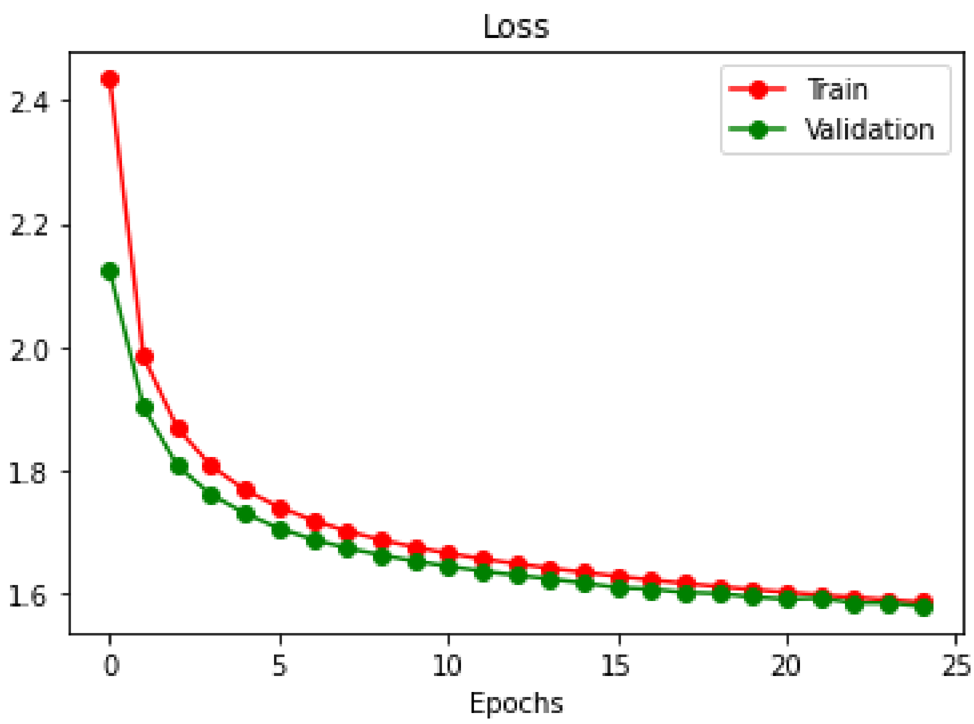
1 Complete the model PoolUpsampleNet

```python
#V202102171018
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 ###############
        # Input: [BS, NIC, 32, 32]
        self.block0 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels,
                      out_channels=num_filters,
                      kernel_size=kernel,
                      padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        # Output: [BS, NF, 16, 16]
        self.block1 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters,
                      out_channels=2*num_filters,
                      kernel_size=kernel,
                      padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        # Output: [BS, 2NF, 8, 8]
        self.block2 = nn.Sequential(
            nn.Conv2d(in_channels=2*num_filters,
                      out_channels=num_filters,
                      kernel_size=kernel,
                      padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        # Output: [BS, NF, 16, 16]
        self.block3 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters,
                      out_channels=num_colours,
                      kernel_size=kernel,
                      padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        # Output: [BS, NC, 32, 32]
        self.block4 = nn.Sequential(
            nn.Conv2d(in_channels=num_colours,
                      out_channels=num_colours,
                      kernel_size=kernel,
                      padding=padding)
        )
        # Output: [BS, NC, 32, 32]
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        for block in (self.block0, self.block1, self.block2,
                      self.block3, self.block4):
            x = block.forward(x)
        return x
        ###################################################
```

2 Run main training loop of PoolUpsampleNet

```
...
Epoch [25/25], Loss: 1.5852, Time (s): 33
Epoch [25/25], Val Loss: 1.5785, Val Acc: 41.6%, Time(s): 34.18
```





Do these results look good to you? Why or why not?

The images appear blurry because in the network they were up-sampled from 8x8 back to 32x32. The background and saddles also appear poorly coloured.

3 Compute the number of weights, outputs, and connections in the model

For 32x32 inputs:

Kernel Size	K	3
Number of Input Channels	NIC	3
Number of Filters	NF	32
Number of Colours	NC	24
Image Width	W	32
Image Height	H	32

