Programming Assignment 2: Convolutional Neural Networks

Bowen Xu 1006411786

Note: some results in this writeup file does not match with the output in ipynb file since some codes are re-ran after the update. Thank you for your understanding☺

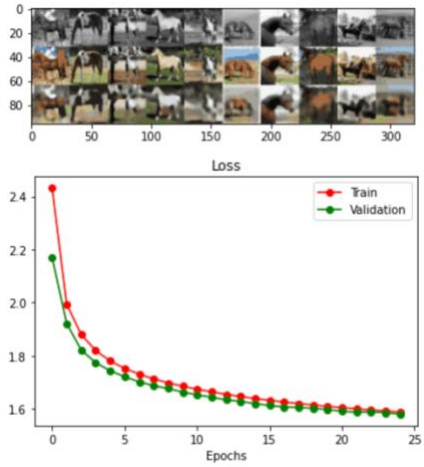
Image Colourization as Classification

Part A: Pooling and Upsampling

Q1.

```
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.firstlayer = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_filters),
            nn.ReLU())

        self.secondlayer = nn.Sequential(
            nn.Conv2d(num_filters, num_filters*2, kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_filters*2),
            nn.ReLU())

        self.thirdlayer = nn.Sequential(
            nn.Conv2d(num_filters*2, num_filters, kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_filters),
            nn.ReLU())

        self.fourthlayer = nn.Sequential(
            nn.Conv2d(num_filters, num_colours, kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_colours),
            nn.ReLU())

        self.finallayer = nn.Conv2d(num_colours, num_colours, kernel_size=kernel,padding=padding)
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        self.first = self.firstlayer(x)
        self.second = self.secondlayer(self.first)
        self.third = self.thirdlayer(self.second)
        self.fourth = self.fourthlayer(self.third)
        self.output = self.finallayer(self.fourth)
        return self.output
        ###################################################
```

Q2.



The results don't look good to me since the pictures after colourization is not very realistic and looked very blurred. Some colourization is not accurately applied: some colors applied to the ground instead of the horses. Although the gap between training and validation loss curves seems very small, the validation accuracy (40.9%) is quite low.

Q3.

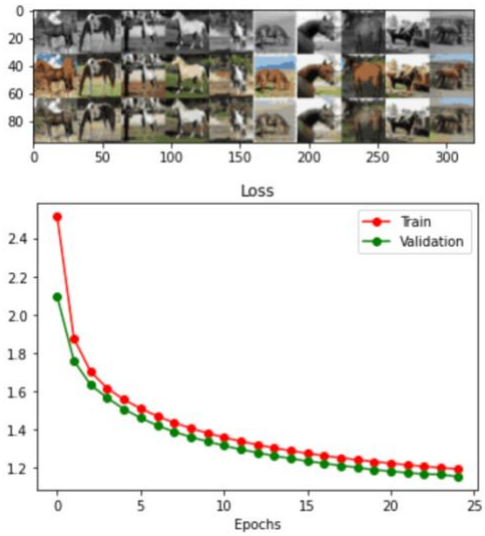
	Weight	**Output**	**Connections**
Conv2d	$3 \times 3 \times NIC \times NF + NF$	$32 \times 32 \times NF$	$32 \times 32 \times 3 \times 3 \times NIC \times NF$
MaxPool2d		$16 \times 16 \times NF$	$16 \times 16 \times 2 \times 2 \times NF$
BatchNorm2d	$2NF$	$16 \times 16 \times NF$	$16 \times 16 \times NF$
ReLU			
Conv2d	$3 \times 3 \times NF \times 2NF + 2NF$	$16 \times 16 \times 2NF$	$16 \times 16 \times 3 \times 3 \times NF \times 2NF$
MaxPool2d		$8 \times 8 \times 2NF$	$8 \times 8 \times 2 \times 2 \times 2NF$
BatchNorm2d	$4NF$	$8 \times 8 \times 2NF$	$8 \times 8 \times 2NF$
ReLU			
Conv2d	$3 \times 3 \times 2NF \times NF + NF$	$8 \times 8 \times NF$	$8 \times 8 \times 3 \times 3 \times 2NF \times NF$
Upsample		$16 \times 16 \times NF$	$16 \times 16 \times 2 \times 2 \times NF$
BatchNorm2d	$2NF$	$16 \times 16 \times NF$	$16 \times 16 \times NF$
ReLU			
Conv2d	$3 \times 3 \times NF \times NC + NC$	$16 \times 16 \times NC$	$16 \times 16 \times 3 \times 3 \times NF \times NC$
Upsample		$32 \times 32 \times NC$	$32 \times 32 \times 2 \times 2 \times NC$
BatchNorm2d	$2NC$	$32 \times 32 \times NC$	$32 \times 32 \times NC$
ReLU			
Conv2d	$3 \times 3 \times NC \times NC + NC$	$32 \times 32 \times NC$	$32 \times 32 \times 3 \times 3 \times NC \times NC$
Total	$9NIC \times NF + 36NF^2 + 9NF \times NC + 9NC^2 + 12NF + 4NC$	$2880NF + 3328NC$	$9216NIC \times NF + 5760NF^2 + 2304NF \times NC + 9216NC^2 + 3200NF + 5120NC$
Double Width/Length	$no\ change$ $9NIC \times NF + 36NF^2 + 9NF \times NC + 9NC^2 + 12NF + 4NC$	$11520NF + 13312NC$	$36864NIC \times NF + 23040NF^2 + 9216NF \times NC + 36864NC^2 + 12800NF + 20480NC$

