Cloud segmentation in Satellite Images

Introduction

Cloud Detection and why is it important?

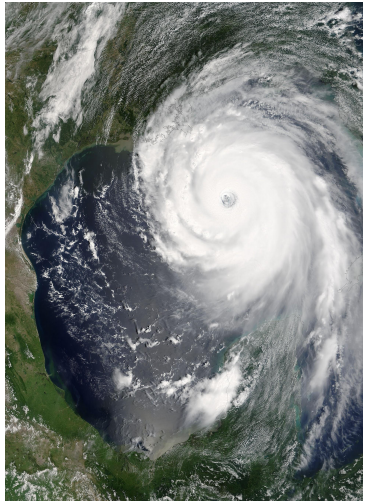
- Recently, detection of the clouds in satellite images is an important issue during analyzing and utilizing these images.
- Cloud being the most uncertain factor in the climate system and has a significant impact on climate change. The most considerable value you get from satellite images is that we can see what's on the ground.
- If there is too much water vapor in front of the satellite imaging sensors, it obscures the Earth's surface, and we might lose all value of the platform. If we have cloudy images, we can't see what we need to see.

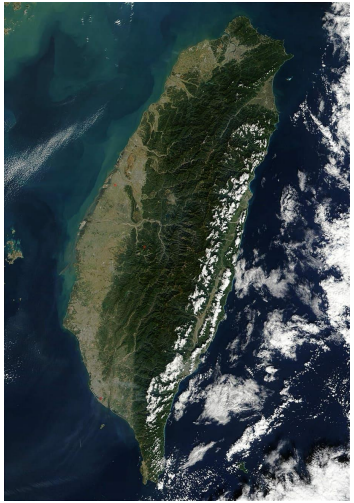
Introduction

Applications of Cloud Detection:

There are several remote sensing based applications where cloud detection plays a crucial role.

- Disaster management (natural disasters such as hurricanes and volcanic eruptions)
- Agricultural crop yield
- Clouds regulate the amount of solar radiation absorbed by a planet and its solar surface irradiance.



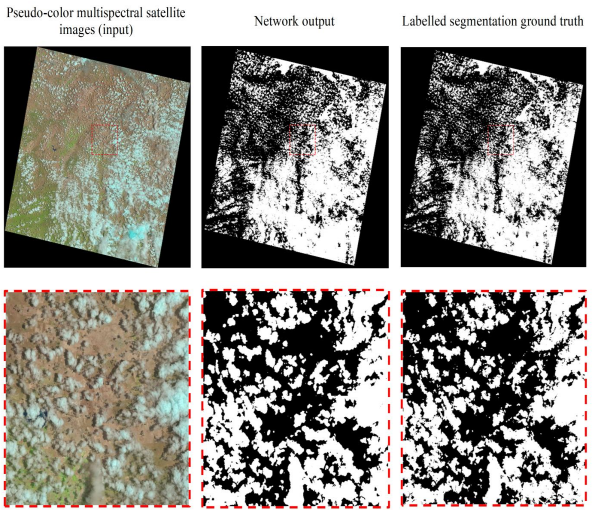


Why Is Cloud Detection Difficult?

Due to various noise disturbances in the remote sensing data and factors such as ice and snow on the ground.

With the rapid development of artificial intelligence technology, deep learning methods have achieved great success in methods such as image processing and classification. Here, we have used the modified U-Net architecture that introduces the attention mechanism for cloud detection.

Understanding the Dataset

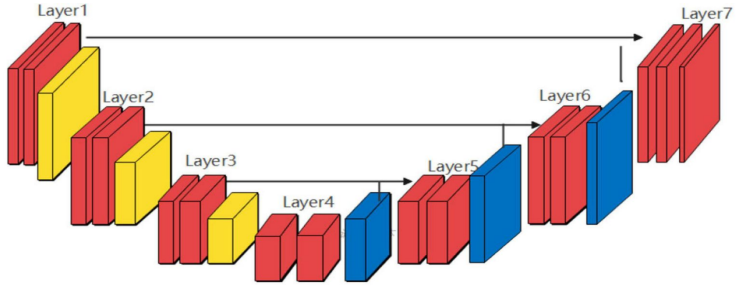


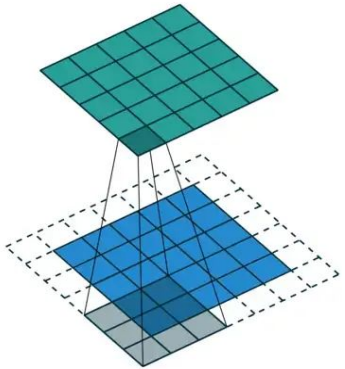
Gathering high-resolution satellite training data from multiple sources is challenging!

