Aerial Satellite Segmentation

Motivation

Nowadays, drones play an important role in our lives. It can not only be used to take photographs or shoot videos, but also can be used for military, search and rescue. And one of the main technologies of drones is Aerial Semantic Segmentation. It helps drones to do object classification, object detection and segmentation. Moreover, Aerial Semantic Segmentation is also widely used in many critical applications, such as autonomous cars, which is the most popular applications in recent years. Thus, we are interested in this topic and want to know more about it.

Problem Description

In this project, we discuss which model is more suitable for Aerial Semantic Segmentation. We talk about 4 models: Simple CNN model, FCN8s+VGG-19, DeepLabV3+ with ResNet101, and the U-Net model.

Contribution

- Simple CNN model – Zhiying Guo
- FCN8s+VGG-19 – Jiajun Ren
- DeepLabV3+ with ResNet101 – Jinlun Zhang
- U-Net model – Xiaofan Cheng

Related Work

Before choosing to use the model, Xiaofan carefully searched for information and found a paper MLCRNet: Multi Level Context Refinement for Semantic Segmentation in Aerial Images (Huang, Z. et al, 2022). This paper mainly uses MLCRNet for aviation semantic segmentation. In the paper, Xiaofan compared the common FCN models. At the end of the paper, the value of IoU of the model is 75.7%, which is good. This time, Xiaofan used the U-net model for semantic segmentation, compared with other algorithms, U-net has a better effect on samples with fewer samples and high-resolution images. The final value of IoU is 77%

Before selecting the model, Zhiying reviewed an article, *Semantic Segmentation of Aerial Imagery Using U-Net in Python* (Davies, A. J, 2022), which used U-net model to do semantic segmentation. It used MBRSC dataset, which contains 72 semantic segmentation annotated images grouped into 6 larger tiles. Zhiying used Semantic segmentation dataset from Kaggle, which contains 72 semantic segmentation annotated images grouped into 8 tiles. Their sizes are similar. And its accuracy is about 0.86 and my model's accuracy is about 0.75. It shows that the U-Net model might be more suitable for Aerial semantic segmentation.

Jiajun reviewed several papers about fully convolutional networks in aerial semantic segmentation in the literature review. Almost all of them mentioned the first paper that proposes FCN, which *is Fully Convolutional Networks for Semantic Segmentation*(J. Long et al, 2015). In that paper, it proposed FCN with VGG-16 as the backbone. With the validation set of PASCAL VOC 2011, the mean IoU score can be reached at 56, and the mean accuracy of FCN-8s can be achieved at 75.9. The strength of FCN is to convert the classification network as a pixel classification network for segmentation to get higher accuracy results. In this project, Jiajun tries to implement FCN-8s+VGG19, and FCN-8s+VGG19 can get better results in IoU and accuracies.

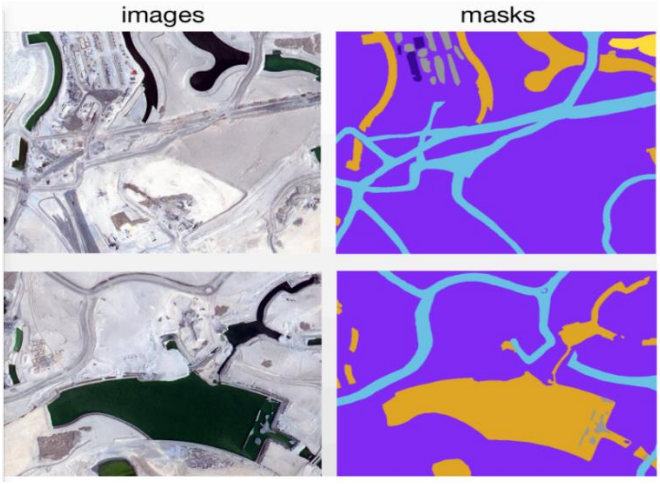
Jinlun reviewed multiple papers about aerial imagery pixel-level segmentation, in which the DeepLabV3+ model is frequently selected to complete the various challenges of semantic segmentation within Computer Vision due to DeepLavbV3+ model's state-of-the-art performance on segmentation datasets, and the fact that its proposed techniques are suitable for aerial imagery challenges (Heffels M, et al, 2020). Thus, Jinlun planned to extend the DeepLavbV3+ model with ResNet50 being the backbone (D & D, 2021). With the validation set consisting of aerial satellite imagery of Dubai obtained by MBRSC satellites, the mean IoU and accuracy scores achieved by this DeepLavbV3+ with ResNet50 backbone model are around 66% and 81%. Other novel models are also considered but not deemed ideal, such as YOLOv4 which mainly focuses on high-speed object detection and HRNet-OCR which has shortcomings regarding to the coarse labeling.

Dataset used

The dataset that we used is *Semantic segmentation of aerial imagery* on Kaggle. It contains 72 images of satellite images of Dubai with 797x644 size.

The dataset is divided into eight folders. Tile 1 to Tile 8 have two folders under each tile: images and masks. Images store the original images, and masks store the ground truth images after semantic segmentation

The Aerial Satellite Segmentation. Dataset includes 6 categories: Building, Land (unpaved area), Road, Vegetation, Water and Unlabeled with colors: '#3C1098', '#8429F6', '#6EC1E4', '#FEDD3A', '#E2A929', and '#9B9B9B'.



