

DLCV HW1 REPORT

R11521701

土木所營管組碩二 程懷恩

Problem 1: Image Classification

1. Draw the network architecture of method A or B

The Model B I chose is ResNet101

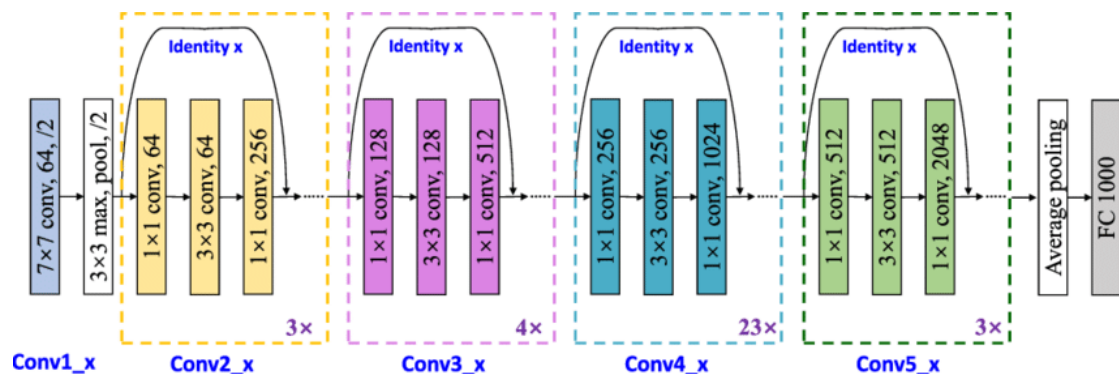


Fig1. Resnet101-architecture

2. Report the accuracy of your models (both A, and B) on the validation set.

1. Model A (from scratch)

Accuracy on validation set: 0.7004

2. Model B (with pre-trained weight)

Accuracy on validation set: 0.8844

3. Report your implementation details of model A.

My Model A from scratch is a simplified version of a Residual Network (ResNet). I referred to the design concept of the ResNet and then implemented a shallow from scratch. Below is the explanation of each component:

- conv1 Block:

This is first a convolutional block with 64 filters of size 3x3. Followed by Batch Normalization and ReLU activation.

- conv2 Block:

Another convolutional block with 128 filters. Followed by Batch Normalization, ReLU, and Max Pooling layer which will down-sample the spatial dimensions by a factor of 2.

- res1 Block:

This first residual block consists of two sub-blocks, each with 128 filters.

Each sub-block is followed by Batch Normalization and ReLU activation.
The output of this block is added element-wise to the output of the conv2 block (skip connection).

- conv3 Block:
Convolutional block with 256 filters.
Followed by Batch Normalization, ReLU, and Max Pooling.
- conv4 Block:
Convolutional block with 512 filters.
Followed by Batch Normalization, ReLU, and Max Pooling.
- res2 Block:
Second residual block consisting of two sub-blocks, each with 512 filters.
Each sub-block is followed by Batch Normalization and ReLU activation.
The output of this block is added element-wise to the output of conv4 block (skip connection).
- classifier Block:
Global Max Pooling layer of size 4x4.
Flatten layer to convert the 2D matrix into a vector.
Dropout layer for regularization with a drop probability of 0.2.
Fully connected layer (nn.Linear) that outputs the log probabilities of each class

	Input size		Layer				Output size				
Layer	Cin	H/W	filters	kernel	stride	pad	Cout	H/W	memory(KB)	params(k)	flop(M)
conv1	3	224	64	3	1	1	64	224	3211	2	287
conv2	64	224	128	3	1	1	128	112	3211	74	2355
res1	128	112	128	3	1	1	128	112	1606	295	1882
conv3	128	112	256	3	1	1	256	56	1606	295	941
conv4	256	56	512	3	1	1	512	28	1606	1181	470
Pooling	512	28	-	4	4	0	512	7	100	0	-
Flatten	512	7	-	-	-	-	25088	1	-	0	-
fc	25088	1	-	-	-	-	50	-	-	1255	-