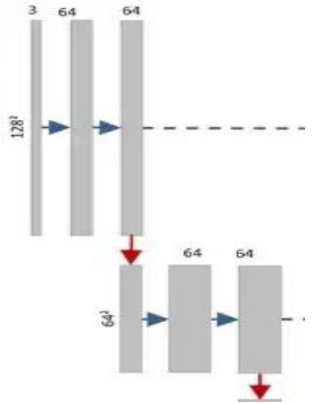
- The data we will be using is a public dataset available at Kaggle called “38-Cloud simple U_Net”, which contains training patches extracted from 57 Landsat 8 scenes.
- The dataset comprises of satellite scenes cropped into 384x384 patches (suitable for deep learning purposes).
- There are 8400 patches for training and 2000 patches for testing, separated into directories for the Red, Green, Blue, and NIR (Near Infrared) bands and an additional directory to store the reference mask (ground truth (gt)).

Model selection

- Semantic segmentation, is an essential task in which we classify each pixel of an image as belonging to a particular class.
- There are many semantic segmentation algorithms such as U-net, Mask R-CNN, and Feature Pyramid Network (FPN).
- The U-Net architecture has a contracting path, which works as a standard CNN model, downsampling the image into several features.
- Then it follows the inverse path, upsampling until the original resolution is obtained.

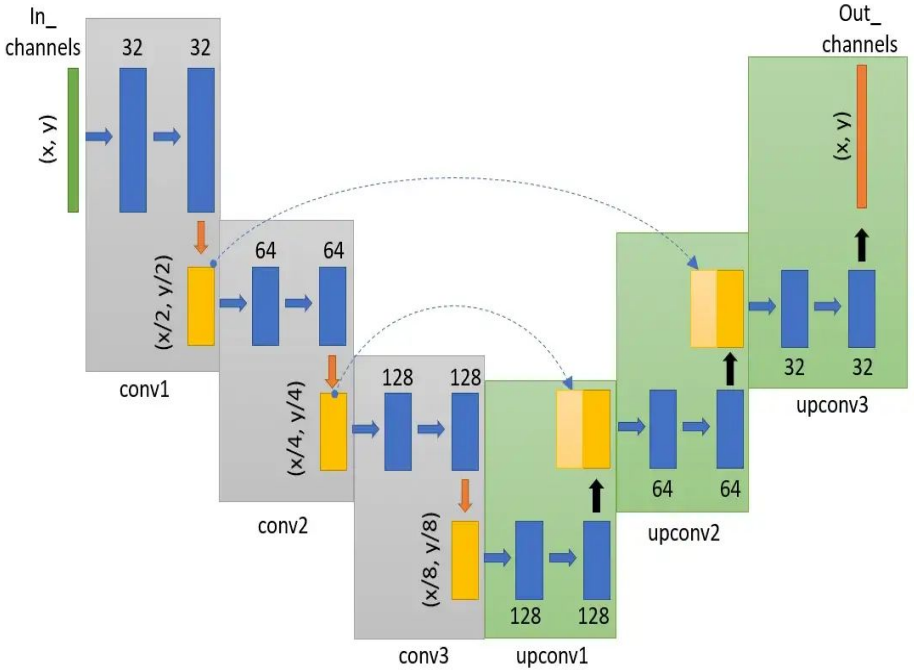






How a U-Net works?

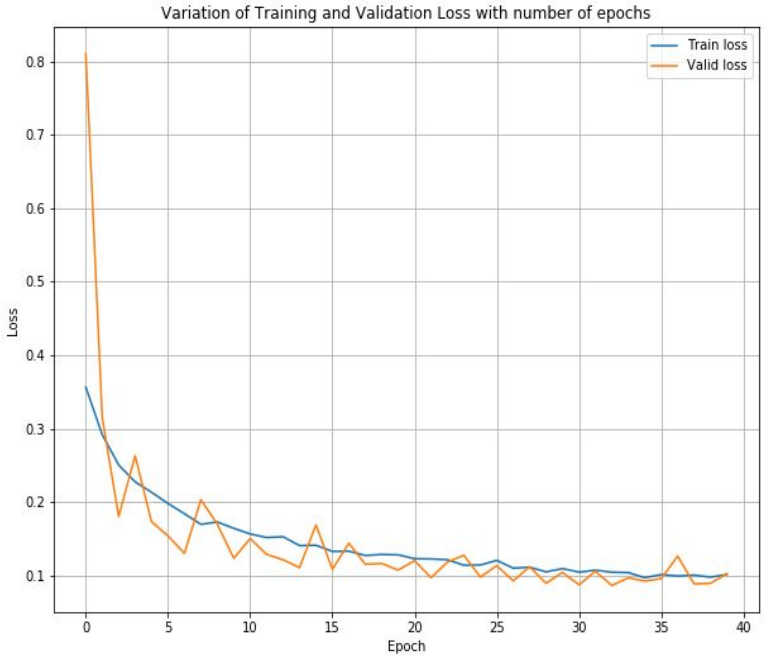
```
UNET(
  (conv1): Sequential(
    (0): Conv2d(4, 32, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 32, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 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()
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (conv3): 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()
    (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
```



