Name - Jinal Vyas ASU ID - 1233053785

Model Training and Evaluation for Segmentation of Glacier Images

Here, I am keeping the initial code from part-2, because it includes comparing three models, and i have done in-detailed post analysis to find the best of the three performing models.

For best visualization of geomaps, i recommend viewing this file in google colab because of its support for google earth engine, however i have plotted the final output files and required plot using matplotlib, so it is platform independent only.

So here, for segmentation, I am using UNet as a base model with three different encoders: ResNet34, ResNet50 and EfficientNet-b4 and Dice Loss as the Evaluation Metric.

Explanation of model choice: ResNet34, ResNet50, and EfficientNet-B4 were chosen as they are popular backbone architectures with varying complexities, enabling a comparison of performance based on model depth and efficiency: ResNet34: A relatively lightweight model suitable for smaller datasets with faster training and inference. ResNet50: A deeper model with higher capacity, useful for more complex datasets but prone to overfitting. EfficientNet-B4: A highly efficient model that balances accuracy and computational cost using compound scaling.

Dice Loss was chosen as the evaluation metric because: It directly measures the overlap between the predicted and ground truth segmentation masks. It is particularly effective for imbalanced datasets, ensuring small regions are not overlooked. Dice Loss focuses on pixel-level accuracy, making it ideal for image segmentation tasks where precise boundaries are critical. By optimizing Dice Loss, the model maximizes overlap and minimizes false positives/negatives, ensuring better segmentation performance.

```
# Install the Rasterio library, which is used for reading and writing geospatial raster data
!pip install rasterio
```

```
Collecting rasterio
  Downloading rasterio-1.4.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.1 kB
Collecting affine (from rasterio)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.10/dist-packages (from rasterio) (24.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from rasterio) (2024.8
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.10/dist-packages (from rasterio) (8.1
Collecting cligj>=0.5 (from rasterio)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.10/dist-packages (from rasterio) (1
Collecting click-plugins (from rasterio)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from rasterio) (3.2
Downloading rasterio-1.4.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (22.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.2/22.2 MB 25.1 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Installing collected packages: cligj, click-plugins, affine, rasterio
Successfully installed affine-2.4.0 click-plugins-1.1.1 cligj-0.7.2 rasterio-1.4.3
```

```
# Install the segmentation_models_pytorch library for pretrained segmentation model, U-Net in this c
!pip install segmentation_models_pytorch
```