Data preprocessing

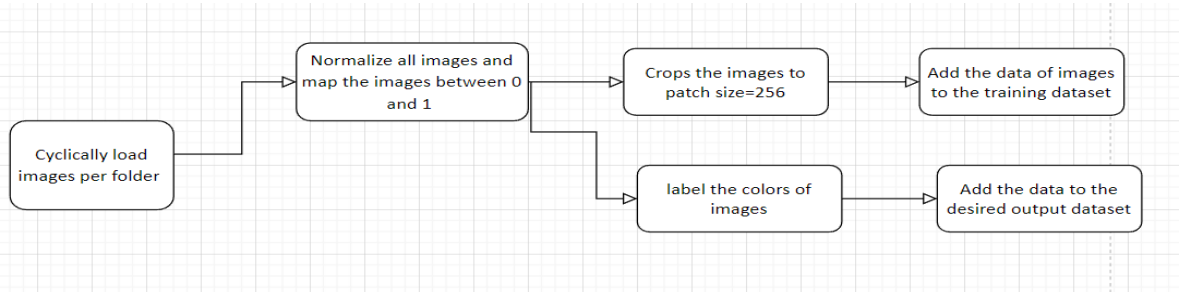


Figure 1. Flowchart of data preprocessing process

The preprocessing process first normalized all the data of images between 0 and 1 to put all data in the same scale. Then it crops the image data to patch size=256 since 256x256 is the common input size of the semantic segmentation models.

Individual part: Simple CNN – Zhiying Guo

Experimental setup

I used Google Colab to do the implementation. And I used keras-segmentation package for creating the model.

Here is my data partition for the input:

```
X_train shape:  (928, 256, 256, 3)
y_train shape:  (928, 256, 256, 6)
X_test shape:  (232, 256, 256, 3)
y_test shape:  (232, 256, 256, 6)
```

Training

This structure is also called encoder-decoder structure. Its basic logic is: first, encode the information about the objects in the input images, and then decode the information and produce the segmentation maps. My model consists of 3 encoder layers and 3 decoder layers. Each encoder layer has two convolution layers and one max pooling layer which would downsample the image by a factor of two. Each decoder layer also has two convolution layers. Between each decoder layer, there is a concatenate layer which is upsampling the tensor. And the final output layer filters the tensor to the number of the classes of the input images.

```python
def cnn_model(n_classes, input_shape):
    img_input = Input(shape=input_shape)
    # encoder layer 1
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(img_input)
    conv1 = Dropout(0.2)(conv1)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D((2, 2))(conv1)
    # add an addition layer to do the same encoding as layer 1
    conv1_addition = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    conv1_addition = Dropout(0.2)(conv1_addition)
    conv1_addition = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1_addition)
    pool1_addition = MaxPooling2D((2, 2))(conv1_addition)
    # encoder layer 2
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1_addition)
    conv2 = Dropout(0.2)(conv2)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D((2, 2))(conv2)
    # decoder layer 1
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = Dropout(0.2)(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    # concate the intermediate encoder conv2 outputs with the intermediate decoder conv3 outputs
    up1 = concatenate([UpSampling2D((2, 2))(conv3), conv2], axis=-1)

    # decoder layer 2
    conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(up1)
    conv4 = Dropout(0.2)(conv4)
    conv4 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv4)
    # concate the intermediate encoder conv1 outputs with the intermediate decoder conv4 outputs
    up2 = concatenate([UpSampling2D((2, 2))(conv4), conv1], axis=-1)
    # decoder layer 3
    conv5 = Conv2D(32, (3, 3), activation='relu', padding='same')(up2)
    conv5 = Dropout(0.2)(conv5)
    conv5 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv5)
    # output layer
    out = Conv2D( n_classes, (1, 1) , activation='softmax', padding='same')(conv5)

    model = Model(inputs=[img_input], outputs=[out])
    return model
```

Figure 1. The code of the model

The training result is:

```python
history = model.fit(
            X_train, y_train,
                batch_size = 8,
                verbose=1,
                epochs=5,
                validation_data=(X_test, y_test),
                shuffle=True
)
```

```
Epoch 1/5
116/116 [==============================] - 1689s 15s/step - loss: 0.6968 - accuracy: 0.5800 - jaccard_coef: 0.3964 - val_loss: 0.5610 - val_accuracy: 0.6933 - val_jaccard_coef: 0.5182
Epoch 2/5
116/116 [==============================] - 1681s 14s/step - loss: 0.5470 - accuracy: 0.7020 - jaccard_coef: 0.5425 - val_loss: 0.5202 - val_accuracy: 0.6932 - val_jaccard_coef: 0.5368
Epoch 3/5
116/116 [==============================] - 1678s 14s/step - loss: 0.4907 - accuracy: 0.7410 - jaccard_coef: 0.5920 - val_loss: 0.5052 - val_accuracy: 0.7019 - val_jaccard_coef: 0.5496
Epoch 4/5
116/116 [==============================] - 1678s 14s/step - loss: 0.4779 - accuracy: 0.7531 - jaccard_coef: 0.6086 - val_loss: 0.4616 - val_accuracy: 0.7453 - val_jaccard_coef: 0.6021
Epoch 5/5
116/116 [==============================] - 1689s 15s/step - loss: 0.4723 - accuracy: 0.7578 - jaccard_coef: 0.6151 - val_loss: 0.4572 - val_accuracy: 0.7487 - val_jaccard_coef: 0.6069
```

I set the epochs=5 and batch size=8. I chose epochs=5 because each epoch takes a long time. One epoch takes about 30 minutes. The whole training process takes about 2.5 hours. And the accuracy does not change too much after epoch 3. My stopping criteria is when computational bounds are exceeded, the process stops.