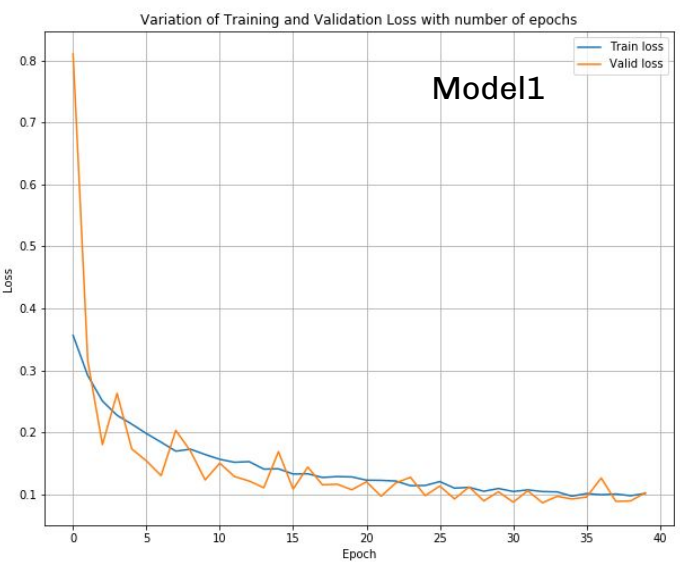
Training the Models

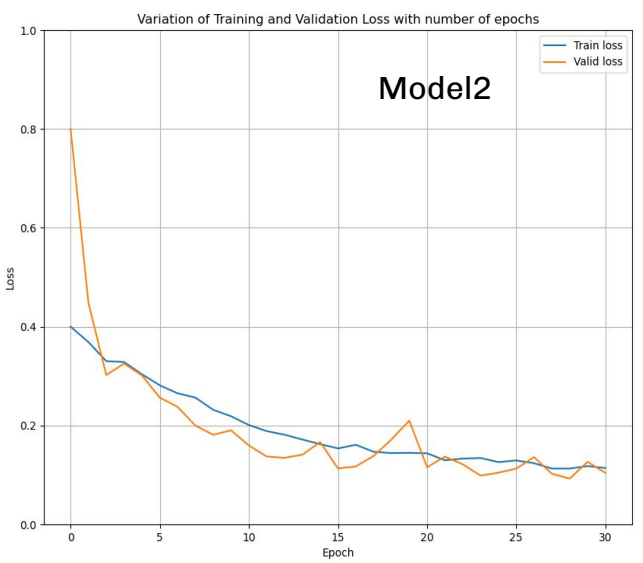
- While training, we have used the **Cross Entropy function** of PyTorch, but for the accuracy we will define our own, that will be basically the number of matched pixels (mask == prediction) divided by the total number of pixels in a batch.
- The optimizer will be **Adam** with a fixed learning rate of 0.01.
- After training it for 40 epochs, we can drastically reduce our training and validation losses.

```
Epoch 39/39
----------
valid Loss: 0.0815 Acc: 0.9679107666015625
----------
Validation loss decreased (0.087083 --> 0.081509).  Saving model ...
Training complete in 126m 53s
```