Fig2. Numbers of hyperparameters in model A (from scratch)

```

(conv1): Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
)
(conv2): Sequential(
  (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(res1): Sequential(
  (0): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (1): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
)
(conv3): Sequential(
  (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(conv4): Sequential(
  (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(res2): Sequential(
  (0): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (1): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
)
(classifier): Sequential(
  (0): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (1): Flatten(start_dim=1, end_dim=-1)
  (2): Dropout(p=0.2, inplace=False)
  (3): Linear(in_features=512, out_features=50, bias=True)
)

```

Fig3. Architecture of model A

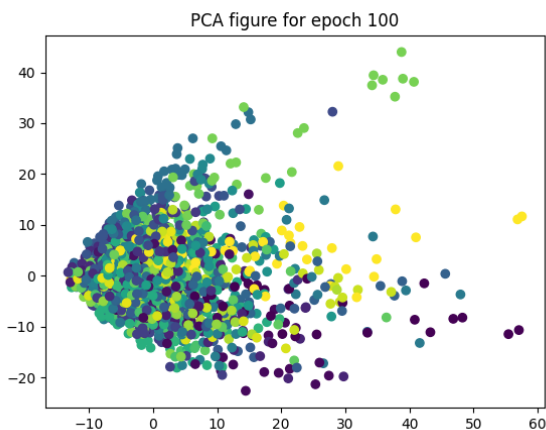
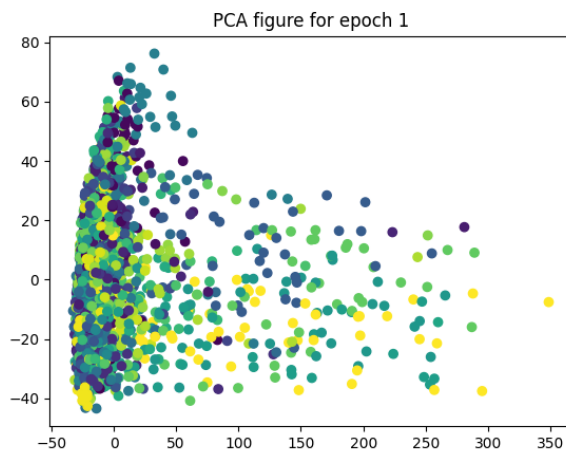
4. Report your alternative model or method in B, and describe its difference from model A.

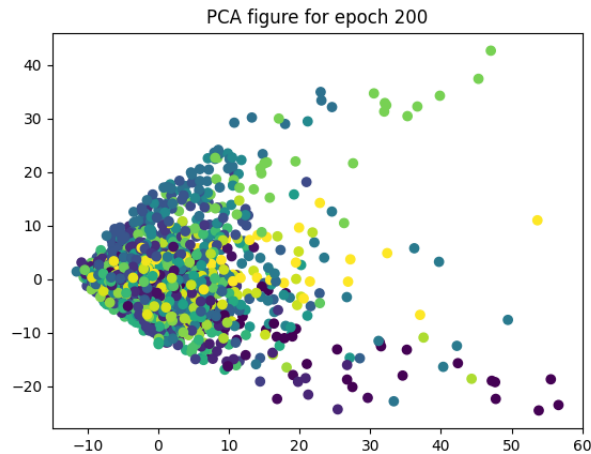
- Bottleneck Blocks: ResNet-101 primarily consists of bottleneck blocks. Each bottleneck block has three convolutional layers (conv1, conv2, and conv3). This design helps in reducing the number of parameters while maintaining the network's depth.
- Skip Connections (or Residual Connections): These connections skip one or more layers during the forward pass. The input to a block is added to the output of the block, helping in training deeper networks by mitigating the vanishing gradient problem.
- Batch Normalization: It normalizes the output of each layer to have zero mean and unit variance. This accelerates training and provides some level of

regularization, reducing the need for other regularization techniques like dropout.