The training result is: loss ≈ 0.47, accuracy ≈ 0.76, IoU ≈ 0.60.

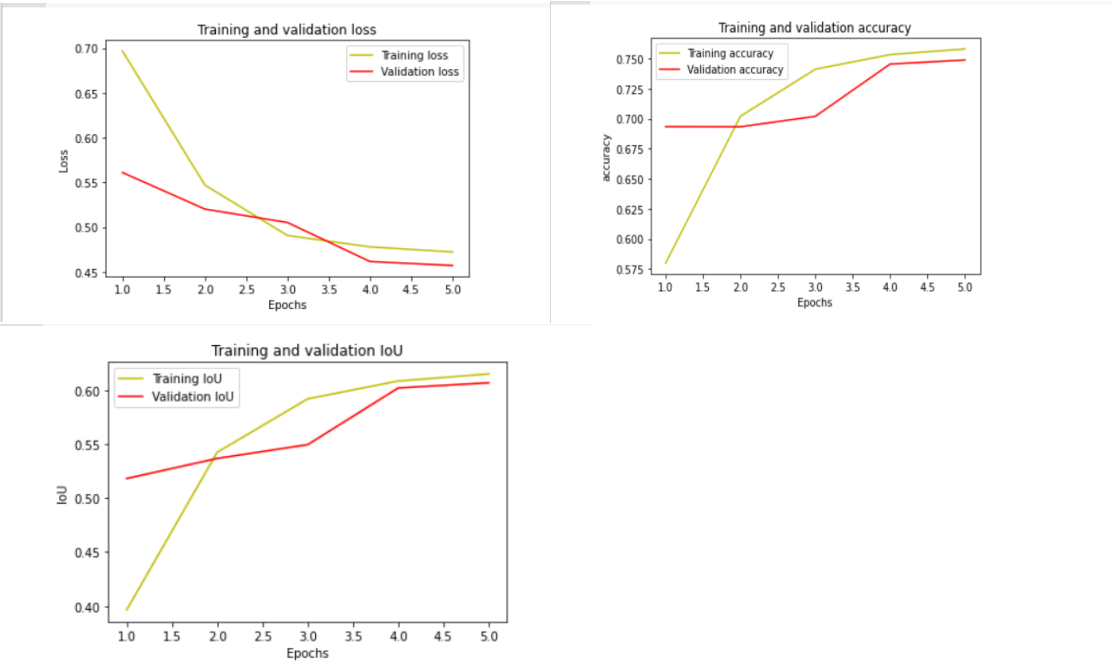
Extension made from the original model

Comparing with the model I showed in the Talk and slides, I add an additional layer which is the same layer as the first encode layer. I want to see whether the accuracy and IoU would increase if I encoded the information twice. The training result of my original model is: loss ≈ 0.45, accuracy ≈ 0.75, IoU ≈ 0.60. Thus, the results of these two models are not much different.

Validation

I use DiceLoss() to calculate the loss of the model.

Here are the graphs of loss, accuracy, and IoU indicator:



The validation result is: loss ≈ 0.46, accuracy ≈ 0.75, IoU ≈ 0.60

Testing

```
8/8 [==============================] - 89s 11s/step
IoU = 0.5979156494140625
```

Figure 2. Testing result of the model

The IoU indicator of the testing process is about 0.60.

Individual part: Jiajun Ren

Model Introductions:

FCN8s+VGG16(original model)

In FCN8s+VGG16, VGG-16 is the backbone of FCN. The input of VGG-16 is a 224*224 RGB image. There are 13 convolutional Layers in VGG-16, and three fully connected layers in VGG-16, which can also be regarded as convolutions covering the entire area. Converting the fully connected layer to a convolutional layer can change the network output from a one-dimensional non-spatial output to a two-dimensional matrix and use the result to generate a heatmap of the input image map. After the first pooling, we can get pool 1, which is 1/2 of the original input size. After the second pooling, we can get pool 2, which is 1/4

of the original input size. After the third, fourth and fifth pooling, we can get pool 3 with the size becoming 1/8 of the original input size, pool 4 with the size becoming 1/16 of the original input size and pool 5 with the size becoming 1/32 of the original input size.

After the convolution of VGG-16, we can get the final pooling layer with 1/32 of the original size. The next step is deconvolution, upsampling the last pooling layer four times, upsampling the pool four-layer two times and upsampling the pool three-layer one time. The Sum of the three upsampling will has the same size as the input. That's the output of FCN8S with VGG-16 as the backbone.

FCN8s+VGG19(improved model)

FCN8s+VGG16 and FCN8s+VGG19 are very similar. Compared with FCN8s+VGG16, FCN8s+VGG19 is a deeper network with three more convolutional layers. So, three convolutional lays are added to the original module to seek a better result of training.