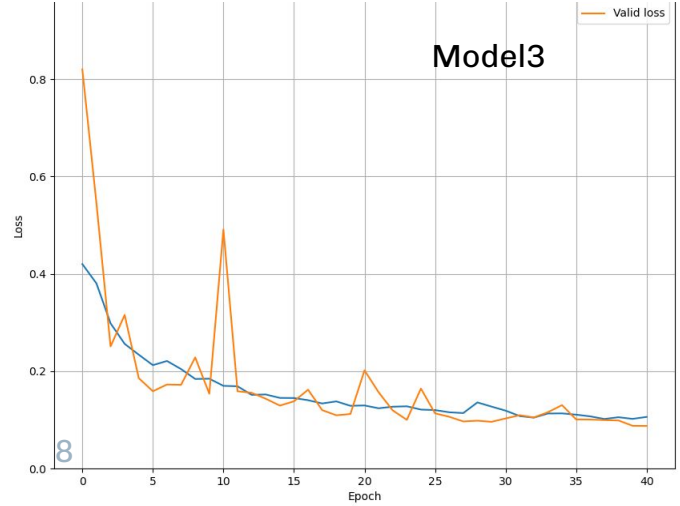


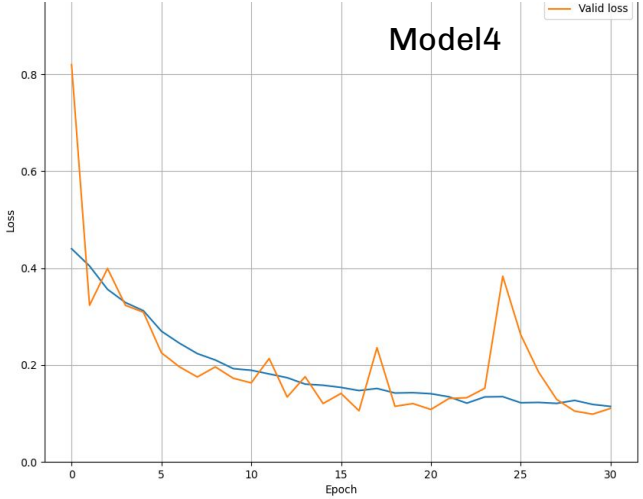
Comparisons of different Models





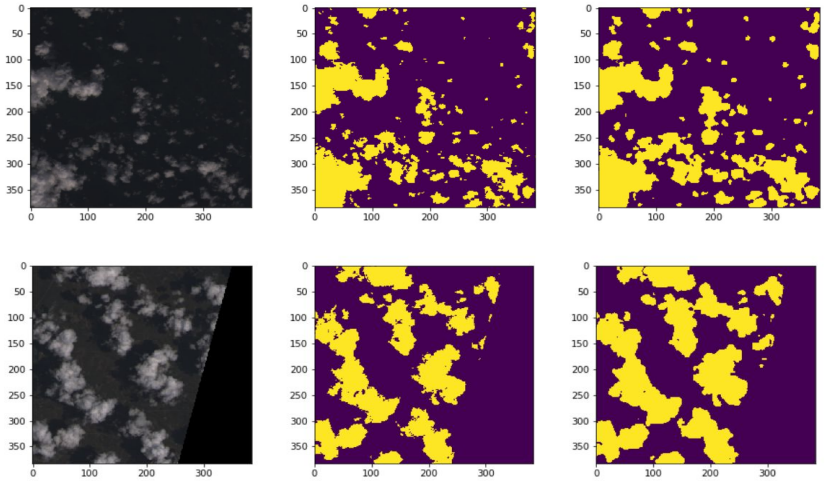
1	Base model with Cross-entropy loss, Adam optimiser, and epochs as 40.
2	Updated model derived from base with increased features in Convolutional layers.
3	Derived from base now we have used SGD instead of ADAM optimiser.
4	Finally we didn't introduced no-skip connections in between layers(No concatenation).



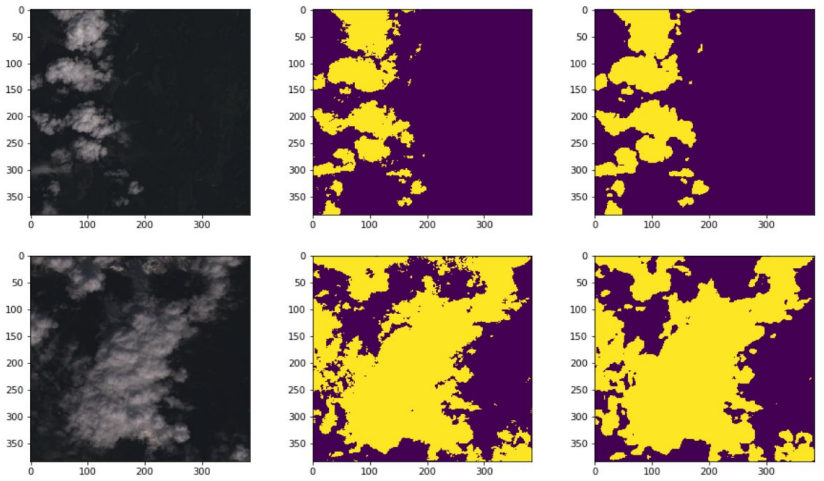


Results:

Model 1



Model 3



Model 2

Model 4

Thank you!

Shankar Ram
M.Tech CAOS

Manya Verma
M.Tech CAOS

Kartik Soni
M.Tech CAOS