

1. Print the network architecture of your VGG16-FCN32s model.

```
(features): Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace)
  (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace)
  (9): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace)
  (16): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace)
  (23): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace)
  (30): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
)
(classifier): Sequential(
  (0): Conv2d(512, 4096, kernel_size=(2, 2), stride=(1, 1))
  (1): ReLU(inplace)
  (2): Dropout2d(p=0.5)
  (3): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
  (4): ReLU(inplace)
  (5): Dropout2d(p=0.5)
  (6): Conv2d(4096, 7, kernel_size=(1, 1), stride=(1, 1))
  (7): ConvTranspose2d(7, 7, kernel_size=(64, 64), stride=(32, 32), bias=False)
)
```

Figure 1. VGG16-FCN32s architecture with parameters

The visualization of the above architecture is shown in Fig 2.

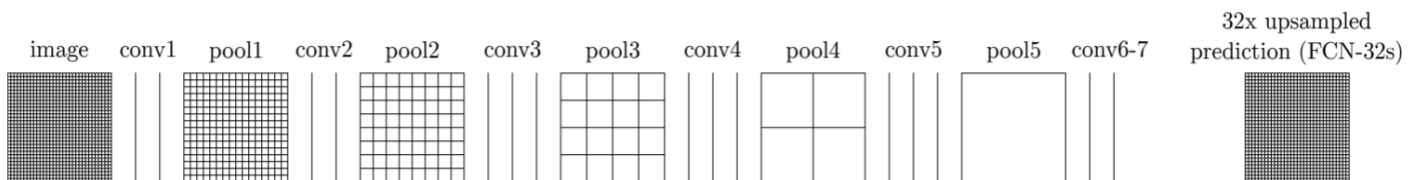




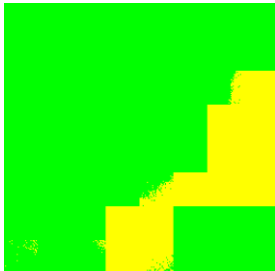


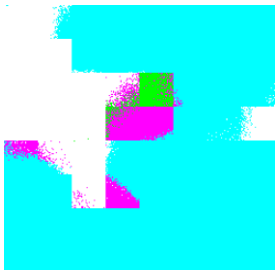



Figure 2. FCN32s plot from the original paper<sup>1</sup>

<sup>1</sup> [https://people.eecs.berkeley.edu/~jonlong/long\\_shelhamer\\_fcn.pdf](https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf)

2. Show the predicted segmentation mask of validation/0008\_sat.jpg, validation/0097\_sat.jpg, validation/0107\_sat.jpg during the early, middle, and the final stage during the training stage. (For example, results of 1st, 10th, 20th epoch)

Table 2. Predicted segmentation on validation set using FCN32s

<b>FCN32s</b> on Validation	<b>Epoch 1</b>	<b>Epoch 10</b>	<b>Epoch 20</b>
0008_sat.jpg			
0097_sat.jpg			
0107_sat.jpg			

3. Implement an improved model which performs better than your baseline model. Print the network architecture of this model.

Here I implement FCN8s as the improved model. The model architecture is shown below:

```

(features): Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace)
  (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace)
  (9): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace)
  (16): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace)
  (23): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace)
  (30): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
)
(classifier): Sequential(
  (0): Conv2d(512, 4096, kernel_size=(2, 2), stride=(1, 1))
  (1): ReLU(inplace)
  (2): Dropout2d(p=0.5)
  (3): Conv2d(4096, 4096, kernel_size=(1, 1), stride=(1, 1))
  (4): ReLU(inplace)
  (5): Dropout2d(p=0.5)
  (6): Conv2d(4096, 7, kernel_size=(1, 1), stride=(1, 1))
  (7): ConvTranspose2d(7, 256, kernel_size=(8, 8), stride=(4, 4), bias=False)
)
(to_pool3): Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace)
  (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace)
  (9): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace)
  (16): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
)
(to_pool4): Sequential(
  (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace)
  (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace)
  (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): ReLU(inplace)
  (6): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
)
(to_pool5): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace)
  (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace)
  (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): ReLU(inplace)
  (6): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
)
(pool4_upsample): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2), bias=False)
(upsample8): ConvTranspose2d(256, 7, kernel_size=(8, 8), stride=(8, 8), bias=False)