- **ReLU Activation:** The Rectified Linear Unit (ReLU) activation function is used to introduce non-linearity into the model. It replaces all negative values in the output tensor with zeros.
- **Convolutional Layers:** The network uses 3x3 and 1x1 convolutional layers primarily. The 1x1 convolutions are used to change the number of channels in the tensor without altering its spatial dimensions.
- **Downsampling:** The spatial dimensions of the tensor are reduced (downsampled) in some blocks using strided convolutions or pooling layers to increase the network's receptive field.

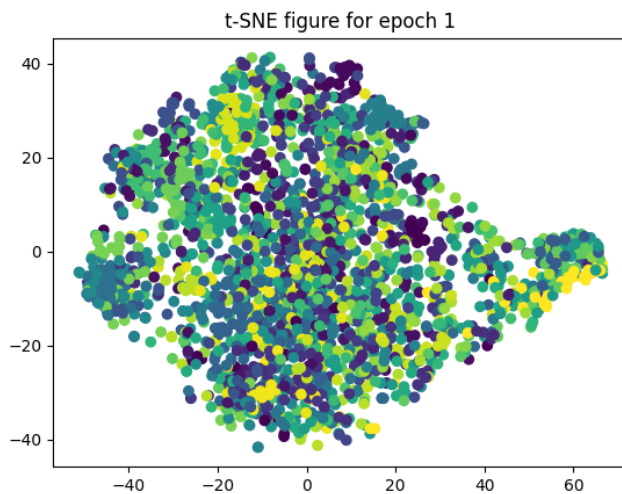
5. Visualize the learned visual representations of model A on the validation set by implementing PCA (Principal Component Analysis) on the output of the second last layer. Briefly explain your result of the PCA visualization.

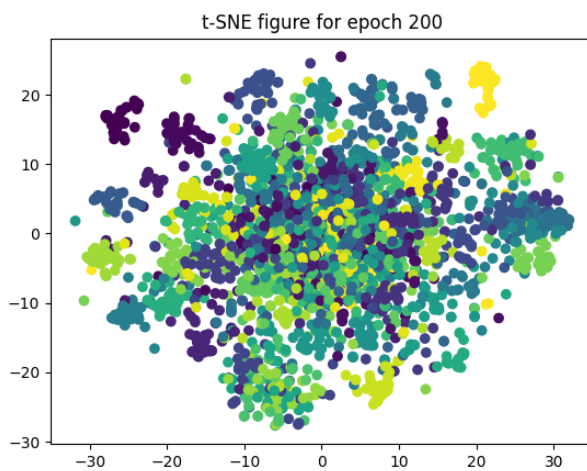
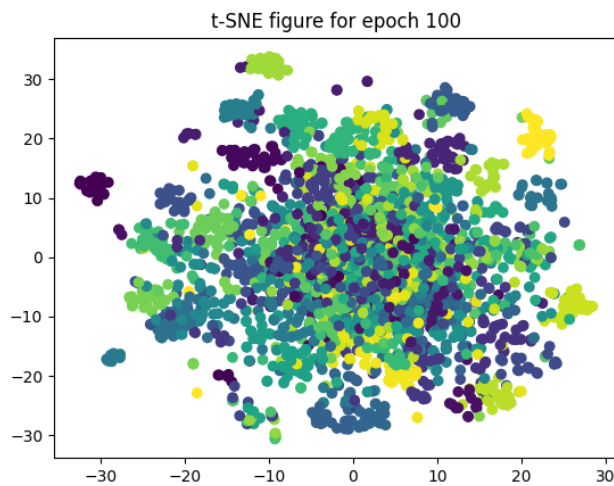




After more and more epochs of learning, the separating performance of the model becomes better. For instance, the “purple” class becomes more independent from others.