	PoolUpsampleNet	Weights		Outputs		Connections	
	Image						
0	Conv2d	NF*NIC*K*K+NF	= 896	NF*H*W	= 32768	(NF*NIC*K*K+NF)*H*W	= 917504
	MaxPool2d			NF*H/2*W/2	= 8192	NF*H*W	= 32768
	BatchNorm2d	NF+NF	= 64	NF*H/2*W/2	= 8192	(H/2*W/2)*(H/2*W/2)	= 65536
	ReLU			NF*H/2*W/2	= 8192		
1	Conv2d	(2*NF)*NF*K*K+(2*NF)	= 18496	(2*NF)*H/2*W/2	= 16384	((2*NF)*NF*K*K+(2*NF))*H/2*W/2	= 4734976
	MaxPool2d			(2*NF)*H/4*W/4	= 4096	(2*NF)*H/2*W/2	= 16384
	BatchNorm2d	(2*NF)+(2*NF)	= 128	(2*NF)*H/4*W/4	= 4096	(H/4*W/4)*(H/4*W/4)	= 4096
	ReLU			(2*NF)*H/4*W/4	= 4096		
2	Conv2d	NF*(2*NF)*K*K+NF	= 18464	NF*H/4*H/4	= 2048	(NF*(2*NF)*K*K+NF)*H/4*W/4	= 1181696
	Upsample			NF*H/2*W/2	= 8192	NF*H/2*W/2	= 8192
	BatchNorm2d	NF+NF	= 64	NF*H/2*W/2	= 8192	(H/2*W/2)*(H/2*W/2)	= 65536
	ReLU			NF*H/2*W/2	= 8192		
3	Conv2d	NC*NF*K*K	= 6912	NC*H/2*W/2	= 6144	(NC*NF*K*K)*H/2*W/2	= 1769472
	Upsample			NC*H*W	= 24576	NC*H*W	= 24576
	BatchNorm2d	NC+NC	= 48	NC*H*W	= 24576	(H*W)*(H*W)	= 1048576
	ReLU			NC*H*W	= 24576		
4	Conv2d	NC*NC*K*K+NC	= 5208	NC*H*W	= 24576	(NC*NC*K*K+NC)*H*W	= 5332992
	Total		50280		217088		15202304

For 64x64 inputs:

	Kernel Size	K	3				
	Number of Input Channels	NIC	3				
	Number of Filters	NF	32				
	Number of Colours	NC	24				
	Image Width	W	64				
	Image Height	H	64				
PoolUpsampleNet		Weights		Outputs		Connections	
	Image						
0	Conv2d	NF*NIC*K*K+NF	= 896	NF*H*W	= 131072	(NF*NIC*K*K+NF)*H*W	= 3670016
	MaxPool2d			NF*H/2*W/2	= 32768	NF*H*W	= 131072
	BatchNorm2d	NF+NF	= 64	NF*H/2*W/2	= 32768	(H/2*W/2)*(H/2*W/2)	= 1048576
	ReLU			NF*H/2*W/2	= 32768		
1	Conv2d	(2*NF)*NF*K*K+(2*NF)	= 18496	(2*NF)*H/2*W/2	= 65536	((2*NF)*NF*K*K+(2*NF))*H/2*W/2	= 18939904
	MaxPool2d			(2*NF)*H/4*W/4	= 16384	(2*NF)*H/2*W/2	= 65536
	BatchNorm2d	(2*NF)+(2*NF)	= 128	(2*NF)*H/4*W/4	= 16384	(H/4*W/4)*(H/4*W/4)	= 65536
	ReLU			(2*NF)*H/4*W/4	= 16384		
2	Conv2d	NF*(2*NF)*K*K+NF	= 18464	NF*H/4*H/4	= 8192	(NF*(2*NF)*K*K+NF)*H/4*W/4	= 4726784
	Upsample			NF*H/2*W/2	= 32768	NF*H/2*W/2	= 32768
	BatchNorm2d	NF+NF	= 64	NF*H/2*W/2	= 32768	(H/2*W/2)*(H/2*W/2)	= 1048576
	ReLU			NF*H/2*W/2	= 32768		
3	Conv2d	NC*NF*K*K	= 6912	NC*H/2*W/2	= 24576	(NC*NF*K*K)*H/2*W/2	= 7077888
	Upsample			NC*H*W	= 98304	NC*H*W	= 98304
	BatchNorm2d	NC+NC	= 48	NC*H*W	= 98304	(H*W)*(H*W)	= 16777216
	ReLU			NC*H*W	= 98304		
4	Conv2d	NC*NC*K*K+NC	= 5208	NC*H*W	= 98304	(NC*NC*K*K+NC)*H*W	= 21331968
	Total		50280		868352		75014144

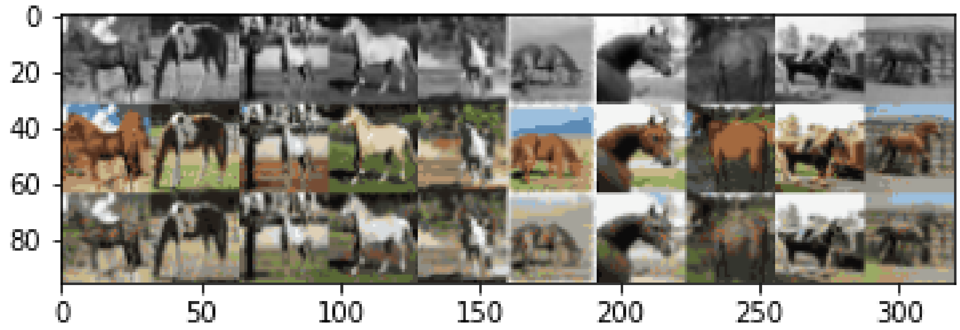
Part B: Strided and Transposed Convolutions