Part B: Strided and Transposed Convolutions

Q1.

```
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.firstlayer = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding,stride=stride),
            nn.BatchNorm2d(num_filters),
            nn.ReLU())

        self.secondlayer = nn.Sequential(
            nn.Conv2d(num_filters, num_filters*2, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(num_filters*2),
            nn.ReLU())

        self.thirdlayer = nn.Sequential(
            nn.ConvTranspose2d(num_filters*2, num_filters, kernel_size=kernel, padding=padding, output_padding=output_padding, stride=stride),
            nn.BatchNorm2d(num_filters),
            nn.ReLU())

        self.fourthlayer = nn.Sequential(
            nn.ConvTranspose2d(num_filters, num_colours, kernel_size=kernel, padding=padding, output_padding=output_padding, stride=stride),
            nn.BatchNorm2d(num_colours),
            nn.ReLU())

        self.finallayer = nn.Conv2d(num_colours, num_colours, kernel_size=kernel,padding=padding)
        ################################################

    def forward(self, x):
        ############## YOUR CODE GOES HERE ##############
        self.first = self.firstlayer(x)
        self.second = self.secondlayer(self.first)
        self.third = self.thirdlayer(self.second)
        self.fourth = self.fourthlayer(self.third)
        self.output = self.finallayer(self.fourth)
        return self.output
        ################################################
```

Q2. Validation Loss: 1.1384 Validation Accuracy: 55.2%



Q3. ConvTransposeNet has a validation loss and a validation accuracy of 1.1384 and 55.2%, respectively, which are both better than the validation loss and accuracy got from PoolUpsampleNet (1.5937 and 40.9%). The reason behind this could be that transpose is a learnable parameter while upsampling is not learnable since it's a simple interpolation method and does not require any weight updates during the training process. More learnable parameters in transpose layer will although make the training process slower, it can improve the performance of the model.

Q4.

Conv2d

- *kernel_size = 4*

$$first\ layer:\quad H_{out} = 16 = \left\lfloor \frac{H_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \right\rfloor$$
$$= \left\lfloor \frac{32 + 2 \times padding - 1 \times (\mathbf{4} - 1) - 1}{2} + 1 \right\rfloor$$

$$second\ layer:\quad H_{out} = 8 = \left\lfloor \frac{H_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \right\rfloor$$
$$= \left\lfloor \frac{16 + 2 \times padding - 1 \times (\mathbf{4} - 1) - 1}{2} + 1 \right\rfloor$$
$$therefore,\qquad \boldsymbol{padding = 1}$$