6. **Visualize the learned visual representation of model A, again on the output of the second last layer, but using t-SNE (t-distributed Stochastic Neighbor Embedding) instead. Depict your visualization from three different epochs including the first one and the last one. Briefly explain the above results.**



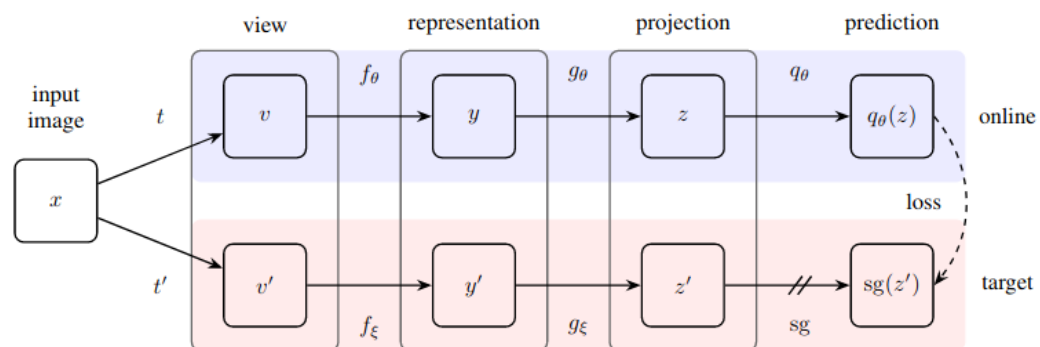


Even though the clustering in the middle is not very distinct, it can be observed that purple forms a cluster on its own, yellow forms a cluster on its own, and green also forms a cluster on its own. This indicates that the clustering is more apparent in the t-SNE dimensionality reduction representation. In conclusion, we can find out that t-SNE can separate classes more clearly.

Problem 2: Self-Supervised Pre-training for Image Classification

1. Describe the implementation details of your SSL method for pre-training the ResNet50 backbone. (Include but not limited to the name of the SSL method you used, data augmentation for SSL, learning rate schedule, optimizer, and batch size setting for this pre-training phase)

I chose the SSL method as BYOL (Bootstrap Your Own Latent), which was introduced by DeepMind. It's designed to learn visual representations without the need for labels.



Due to the default augmentations already present in the original BYOL code, to avoid redundant processing through two sets of transformations, I commented out the default BYOL augmentations. As a result, the loss obtained during training is approximately 0.01. I also compared this with the results when the default augmentations were not commented out, and the loss value went up to around 0.26.

Classifier Architecture:

Model: Multi-Layer Perceptron (MLP) with the following layers and characteristics:

- Input Size: 1000 (backbone model outputs a feature vector of size 1000)
Because I used the full ResNet50 model (including its final 1000-dimensional layer) as a feature extractor. The output from this feature extractor is then passed to my custom Classifier to get the final predictions.
- Hidden Layers:
Layer 1: 512 units, Batch Normalization, ReLU activation, Dropout (0.5)
Layer 2: 256 units, Batch Normalization, ReLU activation, Dropout (0.5)
- Output Layer: 65 units (Assuming a classification task with 65 classes)
- Activation Function: ReLU for hidden layers
- Dropout: 0.5 for regularization in hidden layers

2. Please conduct the Image classification on Office-Home dataset as the downstream task. Also, please complete the following Table, which contains different image classification setting, and discuss/analyze the results.

Setting	Pre-training	Fine-tuning	Validation accuracy
A	-	Train full model	40.15%
B	w/label	Train full model	46.55%
C	w/o label	Train full model	43.10%
D	w/ label	Fix the backbone Train Classifier only	33.99%
E	w/o label	Fix the backbone Train Classifier only	5.67%

Comparison between A, B, and C

In Setting A, using the ResNet-50 model without any pre-training achieves an accuracy of up to 40.15%. However, in Setting C, when I load a model pre-trained with SSL, the performance obtained through training is only slightly higher, around 3% more than the A model. Moreover, the performance of self-supervised learning in setting C (43.10%) slightly lags behind that of supervised learning in setting B (46.55%) and converges more slowly.