```

Figure 3. VGG16-FCN8s architecture with parameters

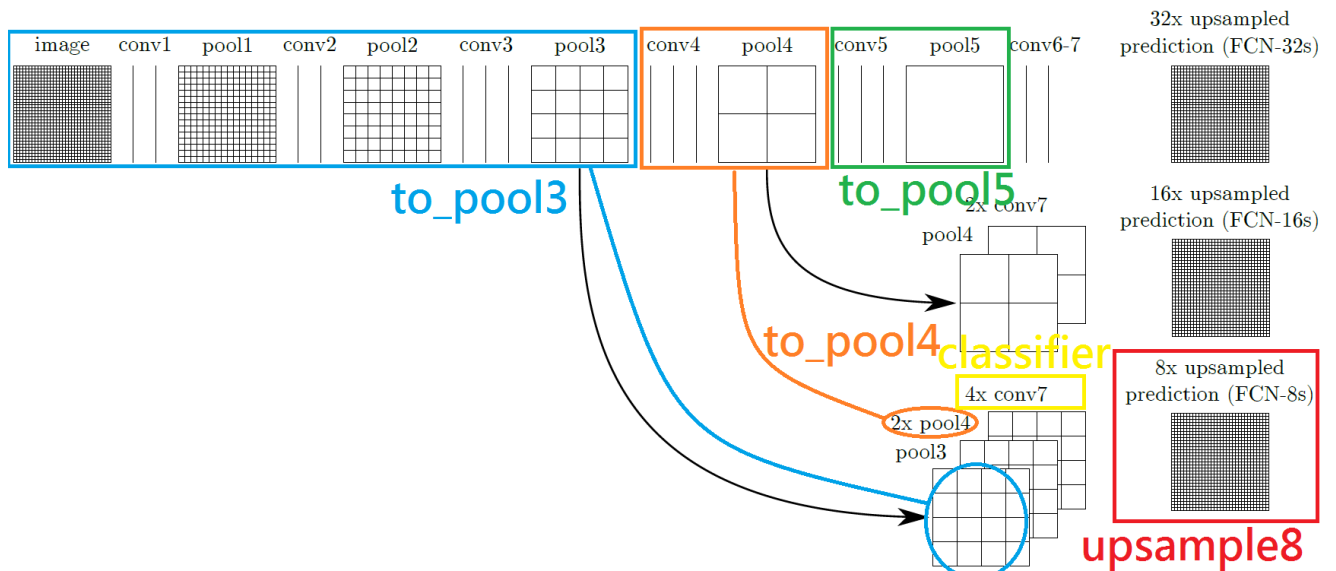


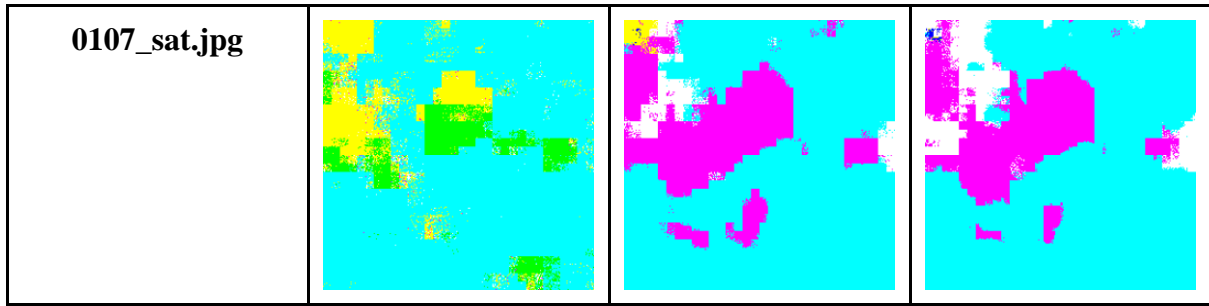
Figure 4. Corresponding layers in the original paper<sup>2</sup>

4. Show the predicted segmentation mask of validation/0008\_sat.jpg, validation/0097\_sat.jpg, validation/0107\_sat.jpg during the early, middle, and the final stage during the training process of this improved model.

Table 2. Predicted segmentation on validation set using FCN8s

<b>FCN8s on Validation</b>	<b>Epoch 1</b>	<b>Epoch 10</b>	<b>Epoch 20</b>
<b>0008_sat.jpg</b>			
<b>0097_sat.jpg</b>			

<sup>2</sup> [https://people.eecs.berkeley.edu/~jonlong/long\\_shelhamer\\_fcn.pdf](https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf)



- Report mIoU score of both models on the validation set. Discuss the reason why the improved model performs better than the baseline one. You may conduct some experiments and show some evidences to support your discussion.