Data Pre-Processing:

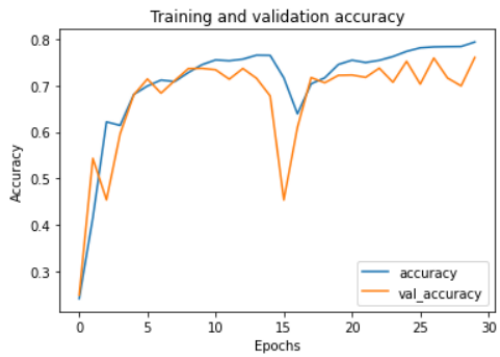
The preprocessing data strategy of FCN8s-VGG16 and FCN8s-VGG19 model is very similar to what we mentioned in the previous data preprocessing part. Since the input size of VGG-16 and VGG-19 must be 224x224 sized images. Thus, instead of resizing the input image to a 256x256 size, my preprocessing data strategy resizes the input image to a 224x224 size.

Experimental setup:

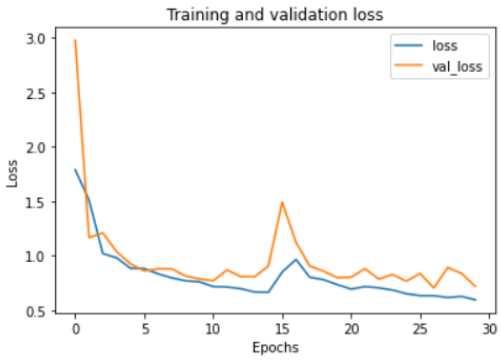
Google Colab Pro is used for whole code implementation. Keras segmentation package is used for setting up the model.

Training/testing/validation:

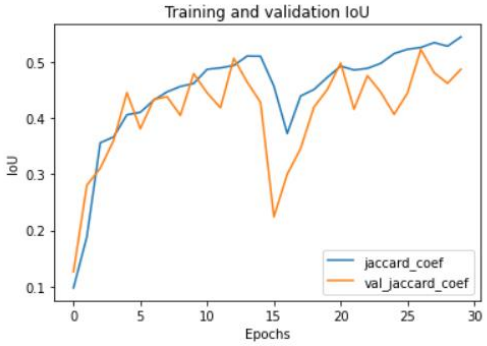
The FCN8-s+VGG-16 model (original model) is trained by 30 epochs. The training used SGD as an optimizer with a learning rate of 0.1. The batch size of the training process is set to 8, and the verbose is set to 2. For the loss function, "categorical_crossentropy" is used during training. The training process took about 2 minutes. After the 30 epochs, the training accuracy reached 0.7944 and the training loss rate reduced to: 0.5965. For validation, validation accuracy reached 0.7610 and validation loss is reduced to 0.7197. For the testing IoU score, the FCN8s with VGG-16 as backbone got 0.623.



FCN8+VGG16 training and validation accuracy

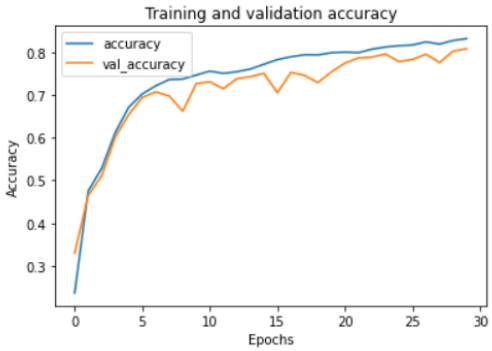


FCN8+VGG16 training and validation loss

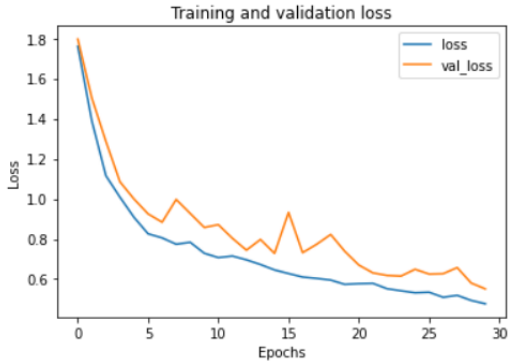


FCN8+VGG16 training and validation IoU

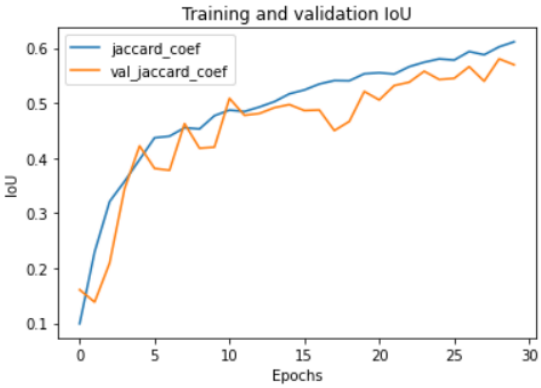
The FCN-8s+VGG-19 model (improved model) is trained by 30 epochs. The training used SGD as an optimizer with a learning rate of 0.033. The batch size of the training process is set to 8, and the verbose is set to 2. For the loss function, "categorical_crossentropy" is used during training. The training process took about 2 minutes. After the 30 epochs, the training accuracy reached 0.8322 and the training loss rate reduced to: 0.4770. For validation, validation accuracy is reached 0.8085 and validation loss is reduced to 0.5511. For the testing IoU score, the FCN8s with VGG-19 as backbone got 0.671. Compared to FCN8s+VGG16, the FCN8s+VGG19 gets better results, and FCN8s+VGG19 gets a 0.0378 increment in training accuracy and reduces 0.1195 more in training loss; it also gets a 0.0475 increment in training accuracy and reduces 0.1686 more in training loss. For the testing IoU, FCN8s+VGG19 gets a 0.048 increment.