- By using similar formula but *kernel_size = 5*: $\boldsymbol{padding = 2}$

ConveTranspose2d

- *kernel_size = 4*

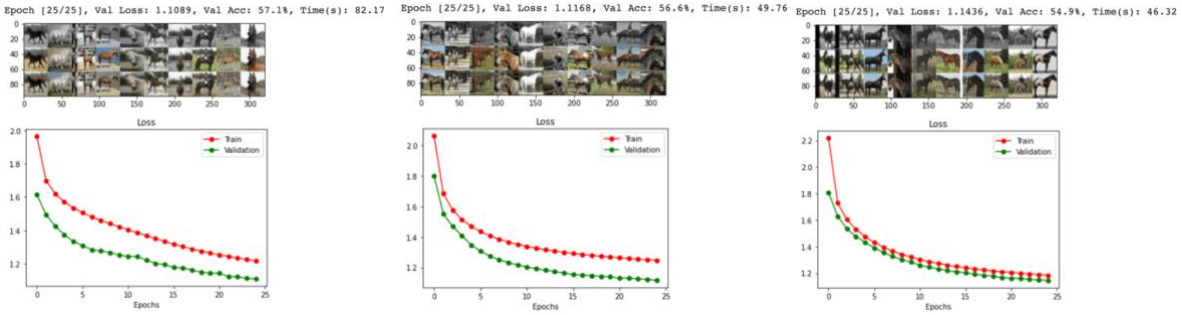
$$first\ layer:\quad H_{out} = 16$$
$$= (H_{in} - 1) \times stride - 2 \times padding + dilation \times (kernel\_size - 1) + output\_padding + 1$$
$$= (8 - 1) \times 2 - 2 \times 2 + 1 \times (\mathbf{4} - 1) + output\_padding + 1$$
$$second\ layer:\quad H_{out} = 32$$
$$= (H_{in} - 1) \times stride - 2 \times padding + dilation \times (kernel\_size - 1) + output\_padding + 1$$
$$= (16 - 1) \times 2 - 2 \times 2 + 1 \times (\mathbf{4} - 1) + output\_padding + 1$$

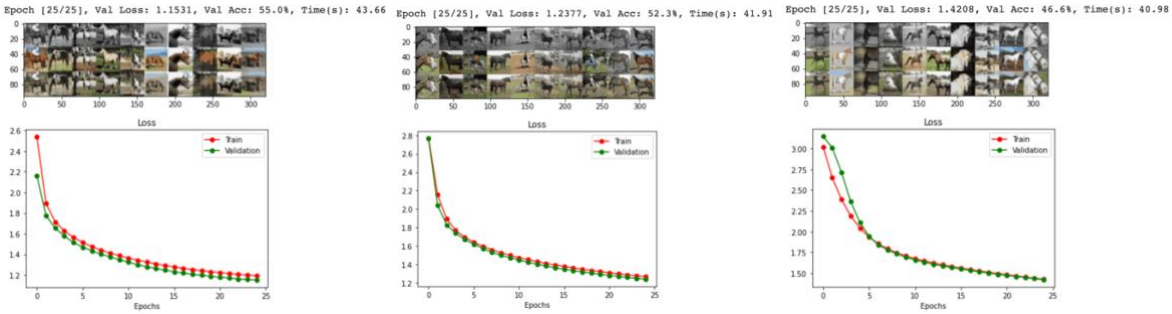
$$therefore,\qquad \boldsymbol{padding = 2, output\_padding = 2}$$

- By using similar formula but *kernel_size = 5*: $\boldsymbol{padding = 2, output\_padding = 1}$

Q5.

batch_size	val_acc	val_loss
10	57.1%	1.1089
20	56.6%	1.1168
50	54.9%	1.1436
100	55.0%	1.1531
200	52.3%	1.2377
500	46.6%	1.4208