Table 3. mIoU score of both models

	FCN32s	FCN8s
mIoU	0.670	0.699

#### mIoU on Validation set

FCN32s, FCN16s & FCN8s

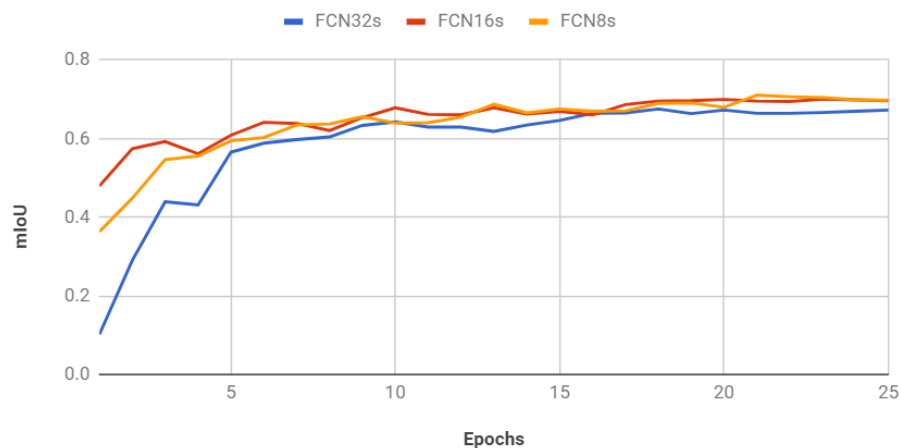


Figure 5. Validation mIoU during training procedure

From the above plot, we can find out that FCN16s and FCN8s perform better than FCN32s. Since both of the improved model combine predictions from the final layer and the former pooling layer (*pool4* and *pool3*, respectively), the model could predict finer details and retaining high-level semantic information. The previous sections have shown the output masks of segmentation by FCN32s and FCN8s. We can find out that the FCN8s indeed improves segmentation detail by fusing information from former layers. In addition, FCN8s uses smaller up-sampling rate before reconstructing the mask. Finally, FCN16s and FCN8s converge faster than FCN32s in this case, which could be observed by the output mask of different epoch in each model.



6. [bonus] Calculate the result of  $d/dw$   $G(w)$ :

**objective function:**

$$G(w) = -\sum_n [t^{(n)} \log x(z^{(n)}; w) + (1 - t^{(n)}) \log (1 - x(z^{(n)}; w))] \geq 0$$

$w^* = \arg \min_w G(w)$  choose the weights that minimise the network's surprise about the training data

$$\frac{d}{dw} G(w) = \sum_n \frac{dG(w)}{dx^{(n)}} \frac{dx^{(n)}}{dw} = -\sum_n (t^{(n)} - x^{(n)}) z^{(n)} = \text{prediction error} \times \text{feature}$$

$w \leftarrow w - \eta \frac{d}{dw} G(w)$  iteratively step down the objective (gradient points up hill) 39

Here  $x(z^{(n)}; w) = \frac{1}{1 + e^{-z^{(n)} w}} = (1 + e^{-z^{(n)} w})^{-1}$

$$G(w) = t^{(n)} \log x(z^{(n)}; w) + (1 - t^{(n)}) \log (1 - x(z^{(n)}; w))$$

Consider  $\log$  as  $\ln$  here; by chain rule  $\frac{d}{dw} G(w) = \sum \frac{dG}{dx} \frac{dx}{dw}$

$$\frac{dG(w)}{dx(z^{(n)}; w)} = \frac{t^{(n)}}{x(z^{(n)}; w)} + \frac{(-1)(1 - t^{(n)})}{1 - x(z^{(n)}; w)}$$

$$\frac{dx(z^{(n)}; w)}{dw} = \frac{z^{(n)} e^{-z^{(n)} w}}{(1 + e^{-z^{(n)} w})^2}$$

$\frac{1 + e^{-z^{(n)} w}}{1 + e^{-z^{(n)} w}} - \frac{1}{1 + e^{-z^{(n)} w}}$

$$\Rightarrow \frac{dG(w)}{dx(z^{(n)}; w)} \cdot \frac{dx(z^{(n)}; w)}{dw} = \left[ t^{(n)} (1 + e^{-z^{(n)} w}) + \frac{(t - 1)(1 + e^{-z^{(n)} w})}{e^{z^{(n)} w}} \right] \frac{z^{(n)} e^{-z^{(n)} w}}{(1 + e^{-z^{(n)} w})^2}$$

$$= \left( \frac{e^{-z^{(n)} w} t^{(n)} z^{(n)}}{1 + e^{-z^{(n)} w}} \right) + \frac{z^{(n)} (t - 1)}{1 + e^{-z^{(n)} w}}$$

$$= z^{(n)} \left[ (1 - x(z^{(n)}; w)) t^{(n)} + (t - 1) x(z^{(n)}; w) \right]$$

$$= z^{(n)} (t^{(n)} - x(z^{(n)}; w))$$

Therefore:

$$\frac{dG(w)}{dw} = \sum_n \frac{dG(w)}{dx^{(n)}} \frac{dx^{(n)}}{dw} = -\sum_n (t^{(n)} - x^{(n)}) z^{(n)}$$