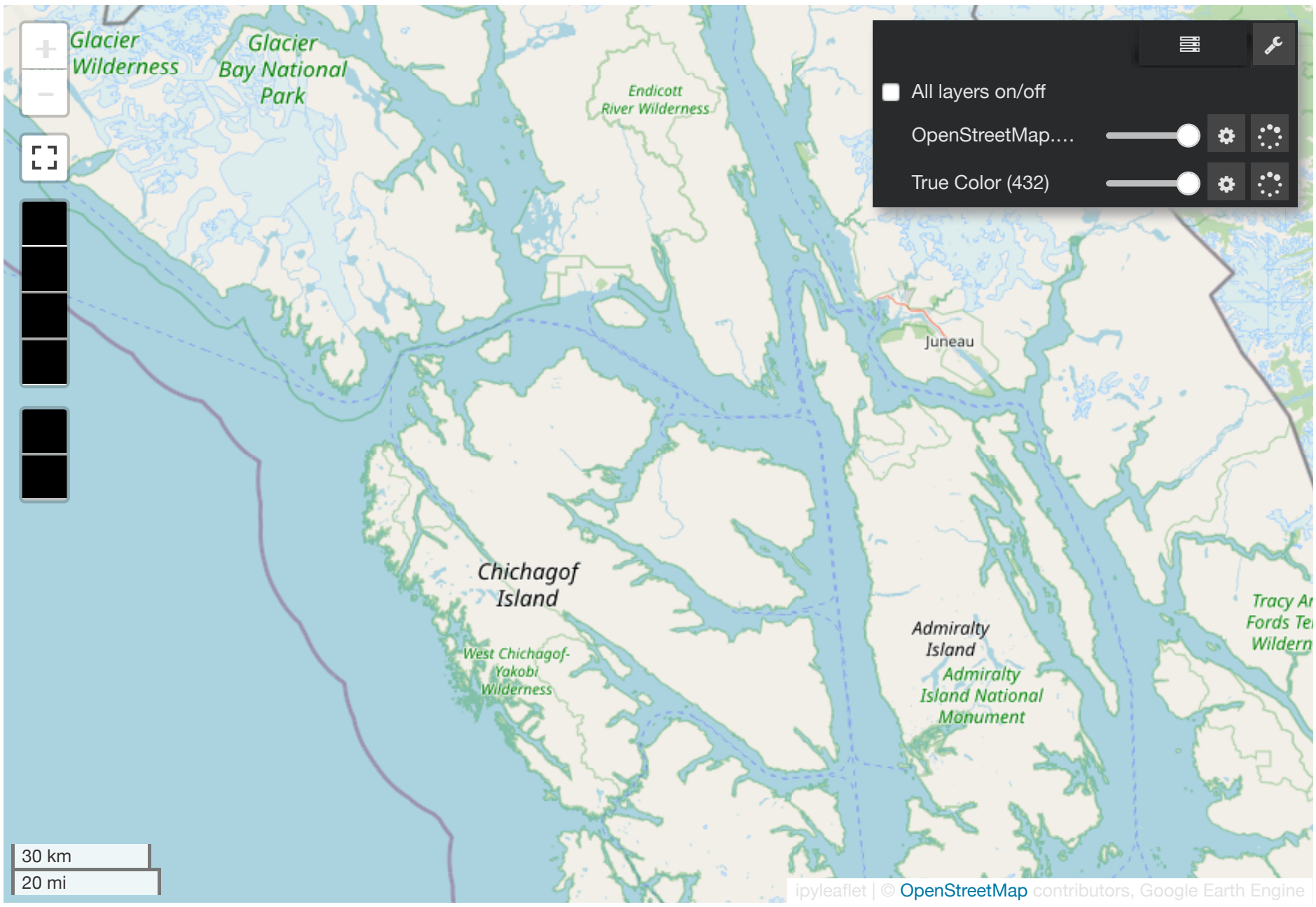
```
Collecting segmentation_models_pytorch
  Downloading segmentation_models_pytorch-0.3.4-py3-none-any.whl.metadata (30 kB)
Collecting efficientnet-pytorch==0.7.1 (from segmentation_models_pytorch)
  Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: huggingface-hub>=0.24.6 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from segmentation_mode
Collecting pretrainedmodels==0.7.4 (from segmentation_models_pytorch)
  Downloading pretrainedmodels-0.7.4.tar.gz (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 4.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from segmentation_models_p
Collecting timm==0.9.7 (from segmentation_models_pytorch)
  Downloading timm-0.9.7-py3-none-any.whl.metadata (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 5.0 MB/s eta 0:00:00
Requirement already satisfied: torchvision>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from segmen
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from segmentation_models_
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from efficientnet-pytorc
Collecting munch (from pretrainedmodels==0.7.4->segmentation_models_pytorch)
  Downloading munch-4.0.0-py2.py3-none-any.whl.metadata (5.9 kB)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from timm==0.9.7->segme
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-packages (from timm==0.9.7->
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingf
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingfa
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision>=0.5.0-
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->efficient
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->efficientne
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch->effi
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy=
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->hu
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from reques
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from reques
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->t
Downloading segmentation_models_pytorch-0.3.4-py3-none-any.whl (109 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 109.5/109.5 kB 9.6 MB/s eta 0:00:00
Downloading timm-0.9.7-py3-none-any.whl (2.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 31.6 MB/s eta 0:00:00
Downloading munch-4.0.0-py2.py3-none-any.whl (9.9 kB)
Building wheels for collected packages: efficientnet-pytorch, pretrainedmodels
  Building wheel for efficientnet-pytorch (setup.py) ... done
  Created wheel for efficientnet-pytorch: filename=efficientnet_pytorch-0.7.1-py3-none-any.whl size=1642
  Stored in directory: /root/.cache/pip/wheels/03/3f/e9/911b1bc46869644912bda90a56bcf7b960f20b5187feea3b
  Building wheel for pretrainedmodels (setup.py) ... done
  Created wheel for pretrainedmodels: filename=pretrainedmodels-0.7.4-py3-none-any.whl size=60944 sha256=
  Stored in directory: /root/.cache/pip/wheels/35/cb/a5/8f534c60142835bfc889f9a482e4a67e0b817032d9c6883b0
Successfully built efficientnet-pytorch pretrainedmodels
Installing collected packages: munch, efficientnet-pytorch, timm, pretrainedmodels, segmentation_models_p
  Attempting uninstall: timm
    Found existing installation: timm 1.0.12
    Uninstalling timm-1.0.12:
      Successfully uninstalled timm-1.0.12
Successfully installed efficientnet-pytorch-0.7.1 munch-4.0.0 pretrainedmodels-0.7.4 segmentation_models_
```

```
# installing required libraries
import ee
import geemap.core as geemap
import torch
import rasterio
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from osgeo import gdal
import seaborn as sns
import time
from tqdm import tqdm
import cv2
from google.colab.patches import cv2_imshow
import os
from PIL import Image
import segmentation_models_pytorch as smp
from sklearn.model_selection import KFold
```

```
# authenticating the access of google earth engine
ee.Authenticate()
ee.Initialize(project='ee-jinalvyasict19')
```

```
# Landsat-8 was launched in february, 2013, hence am considering data of past 10 years starting fro
# I am combining the data of every month into a single image through its median, hence i am generat
# However, due to cloud mask, 38 images were found complelely empty/corrupted, so i dropped them an
# 10% of images among those 82 may have very less data but still I am using it for training and tes
dataset = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2').filterDate(
    '2023-06-01', '2023-12-01'
)
# masking alaska's mendenhall glacier coordinates
alaska_bounds = ee.Geometry.Polygon([[-134.62, 58.42], [-134.62, 58.59], [-134.42, 58.59], [-134.42,

# selecting the appropriate bands required, in the data preprocessing part, i have indetailed expla
# Optical Bands (SR_B3 to SR_B7): Represent visible, near-infrared (NIR), and shortwave infrared (SW
# Thermal Bands (ST_B.*): Capture surface temperature, useful for monitoring heat and environmental
def apply_scale_factors(image):
  global alaska_bounds
  optical_bands = image.select(['SR_B3', 'SR_B4', 'SR_B5', 'SR_B6', 'SR_B7']).multiply(0.0000275).ad
  thermal_bands = image.select('ST_B.*').multiply(0.00341802).add(149.0)
  image_with_bands =  image.addBands(optical_bands, None, True).addBands(
      thermal_bands, None, True
  )
  return image_with_bands.clip(alaska_bounds)

# clouds many times distort the true images, hence it is crucial to apply cloud mask and remove the
def mask_clouds_shadows(image):
  qa_mask = image.select('QA_PIXEL')
  cloud_mask = (qa_mask.bitwiseAnd(1 << 3)).eq(0)
  shadow_mask = (qa_mask.bitwiseAnd(1 << 4)).eq(0)
  mask = cloud_mask.And(shadow_mask)
  return image.updateMask(mask)

dataset = dataset.map(apply_scale_factors)
dataset = dataset.map(mask_clouds_shadows)

visualization = {
    'bands': ['SR_B4', 'SR_B3', 'SR_B2'],
    'min': 0.0,
    'max': 0.3,
}

map = geemap.Map()
map.centerObject(alaska_bounds, 10)
map.add_layer(dataset, visualization, 'True Color (432)')
```

```
43 map
```




```
# applying preprocessing to the images of past ten years i.e. ranging from January, 2014 to Decembe
images = {}

for year in tqdm(range(2014, 2024)):
  for month in tqdm(range(1, 13)):
    if month != 2:
      dataset = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2').filterDate(
        rf'{year}-{month}-01', rf'{year}-{month}-30'
      )
    else:
      dataset = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2').filterDate(
        rf'{year}-{month}-01', rf'{year}-{month}-28'
      )
    dataset = dataset.map(apply_scale_factors)
    dataset = dataset.map(mask_clouds_shadows)
    # the reason I am taking median and not all the images of every month is, the frequency of Lands
    # and also there is a good probability that either of those image may be corrupted due to clouds
    # as median will also be less affacted by the outliers (which may be the NaN values recorded due
    images[f'{year}-{month}'] = dataset.median()
images
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
100%|██████████| 12/12 [00:00<00:00, 216.29it/s]

100%|██████████| 12/12 [00:00<00:00, 257.08it/s]
 20%|██        | 2/10 [00:00<00:00, 16.57it/s]
100%|██████████| 12/12 [00:00<00:00, 129.53it/s]

  0%|          | 0/12 [00:00<?, ?it/s]
100%|██████████| 12/12 [00:00<00:00, 99.21it/s]
 40%|████      | 4/10 [00:00<00:00, 10.70it/s]
  0%|          | 0/12 [00:00<?, ?it/s]
100%|██████████| 12/12 [00:00<00:00, 55.89it/s]

  0%|          | 0/12 [00:00<?, ?it/s]
100%|██████████| 12/12 [00:00<00:00, 74.43it/s]
 60%|██████    | 6/10 [00:00<00:00,  6.84it/s]
  0%|          | 0/12 [00:00<?, ?it/s]
```

```
100%|██████████| 12/12 [00:00<00:00, 55.33it/s]
 70%|███████   | 7/10 [00:01<00:00,  5.86it/s]
  0%|          | 0/12 [00:00<?, ?it/s]
100%|██████████| 12/12 [00:00<00:00, 86.09it/s]
 80%|████████  | 8/10 [00:01<00:00,  5.82it/s]
100%|██████████| 12/12 [00:00<00:00, 125.52it/s]
 90%|█████████ | 9/10 [00:01<00:00,  6.44it/s]
  0%|          | 0/12 [00:00<?, ?it/s]
100%|██████████| 12/12 [00:00<00:00, 53.14it/s]
100%|██████████| 10/10 [00:01<00:00,  6.48it/s]
{'2014-1': <ee.image.Image at 0x7d7b9dc72fe0>,
 '2014-2': <ee.image.Image at 0x7d7b9d70c040>,
 '2014-3': <ee.image.Image at 0x7d7b9dcab130>,
 '2014-4': <ee.image.Image at 0x7d7b9dcaab60>,
 '2014-5': <ee.image.Image at 0x7d7b9dd15c00>,
 '2014-6': <ee.image.Image at 0x7d7b9dd4b070>,
 '2014-7': <ee.image.Image at 0x7d7b9d2352d0>,
 '2014-8': <ee.image.Image at 0x7d7b9d236800>,
 '2014-9': <ee.image.Image at 0x7d7b9d2342b0>,
 '2014-10': <ee.image.Image at 0x7d7b9d29c640>,
 '2014-11': <ee.image.Image at 0x7d7b9d29d6f0>,
 '2014-12': <ee.image.Image at 0x7d7b9d29e6e0>,
 '2015-1': <ee.image.Image at 0x7d7b9dd04940>,
 '2015-2': <ee.image.Image at 0x7d7b9dca9a50>,
 '2015-3': <ee.image.Image at 0x7d7b9dd061d0>,
 '2015-4': <ee.image.Image at 0x7d7b9dc62770>,
 '2015-5': <ee.image.Image at 0x7d7b9dc62260>,
 '2015-6': <ee.image.Image at 0x7d7b9dc63e20>,
 '2015-7': <ee.image.Image at 0x7d7b9de7ea40>,
 '2015-8': <ee.image.Image at 0x7d7b9d724640>,
 '2015-9': <ee.image.Image at 0x7d7b9d726e60>,
 '2015-10': <ee.image.Image at 0x7d7b9d726140>,
 '2015-11': <ee.image.Image at 0x7d7b9d724b20>,
 '2015-12': <ee.image.Image at 0x7d7b9c49c520>,
 '2016-1': <ee.image.Image at 0x7d7b9c49d600>,
 '2016-2': <ee.image.Image at 0x7d7b9c49e5f0>,
 '2016-3': <ee.image.Image at 0x7d7b9c49f310>,
 '2016-4': <ee.image.Image at 0x7d7b9c4ac220>,
 '2016-5': <ee.image.Image at 0x7d7b9c4ad210>,
 '2016-6': <ee.image.Image at 0x7d7b9c4ae200>,
 '2016-7': <ee.image.Image at 0x7d7b9c4af130>,
 '2016-8': <ee.image.Image at 0x7d7b9c4afee0>,
 '2016-9': <ee.image.Image at 0x7d7b9c4b8f10>,
 '2016-10': <ee.image.Image at 0x7d7b9c4b9f00>,
 '2016-11': <ee.image.Image at 0x7d7b9c4babf0>,
 '2016-12': <ee.image.Image at 0x7d7b9c4bbbb0>,
 '2017-1': <ee.image.Image at 0x7d7b9c4c8cd0>,
 '2017-2': <ee.image.Image at 0x7d7b9c4c9a80>,
 '2017-3': <ee.image.Image at 0x7d7b9c4cabc0>,
 '2017-4': <ee.image.Image at 0x7d7b9c4cb8b0>,
 '2017-5': <ee.image.Image at 0x7d7b9c4d8850>,
 '2017-6': <ee.image.Image at 0x7d7b9c4d9840>,
 '2017-7': <ee.image.Image at 0x7d7b9c4da830>,
 '2017-8': <ee.image.Image at 0x7d7b9c4db820>,
 '2017-9': <ee.image.Image at 0x7d7b9c4e41f0>,
 '2017-10': <ee.image.Image at 0x7d7b9c4e5480>,
 '2017-11': <ee.image.Image at 0x7d7b9c4e6470>,
 '2017-12': <ee.image.Image at 0x7d7b9c4e7460>,
 '2018-1': <ee.image.Image at 0x7d7b9c4f4040>,
 '2018-2': <ee.image.Image at 0x7d7b9c4f5360>,
 '2018-3': <ee.image.Image at 0x7d7b9c4f6320>,
 '2018-4': <ee.image.Image at 0x7d7b9c4f7310>,
 '2018-5': <ee.image.Image at 0x7d7b9c504340>,
 '2018-6': <ee.image.Image at 0x7d7b9c504c10>,
 '2018-7': <ee.image.Image at 0x7d7b9c505f60>,
 '2018-8': <ee.image.Image at 0x7d7b9c506f50>,
 '2018-9': <ee.image.Image at 0x7d7b9c507f40>,
 '2018-10': <ee.image.Image at 0x7d7b9c518bb0>,
 '2018-11': <ee.image.Image at 0x7d7b9c519c60>,
 '2018-12': <ee.image.Image at 0x7d7b9c51ac50>,
 '2019-1': <ee.image.Image at 0x7d7b9c51ae90>,
 '2019-2': <ee.image.Image at 0x7d7b9c528d60>,
 '2019-3': <ee.image.Image at 0x7d7b9c529990>,
 '2019-4': <ee.image.Image at 0x7d7b9c52a980>,
 '2019-5': <ee.image.Image at 0x7d7b9c52b970>,
 '2019-6': <ee.image.Image at 0x7d7b9c5349a0>,
 '2019-7': <ee.image.Image at 0x7d7b9c535990>,
 '2019-8': <ee.image.Image at 0x7d7b9c5366b0>,
 '2019-9': <ee.image.Image at 0x7d7b9c537790>,
```

```
'2019-10': <ee.image.Image at 0x7d7b9c54c640>,
'2019-11': <ee.image.Image at 0x7d7b9c54d630>,
'2019-12': <ee.image.Image at 0x7d7b9c54e410>,
'2020-1': <ee.image.Image at 0x7d7b9c554220>,
'2020-2': <ee.image.Image at 0x7d7b9c54f7f0>,
'2020-3': <ee.image.Image at 0x7d7b9c5553c0>,
'2020-4': <ee.image.Image at 0x7d7b9c5560b0>,
'2020-5': <ee.image.Image at 0x7d7b9c557130>,
'2020-6': <ee.image.Image at 0x7d7b9c568160>,
'2020-7': <ee.image.Image at 0x7d7b9c5691b0>,
'2020-8': <ee.image.Image at 0x7d7b9c56a4d0>,
'2020-9': <ee.image.Image at 0x7d7b9c56b1f0>,
'2020-10': <ee.image.Image at 0x7d7b9c5740d0>,
'2020-11': <ee.image.Image at 0x7d7b9c5750f0>,
'2020-12': <ee.image.Image at 0x7d7b9c5760e0>,
'2021-1': <ee.image.Image at 0x7d7b9c576dd0>,
'2021-2': <ee.image.Image at 0x7d7b9c577dc0>,
'2021-3': <ee.image.Image at 0x7d7b9c38cdf0>,
'2021-4': <ee.image.Image at 0x7d7b9c38dde0>,
'2021-5': <ee.image.Image at 0x7d7b9c38edd0>,
'2021-6': <ee.image.Image at 0x7d7b9c38f550>,
'2021-7': <ee.image.Image at 0x7d7b9c398a60>,
'2021-8': <ee.image.Image at 0x7d7b9c399a50>,
'2021-9': <ee.image.Image at 0x7d7b9c39aa40>,
'2021-10': <ee.image.Image at 0x7d7b9c39ba30>,
'2021-11': <ee.image.Image at 0x7d7b9c3a8760>,
'2021-12': <ee.image.Image at 0x7d7b9c3a9750>,
'2022-1': <ee.image.Image at 0x7d7b9c3aa890>,
'2022-2': <ee.image.Image at 0x7d7b9c3ab6d0>,
'2022-3': <ee.image.Image at 0x7d7b9c3bc610>,
'2022-4': <ee.image.Image at 0x7d7b9c3bd600>,
'2022-5': <ee.image.Image at 0x7d7b9c3be5f0>,
'2022-6': <ee.image.Image at 0x7d7b9c3bf640>,
'2022-7': <ee.image.Image at 0x7d7b9c3c4250>,
'2022-8': <ee.image.Image at 0x7d7b9c3c5300>,
'2022-9': <ee.image.Image at 0x7d7b9c3c62f0>,
'2022-10': <ee.image.Image at 0x7d7b9c3c72e0>,
'2022-11': <ee.image.Image at 0x7d7b9c3bffa0>,
'2022-12': <ee.image.Image at 0x7d7b9c3dd000>,
'2023-1': <ee.image.Image at 0x7d7b9c3de050>,
'2023-2': <ee.image.Image at 0x7d7b9c3df0d0>,
'2023-3': <ee.image.Image at 0x7d7b9c3df970>,
'2023-4': <ee.image.Image at 0x7d7b9c3eceb0>,
'2023-5': <ee.image.Image at 0x7d7b9c3edea0>,
'2023-6': <ee.image.Image at 0x7d7b9c3eee90>,
'2023-7': <ee.image.Image at 0x7d7b9c3eff40>,
'2023-8': <ee.image.Image at 0x7d7b9c3f4970>,
```

```python
# mounting google drive, for easy storage and access of data
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```python
# Creating tasks for downloading the images to a folder named, 'Alasks Dataset', in the drive, it is
for date, image in tqdm(images.items()):
    export_task = ee.batch.Export.image.toDrive(
        image=image,
        description=f'image_{date}',
        folder='Alaska_Dataset',
        region=alaska_bounds,
        scale=30,
        fileFormat = 'GeoTIFF',
        maxPixels=1e13
    )
    export_task.start()
    while export_task.active():
      pass
```

```
100%|██████████| 120/120 [1:16:02<00:00, 38.02s/it]
```

```
# Here, using the tiff files derived from the above task, I am extracting the bands 3 to 7, replacin
# Then, using bands 3, 4 and 5, I am plotting RGB images, and also NDSI ( Normalized Difference Snow
# Here, as visually representable, RGB images better distinguish the glacier from NDSI images, hence

# I am also applying thresholding of rgb_image.max() == 1, such that months producing empty tif file
for files in images.keys():
  file_path = f"/content/drive/My Drive/Alaska_Dataset/image_{files}.tif"
  try:
      with rasterio.open(file_path) as src:
          band_3 = src.read(3)
          band_4 = src.read(4)
          band_5 = src.read(5)
          band_6 = src.read(6)
          band_7 = src.read(7)

      band_3 = np.nan_to_num(band_3, nan=0)
      band_4 = np.nan_to_num(band_4, nan=0)
      band_5 = np.nan_to_num(band_5, nan=0)
      band_6 = np.nan_to_num(band_6, nan=0)
      band_7 = np.nan_to_num(band_7, nan=0)

      band_3_flat = band_3.flatten()
      band_4_flat = band_4.flatten()
      band_5_flat = band_5.flatten()
      band_6_flat = band_6.flatten()
      band_7_flat = band_7.flatten()
      band_5_norm = band_5 / band_5.max()
      band_4_norm = band_4 / band_4.max()
      band_3_norm = band_3 / band_3.max()

      rgb_image = np.stack((band_5_norm, band_4_norm, band_3_norm), axis=-1)
      NDSI = (band_3 - band_6) / (band_3 + band_6)
      if rgb_image.max() == 1:
        plt.figure(figsize=(10, 10))
        plt.imshow(rgb_image)
        plt.title("False Color Composite (NIR, Red, Green)")
        plt.axis('off')
        plt.savefig(f"/content/drive/My Drive/Alaska_Dataset/image_{files}.png")
        plt.show()
        plt.figure(figsize=(7, 7))
        plt.imshow(NDSI, cmap='Spectral')
        plt.title('Normalized Difference Snow Index (NDSI)')
        plt.colorbar()
        plt.axis('off')
        plt.clim(-5, 1)
        plt.savefig(f"/content/drive/My Drive/Alaska_Dataset/image_{files}_NDSI.png")
        plt.show()
  except Exception as e:
    print(e)
```

```
# I am creating binary masks using the RGB images and OpenCV, first I am converting them to grayscal
# and the non-glacier region, including land and vegetation will be marked as black. I am also cropp
# as it can cause discrepancy in the segmentation.
for files in images.keys():
  file_path = rf"/content/drive/My Drive/Alaska_Dataset/image_{files}.png"
  if not os.path.exists(file_path):
    print(f'image_{files}.png was not saved, because it contained no data!')
    continue
  file_path = rf"/content/drive/My Drive/Alaska_Dataset/image_{files}.png"
  img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
  # print(img)
  _, mask = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
  # cv2.imwrite(f"/content/drive/My Drive/Alaska_Dataset/mask_{files}.png", mask)
  x, y, w, h = 55, 82, 400, 340
  cropped_mask = mask[y:y+h, x:x+w]
  cv2_imshow(cropped_mask)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
  cv2.imwrite(f"/content/drive/My Drive/Alaska_Dataset/mask_{files}.png", cropped_mask)
```

```
# I am creating a custom dataset such that it applies transformation to the images and masks, here i
# modified, for example, larger models will function better on 512*512 images, and rather overfit on
# hence 224*224 seems like an appropriate choice.
class SegmentationDataset(Dataset):
  def __init__(self, image_path, mask_path, transformr=None):
    self.image_paths = image_path
    self.mask_paths = mask_path
    self.transform = transforms.Compose([
        transforms.Resize((224, 224)),  # Resize the image
        transforms.ToTensor(),                        # Convert to tensor
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])  # Normalize using ImageNet stats
    ])
    self.mask_transform = transforms.Compose([
        transforms.Resize((224, 224)),  # Resize the image
        transforms.ToTensor()
    ])
  def __len__(self):
    return len(self.image_paths)
  def __getitem__(self, idx):
    image = Image.open(self.image_paths[idx]).convert("RGB")
    x, y, w, h = 55, 82, 400, 340
    cropped_image = self.transform(image.crop((x, y, x + w, y + h)))
    mask = self.mask_transform(Image.open(self.mask_paths[idx]))
    # print(image.shape, mask.shape)
    return cropped_image, mask
```

```
# I am appending all the rgb image paths and mask paths to two variables, which I will later use fo
image_paths = []
mask_paths = []
for files in images.keys():
  image_path = rf"/content/drive/My Drive/Alaska_Dataset/image_{files}.png"
  if not os.path.exists(image_path):
    print(f'image_{files}.png was not saved, because it contained no data!')
    continue
  mask_path = rf"/content/drive/My Drive/Alaska_Dataset/mask_{files}.png"
  image_paths.append(image_path)
  mask_paths.append(mask_path)
image_paths, mask_paths
```

```
image_2014-1.png was not saved, because it contained no data!
image_2014-9.png was not saved, because it contained no data!
image_2014-12.png was not saved, because it contained no data!
image_2015-1.png was not saved, because it contained no data!
image_2015-2.png was not saved, because it contained no data!
image_2015-11.png was not saved, because it contained no data!
image_2015-12.png was not saved, because it contained no data!
image_2016-1.png was not saved, because it contained no data!
image_2016-8.png was not saved, because it contained no data!
```

```
image_2016-11.png was not saved, because it contained no data!
image_2016-12.png was not saved, because it contained no data!
image_2017-1.png was not saved, because it contained no data!
image_2017-10.png was not saved, because it contained no data!
image_2017-12.png was not saved, because it contained no data!
image_2018-1.png was not saved, because it contained no data!
image_2018-8.png was not saved, because it contained no data!
image_2018-11.png was not saved, because it contained no data!
image_2018-12.png was not saved, because it contained no data!
image_2019-1.png was not saved, because it contained no data!
image_2019-11.png was not saved, because it contained no data!
image_2019-12.png was not saved, because it contained no data!
image_2020-1.png was not saved, because it contained no data!
image_2020-6.png was not saved, because it contained no data!
image_2020-8.png was not saved, because it contained no data!
image_2020-11.png was not saved, because it contained no data!
image_2020-12.png was not saved, because it contained no data!
image_2021-1.png was not saved, because it contained no data!
image_2021-11.png was not saved, because it contained no data!
image_2021-12.png was not saved, because it contained no data!
image_2022-1.png was not saved, because it contained no data!
image_2022-2.png was not saved, because it contained no data!
image_2022-7.png was not saved, because it contained no data!
image_2022-10.png was not saved, because it contained no data!
image_2022-12.png was not saved, because it contained no data!
image_2023-1.png was not saved, because it contained no data!
image_2023-4.png was not saved, because it contained no data!
image_2023-11.png was not saved, because it contained no data!
image_2023-12.png was not saved, because it contained no data!
(['/content/drive/My Drive/Alaska_Dataset/image_2014-2.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-3.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-4.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-5.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-6.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-7.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-8.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-10.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-11.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-3.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-4.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-5.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-6.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-7.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-8.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-9.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2015-10.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-2.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-3.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-4.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-5.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-6.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-7.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-9.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2016-10.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-2.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-3.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-4.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-5.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-6.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-7.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-8.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-9.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2017-11.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-2.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-3.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-4.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-5.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-6.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-7.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-9.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2018-10.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-2.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-3.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-4.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-5.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-6.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-7.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-8.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2019-9.png',
```

```
 '/content/drive/My Drive/Alaska_Dataset/image_2019-10.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2020-2.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2020-3.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2020-4.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2020-5.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2020-7.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2020-9.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2020-10.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-2.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-3.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-4.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-5.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-6.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-7.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-8.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-9.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2021-10.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2022-3.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2022-4.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2022-5.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2022-6.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2022-8.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2022-9.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2022-11.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-2.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-3.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-5.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-6.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-7.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-8.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-9.png',
 '/content/drive/My Drive/Alaska_Dataset/image_2023-10.png'],
['/content/drive/My Drive/Alaska_Dataset/mask_2014-2.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-3.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-4.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-5.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-6.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-7.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-8.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-10.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2014-11.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-3.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-4.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-5.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-6.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-7.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-8.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-9.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2015-10.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-2.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-3.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-4.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-5.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-6.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-7.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-9.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2016-10.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-2.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-3.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-4.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-5.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-6.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-7.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-8.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-9.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2017-11.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-2.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-3.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-4.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-5.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-6.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-7.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-9.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2018-10.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2019-2.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2019-3.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2019-4.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2019-5.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2019-6.png',
 '/content/drive/My Drive/Alaska_Dataset/mask_2019-7.png',
```

```
        '/content/drive/My Drive/Alaska_Dataset/mask_2019-8.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2019-9.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2019-10.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2020-2.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2020-3.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2020-4.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2020-5.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2020-7.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2020-9.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2020-10.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-2.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-3.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-4.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-5.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-6.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-7.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-8.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-9.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2021-10.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2022-3.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2022-4.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2022-5.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2022-6.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2022-8.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2022-9.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2022-11.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2023-2.png',
        '/content/drive/My Drive/Alaska_Dataset/mask_2023-3.png',
```

```
# as explained earlier, these arer total number of images I got after filtering cloud mask, and take
len(image_paths)
```

```
82
```

```
# defining device type, here I am using T4 GPU provided by google colab.
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cuda')
```

```
# defining the train function for single epoch, which returns the total loss per epoch i.e. the loss
def train(model, train_loader, optimizer, loss, device):
  model.train()
  epoch_loss = 0
  for images, masks in train_loader:
    images = images.to(device)
    masks = masks.to(device)
    outputs = model(images)
    # print(outputs.shape, masks.shape)
    batch_loss = loss(outputs, masks)
    optimizer.zero_grad()
    batch_loss.backward()
    optimizer.step()
    epoch_loss += batch_loss.item()
  return epoch_loss/len(train_loader)
```

```
# defining the testing function, which return the total testing loss per epoch, averaged over the ba
def test(model, test_loader, loss, device):
  model.eval()
  test_loss = 0
  with torch.no_grad():
    for images, masks in test_loader:
      images = images.to(device)
      masks = masks.to(device)
      outputs = model(images)
      batch_loss = loss(outputs, masks)
      test_loss += batch_loss.item()
  return test_loss/len(test_loader)
```

```
# this function is used to plot all the images in the test loader. there are three images being plot
def test_image_plot(test_loader, model):
  for images, masks in test_loader:
        images = images.to(device)
        masks = masks.to(device)
        outputs = model(images)
        pred_masks = (outputs > 0.5).float()
        for image, mask, pred_mask in zip(images, masks, pred_masks):
          fig, axs = plt.subplots(1, 3, figsize=(15, 5))
          axs[0].imshow(image.cpu().numpy().transpose(1, 2, 0))  # Convert tensor to numpy (C, H, W)
          axs[0].set_title("Original Image")
          axs[1].imshow(mask.cpu().numpy().squeeze(), cmap='gray')  # Mask is single channel, squeez
          axs[1].set_title("Ground Truth Mask")
          axs[1].axis('off')
          axs[2].imshow(pred_mask.cpu().numpy().squeeze(), cmap='gray')  # Predicted mask (squeeze t
          axs[2].set_title("Predicted Mask")
          axs[2].axis('off')
          plt.show()
```

```
# sorting the images and masks as per date and time, to make sure the indexes of both the lists are
image_paths.sort()
mask_paths.sort()
image_paths[:5], mask_paths[:5]
```

```
(['/content/drive/My Drive/Alaska_Dataset/image_2014-10.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-11.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-2.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-3.png',
  '/content/drive/My Drive/Alaska_Dataset/image_2014-4.png'],
 ['/content/drive/My Drive/Alaska_Dataset/mask_2014-10.png',
  '/content/drive/My Drive/Alaska_Dataset/mask_2014-11.png',
  '/content/drive/My Drive/Alaska_Dataset/mask_2014-2.png',
  '/content/drive/My Drive/Alaska_Dataset/mask_2014-3.png',
  '/content/drive/My Drive/Alaska_Dataset/mask_2014-4.png'])
```

```
# For testing the model, I am using 20% of the data, i.e. I will divide these indices into 80-20 sp
# However, keeping one image separate to compare segmentation output mask produced the best wieghts

indices = list(range(len(image_paths[:81])))
indices
```

```
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
```

```
14,
15,
16,
17,
18,
19,
20,
21,
22,
23,
24,
25,
26,
27,
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
```

```
# Initialize K-Fold Cross-Validation with 5 splits
# shuffle=True ensures the data is randomly shuffled before splitting
# random_state=42 ensures reproducibility of the splits
k_folds = 5
kf = KFold(n_splits=k_folds, shuffle=True, random_state=42)
kf
```

```
KFold(n_splits=5, random_state=42, shuffle=True)
```