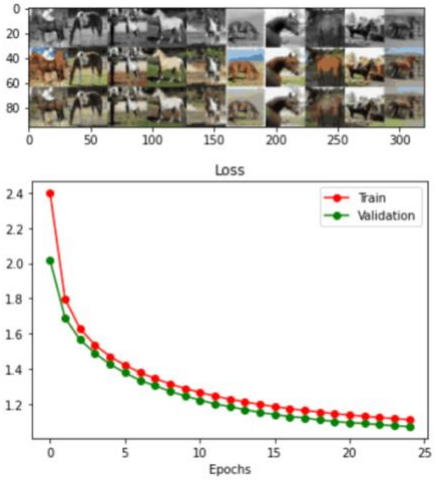
As we can see from the table, the smaller the batch_size is, the lower the validation losses and higher the validation accuracy. By observing the six screenshots of the results and training/validation curves, we can clearly see the gap between two curves are increasing as the batch_size decreases, and the results have better quality since the colourization seems accurately applied and more realistic.

Part C: Skip Connections

Q1.

```
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.firstlayer = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=1,stride=stride),
            nn.BatchNorm2d(num_filters),
            nn.ReLU())

        self.secondlayer = nn.Sequential(
            nn.Conv2d(num_filters, num_filters*2, kernel_size=kernel, padding=1, stride=stride),
            nn.BatchNorm2d(num_filters*2),
            nn.ReLU())

        self.thirdlayer = nn.Sequential(
            nn.ConvTranspose2d(num_filters*2, num_filters, kernel_size=kernel, padding=padding, output_padding=output_padding, stride=stride),
            nn.BatchNorm2d(num_filters),
            nn.ReLU())

        self.fourthlayer = nn.Sequential(
            nn.ConvTranspose2d(num_filters+num_filters, num_colours, kernel_size=kernel, padding=padding, output_padding=output_padding, stride=stride),
            nn.BatchNorm2d(num_colours),
            nn.ReLU())

        self.finallayer = nn.Conv2d(num_colours+num_in_channels, num_colours, kernel_size=kernel,padding=padding)
        ################################################

    def forward(self, x):
        ############## YOUR CODE GOES HERE ##############
        self.first = self.firstlayer(x)
        self.second = self.secondlayer(self.first)
        self.third = self.thirdlayer(self.second)
        self.fourth = self.fourthlayer(torch.cat((self.first, self.third), dim=1))
        self.output = self.finallayer(torch.cat((x, self.fourth), dim=1))
        return self.output
        ################################################
```

Q2. Validation loss = 1.0651 and validation accuracy = 58.1%



Q3. UNet has lower validation loss and higher validation accuracy than the ConvTransposeNet. The quality of the results improves as well since it seems like more colourization are added to the horses instead of the ground and more colorful than the result got from Part B. The two reasons behind this could be:

- Concatenating two layers as the input to the latter layers will increase the number of trainable parameters in the model and therefore make the model have more expressive power to make predictions.
- Some very useful information may exist in the input, however, some of them may get lost during transpose convolutions process in the first couple of layers. The skip connections structure can bring those lost but useful information back into the model and allow the model to learn more from different scales and get a better performance.

Image Segmentation as Classification

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

Q1.

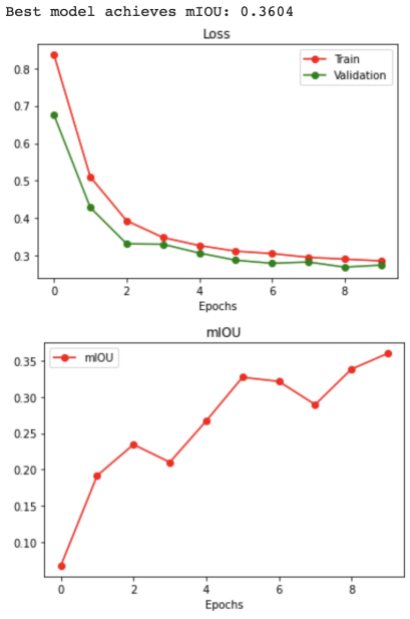
```
def train(args, model):

    # 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 ######################
    # Around 3 lines of code
    # Hint:
    # - use a for loop to loop over all model.named_parameters()
    # - append the parameters (both weights and biases) of the last layer (prefix: classifier.4) to the learned_parameters list
    for name, param in model.named_parameters():
        if name.startswith('classifier.4'):
            learned_parameters.append(param)
    ######################################################


    # Adam only updates learned_parameters
    optimizer = torch.optim.Adam(learned_parameters, lr=args.learn_rate)

    train_loader, valid_loader = initialize_loader(args.train_batch_size, args.val_batch_size)
    print(
        "Train set: {}, Test set: {}".format(
            train_loader.dataset.num_files, valid_loader.dataset.num_files
        )
    )
```

