CSC413 Programming Assignment 2: Convolutional Neural Networks

Tianyu Du (1003801647)

Saturday 22nd February, 2020

1 Part A: Colourization as Classification

Question 1 See `ipynb` file for implementation.

Question 2 The result was terrible, the model merely add some brown pixels to the dark portion of the input image. Backgrounds such as sky and grassland are not properly identified and coloured.

Question 3 The table below summarizes configurations for all layers. For the ℓ^{th} layer. Assumes that all convolution layers have the same kernel size k.
The prefix of `B1.MyConv2D-NF` denotes the convolution layer in the first block of the entire network. There are

$$k^2\mathrm{NC}^2 + k^2\mathrm{NCNF} + 8k^2\mathrm{NF}^2 + k^2\mathrm{NF} + 2\mathrm{NC} + 12\mathrm{NF} \tag{1.1}$$

weights in total, and

$$1024k^2\mathrm{NC}^2 + 256k^2\mathrm{NCNF} + 896k^2\mathrm{NF}^2 + 1024k^2\mathrm{NF} \tag{1.2}$$

connections in convolution layers.
When the image size is doubled, the total number of weights is unchanged. The weight and height of each output (i.e., the first two dimensions of the output) are doubled. The total number of connections is 400% of the network taking 32×32 images.

Table 1: Summary of ConvNet

Layer	#Weights	Output Shape	#Connections
B1.MyConv2D-NF	k^2×NF	32×32×NF	32×32×NF×k^2
B1.MaxPool	0	16×16×NF	
B1.BatchNorm-NF	2×NF	16×16×NF	
B1.ReLu	0	16×16×NF	
B2.MyConv2D-2NF	k^2×NF ×2NF	16×16×2×NF	16^2×NF ×2NF×k^2
B2.MaxPool	0	8×8×2×NF	
B2.BatchNorm-2NF	4×NF	8×8×2×NF	
B2.ReLu	0	8×8×2×NF	
B3.MyConv2D-2NF	k^2×2NF ×2NF	8×8×2×NF	8^2×2NF ×2NF×k^2
B3.BatchNorm-2NF	4×NF	8×8×2×NF	
B3.ReLU	0	8×8×2×NF	
B4.MyConv2D-NF	k^2 ×2NF×NF	8×8×NF	8^2 ×2NF ×NF ×k^2
B4.Upsample	0	16×16×NF	
B4.BatchNorm-NF	2×NF	16×16×NF	
B4.ReLU	0	16×16×NF	
B5.MyConv2D-NC	k^2×NF×NC	16×16×NC	16^2×NF×NC×k^2
B5.Upsample	0	32×32×NC	
B5.BatchNorm-NC	2×NC	32×32×NC	
B5.ReLu	0	32×32×NC	
Out.MyConv2D-NC	k^2 ×NC ×NC	32×32×NC	32^2×NC^2 ×k^2

Question 4 The colourization should be exactly the same. The proposed affine preprocessing is equivalently adding one more linear layer before the first convolution layer. Since convolution operator is linear, the composite of the additional linear layer and the convolution layer is linear as well. The optimizer should be able to adjust weights in the convolution layer to offset the linear layer. In particular, each weight $\mathbf{w}$ will be adjusted to $\frac{\mathbf{w}-b}{a}$.

2 Part B: Skip Connections

Question 1 & 2 See `ipynb` file for implementation.

Question 3

(i) The coloured images are now more reasonable, the model with skip connections can now colour part of sky and grassland correctly.

(ii) Skip connections successfully help improve performances. The validation loss reduces from 1.83 to 1.35 (-26%), and the validation accuracy increases from 33% to 49%.

(iii) The quality of coloured images is improved significantly compared with outcome of the previous CNN without skip connections.

(iv) The model with skip connections has more trainable parameters than the previous CNN, it is more likely for it to pick up more complicated patterns.

(v) In the CNN without skip connections, the input to the very last convolution layer is an abstract representation of the original image. It could be that the representation is too abstract and the last convolution layer fails to infer colouring information from such an abstract representation. Adding skip connections helps the last convolution layer recall the previous information such as the raw image and activations of previous layers. In this case, the CNN with skip connections outperforms the vanilla CNN.

Question 4 The training/validation loss and final images are included below. And the table below summarizes the validation loss and accuracy after 25 epochs. In most coloured pictures from models trained with small batch sizes (50 and 100), the colouring is finer and more accurate especially at edges of objects. Evaluating metrics suggest a smaller batch size helps improve both models' losses and accuracy on the validation set.