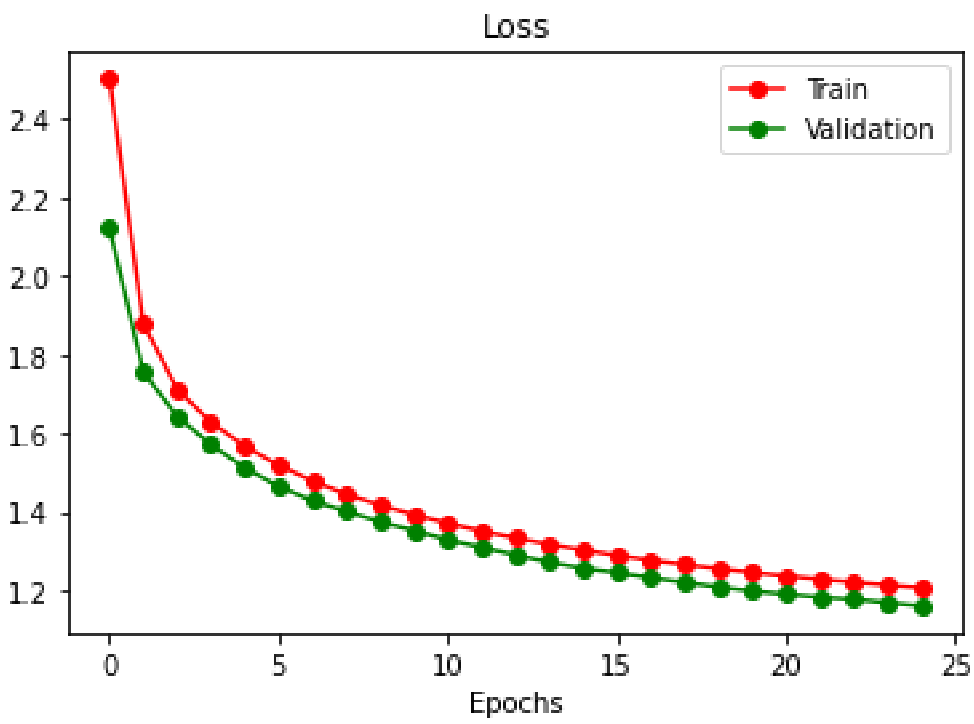
1 Complete the model `ConvTransposeNet`

```
#V202102171234
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 ###############
        # Input: [BS, NIC, 32, 32]
        self.block0 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels,
                      out_channels=num_filters,
                      kernel_size=kernel,
                      padding=padding,
                      stride=stride),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        # Output: [BS, NF, 16, 16]
        self.block1 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters,
                      out_channels=2*num_filters,
                      kernel_size=kernel,
                      padding=padding,
                      stride=stride),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        # Output: [BS, 2NF, 8, 8]
        self.block2 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=2*num_filters,
                      out_channels=num_filters,
                      kernel_size=kernel,
                      padding=padding,
                      output_padding=output_padding,
                      stride=stride),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        # Output: [BS, NF, 16, 16]
        self.block3 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=num_filters,
                      out_channels=num_colours,
                      kernel_size=kernel,
                      padding=padding,
                      output_padding=output_padding,
                      stride=stride),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        # Output: [BS, NC, 32, 32]
        self.block4 = nn.Sequential(
            nn.Conv2d(in_channels=num_colours,
                      out_channels=num_colours,
                      kernel_size=kernel,
                      padding=padding)
        )
        # Output: [BS, NC, 32, 32]
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        for block in (self.block0, self.block1, self.block2,
                      self.block3, self.block4):
            x = block.forward(x)
        return x
        ###################################################
```

Train the model

```
...
Epoch [25/25], Loss: 1.2070, Time (s): 104
Epoch [25/25], Val Loss: 1.1603, Val Acc: 54.7%, Time(s): 105.78
```





3 How do the result compare to Part A?

The result qualitatively appear similar as before. The current `ConvTransposeNet` model seems to be worse at colouring brown horses.

The current model resulted in *lower* validation loss (1.2) compared to the previous model (1.6). The `ConvTranspose2d` layers produce lower loss than `Upsample` because the transposed convolutions fits additional weights to "reverse" the previous skipping done by the convolution, resulting in a better numeric fitting and lower loss.

4 padding parameter

	`nn.Conv2d`	`nn.ConvTranspose2d`
if kernel size = 4	`padding=1`	`padding=1` `output_padding=0`
if kernel size = 5	`padding=2`	`padding=2` `output_padding=1`

5 Describe the effect of batch sizes on the training/validation loss, and the final image output quality.

Smaller batch sizes were associated with lower training/validation loss and the output quality was better with smaller batch sizes.