UNet with Encoder ResNet34 - Training and Evaluation

```
# here i will be iterating over 5 folds and in each fold, i am dividing the data into 80-20 split i
# Hence as visible, there are 65-66 training images every iteration and 16-17 test images.
# I am globally defining one variable called model_best_loss, which will track when the TEST loss fu
# the best model obtained.
# I am using resnet34 as my encoder here, with pretrained weights.
k_folds_results = []
resnet34_best_test_loss = 10000
# As explained earlier in the initial cells, I am using Dice Loss as my evaluation metric as it min
loss = smp.losses.DiceLoss(mode='binary')
for fold, (train_index, test_index) in enumerate(kf.split(indices)):
  train_image_paths = [image_paths[i] for i in train_index]
  train_mask_paths = [mask_paths[i] for i in train_index]
  test_image_paths = [image_paths[i] for i in test_index]
  test_mask_paths = [mask_paths[i] for i in test_index]
  print(len(train_index), len(test_index))
  train_dataset = SegmentationDataset(train_image_paths, train_mask_paths)
  test_dataset = SegmentationDataset(test_image_paths, test_mask_paths)
  train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
  test_loader = DataLoader(test_dataset, batch_size=8, shuffle=True)
  model = smp.Unet(
    encoder_name="resnet34",  #encoder defined
    encoder_weights="imagenet",  #imagenet pretrained weights are used
    in_channels=3, #3 classes as input images are RGB
    classes=1 #as output is supposed to be a binary mask, hence 1 output classes only.
  )
  model = model.to(device)
  # here I am using the classic Adam as an optimizer with learning rate 1e-4, decreasing it from he
  # Adam optimizer was chosen because:
  # - It combines the benefits of Momentum and RMSprop, adapting the learning rate for each paramete
  # - It is computationally efficient and works well for large datasets and high-dimensional proble
  # - Adam performs well with minimal hyperparameter tuning, making it a reliable choice for most ta
  # - Its adaptive nature ensures faster convergence and robustness to noisy gradient
  # And the model didn't overfit with the given hyperparameters, hence i didn't provide weight deca
  optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
  for epoch in tqdm(range(30)):
    train_loss = train(model, train_loader, optimizer, loss, device)
    test_loss = test(model, test_loader, loss, device)
    if test_loss < resnet34_best_test_loss:
        resnet34_best_test_loss = test_loss
        torch.save(model.state_dict(), f"Best_model_resnet34.pth")
        print('\nBest Model Saved with Test Loss: ', resnet34_best_test_loss)
        best_test_loader = test_loader
  k_folds_results.append(test_loss)
print("\nCross Validation Results: ", k_folds_results)
print("\nAverage_loss: ", sum(k_folds_results)/len(k_folds_results))
```

```
64 17
Downloading: "https://download.pytorch.org/models/resnet34-333f7ec4.pth" to /root/.cache/torch/hub/checkp
100%|██████████| 83.3M/83.3M [00:00<00:00, 178MB/s]
  3%|▎         | 1/30 [01:05<31:40, 65.53s/it]
Best Model Saved with Test Loss:  0.43397557735443115
  7%|▋         | 2/30 [01:08<13:26, 28.82s/it]
Best Model Saved with Test Loss:  0.4210205078125
 10%|█         | 3/30 [01:12<07:49, 17.38s/it]
Best Model Saved with Test Loss:  0.34245532751083374
 13%|█▎        | 4/30 [01:16<05:15, 12.13s/it]
Best Model Saved with Test Loss:  0.20578938722610474
 17%|█▋        | 5/30 [01:19<03:38,  8.72s/it]
Best Model Saved with Test Loss:  0.1909628709157308
 23%|██▎       | 7/30 [01:24<02:02,  5.31s/it]
Best Model Saved with Test Loss:  0.1621051033337911
 30%|███       | 9/30 [01:29<01:18,  3.75s/it]
Best Model Saved with Test Loss:  0.14238115151723227
 40%|████      | 12/30 [01:36<00:53,  2.95s/it]
Best Model Saved with Test Loss:  0.1290393869082133
 47%|████▋     | 14/30 [01:41<00:42,  2.69s/it]
Best Model Saved with Test Loss:  0.12479044993718465
 50%|█████     | 15/30 [01:44<00:38,  2.59s/it]
Best Model Saved with Test Loss:  0.11434956391652425
```

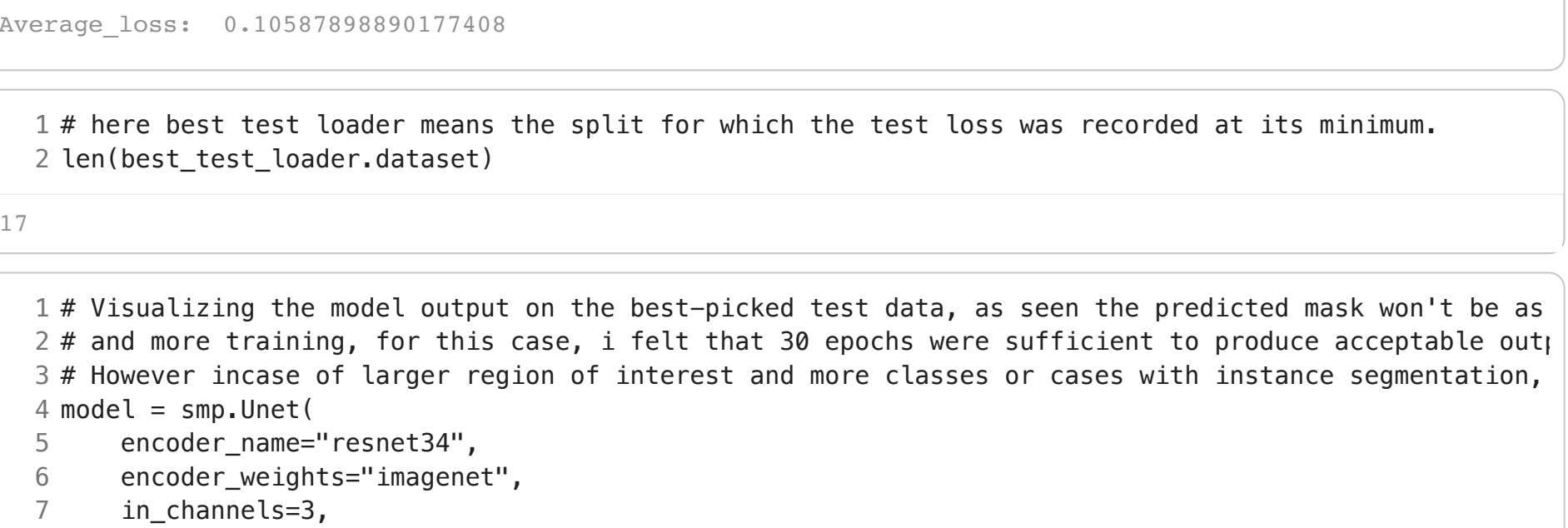
```
 53%|█████     | 16/30 [01:47<00:37,  2.71s/it]
Best Model Saved with Test Loss:  0.11301668485005696
 57%|██████    | 17/30 [01:49<00:35,  2.72s/it]
Best Model Saved with Test Loss:  0.10837630430857341
 60%|██████    | 18/30 [01:52<00:32,  2.71s/it]
Best Model Saved with Test Loss:  0.10548104842503865
 63%|███████   | 19/30 [01:55<00:29,  2.73s/it]
Best Model Saved with Test Loss:  0.09955310821533203
 73%|████████  | 22/30 [02:02<00:21,  2.65s/it]
Best Model Saved with Test Loss:  0.0928419828414917
 80%|████████  | 24/30 [02:07<00:15,  2.54s/it]
Best Model Saved with Test Loss:  0.08815499146779378
 87%|█████████ | 26/30 [02:12<00:09,  2.49s/it]
Best Model Saved with Test Loss:  0.08766223986943562
 90%|█████████ | 27/30 [02:15<00:08,  2.69s/it]
Best Model Saved with Test Loss:  0.08430798848470052
100%|██████████| 30/30 [02:24<00:00,  4.80s/it]
Best Model Saved with Test Loss:  0.08100565274556477
65 16

100%|██████████| 30/30 [01:09<00:00,  2.32s/it]
65 16
100%|██████████| 30/30 [01:09<00:00,  2.31s/it]
65 16
100%|██████████| 30/30 [01:09<00:00,  2.31s/it]
65 16
100%|██████████| 30/30 [01:09<00:00,  2.31s/it]
Cross Validation Results:  [0.08100565274556477, 0.09677454829216003, 0.14236590266227722, 0.117516070604

Average_loss:  0.10587898890177408
```

```
# here best test loader means the split for which the test loss was recorded at its minimum.
len(best_test_loader.dataset)
```