Table 2: Model Performances with Different Batch Sizes

Batch Size	Validation Loss	Validation Accuracy
50	1.32	50%
100	1.36	49%
500	1.52	43%
1,000	1.63	41%

Figure 1: Final Image Outputs with Batch Size = 50, 100, 500, 1000 (from top to bottom)

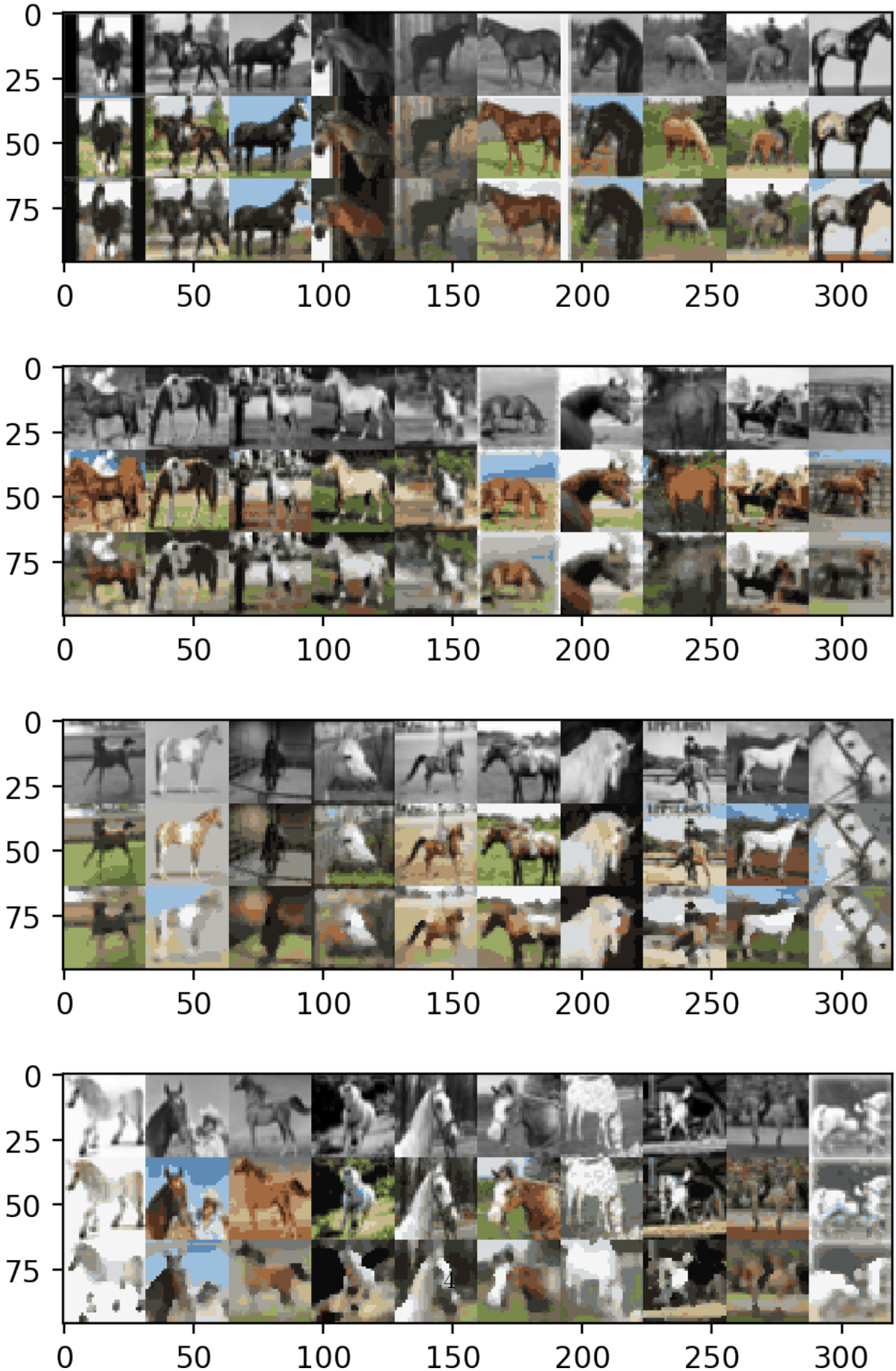


Figure 2: Batchsize=50(left) and 100(right)