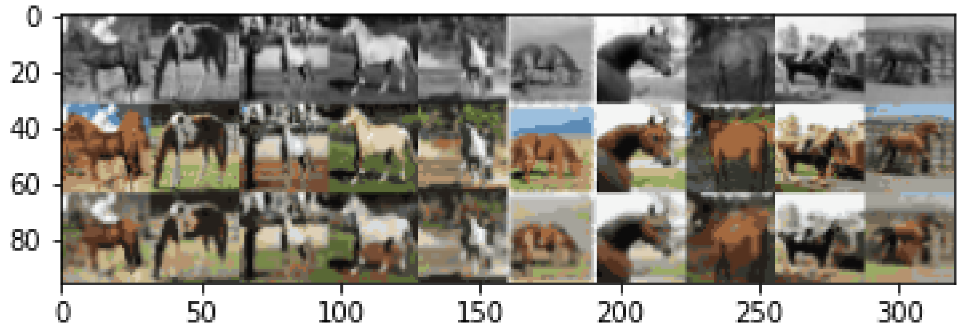
Part C: Skip Connections

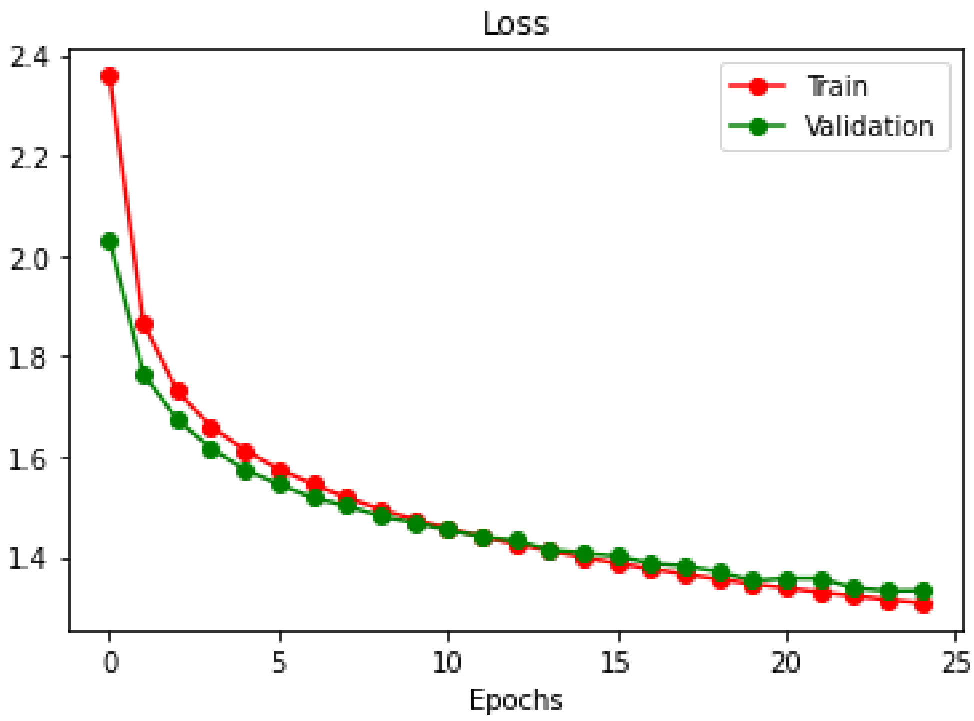
1 Add a skip connection...

```
#V202102200047
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 ###############
        # Input: [BS, NIC, 32, 32]
        self.block0 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels,
                      out_channels=num_filters,
                      kernel_size=kernel,
                      padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        # Output: [BS, NF, 16, 16]
        self.block1 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters,
                      out_channels=2*num_filters,
                      kernel_size=kernel,
                      padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        # Output: [BS, 2NF, 8, 8]
        self.block2 = nn.Sequential(
            nn.Conv2d(in_channels=2*num_filters,
                      out_channels=num_filters,
                      kernel_size=kernel,
                      padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        # Concatenate
        # Output: [BS, NF+NF, 16, 16]
        self.block3 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters+num_filters,
                      out_channels=num_colours,
                      kernel_size=kernel,
                      padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        # Concatenate
        # Output: [BS, NC+NIC, 32, 32]
        self.block4 = nn.Sequential(
            nn.Conv2d(in_channels=num_colours+num_in_channels,
                      out_channels=num_colours,
                      kernel_size=kernel,
                      padding=padding)
        )
        # Output: [BS, NC, 32, 32]
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        x_input = x
        x = self.block0(x)
        x_1 = x
        x = self.block1(x)
        x = self.block2(x)
        x = torch.cat((x_1, x), dim=1)
        x = self.block3(x)
        x = torch.cat((x_input, x), dim=1)
        x = self.block4(x)
        return x
        ###################################################
```

2 Train the model

```
...
Epoch [25/25], Loss: 1.3075, Time (s): 33
Epoch [25/25], Val Loss: 1.3323, Val Acc: 48.2%, Time(s): 34.52
```





3 How does the result compare to the previous mode

The skip connections did not improve validation loss as compared to the previous model, but improved the output qualitatively. The brown horses were coloured closer to the actual brown colour. The improvements come from:

1. The skip connections allows high resolution, low level features that did not need convolution operation or only one convolution to be directly passed down to the final layers.
2. More channels in the final two layers increased the complexity and parameters of the model.