FCN8+VGG19 training and validation accuracy



FCN8+VGG19 training and validation loss



FCN8+VGG16 training and validation IoU

Individual part: Jinlun Zhang

Data Pre-Processing and Model Architecture Description:

Though sharing the same dataset with the other members in the group, my model implements a slightly different data pre-processing strategy as shown below in the dataflow Figure 1.

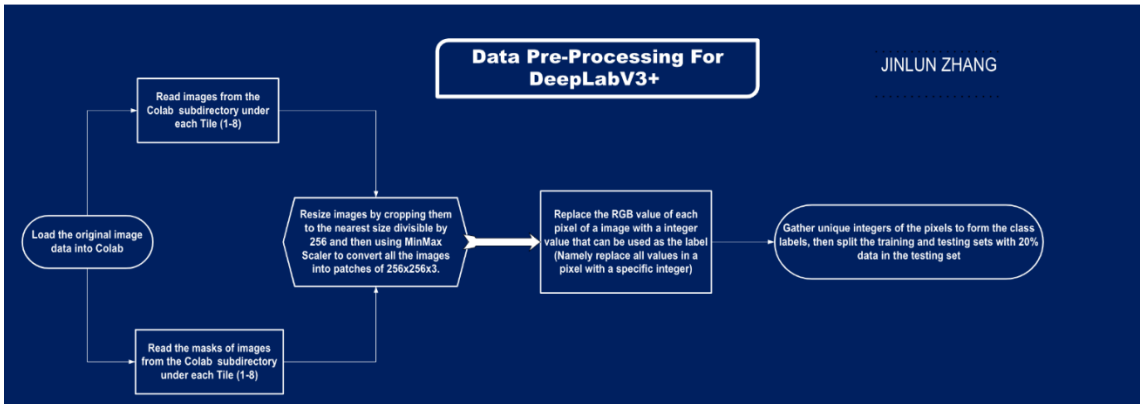


Figure 1: Dataflow of Data Pre-Processing for DeepLabV3+

After loading the data directory into the Colab cloud platform, I resize the images and the masks of the images by cropping them to the nearest size divisible by 256, and then applying the Min-Max normalization (restrict the values to fall within a certain range without changing the original value distribution) to convert all the images into patches of 256*256*3. This resizing strategy may yield a better model performance than simply dividing the images by 255 (as the information loss due to the cropping is minimized). Afterwards, we replace the RGB vector of each pixel of a given image with an integer (scalar) value that can be used as the class label (in this case, we have a total of 6 unique integers assigned to the pixels to represent the 6 classes / categories our dataset contains). Finally, using the 'train_test_split' API, we have gained our training and testing sets with 20% of data patterns being placed to the testing set.

Moving to the model architecture description section, the model I selected for this aerial satellite segmentation task is called DeepLabV3+, which is one of the most promising models for multi-class semantic image segmentation in Deep Learning. As a state-of-art model for semantic image segmentation, its goal is to assign semantic labels to every pixel in the input image (which is the reason why we replace the RGB vector of each pixel with an integer value in the data pre-processing section). The architecture of the DeepLavV3+ model is shown below in Figure 2.

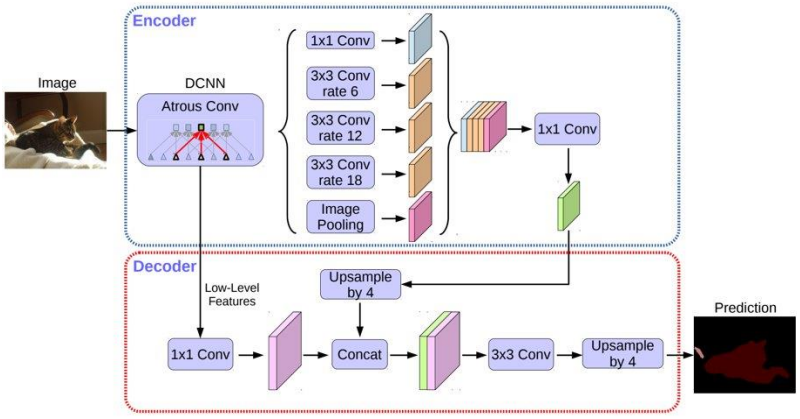


Figure 2: Architecture of DeepLabV3+

As illustrated in Figure 2, the DeepLabv3+ model has an encoding phase and a decoding phase. The encoding phase extracts the essential information from the image using CNN (with pooling layers) to gradually reduce the feature maps and to capture higher-level semantic information, whereas the decoding phase reconstructs the output of appropriate dimensions based on the information obtained from the encoder phase. Moreover, the decoder module is added to give better segmentation results along object boundaries and gradually recovers the spatial information.