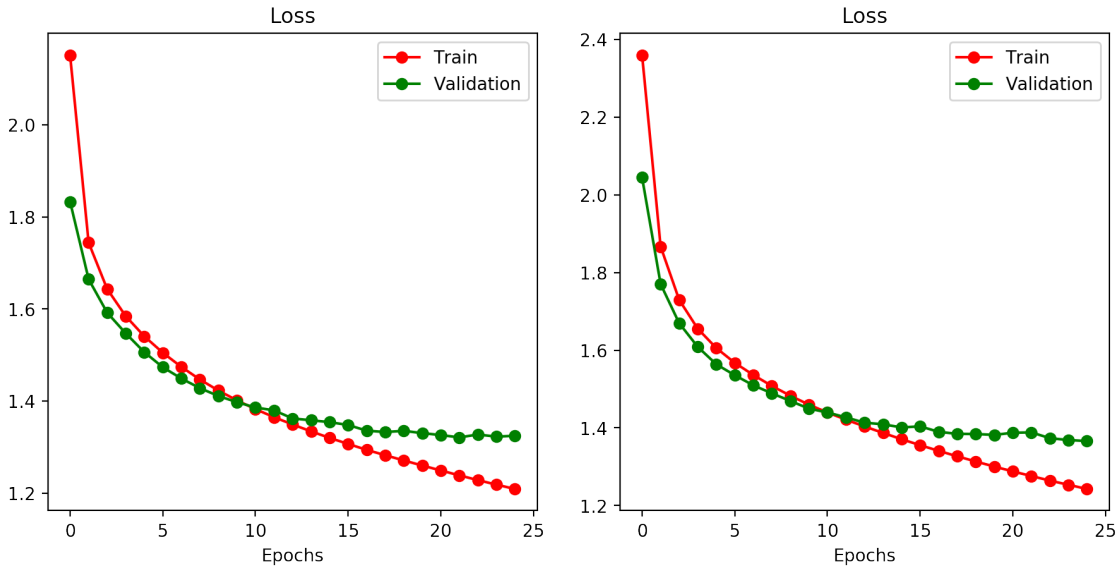
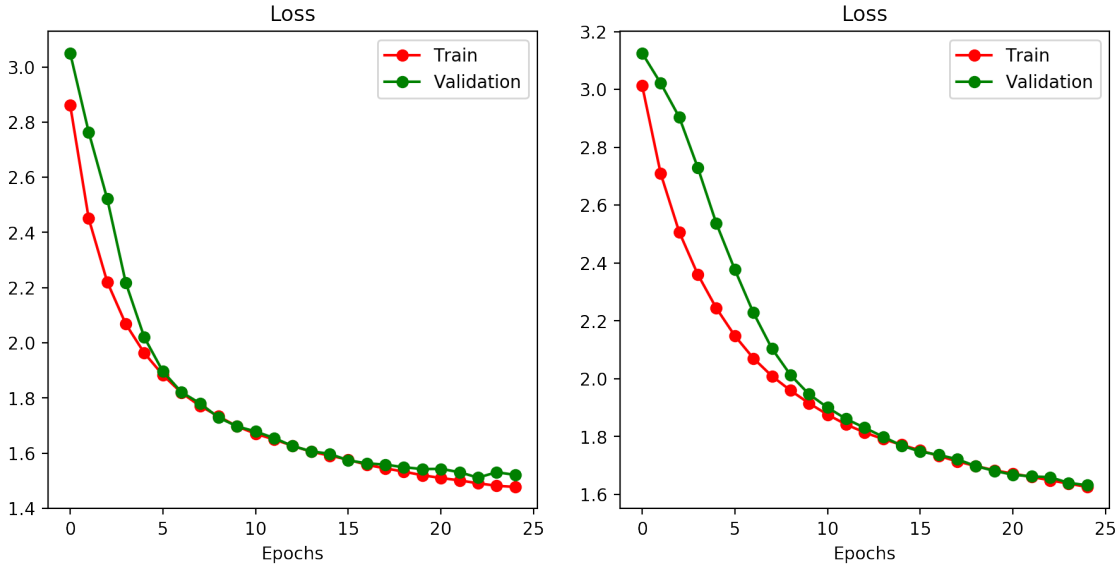


Figure 3: Batchsize=500(left) and 1,000(right)



3 Fine-tune Semantic Segmentation Model

Question 1 Implementation:

```
for name, param in model.named_parameters():
    if name.startswith("classifier.4"):
        learned_parameters.append(param)