Part D.1. Fine tune Semantic Segmentation Model with Cross Entropy Loss

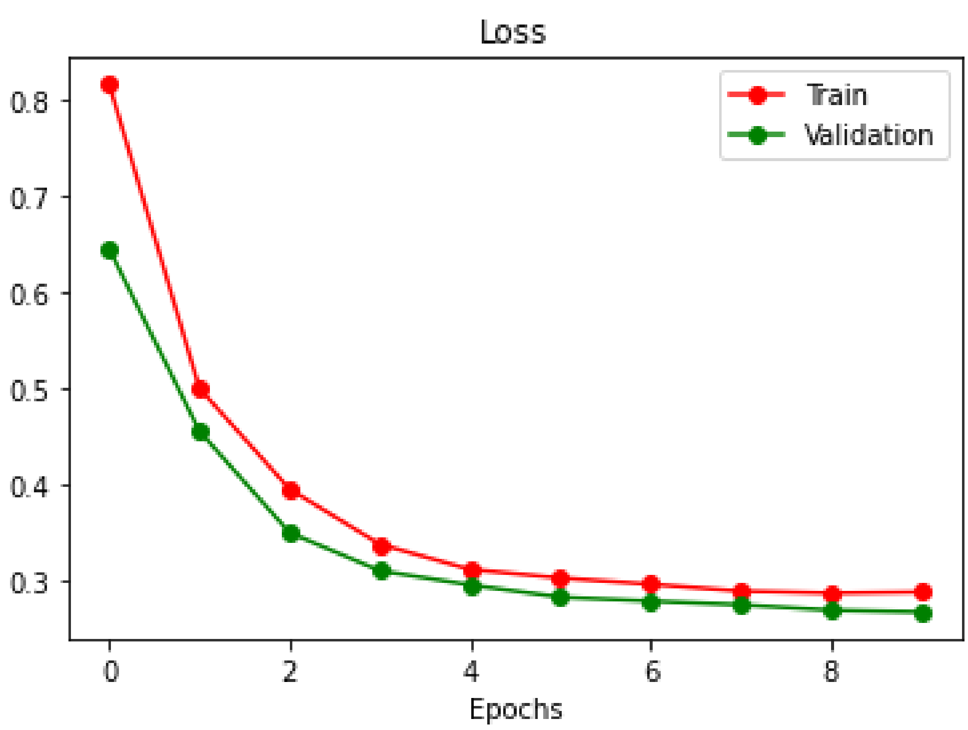
1 Complete the train function

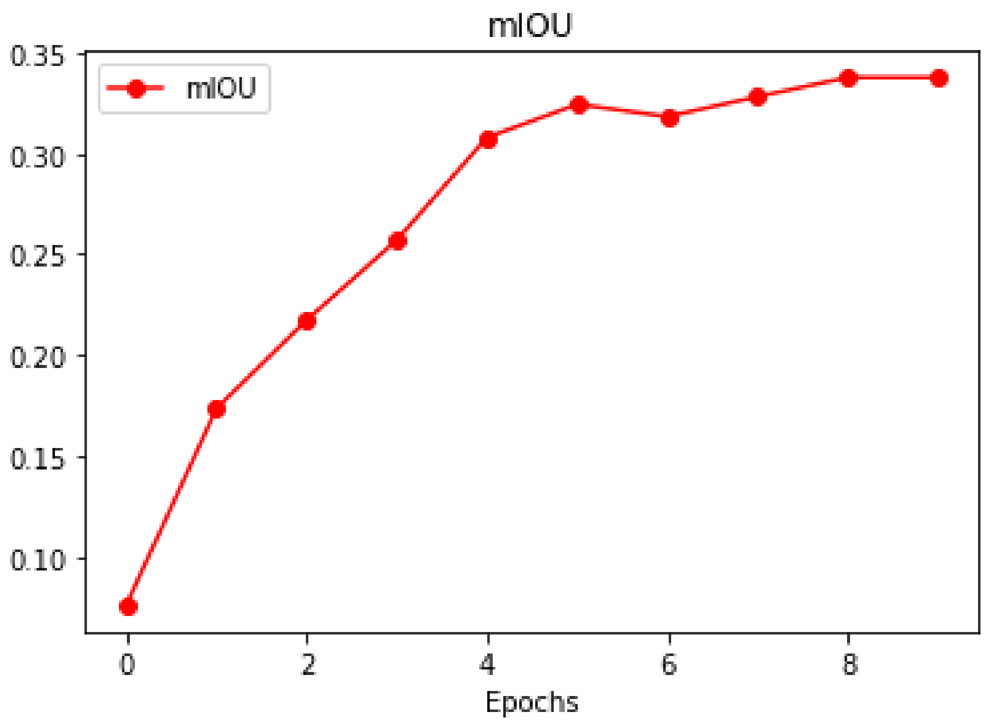
```
    # We only learn the last layer and freeze all the other weights
    ################ Code goes here ######################
    # Around 2-3 lines of code
    for name, param in model.named_parameters():
        if name.startswith("classifier.4"):
            print("Appending " + name + " to learned_parameters")
            learned_parameters.append(param)
    #####################################################
```