In addition, as shown in the Figure 3, DeepLabv3+ also implements the Atrous Spatial Pyramid Pooling (ASPP) scheme. The idea is to apply multiple atrous convolutions to the input feature map, and fuse the results together, where the atrous convolution is a tool for refining the field of view by modifying a parameter termed atrous rate to enlarge the field of view of filters without affecting the computation or reducing the spatial dimension. The reason of implementing this scheme is that since objects of the same class can have different scales in the image, using ASPP can help to account for the different object scales in the images to improve the overall accuracy of the model predictions.

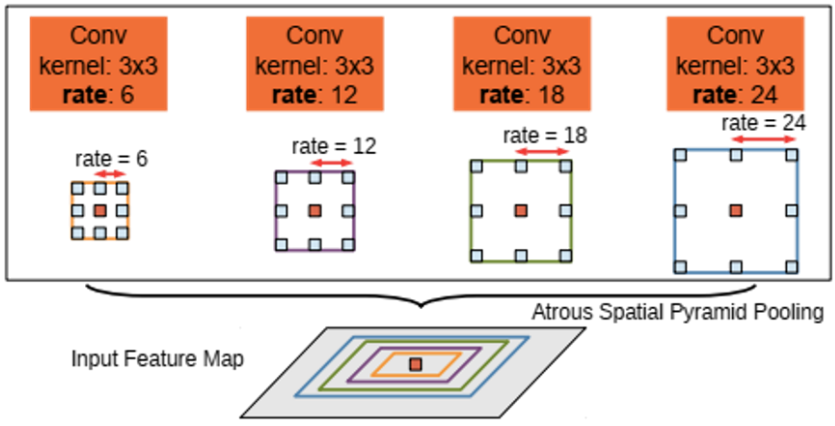


Figure 3: Illustration of Atrous Spatial Pyramid Pooling

Furthermore, for this DeepLabv3+ model I am going to explore, the Residual Network, Resnet, is implemented as the backbone for the training of this DeepLabv3+ model. ResNet is famous for its

utilization of skip connection to add the output from an earlier layer to a later layer, which helps to mitigate the vanishing gradient problem. This means that using ResNet can allow us to train extremely deep neural networks, such as the one with 200 layers, successfully. The position of the ResNet in the DeepLabv3+ model is displayed in Figure 4, where ResNet50 (used in the original code) is a variant of ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer.

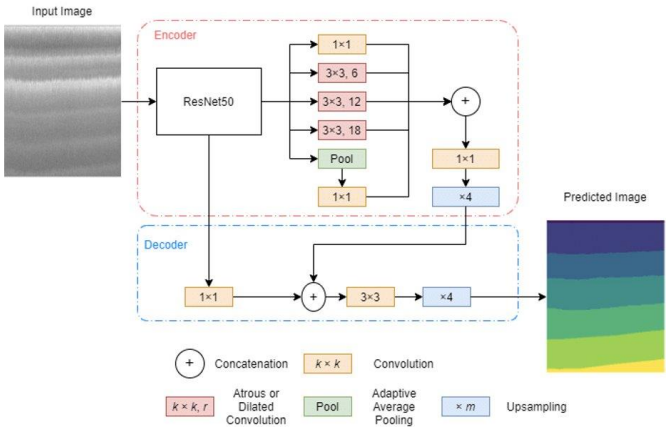


Figure 4: Another illustration of the Architecture of DeepLabv3+ model, where ResNet50 is used in the Encoder section

Experimental Setup:

Google Colab platform is utilized for the model implementation, and the additional Python libraries needed to be downloaded in Colab are 'segmentation-models' and 'patchify' through '!pip install' commend. The image dataset is uploaded to Colab in the zip file and unzipped in Colab with the commend '!unzip /content/semantic_segmentation_dataset.zip'

Training and Testing:

With using the Categorical Crossentropy as the loss function and the Adam (with learning rate being 0.001) as the optimizer, the training and testing results produced by the original DeepLabv3+ model without any extension for 10 epochs are shown in Figure 5. It can be observed from the training result that the training accuracy is gradually increasing and reaches a satisfactory result of 86% at the end of the training process. However, there is a sharp drop in the testing accuracy at the epoch 3, indicating a possibility of overfitting or bad luck during the optimization process. Since I only train the model for 10 epochs due to the limited computational resources I have access to, this unexpected drop may be attributed to the insufficient number of epochs, and this sharp drop should become negligible when the number of epochs becomes large. Thus, there is a possibility of the model being underfitted.

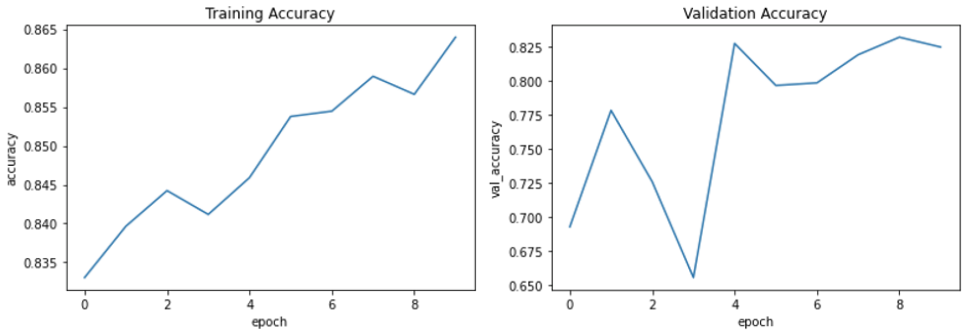


Figure 5: Training and Testing Accuracy Produced by the Original DeepLabv3+ without any Extension

As for the main evaluation criteria our group chose for the model performance, the IoU evaluation score (ratio of the overlap area to the combined area of prediction and ground truth) the original DeepLabV3+ achieved is around 56%, which is not ideal, and we can assume the existence of underfitting in the model training.