```

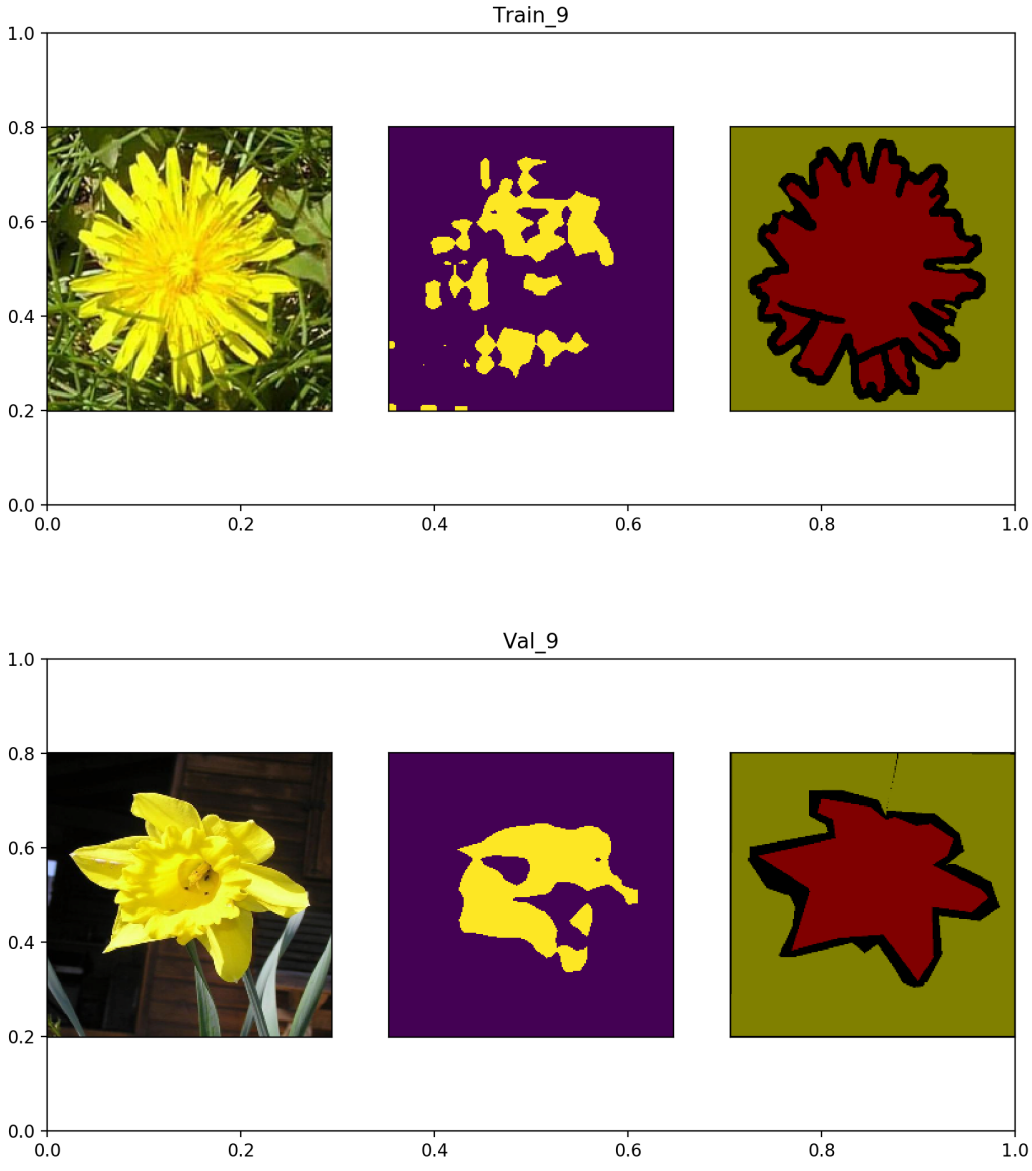
Question 2 Implementation:

```
model.requires_grad_(False)
model.classifier[4] = nn.Conv2d(
    256, 21, kernel_size=(1, 1), stride=(1, 1)
)

```

Question 3 The visualized predictions:

Figure 4: Prediction on Training (top) and Validation (bottom) Set



Question 4 The memory required of tuning the entire model involves storing all trainable parameters and the gradient with respect to each trainable parameter. The overall memory complexity of tuning the entire model is in $\mathcal{O}(n)$, and tuning only one layer is in $\mathcal{O}(1)$.
The computational complexity of backward propagation is approximately 200% of the forward passing. Since the number of connections between two consecutive layers is constant, the computational complexity of training n layers is in $\mathcal{O}(n)$.
Hence, tuning the entire model has computational complexity of $\mathcal{O}(n)$ while tuning only the last layer costs $\mathcal{O}(1)$.

Question 5 The number of trainable parameters in each layer within this model only depends on kernel sizes, number of input channels and number of output channels. These numbers are unchanged when the size of input image changes. Therefore, the total number of parameters and memory needed to store the model is unchanged. However, the new dataset with larger images now requires 400% memory, compared with the original dataset, to be stored.