Q2.

```python
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 ######################
# Around 4 lines of code
# Hint:
# - replace the classifier.4 layer with the new Conv2d layer (1 line)
# - no need to consider the aux_classifier module (just treat it as don't care)
# - freeze the gradient of other layers (3 lines)
model.requires_grad_(False)
model.classifier[4] = nn.Conv2d(in_channels=256, out_channels=2, kernel_size=(1,1))
###########################################################
```

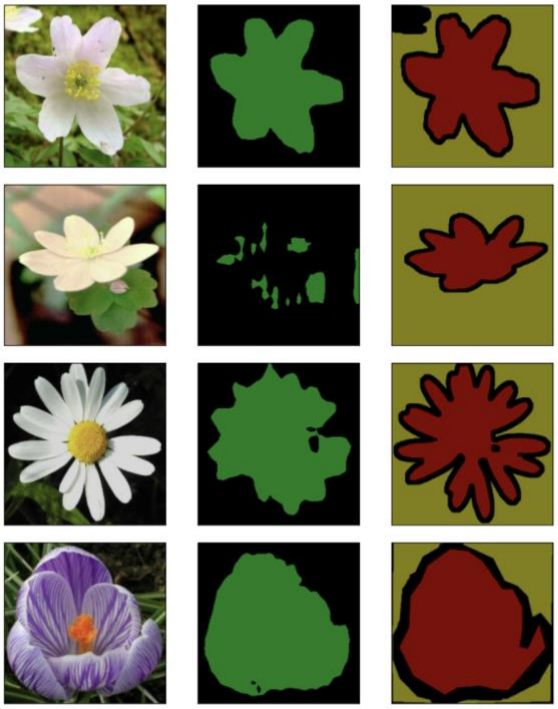
The best model achieves 0.3604 of mIOU.

Q3.



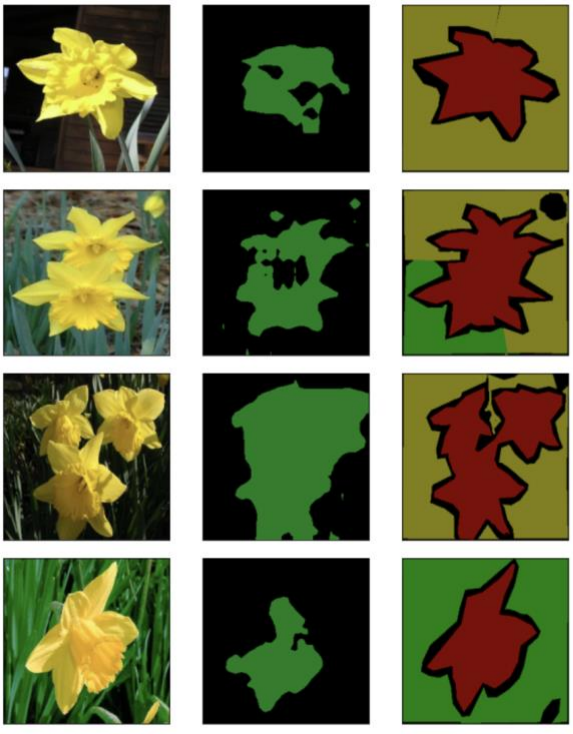
```
plot_prediction(args, model, is_train=True, index_list=[0, 1, 2, 3])
```

```
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).
```



Part D.2 *Fine-tune Semantic Segmentation Model with IoU Loss*

Q1.