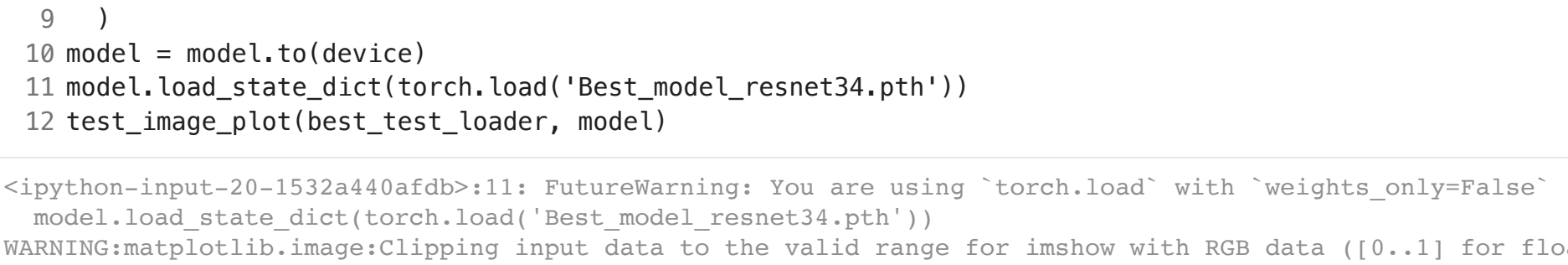
```
17
```

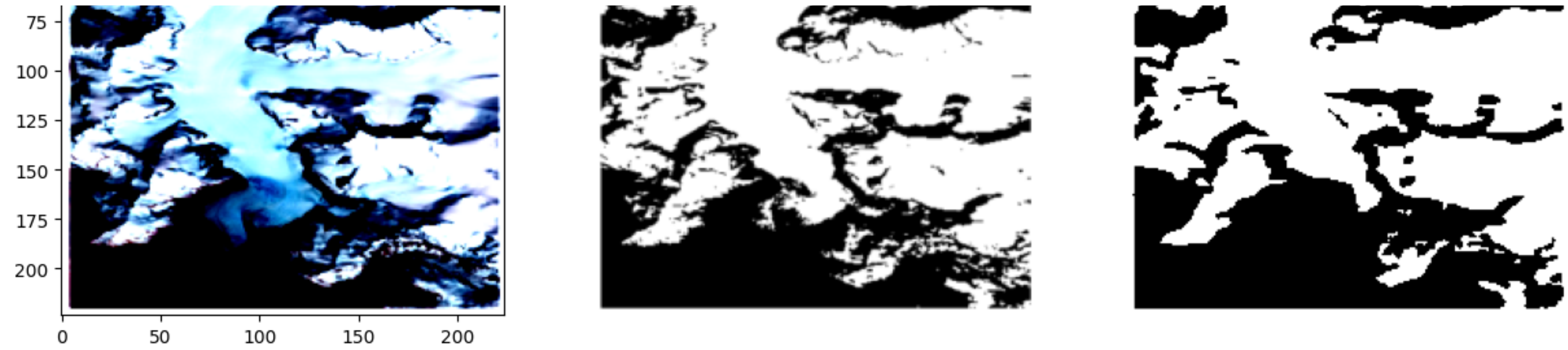
```
# Visualizing the model output on the best-picked test data, as seen the predicted mask won't be as
# and more training, for this case, i felt that 30 epochs were sufficient to produce acceptable outp
# However incase of larger region of interest and more classes or cases with instance segmentation,
model = smp.Unet(
    encoder_name="resnet34",
    encoder_weights="imagenet",
    in_channels=3,
    classes=1
  )
model = model.to(device)
model.load_state_dict(torch.load('Best_model_resnet34.pth'))
test_image_plot(best_test_loader, model)
```

```
<ipython-input-20-1532a440afdb>:11: FutureWarning: You are using `torch.load` with `weights_only=False` (
  model.load_state_dict(torch.load('Best_model_resnet34.pth'))
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

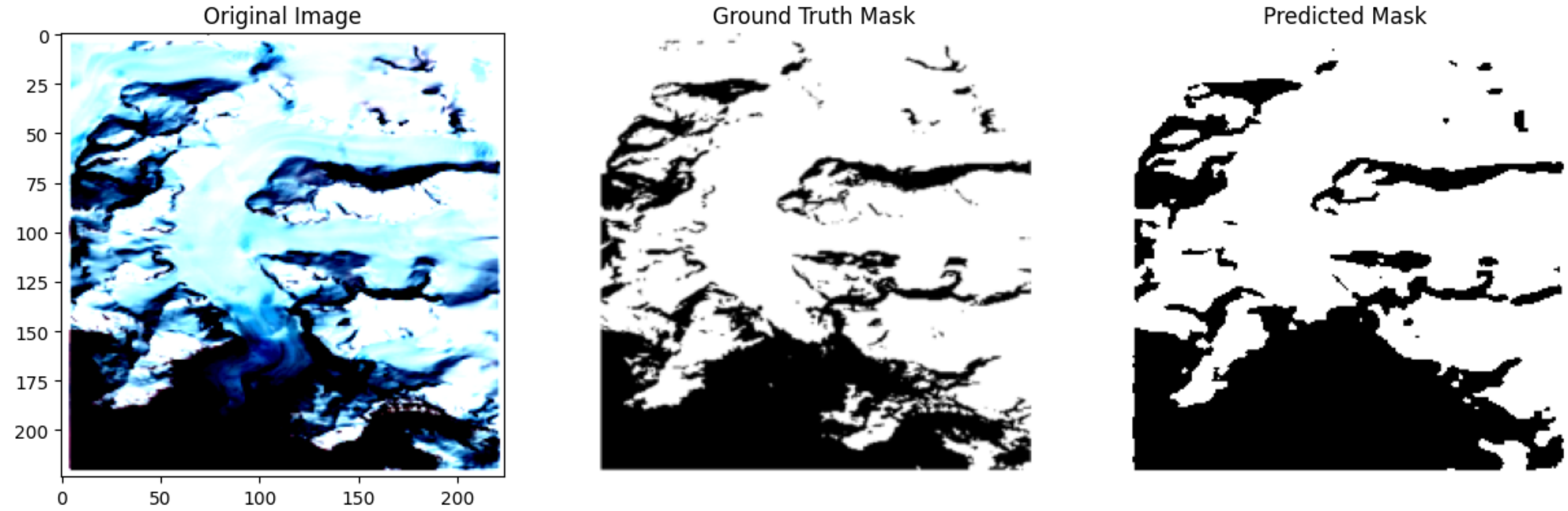



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



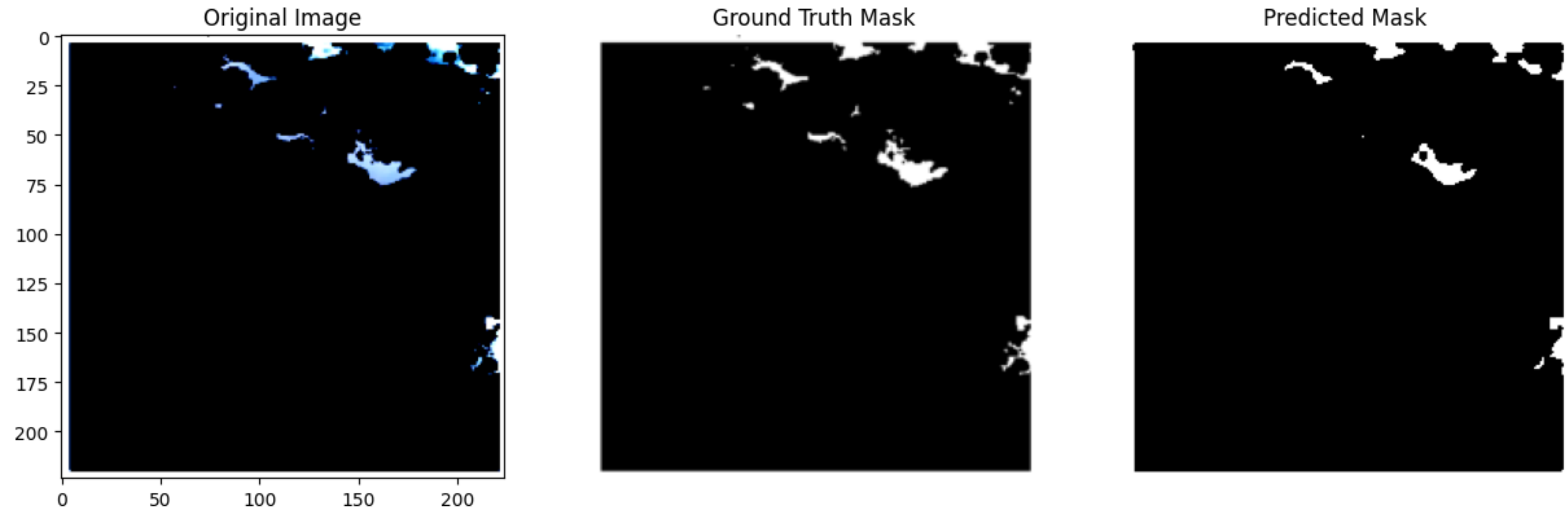


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

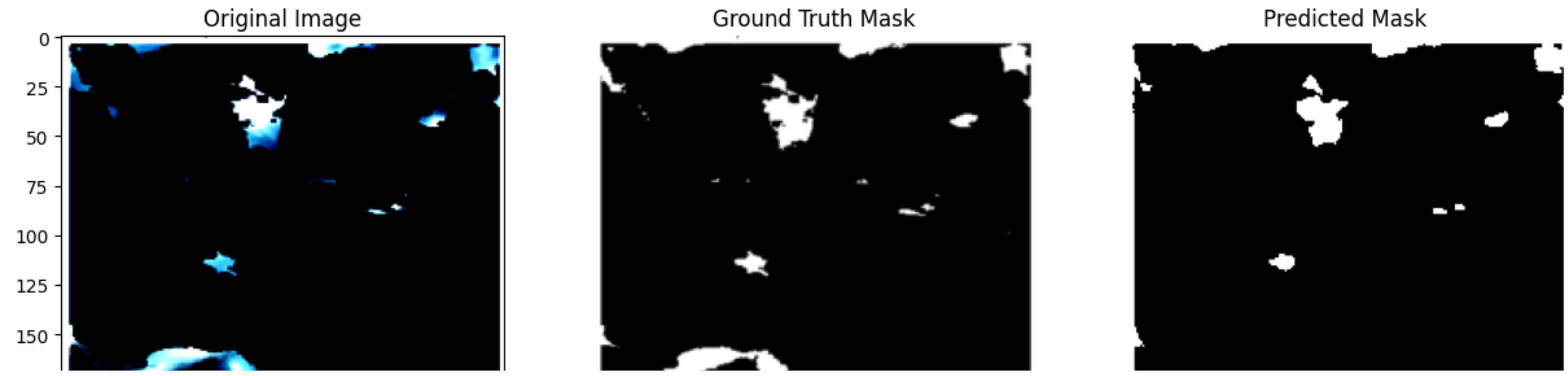



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

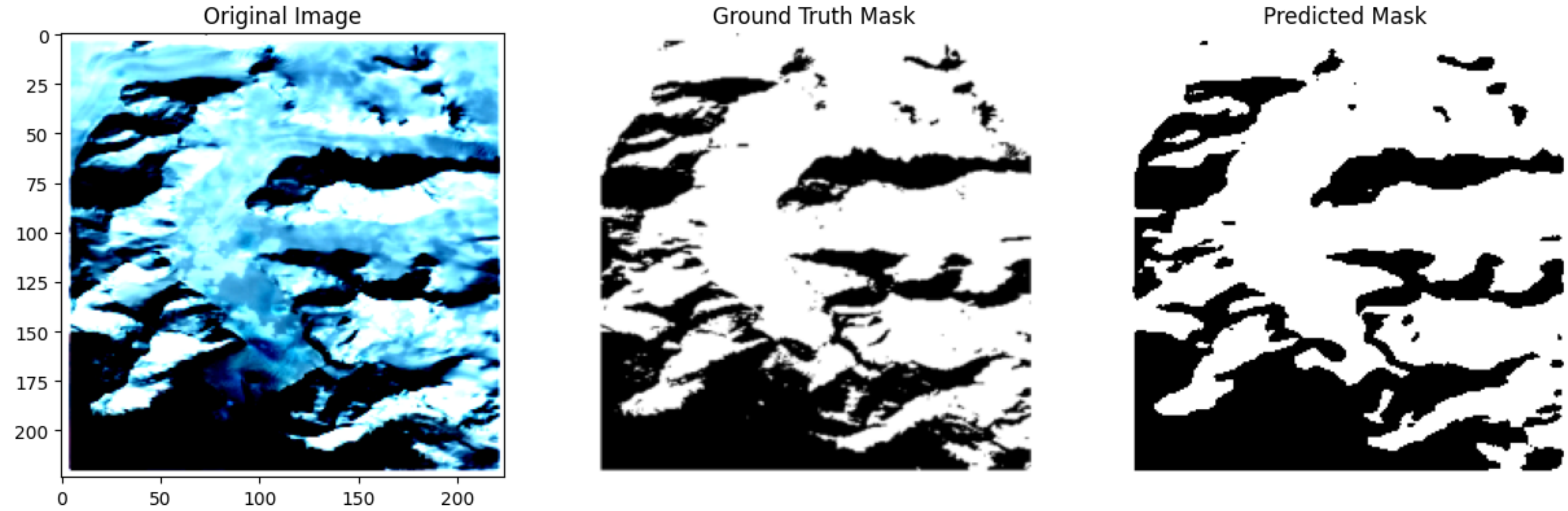

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



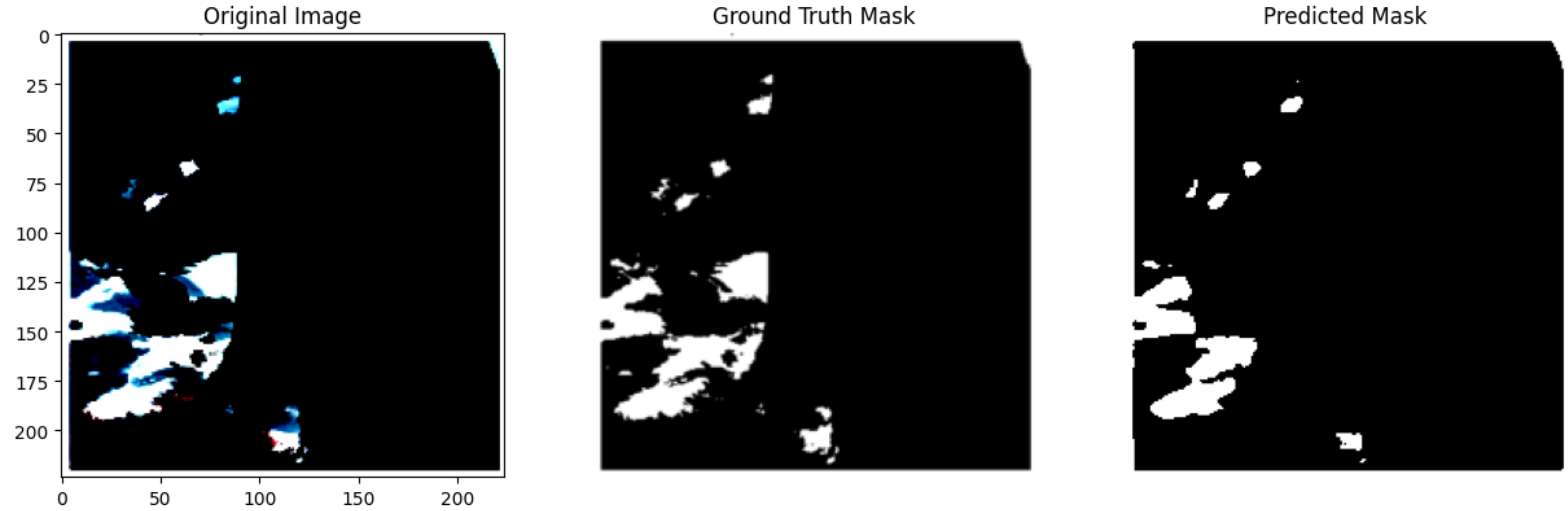

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



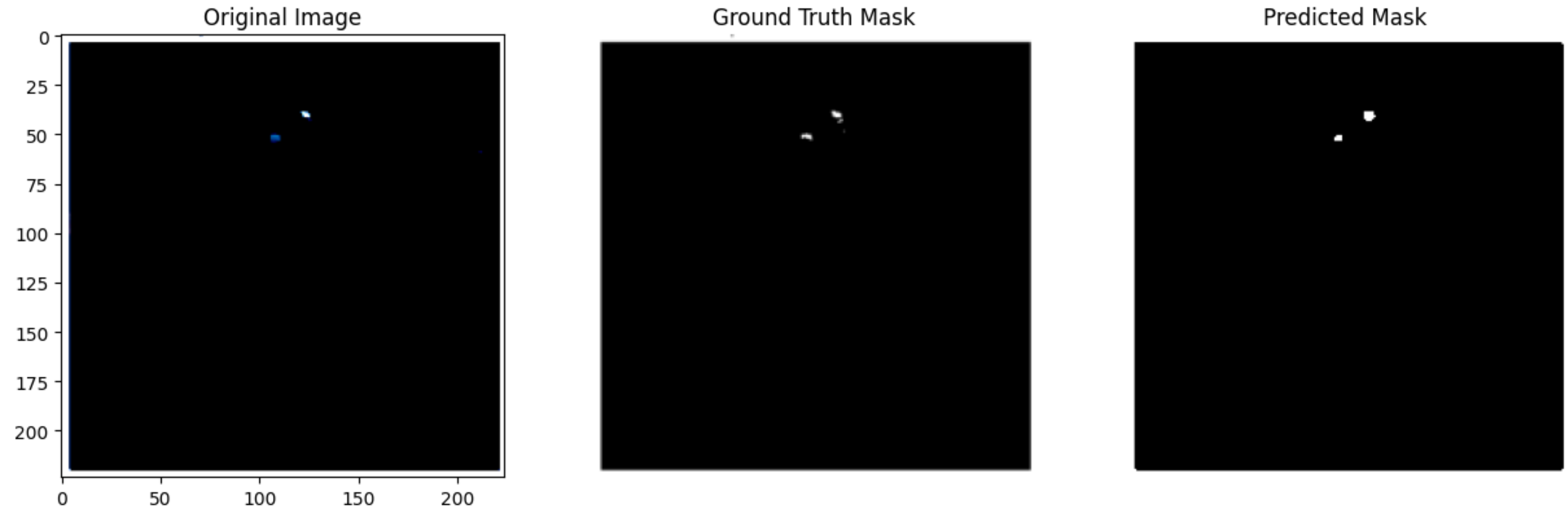
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



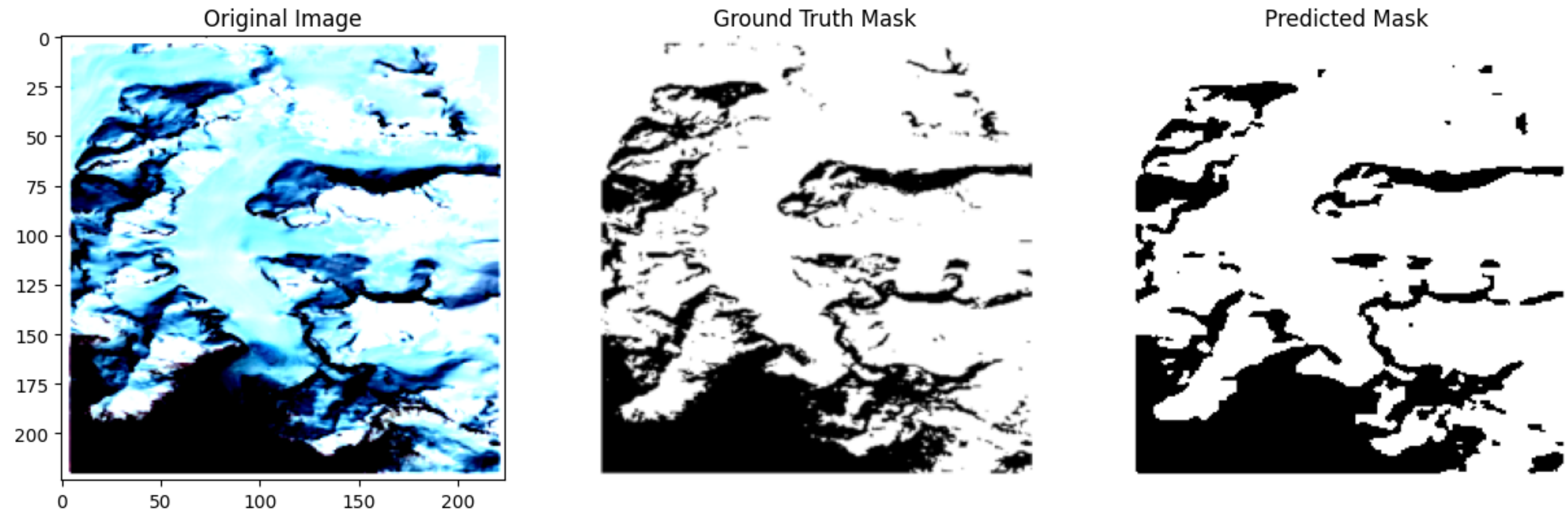

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



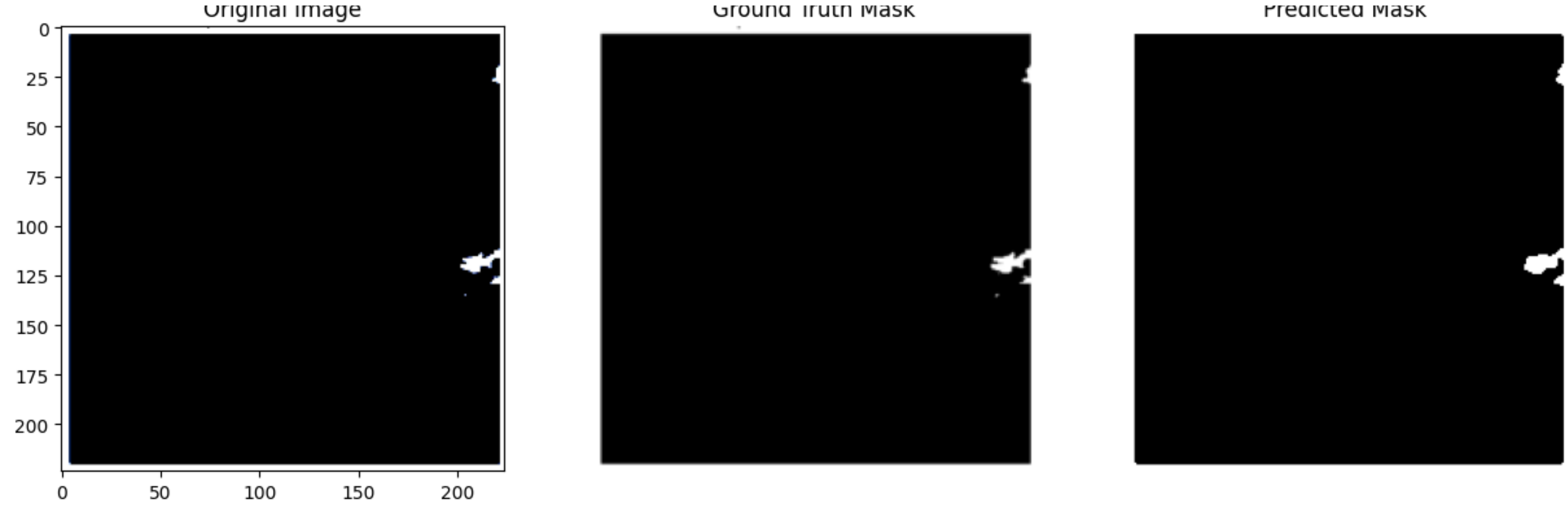

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



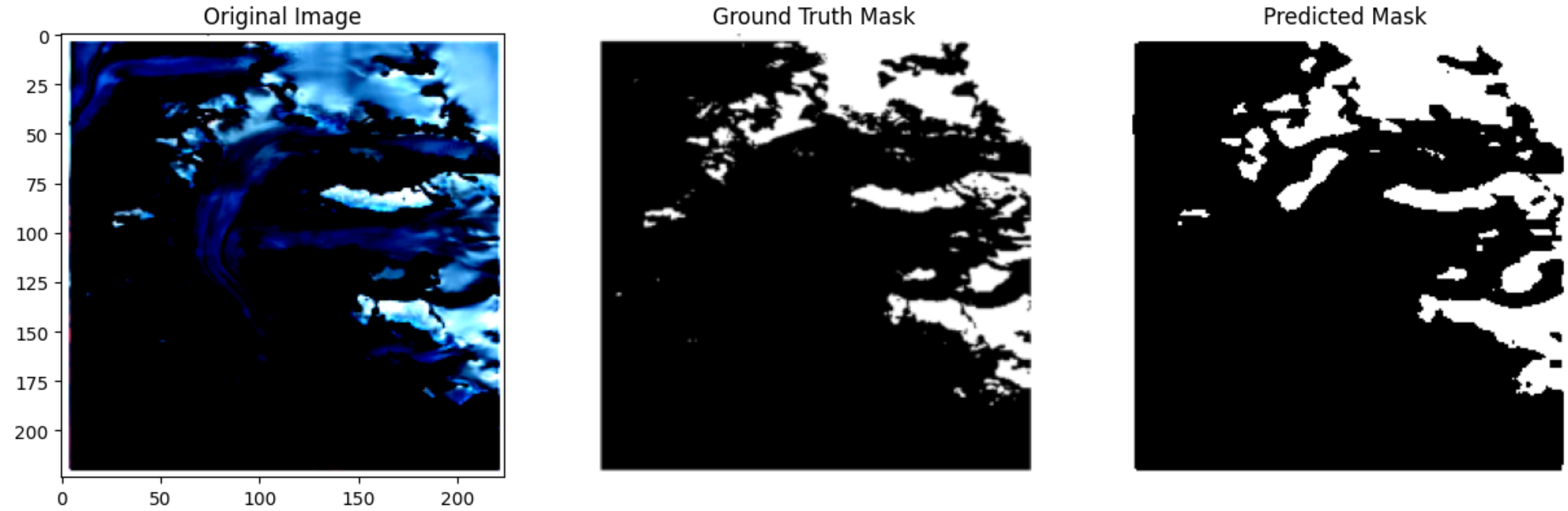

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



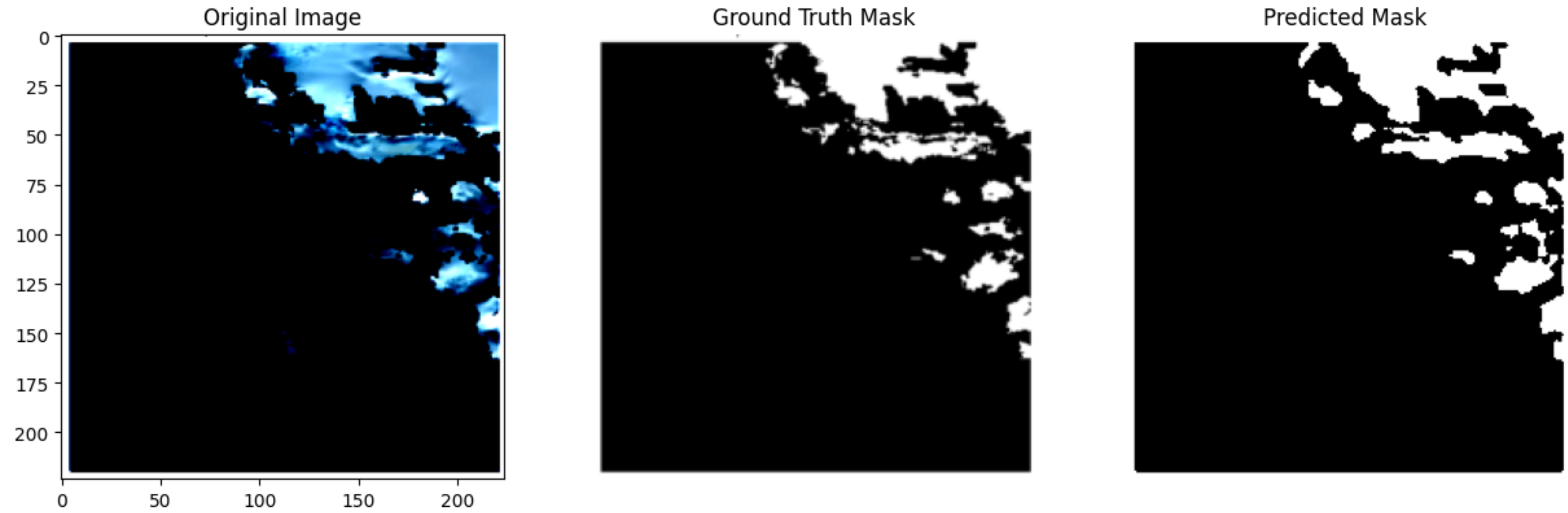

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



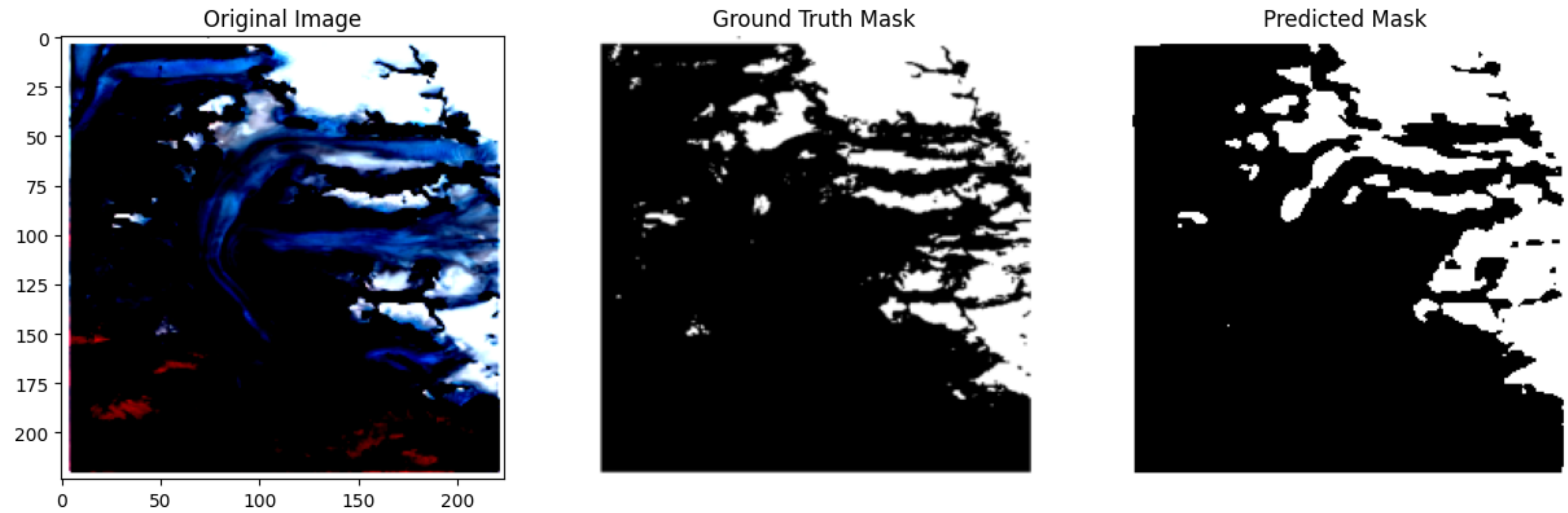

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

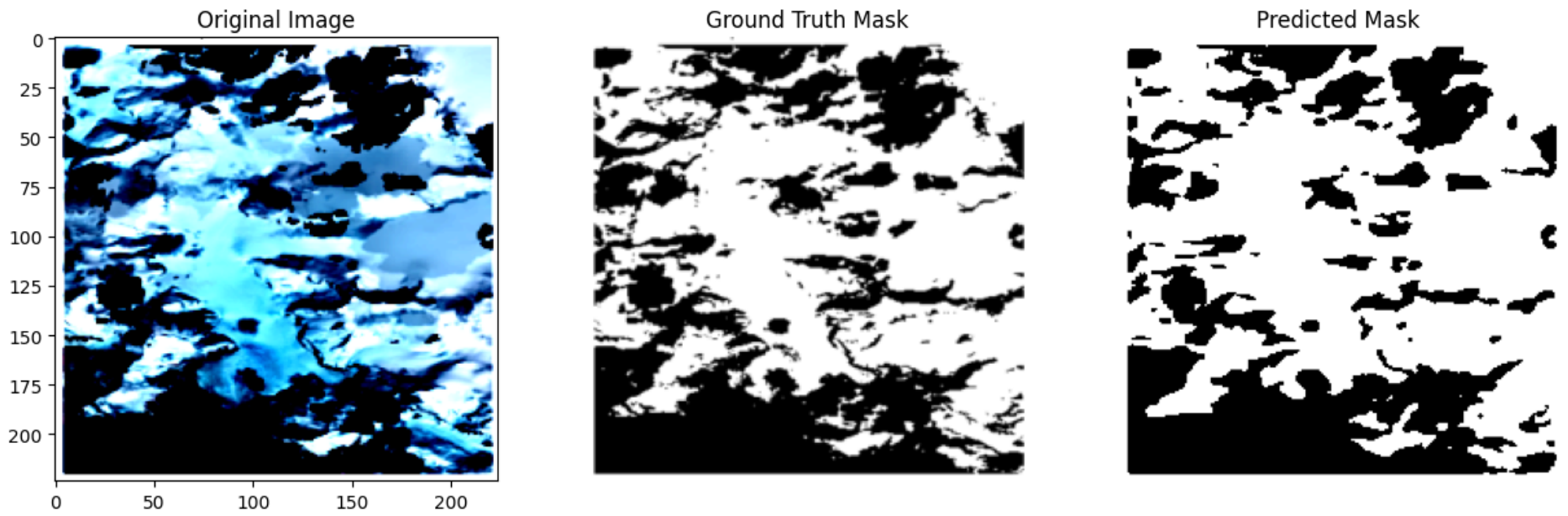



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

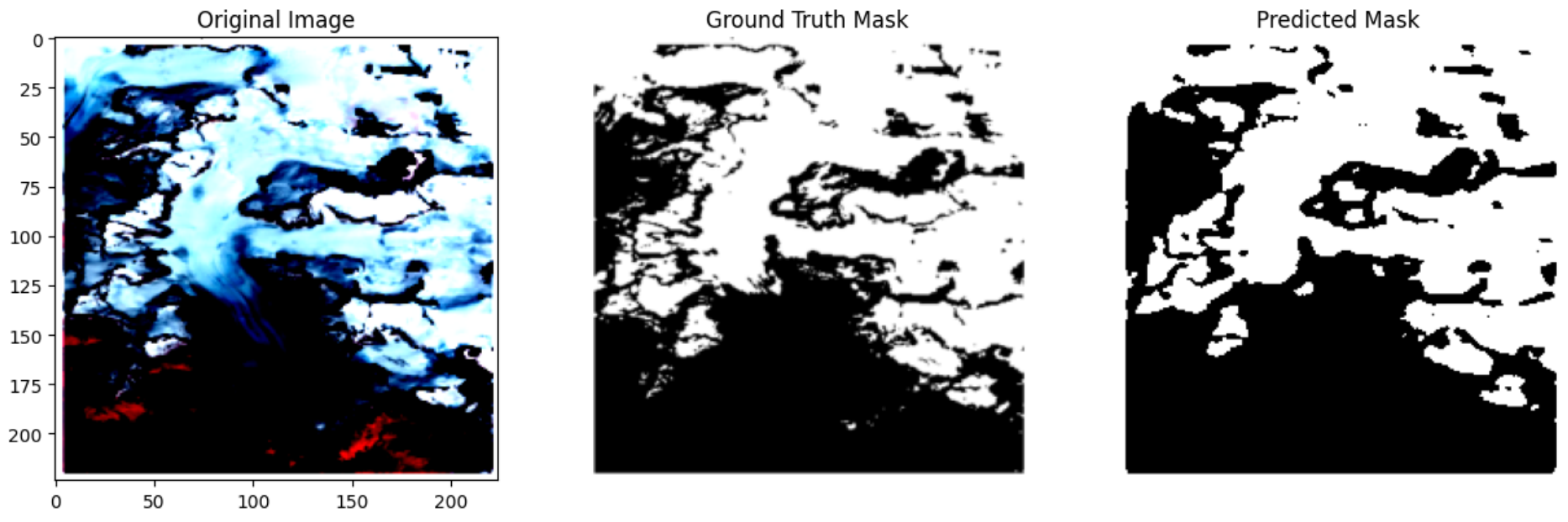



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



UNet with Encoder ResNet50 - Training and Evaluation

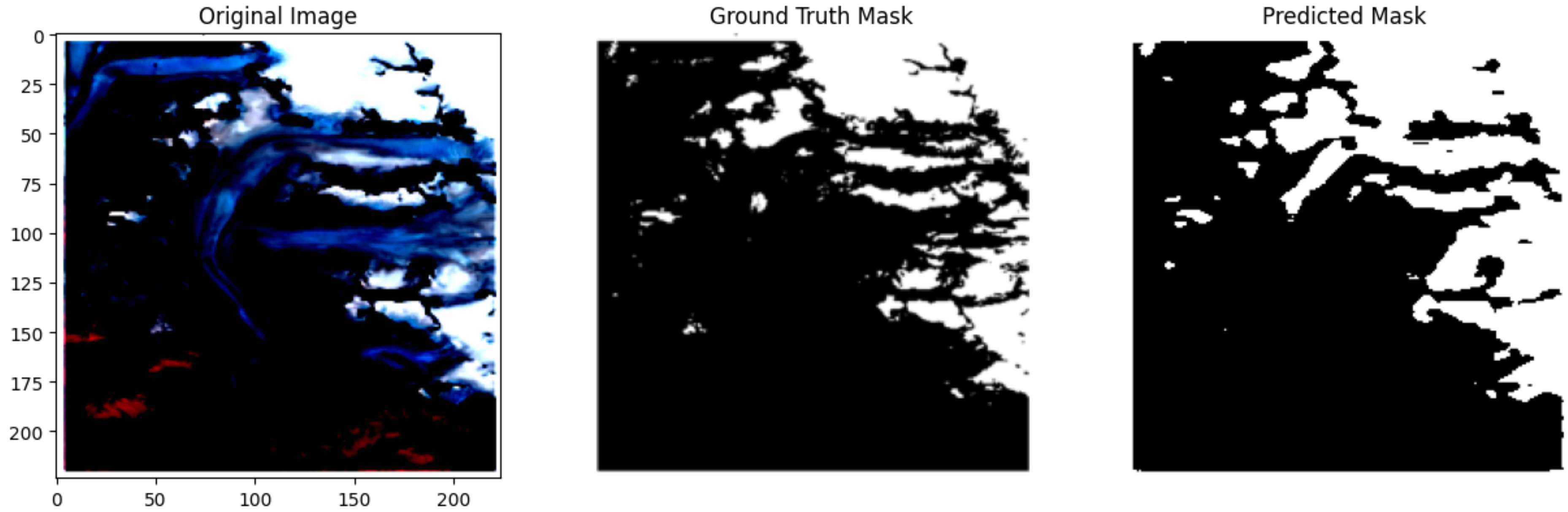
```
# keeping all the parameters same for fair comparison, except the encoder, this time, I am using Res
k_folds_results = []
resnet50_best_test_loss = 10000
loss = smp.losses.DiceLoss(mode='binary')
for fold, (train_index, test_index) in enumerate(kf.split(indices)):
  train_image_paths = [image_paths[i] for i in train_index]
  train_mask_paths = [mask_paths[i] for i in train_index]
  test_image_paths = [image_paths[i] for i in test_index]
  test_mask_paths = [mask_paths[i] for i in test_index]
  print(len(train_index), len(test_index))
  train_dataset = SegmentationDataset(train_image_paths, train_mask_paths)
  test_dataset = SegmentationDataset(test_image_paths, test_mask_paths)
  train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
  test_loader = DataLoader(test_dataset, batch_size=8, shuffle=True)
  model = smp.Unet(
    encoder_name="resnet50",
    encoder_weights="imagenet",
    in_channels=3,
    classes=1
  )
  model = model.to(device)
  optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
  for epoch in tqdm(range(30)):
    train_loss = train(model, train_loader, optimizer, loss, device)
    test_loss = test(model, test_loader, loss, device)
    if test_loss < resnet50_best_test_loss:
        resnet50_best_test_loss = test_loss
        torch.save(model.state_dict(), f"Best_model_resnet50.pth")
        print('\nBest Model Saved with Test Loss: ', resnet50_best_test_loss)
        resnet50_best_test_loader = test_loader
  k_folds_results.append(test_loss)
print("\nCross Validation Results: ", k_folds_results)
print("\nAverage_loss: ", sum(k_folds_results)/len(k_folds_results))
```