Regarding to my extension to this existing DeepLabv3+ model, I firstly change the ResNet50 backbone to ResNet101, which is a deeper Resnet structure than ResNet50, because doing so, we can retrieve more accurate information from the input images in the encoder section of the model to train the model more effectively. Moreover, as a deeper ResNet resolves the problem of gradient vanishing in a deeper NN structure with using the concept of skip connection, and it improves the efficiency of deep neural networks with more layers while minimizing the percentage of errors, I add one additional convolutional block in the decoder section of the model to maximize the model improvement from using a deeper ResNet backbone. Furthermore, the additional convolutional layers can help to solve complex problems more efficiently as the different layers could be trained for varying tasks to get higher accurate aggregated results. Thus, I am expecting to get better performance from adding more layers to this existing DeepLabv3+ model, which may increase the risks of encountering overfitting.

The training and testing results produced by my extended DeepLabv3+ model for 10 epochs are shown in Figure 6. The training accuracy is increasing smoothly but reaches a slightly lower accuracy at the end of the 10 epochs of training than the original model, which again can be attributed to the insufficient number of training epochs as a deeper NN model is required to have an increased number of training iterations to outperform the others. As for the validation accuracy, it is still oscillating on a 10% range at the end of the training, indicating clear evidence of underfitting.

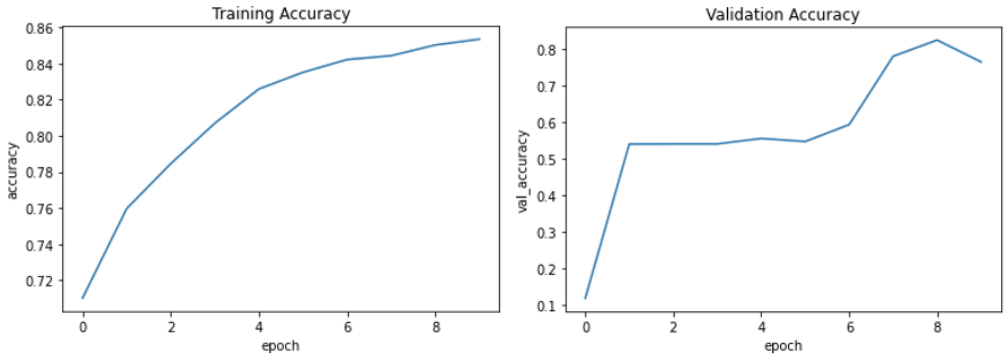


Figure 6: Training and Testing Accuracy Produced by my Extended DeepLabV3+

This extended model achieves a worse IoU score. Comparing to the 56% achieved on the original model, it only yields 43% IoU, which implies that we should adopt to a simpler model structure when the amount of computation resources and time are limited for better model maintenance, and my extended model may outperform the original model when the number of training epochs is sufficient

As for the IoU evaluation score, comparing to the 56% achieved on the original model, my extended model only yields 43% on the IoU evaluation score, which implies that we should adopt to a simpler or smaller model architecture when the amount of computation resources and time are limited for better model performance, and I expect my extended model to outperform the original model when the number of training epochs is sufficiently large. Hence, one of the future works I am considering to do is to exploit the cloud computing resources (such as the Microsoft Azure) to train more epochs in my extended DeepLabV3+ model to improve its accuracy and conduct the hyper-parameter tuning to gain the optimal model performance.

Individual part: Xiaofan Cheng

Experimental setup

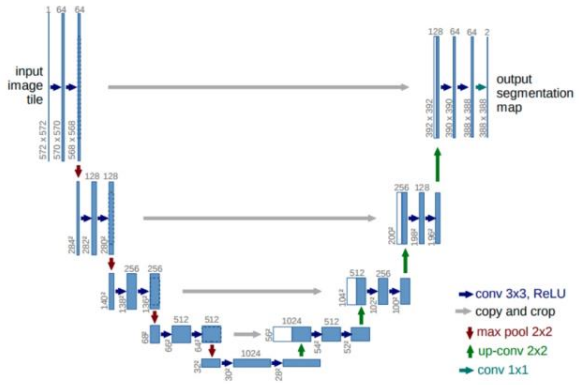
I use anconda and jupyter notebook for programming, The main third-party python libraries are as follows

```
numpy
pandas
imageio
matplotlib
json
tensorflow
sklearn
segmentation_models
```

TensorFlow mainly carries out U-net network design, segmentation_ Models mainly use the DiceLoss() function to optimize losses.