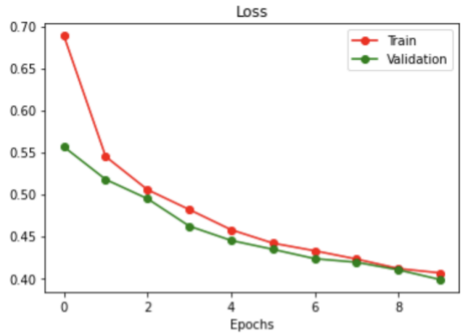
```
def compute_iou_loss(pred, gt, SMOOTH=1e-6):
    # Compute the IoU between the pred and the gt (ground truth)
    ################ YOUR CODE GOES HERE ######################
    # Around 5 lines of code
    # Hint:
    # - apply softmax on pred along the channel dimension (dim=1)
    # - only have 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, :, :])
    # - compute intersection between foreground and gt
    # - compute union between foreground and gt
    # - compute loss using the computed intersection and union
    m = nn.Softmax(dim=1)
    softmaxed_pred = m(pred)
    num = torch.sum(softmaxed_pred[:, 1, :, :] * gt)
    den = torch.sum(softmaxed_pred[:, 1, :, :] + gt - softmaxed_pred[:, 1, :, :] * gt)
    loss = 1 - num/(den+SMOOTH)
    ###########################################################

    return loss
```

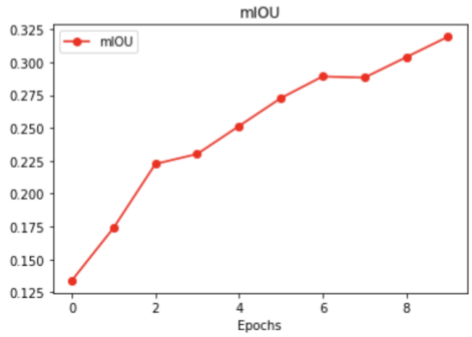
```
args = AttrDict()
# You can play with the hyperparameters here, but to finish the assignment,
# there is no need to tune the hyperparameters here.
args_dict = {
    "gpu": True,
    "checkpoint_name": "finetune-segmentation",
    "learn_rate": 0.05,
    "train_batch_size": 128,
    "val_batch_size": 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 ######################
# Around 4 lines of code
# Hint:
# - replace the classifier.4 layer with the new Conv2d layer (1 line)
# - no need to consider the aux_classifier module (just treat it as don't care)
# - freeze the gradient of other layers (3 lines)
model.requires_grad_(False)
model.classifier[4] = nn.Conv2d(in_channels=256, out_channels=2, kernel_size=(1,1))
###########################################################


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

The best model achieves 0.3193 of mIOU which is lower than the model trained with cross entropy loss

Q2.

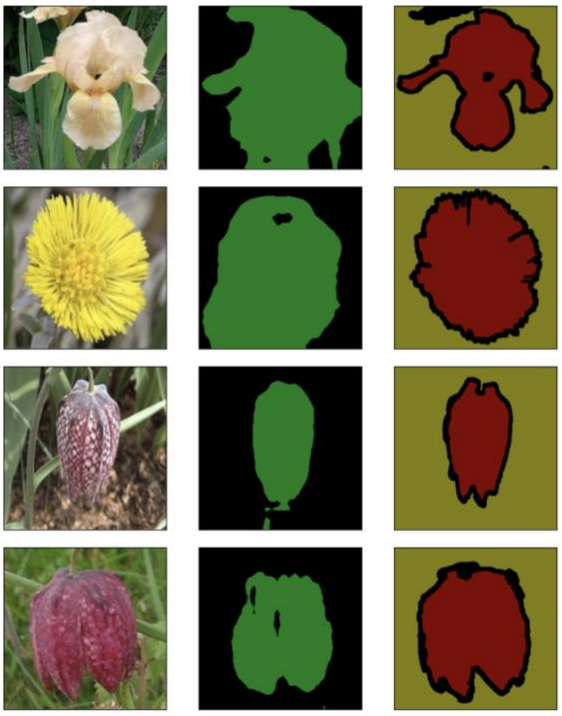
```
Best model achieves mIOU: 0.3193
```






```
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).
```