```
64 17
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.cache/torch/hub/checkp
100%|██████████| 97.8M/97.8M [00:00<00:00, 216MB/s]
  3%|▎         | 1/30 [00:03<01:34,  3.25s/it]
Best Model Saved with Test Loss:  0.5683464209238688
  7%|▋         | 2/30 [00:06<01:29,  3.20s/it]
Best Model Saved with Test Loss:  0.31959615151087445
 13%|█▎        | 4/30 [00:12<01:19,  3.05s/it]
Best Model Saved with Test Loss:  0.20027210315068564
 30%|███       | 9/30 [00:26<01:01,  2.91s/it]
Best Model Saved with Test Loss:  0.1463657021522522
 33%|███       | 10/30 [00:30<01:00,  3.03s/it]
Best Model Saved with Test Loss:  0.1407331625620524
 37%|███▋      | 11/30 [00:33<00:58,  3.06s/it]
Best Model Saved with Test Loss:  0.14008078972498575
 43%|████▎     | 13/30 [00:39<00:52,  3.10s/it]
Best Model Saved with Test Loss:  0.1374316414197286
 50%|█████     | 15/30 [00:45<00:44,  2.97s/it]
Best Model Saved with Test Loss:  0.12121532360712688
 53%|█████▎    | 16/30 [00:48<00:44,  3.19s/it]
Best Model Saved with Test Loss:  0.11623650789260864
 60%|██████    | 18/30 [00:55<00:37,  3.13s/it]
Best Model Saved with Test Loss:  0.11049691836039226
 63%|██████▎   | 19/30 [00:58<00:34,  3.15s/it]
Best Model Saved with Test Loss:  0.10581533114115398
 70%|███████   | 21/30 [01:04<00:28,  3.19s/it]
Best Model Saved with Test Loss:  0.10520986715952556
 83%|████████▎ | 25/30 [01:16<00:14,  2.98s/it]
Best Model Saved with Test Loss:  0.09534839789072673
 87%|████████▋ | 26/30 [01:19<00:12,  3.15s/it]
Best Model Saved with Test Loss:  0.09269265333811443
 93%|█████████▎| 28/30 [01:25<00:06,  3.07s/it]
Best Model Saved with Test Loss:  0.09195496638615926
100%|██████████| 30/30 [01:31<00:00,  3.06s/it]
Best Model Saved with Test Loss:  0.08649138609568278
65 16

 27%|██▋       | 8/30 [00:21<00:59,  2.70s/it]
```

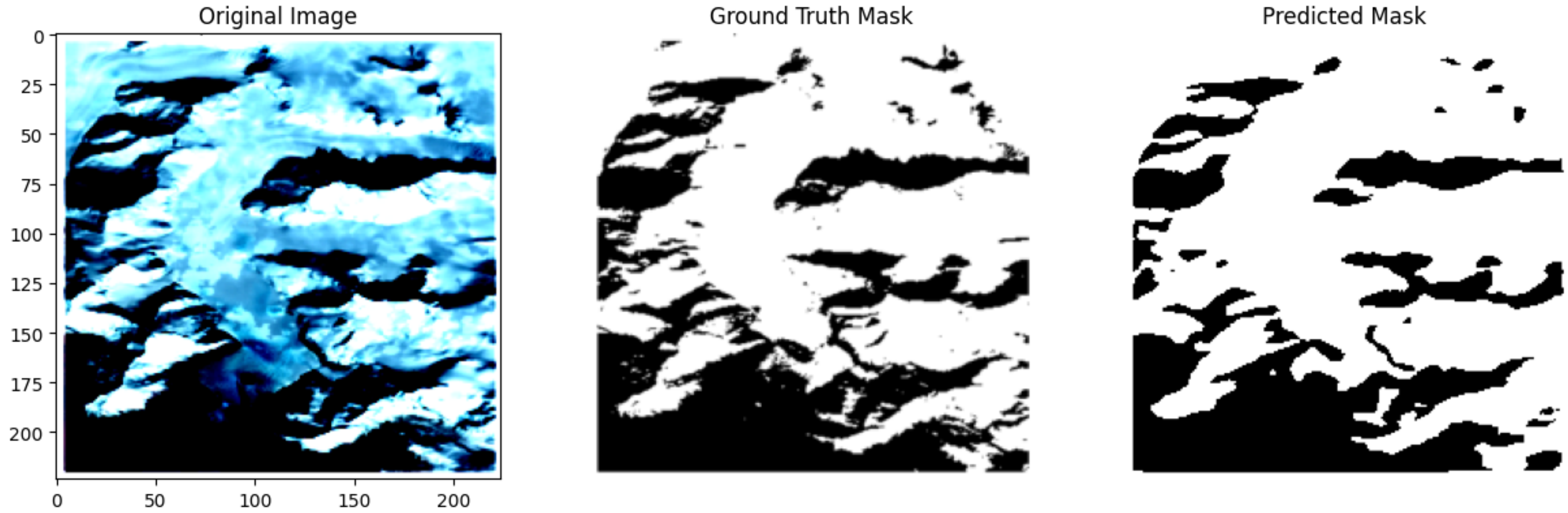
```
# visualizing the segmented output, similarly, on the best recorded test split.
```