Comparison between D and E

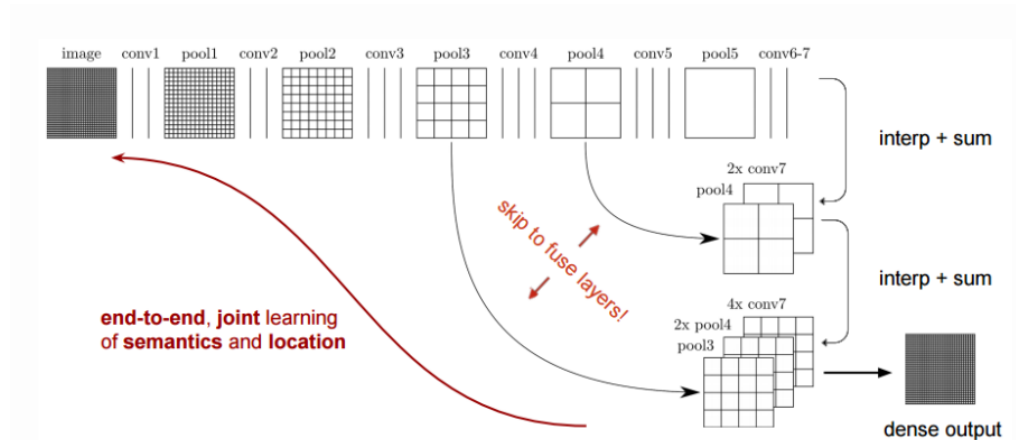
At the same time, under the condition of fixing the backbone architecture, when we assess the performance in Settings D and E, we find that the accuracy of the SSL method is not satisfactory. Even though the supervised backbone was expected to work better, with E showing extremely poor performance, I don't think it's solely due to the SSL method's shortcomings, but rather there may be issues with parameter settings. In my point of view based on the experiments, there are more training and experiments are needed for adjustment.

Conclusion

Based on the results of my experiments, it seems that the BYOL pre-trained model's performance does not match the low loss values observed.

Problem 3: Semantic Segmentation

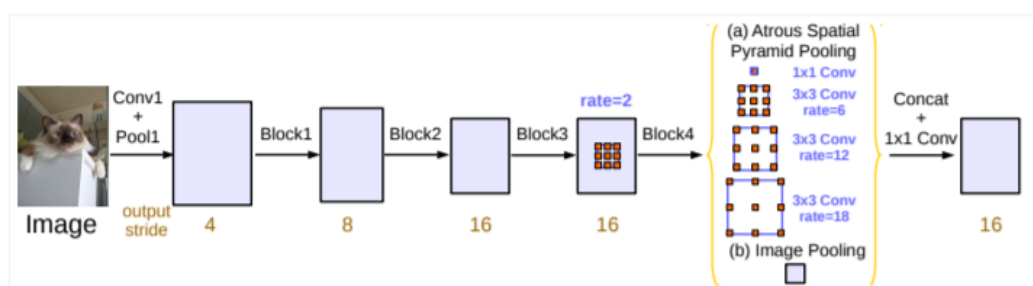
1. Draw the network architecture of your VGG16-FCN32s model (model A).



With 32x unsampled prediction (FCN-32s)

A classification network predicts an entire image as a certain class. FCN (Fully Convolutional Network) improves upon the classification network by replacing fully connected layers with convolutional layers, enabling end-to-end training for semantic segmentation. The network structure from 'conv1' to 'pool5' follows the VGG16 architecture, which reduces the image size by 32 times. FCN replaces fully connected layers with convolutional layers, resulting in an output size of $4096 \times (h/32) \times (w/32)$, where 'class' represents the number of classes to classify. This allows classification for each pixel. The convolutional results are then upsampled by a factor of 32 to restore them to the original size.

2. Draw the network architecture of the improved model (model B) and explain it differs from your VGG16-FCN32s model.



Parallel modules with atrous convolution (ASPP), augmented with image-level features.

Taken from <https://arxiv.org/pdf/1706.05587.pdf>

The most significant difference from VGG16 is that Deeplab V3 utilizes Atrous Spatial Pyramid Pooling (ASPP) to capture a range of information from fine to coarse details.