2 Complete the script

```
# 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 ######################
# Around 2 lines of code
for param in model.parameters():
    param.requires_grad = False
model.classifier[-1] = nn.Conv2d(256, 2, kernel_size=(1, 1), stride=(1, 1))
#####################################################
```

```
...
Epoch [10/10], Loss: 0.2869, Time (s): 44
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
    [0..255] for integers).
Epoch [10/10], Loss: 0.2661, mIOU: 0.3377, Validation time (s): 12
Saving model...
Best model achieves mIOU: 0.3377
```

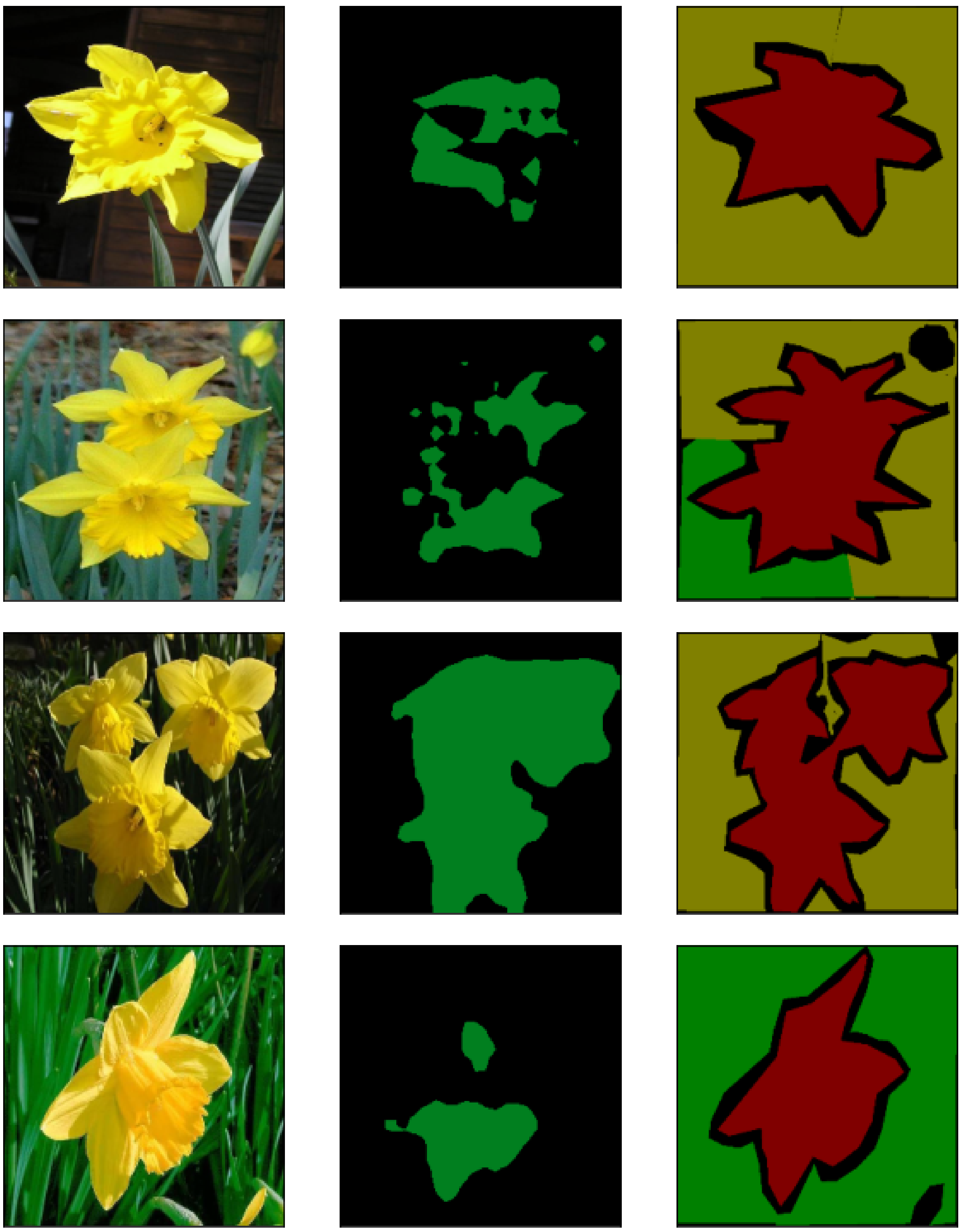




3 Visualize Predictions

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


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



Part D.2. Finetune Semantic Segmentation Model with IoU Loss

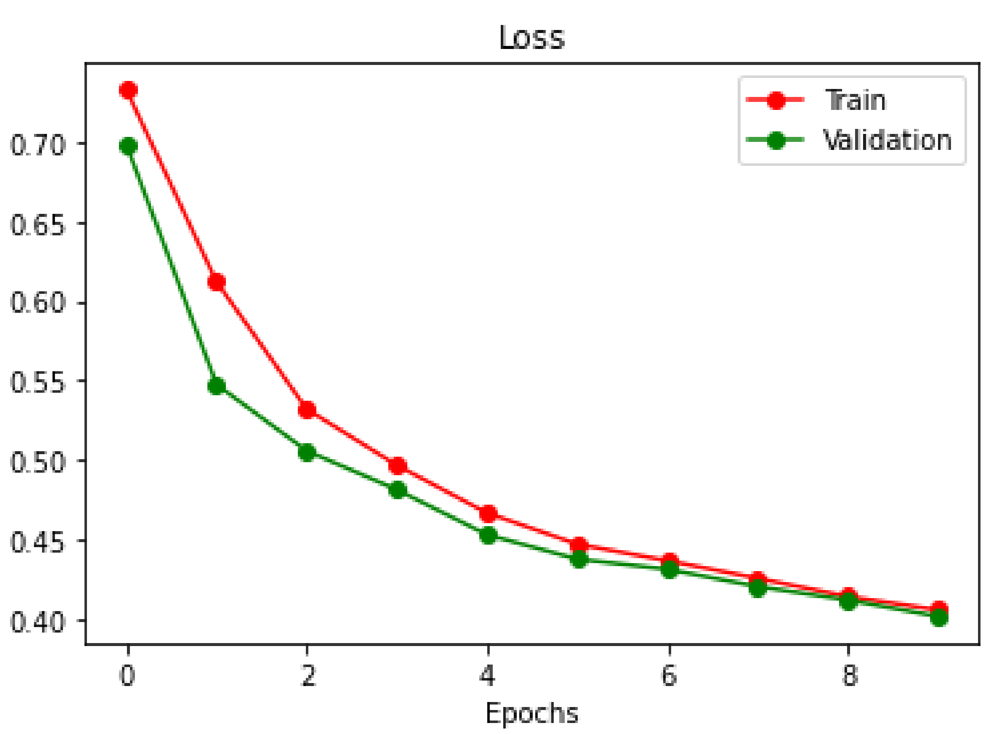
1 Change the loss function from cross entropy used in part D.1 to the (soft) IoU loss

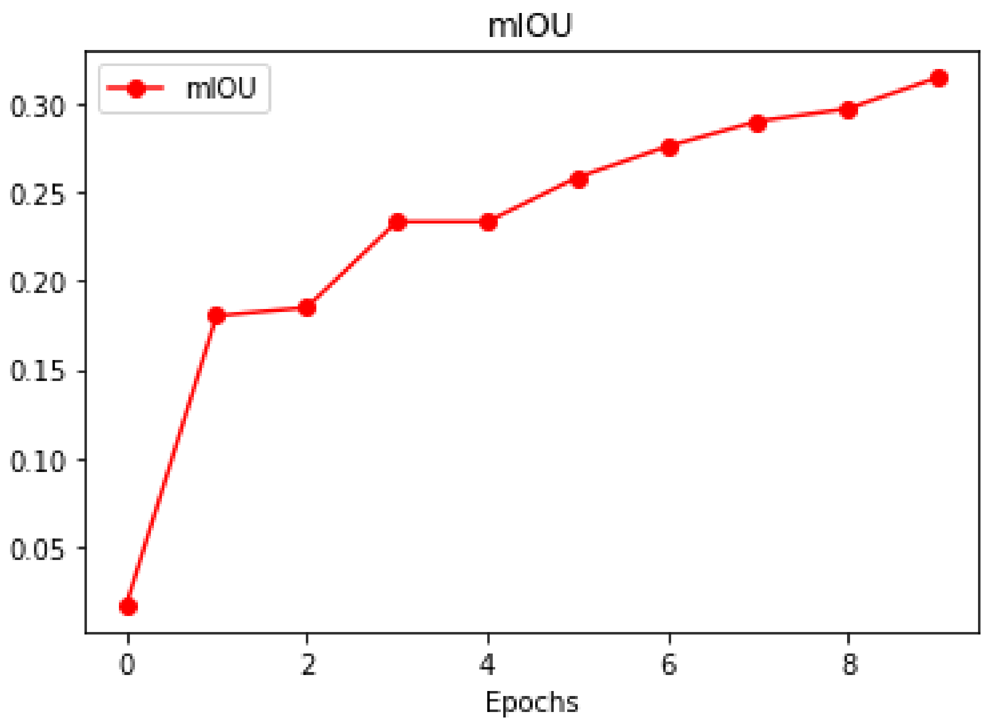
```python
# def compute_IoU_loss(pred, gt):
def compute_iou_loss(pred, gt):
    # Compute the IoU between the pred and the gt (ground truth)
    ################ YOUR CODE GOES HERE ######################
    # Around 2-3 lines of code

    # - apply softmax on pred along the channel dimension (dim=1)
    softmaxed_pred = nn.functional.softmax(pred, dim=1)

    # - only have to compute IoU between gt and the foreground channel of pred
    # - no need to consider IoU for the background channel of pred
    # - extract foreground from the softmaxed pred (e.g., softmaxed_pred[:, 1, :, :])
    softmaxed_pred_fg = softmaxed_pred[:, 1, :, :]

    # - compute intersection between foreground and gt
    intersection = (softmaxed_pred_fg * gt).sum()

    # - compute union between foreground and gt
    union = (softmaxed_pred_fg + gt - softmaxed_pred_fg * gt).sum()

    # - compute loss using the computed intersection and union
    loss = 1.0 - intersection / union

    #####################################################
    return loss

...

# 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 ######################
# Around 2 lines of code
for param in model.parameters():
    param.requires_grad = False
model.classifier[-1] = nn.Conv2d(256, 2, kernel_size=(1, 1), stride=(1, 1))
#####################################################
```

What is the validation mIoU (mean IoU)? How does this compare with the mIoU when training with the cross entropy?