```
model = smp.Unet(
    encoder_name="resnet50",
    encoder_weights="imagenet",
    in_channels=3,
    classes=1
)
model = model.to(device)
model.load_state_dict(torch.load('Best_model_resnet50.pth'))
test_image_plot(resnet50_best_test_loader, model)
```

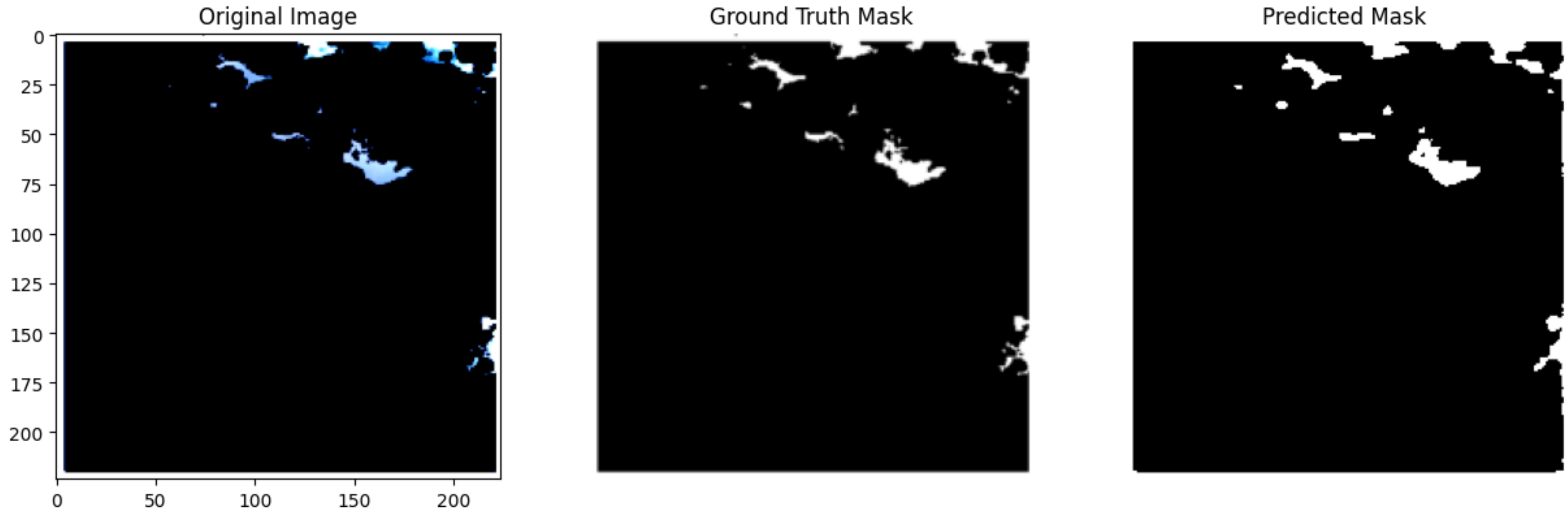
```
<ipython-input-22-01c725dc0aa7>:9: FutureWarning: You are using `torch.load` with `weights_only=False` (t
  model.load_state_dict(torch.load('Best_model_resnet50.pth'))
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



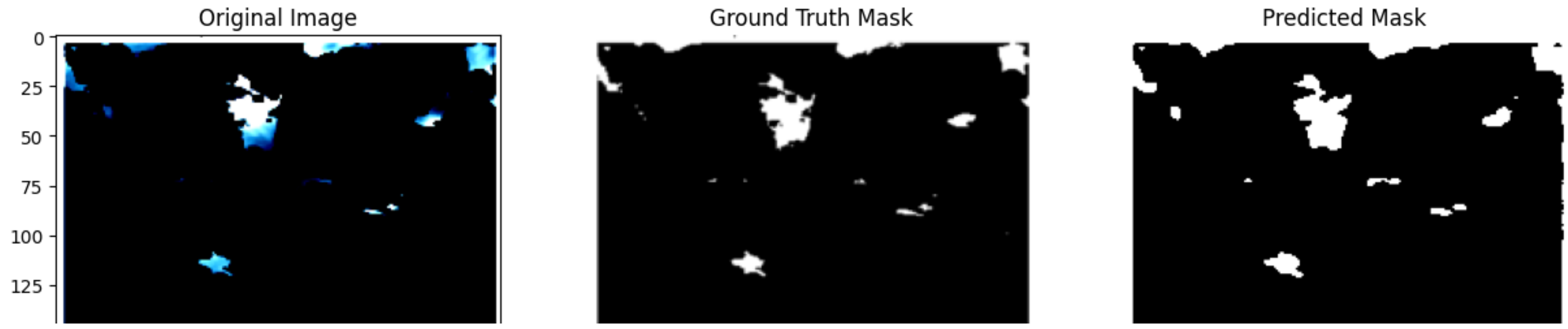

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



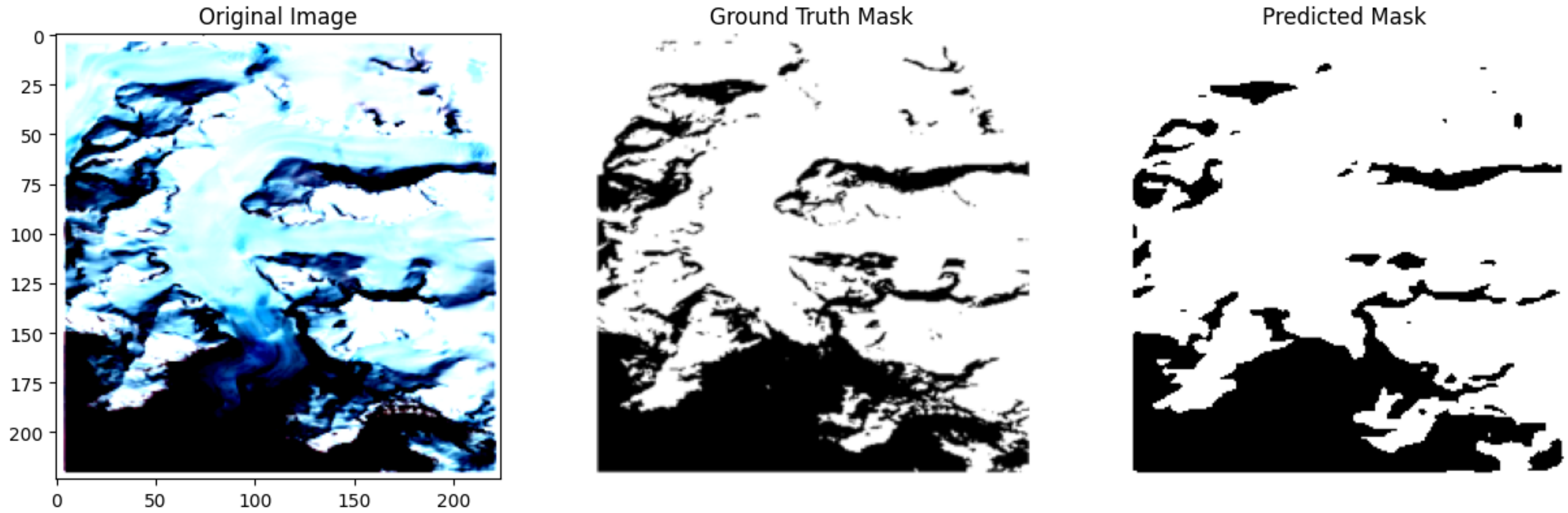

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



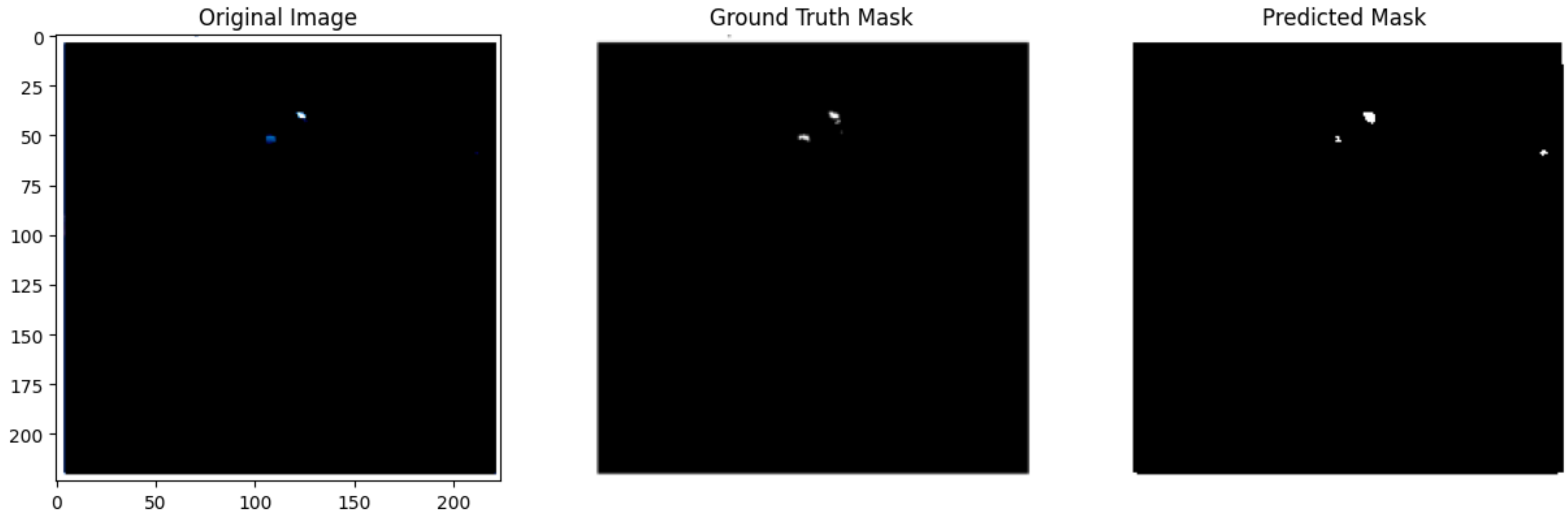

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



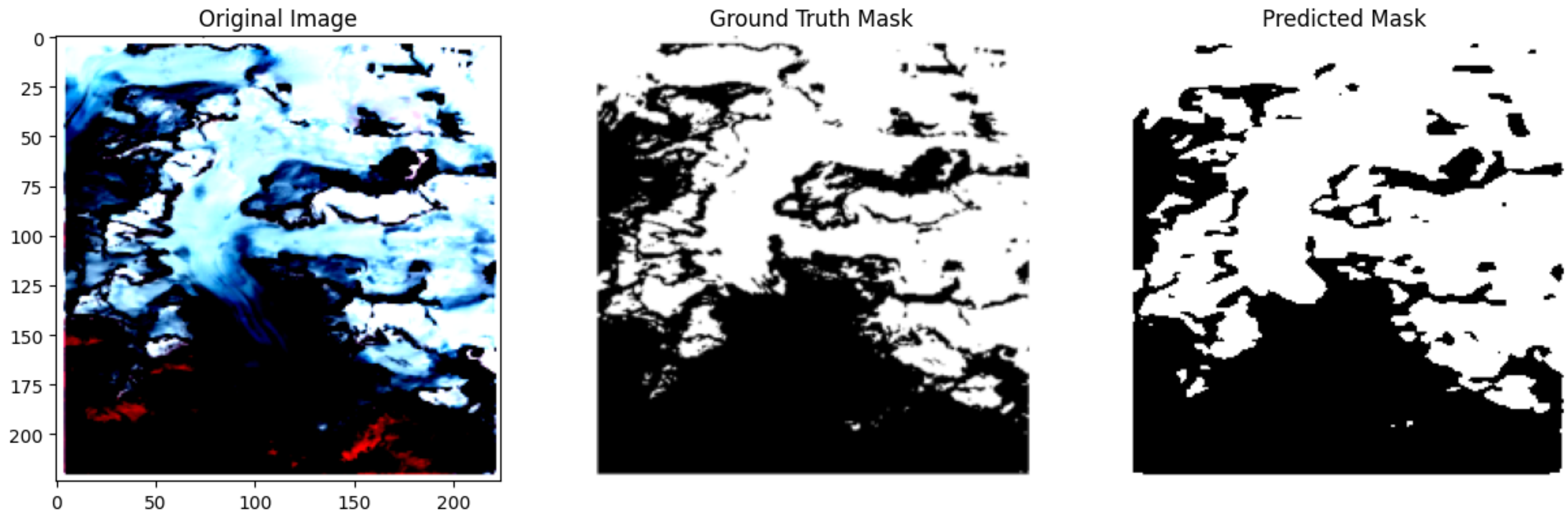
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa



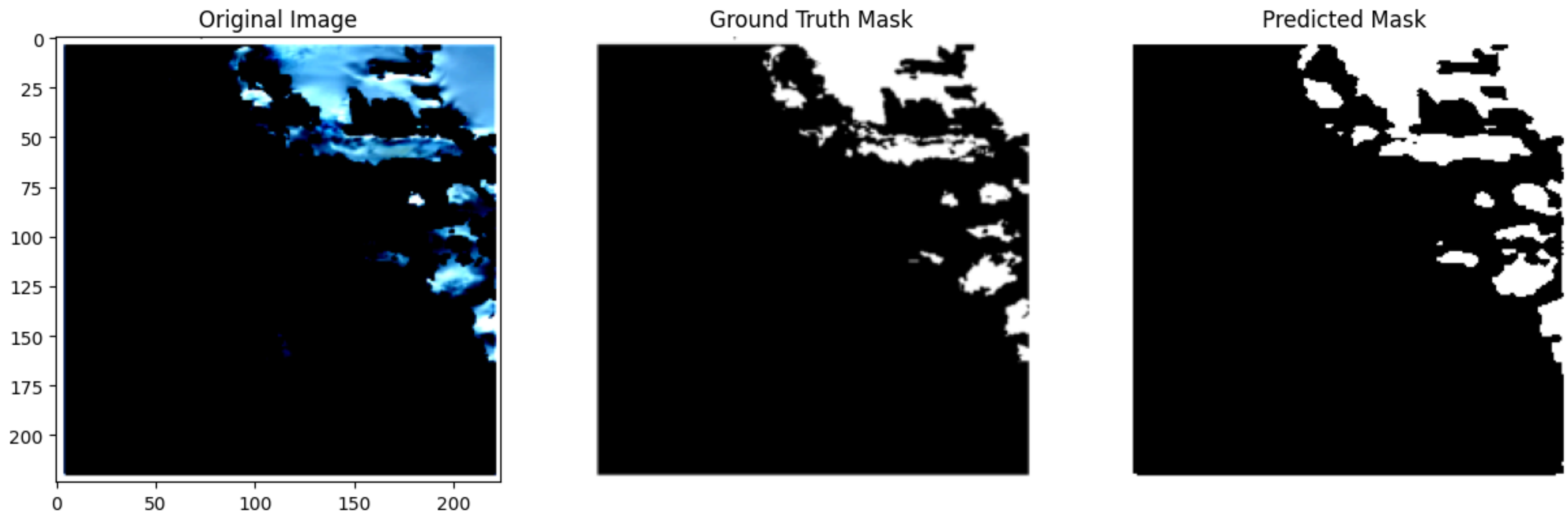
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa



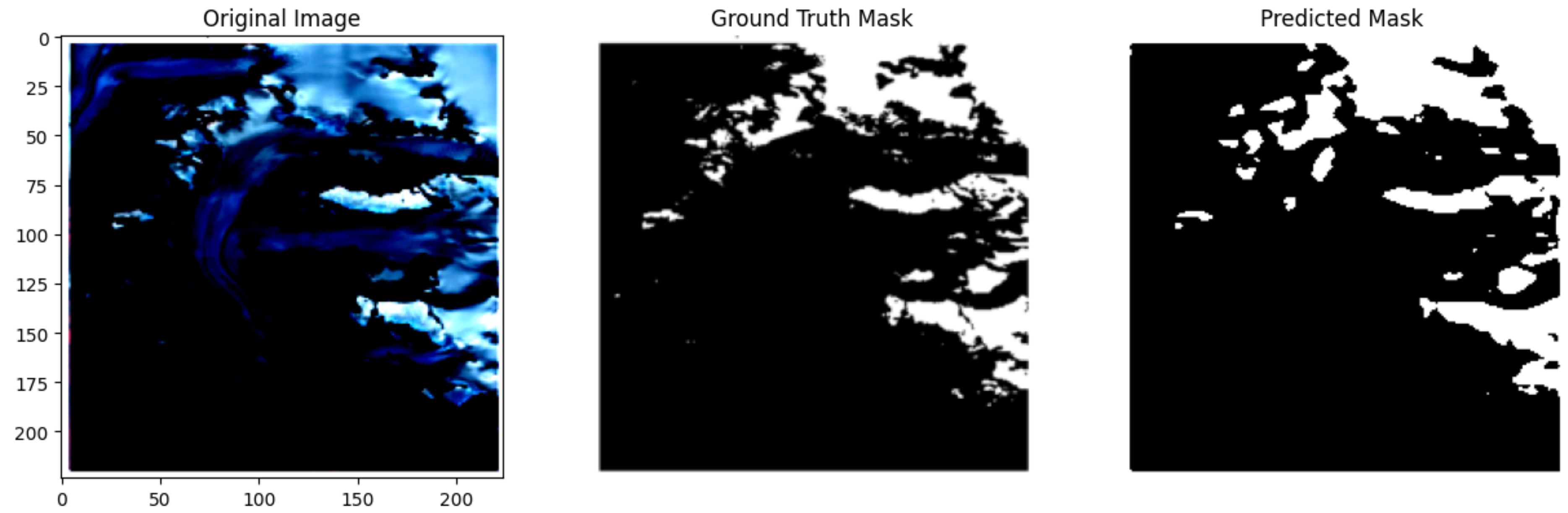
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa



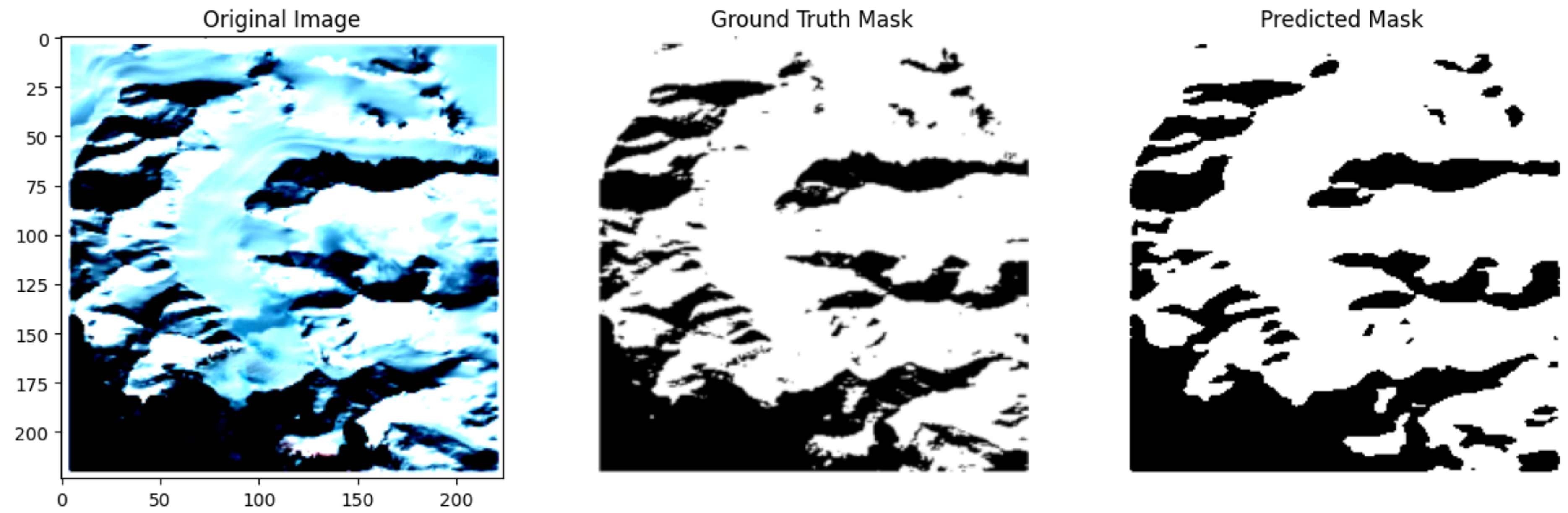
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa



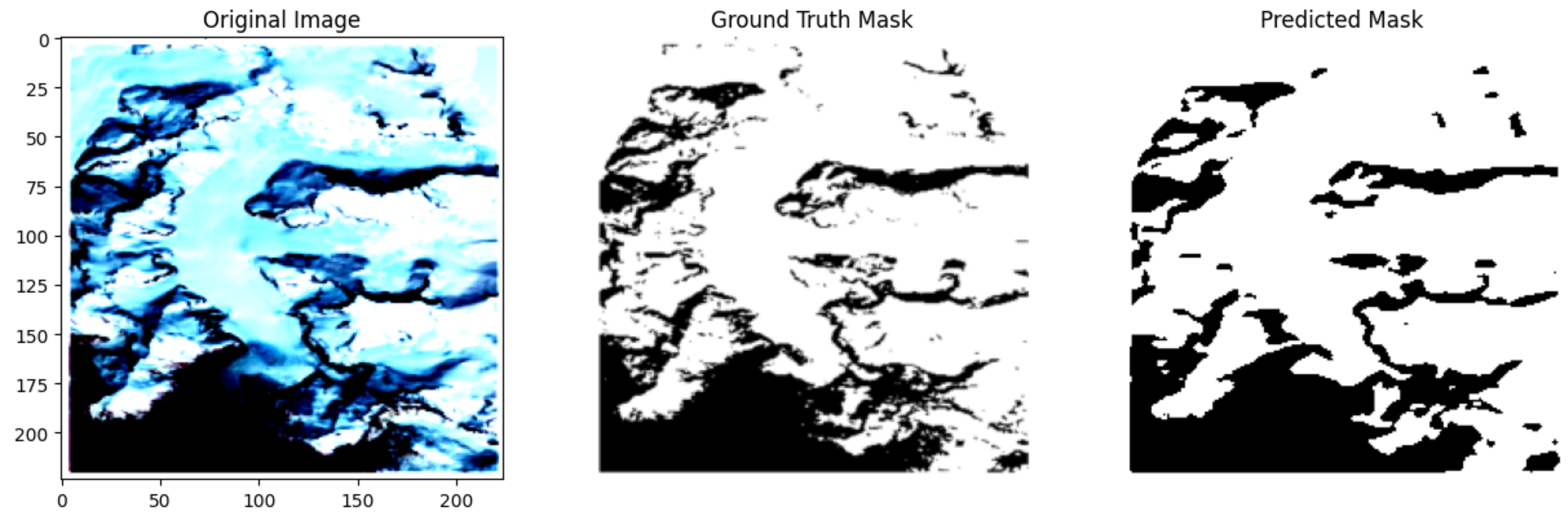
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



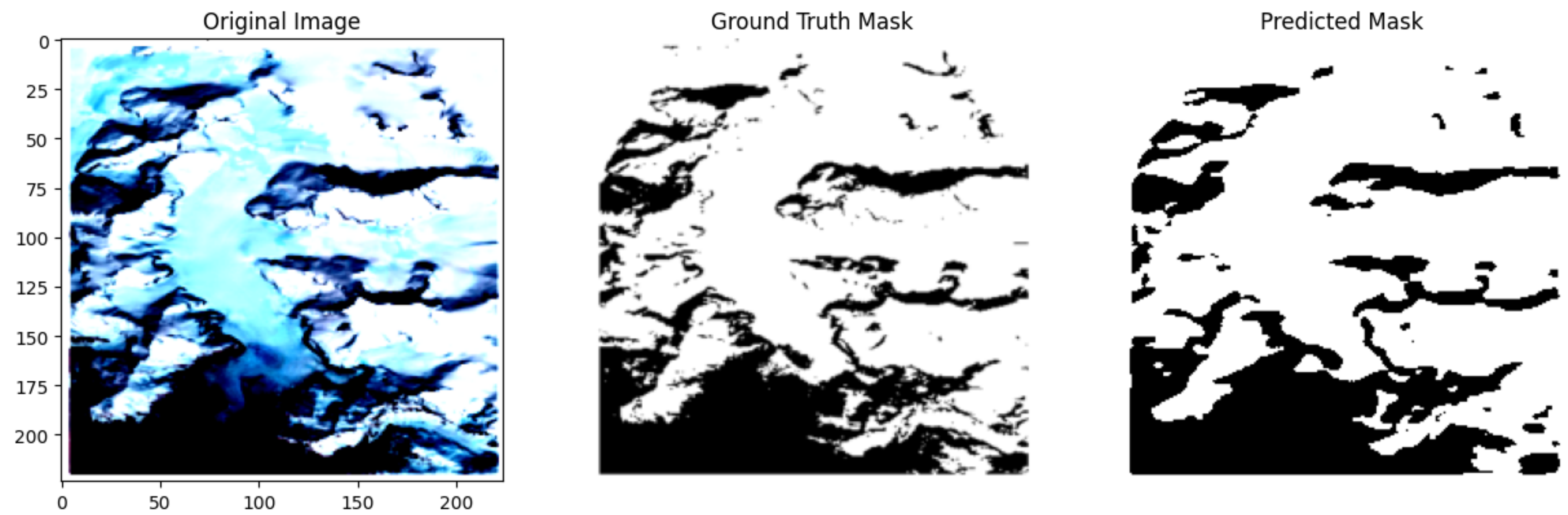

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

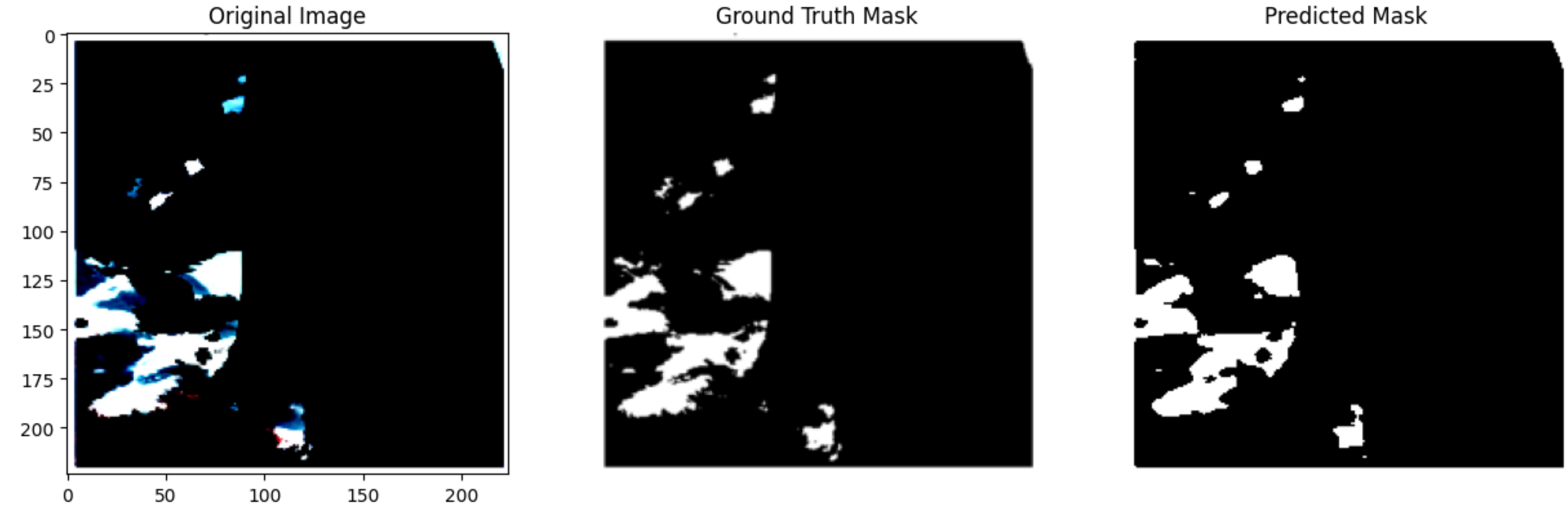



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

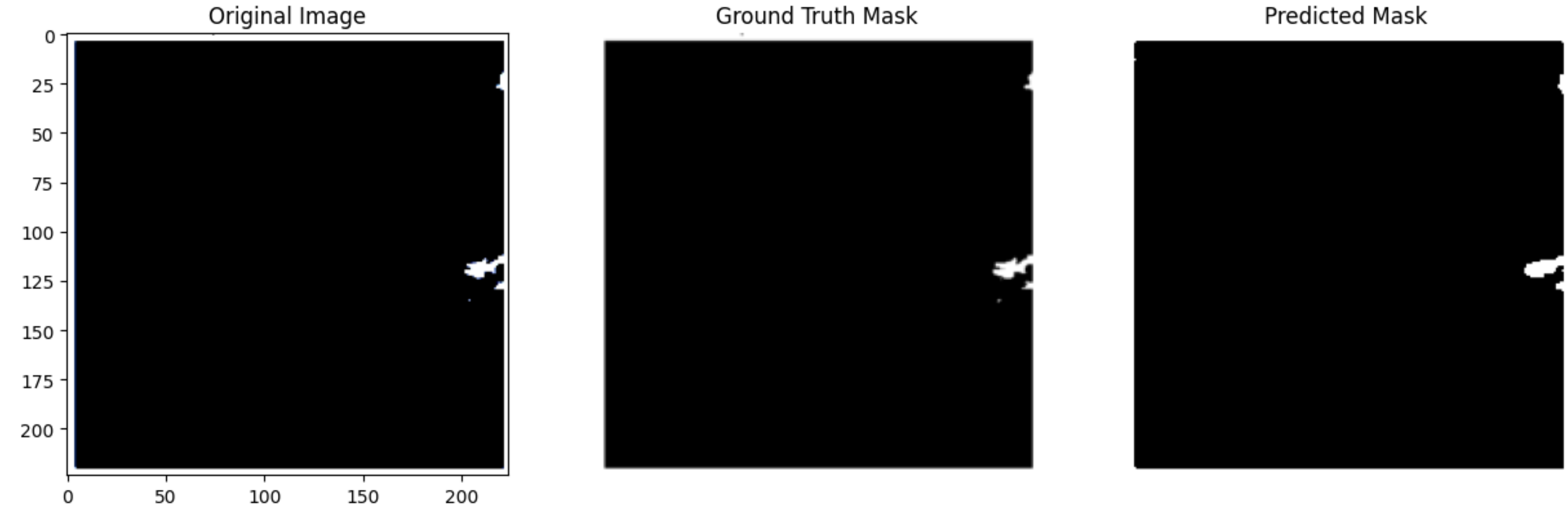



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

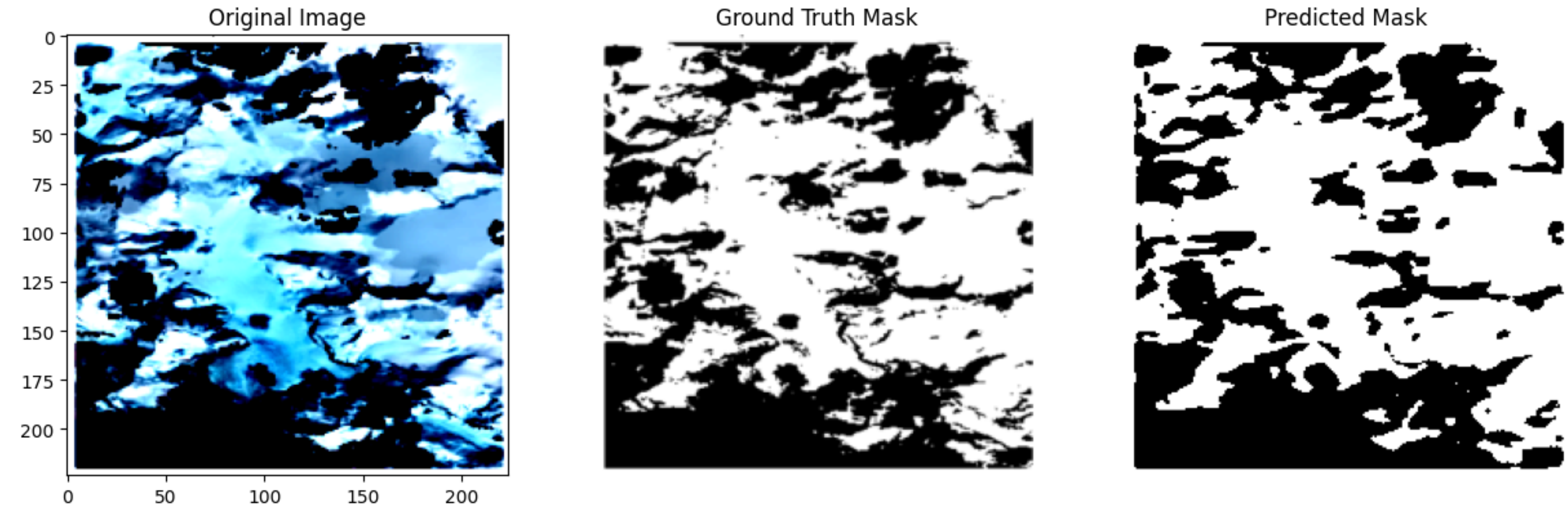
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