In Conclusion, DeepLab v3 incorporates several key innovations, such as atrous convolutions and ASPP, to enhance the capture of multi-scale context and generate highly accurate segmentation maps. It's worth noting that earlier models like VGG16-FCN32s played a foundational role in the development of these advancements within the field of deep learning-based semantic segmentation.

3. Report mIoUs of two models on the validation set.

	Validation mIoU	Trained Epoch
Model A (VGG16-FCN-32s)	0.5527	20
Model B (Deeplab v3)	0.7529	20

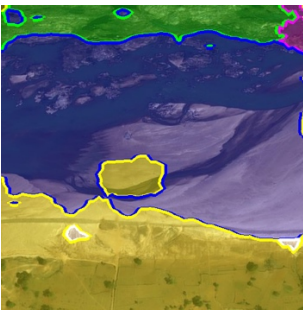



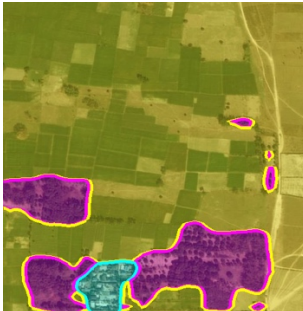

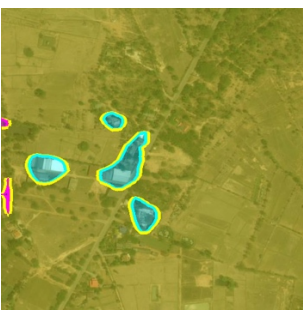


4. Show the predicted segmentation mask of “validation/0013_sat.jpg”, “validation/0062_sat.jpg”, “validation/0104_sat.jpg” during the early, middle, and the final stage during the training process of the improved model.

Trainin Epoch: 20

batch_size = 12

criterion = nn.CrossEntropyLoss()

The understood issue in object detection is that in the foreground and background, there is often an extreme imbalance in the number of backgrounds and targets. When the number of backgrounds is extremely high, it significantly impacts the loss during training. The model tends to classify the results as backgrounds, either because it filters out the majority of backgrounds first or because the influence of easy negatives on classification is relatively low. Nonetheless, when directly compared, using both cross-entropy and focal loss methods, focal loss didn't notably enhance the model's performance. Therefore, I opted to stick with the former.

id/epoch	1	10	20
0013			
0062			
0104			

0013_sat.jpg:

In the 1st epoch, there is a noticeable gap between the model's predictions and the actual targets, especially in the central lake area. By the 10th epoch, the model's predictive performance has significantly improved, particularly in accurately identifying the lake's shape. At the 20th epoch, the prediction results are fairly consistent with those at the 10th epoch, with some enhancements in certain edge details.

0062_sat.jpg:

In the 1st epoch, the model's prediction differs substantially from the actual targets. By the 10th epoch, there is a significant improvement in the prediction results, notably in identifying the majority of the farmland areas. In the 20th epoch, compared to the 10th epoch, further refinements are made. However, in the upper-right corner, there are some areas that might be misclassified and could require running more epochs for further

observation, even though the IoU score is still higher.

0104_sat.jpg:

In the 1st epoch, the model's predictive performance for this photo is relatively better, though there are some discrepancies in small areas. At both the 10th and 20th epochs, the model's prediction results appear very similar and are in close proximity to the targets.

Reference:

- [1706.05587] Rethinking Atrous Convolution for Semantic Image Segmentation(arXiv.org)
- Deeplab v3 csdn https://blog.csdn.net/weixin_39357271/article/details/124066688
- Keras 建構 FCN32,16,8 語意分割模型
https://blog.csdn.net/qq_37923586/article/details/106843736
- DeepLabv3 — Atrous Convolution (Semantic Segmentation)
<https://medium.com/image-processing-and-ml-note/deeplabv3-atrous-convolution-semantic-segmentation-e8cbc111c792>
- [1708.02002] Focal Loss for Dense Object Detection(arXiv.org)
- ChatGPT (Assisting in refining my programs.)

Collaborator:

R11521613 Thank you, Lin, for the collaboration and discussions to complete p1 and p2.