Best model achieves mIOU: 0.3142. Previously with cross entropy the best model mIOU was 0.3377. The current method achieved a better model than the previous best model. See below:

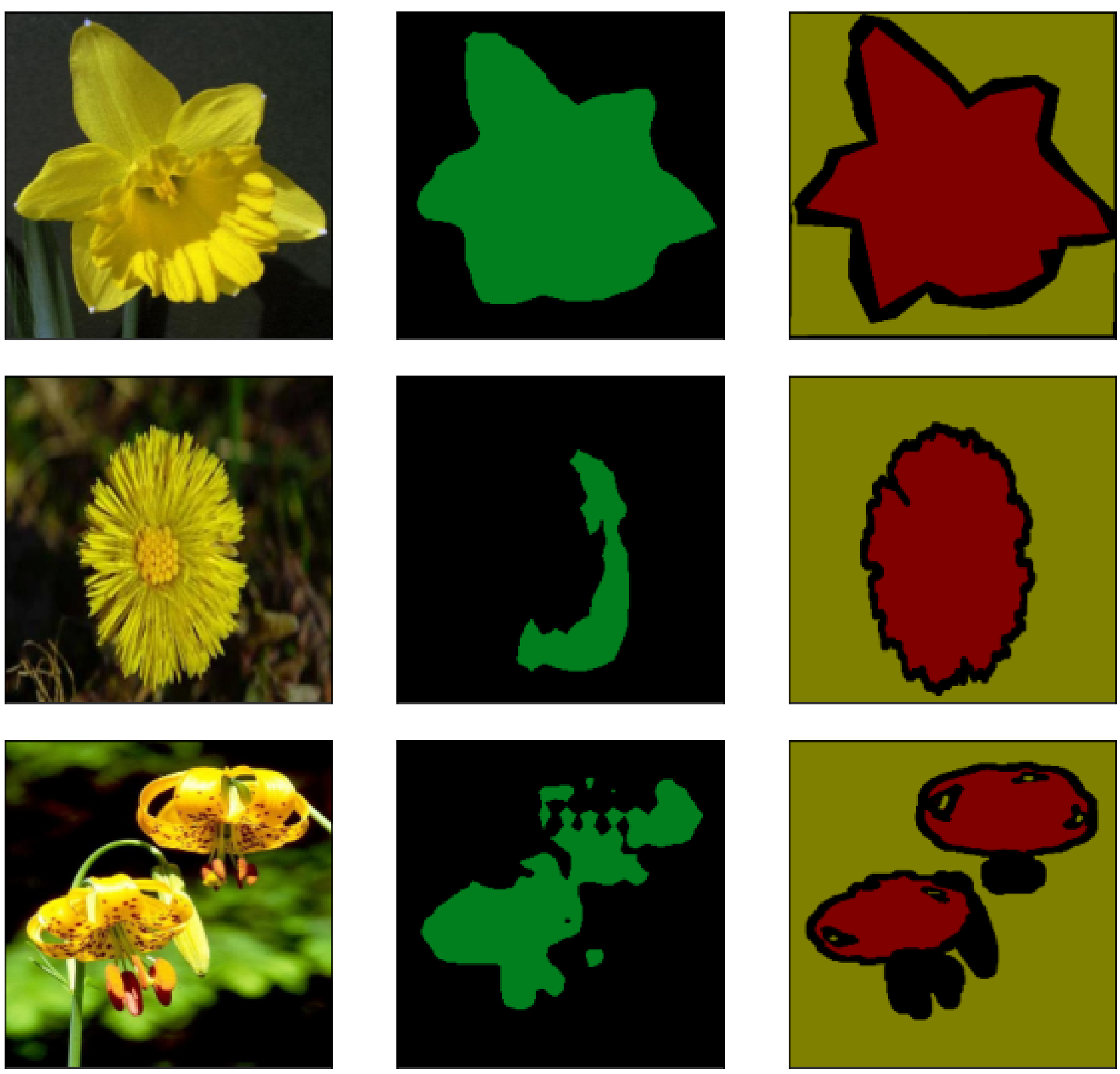
```
...
Epoch [10/10], Loss: 0.4062, Time (s): 46
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
    [0..255] for integers).
Epoch [10/10], Loss: 0.4022, mIOU: 0.3142, Validation time (s): 12
Saving model...
Best model achieves mIOU: 0.3142
```





2 Visualize the predictions

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



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