Training

U-Net structure



According to the U-net network, we use tensorflow to design and implement

```
def unet_model(n_classes=n_classes, IMG_HEIGHT=patch_size, IMG_WIDTH=patch_size,
IMG_CHANNELS=3):
    # Build the model
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    # s = Lambda(lambda x: x / 255)(inputs)   #No need for this if we normalize our
inputs beforehand
    s = inputs

    # Contraction path
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(s)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(p1)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal',
padding='same')(p2)
```

Instantiate U-net model and define loss function, use IoU as model accuracy evaluation indicator, and use Adam as optimizer

```
metrics=['accuracy', jaccard_coef]
loss = sm.losses.DiceLoss()
model= unet_model()
model.summary()
model.compile(optimizer='adam', loss=loss, metrics=metrics)
```

train set and test set split

```
X_train, X_val, y_train, y_val = train_test_split(train_set, train_lab, test_size =
0.20, random_state = 42)
```

Training

```
history = model.fit(X_train, y_train,
                    batch_size = 8,
                    verbose=1,
                    epochs=30,
                    validation_data=(X_val, y_val),
                    shuffle=True)
```

batch_ Size=8 is to take out 8 pictures for training each time, epochs=30 is to update parameters for 30 gradient descent, validation_ Data=(X_val, y_val) is the verification set to prevent over fitting during gradient descent

Validation

In the process of training, we verify , validation_ Data=(X_val, y_val) is the verification set to prevent over fitting during gradient descent

```
history = model.fit(X_train, y_train,
                    batch_size = 8,
                    verbose=1,
                    epochs=30,
                    validation_data=(X_val, y_val),
                    shuffle=True)
```

Testing

Model optimization comparison:

Main optimization means: optimize the model # On the basis of the original U-net model, after each convolution layer is calculated, perform a Dropout to optimize the u-net network model, in order to prevent the model from over fitting and improve the model's generalization ability on the test set.

```
def unet_model2(n_classes=n_classes, IMG_HEIGHT=patch_size, IMG_WIDTH=patch_size, IMG_CHANNELS=3):
#Build the model
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    #s = Lambda(lambda x: x / 255)(inputs)   #No need for this if we normalize our inputs beforehand
    s = inputs

    #Contraction path
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
    # Dropout
    c1 = Dropout(0.1)(c1)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
    c3 = Dropout(0.1)(c3)
    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
    c4 = Dropout(0.1)(c4)
    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
    p4 = MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
    c5 = Dropout(0.2)(c5)

    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c5)
```

Whether to improve after optimization:

IoU value before optimization is 0.70

```
5/5 [==============================] - 13s 2s/step
IoU = 0.7094420790672302
```

The value of IoU after optimization is 0.774

Results and discussion

State-of-Arts Models	Validation Result
DeepLabV3+ with ResNet50	Loss ≈ 0.56, accuracy ≈ 0.8, IoU ≈ 0.66
Extended Models	
Simple CNN	loss ≈ 0.46, accuracy ≈ 0.75, IoU ≈ 0.60
FCN8s+VGG-19	Loss = 0.55, accuracy = 0.81, IoU = 0.623
DeepLabV3+ with ResNet101	Loss ≈ 0.68, accuracy ≈ 0.77, IoU ≈ 0.44
U-Net model	Loss = 0.3901, accuracy = 0.8723, IoU= 0.774

The Simple CNN model is not trained for an adequate number of epochs. The same issue is shared with the DeepLabV3+ with ResNet101 model, whose performance is unsatisfactory due to the insufficient number of training epochs, which is attributed to the amount of computational resources and time available. Although FCN8s+VGG19 can run smoothly through 20 epochs, pixel-to-pixel relationships (such as discontinuity and similarity) are not fully considered in FCN8s+VGG19. FCN8s+VGG19 ignores the spatial regularization step, commonly used in most other pixel-level classifications, so it lacks spatial consistency. That's why it can't get the best result of loss, accuracy and IoU within the 4 models. The U-net model is used for semantic segmentation. Compared with other algorithms, U-net has a better segmentation effect on dataset with fewer samples and higher image resolution, but it is not sensitive to edge classification.

Conclusion and future work

The best model we implemented is the U-Net model. It has the lowest loss=0.375, the highest accuracy=87.23%, and the highest IoU indicator=0.774. In the future work, we can use a bigger dataset to allow our models to have more data to learn. Moreover, we should try to increase the number of epochs to see whether the model is improved or not.

Reference

Davies, A. J. (2022, March 31). Semantic Segmentation of Aerial Imagery Using U-Net in Python [web log]. Retrieved September 18, 2022, from https://towardsdatascience.com/semantic-segmentation-of-aerial-imagery-using-u-net-in-python-552705238514.

D, N., & D, K. (2021, December 29). https://github.com/nive927/Dubai-Satellite-Imagery-Multiclass-Segmentation/blob/main/Report/Report-Nivedhitha-Karthik-Dubai-Dataset.pdf. GitHub. Retrieved December 1, 2022, from https://github.com/nive927/Dubai-Satellite-Imagery-Multiclass-Segmentation/blob/main/Report/Report-Nivedhitha-Karthik-Dubai-Dataset.pdf.

Heffels, M. R., & Vanschoren, J. (2020). Aerial Imagery Pixel-level Segmentation. arXiv. https://doi.org/10.48550/arXiv.2012.02024.

Huang, Z., Zhang, Q., & Zhang, G. (2022). MLCRNet: Multi-Level Context Refinement for Semantic Segmentation in Aerial Images. Remote Sensing, 14(6), 1498. MDPI AG. Retrieved from http://dx.doi.org/10.3390/rs14061498.

J. Long, E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3431-3440, doi: 10.1109/CVPR.2015.7298965.