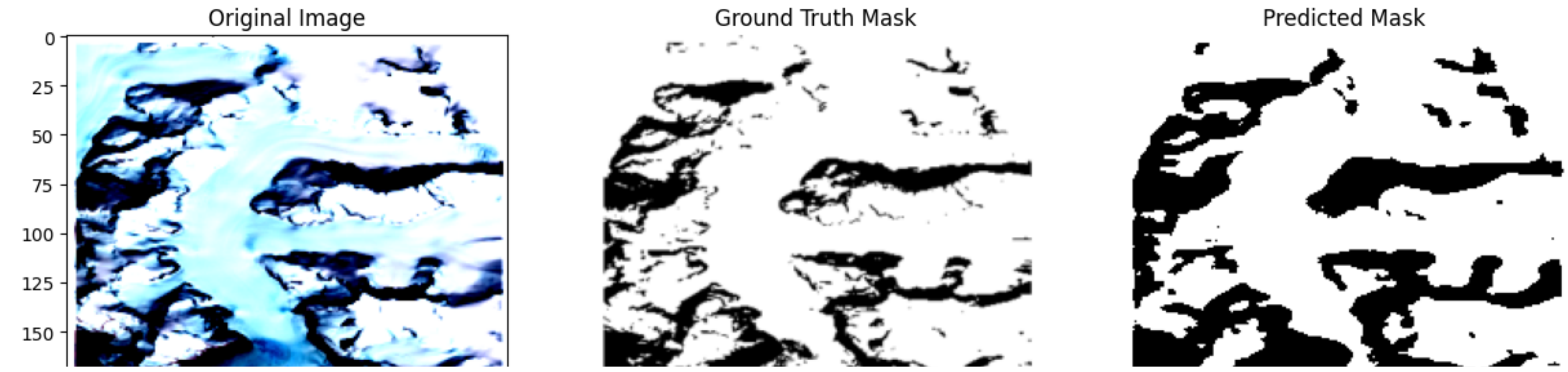

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

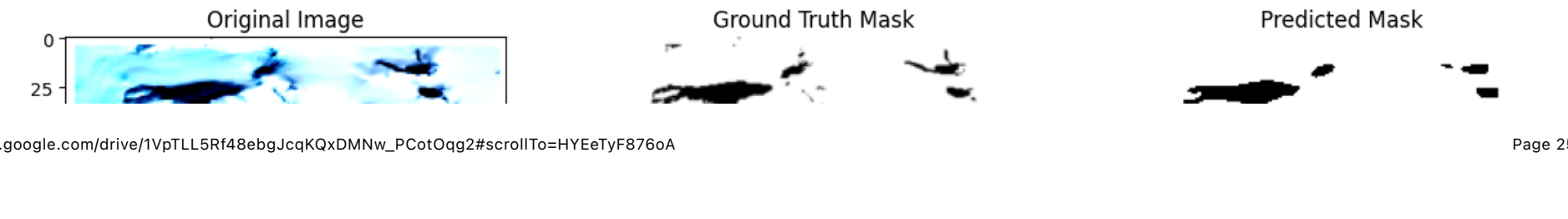


UNet with Encoder EfficientNet-b4: Training and Evaluation

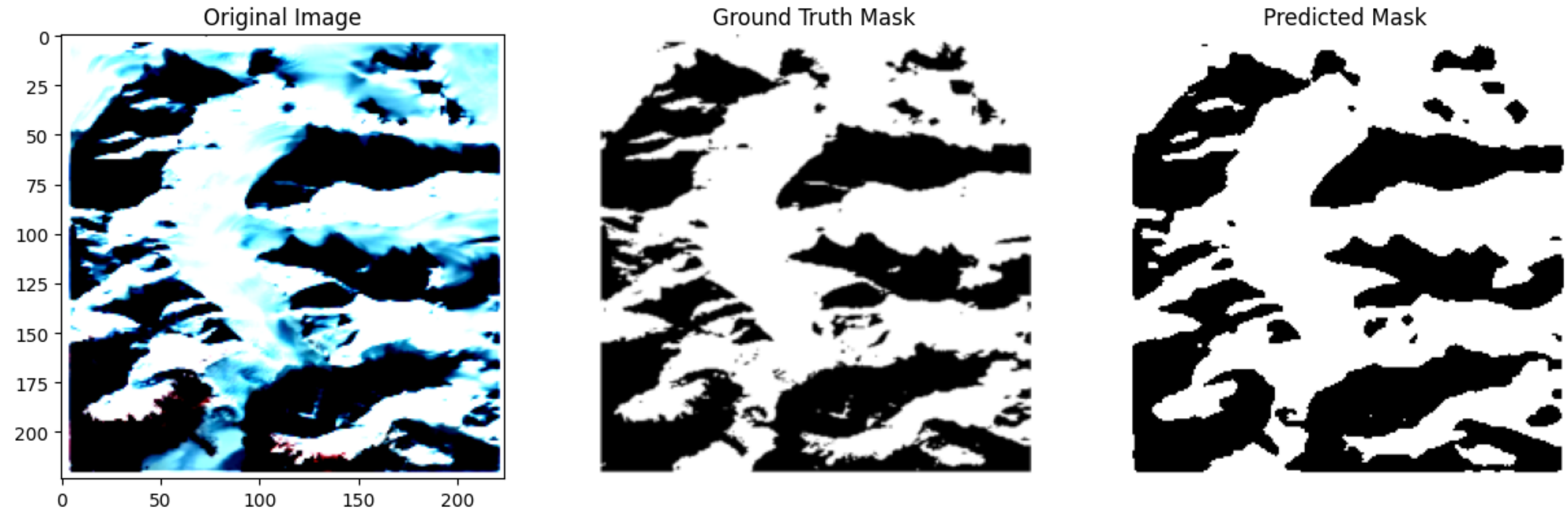
```
# e# keeping all the parameters same for fair comparison, except the encoder, this time, I am using
k_folds_results = []
efficientb4_best_test_loss = 10000
loss = smp.losses.DiceLoss(mode='binary')
for fold, (train_index, test_index) in enumerate(kf.split(indices)):
  train_image_paths = [image_paths[i] for i in train_index]
  train_mask_paths = [mask_paths[i] for i in train_index]
  test_image_paths = [image_paths[i] for i in test_index]
  test_mask_paths = [mask_paths[i] for i in test_index]
  print(len(train_index), len(test_index))
  train_dataset = SegmentationDataset(train_image_paths, train_mask_paths)
  test_dataset = SegmentationDataset(test_image_paths, test_mask_paths)
  train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
  test_loader = DataLoader(test_dataset, batch_size=8, shuffle=True)
  model = smp.Unet(
    encoder_name="efficientnet-b4",
    encoder_weights="imagenet",
    in_channels=3,
    classes=1
  )
  model = model.to(device)
  optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
  for epoch in tqdm(range(30)):
    train_loss = train(model, train_loader, optimizer, loss, device)
    test_loss = test(model, test_loader, loss, device)
    if test_loss < efficientb4_best_test_loss:
        efficientb4_best_test_loss = test_loss
        torch.save(model.state_dict(), f"Best_model_efficientnet-b4.pth")
        print('\nBest Model Saved with Test Loss: ', efficientb4_best_test_loss)
        best_test_loader = test_loader
  k_folds_results.append(test_loss)
print("\nCross Validation Results: ", k_folds_results)
print("\nAverage_loss: ", sum(k_folds_results)/len(k_folds_results))
```

```
# again doing the visualization of model performance on segmented mask, using the best test split, l
model = smp.Unet(
    encoder_name="efficientnet-b4",
    encoder_weights="imagenet",
    in_channels=3,
    classes=1
)
model = model.to(device)
model.load_state_dict(torch.load('Best_model_efficientnet-b4.pth'))
test_image_plot(best_test_loader, model)
```

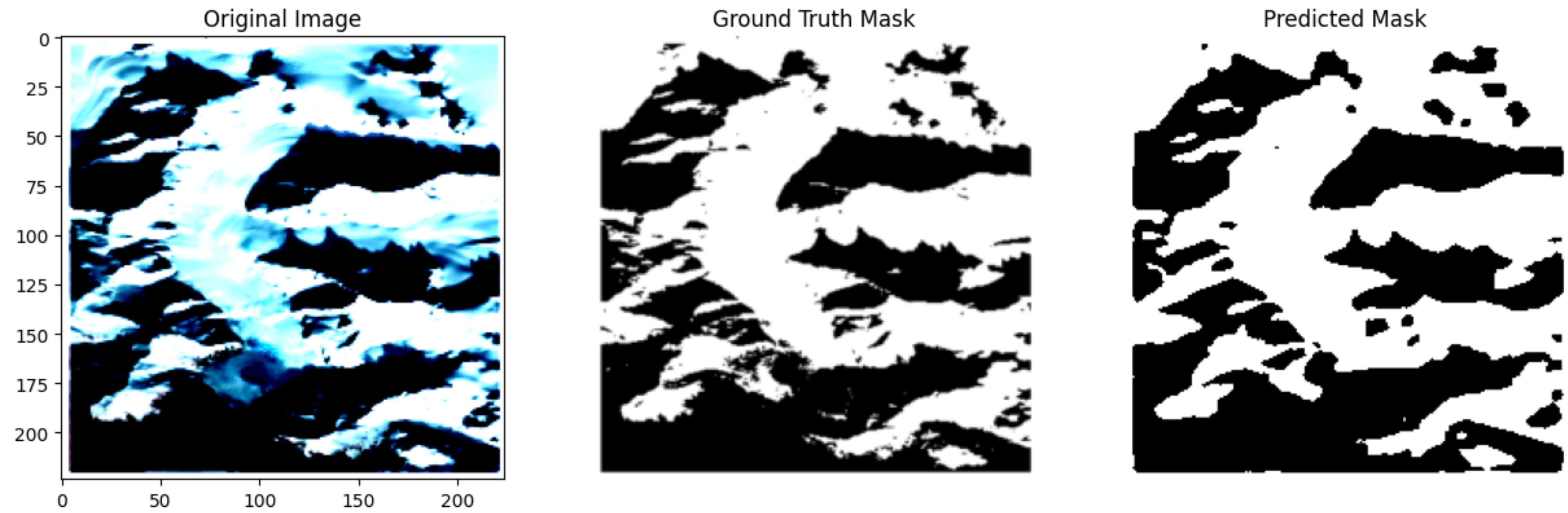
```
<ipython-input-24-d3efdd47aea9>:9: FutureWarning: You are using `torch.load` with `weights_only=False` (t
  model.load_state_dict(torch.load('Best_model_efficientnet-b4.pth'))
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



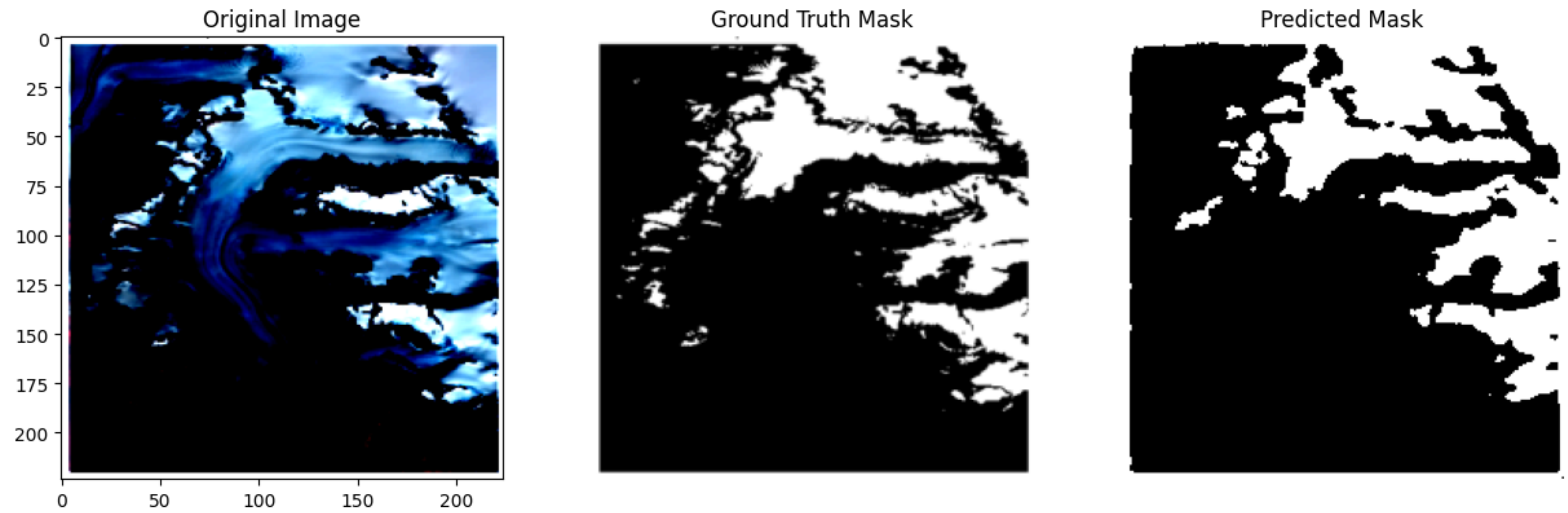
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

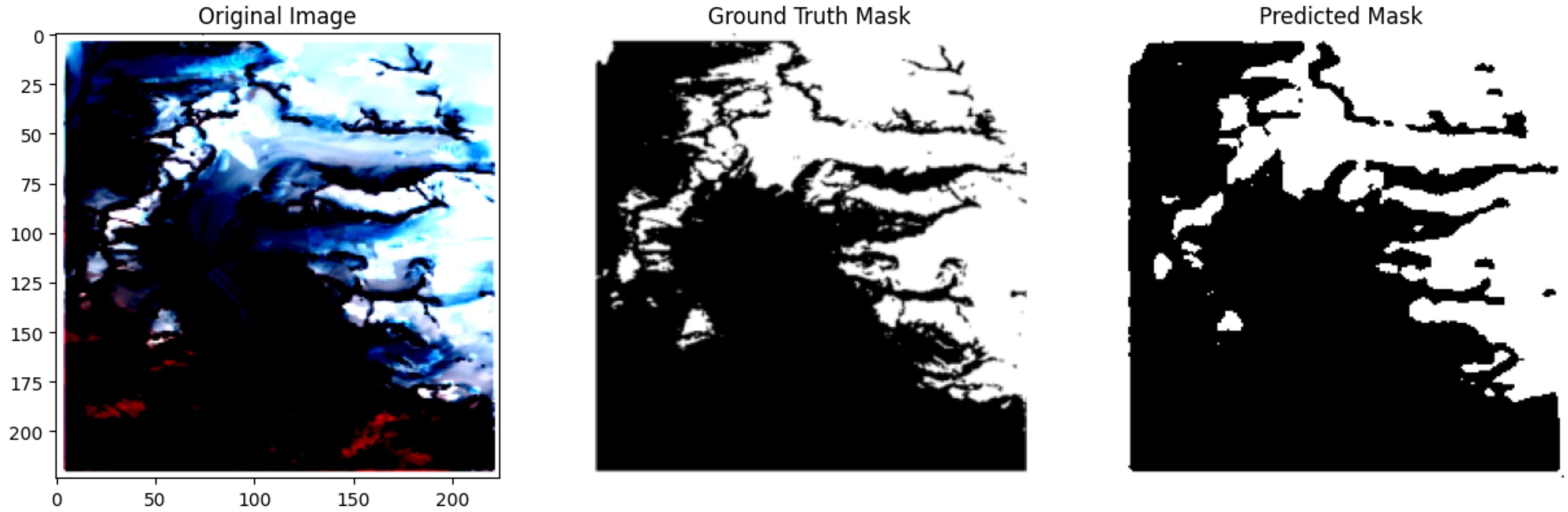



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

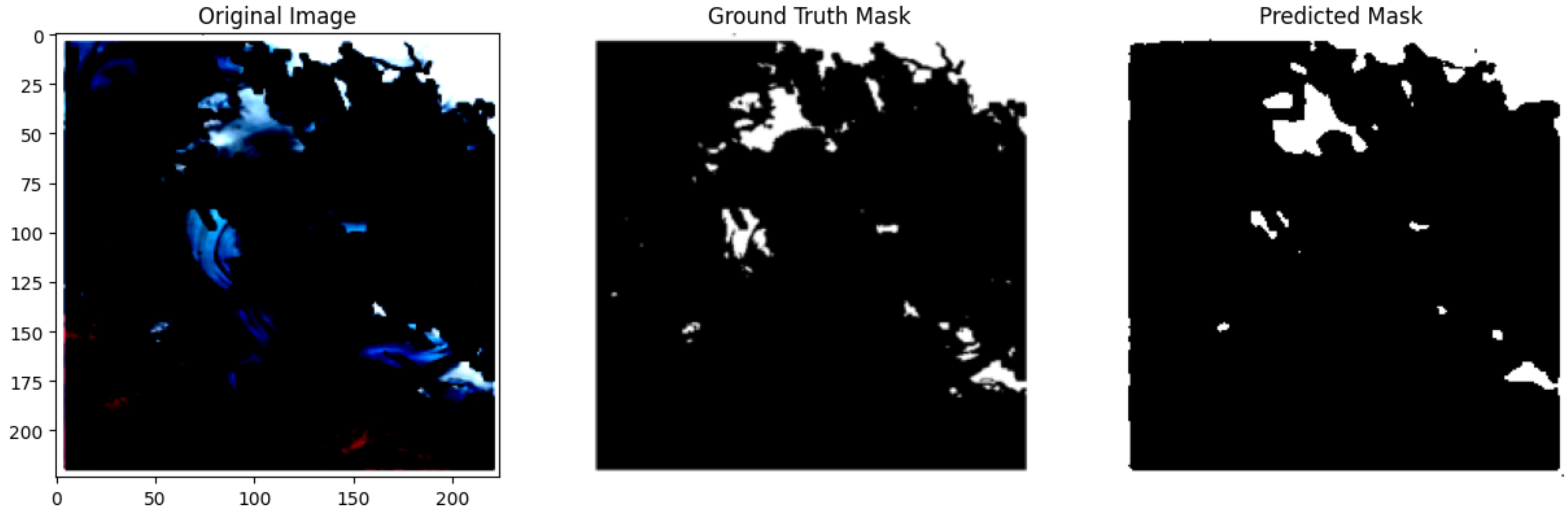



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

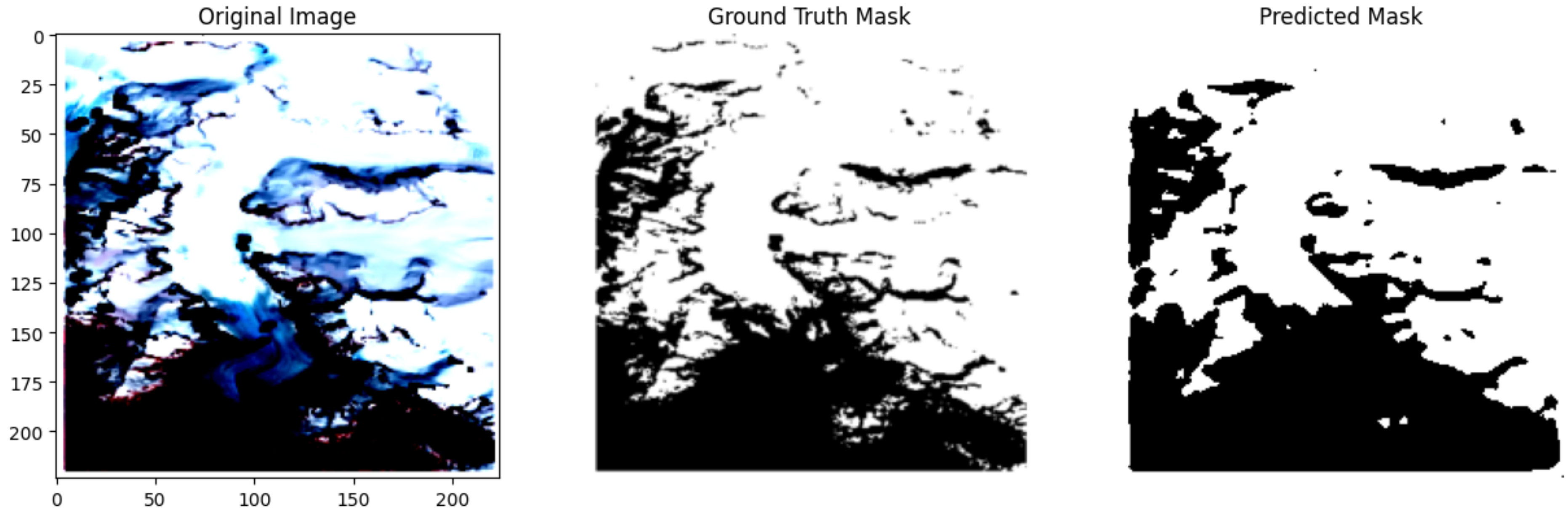
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa



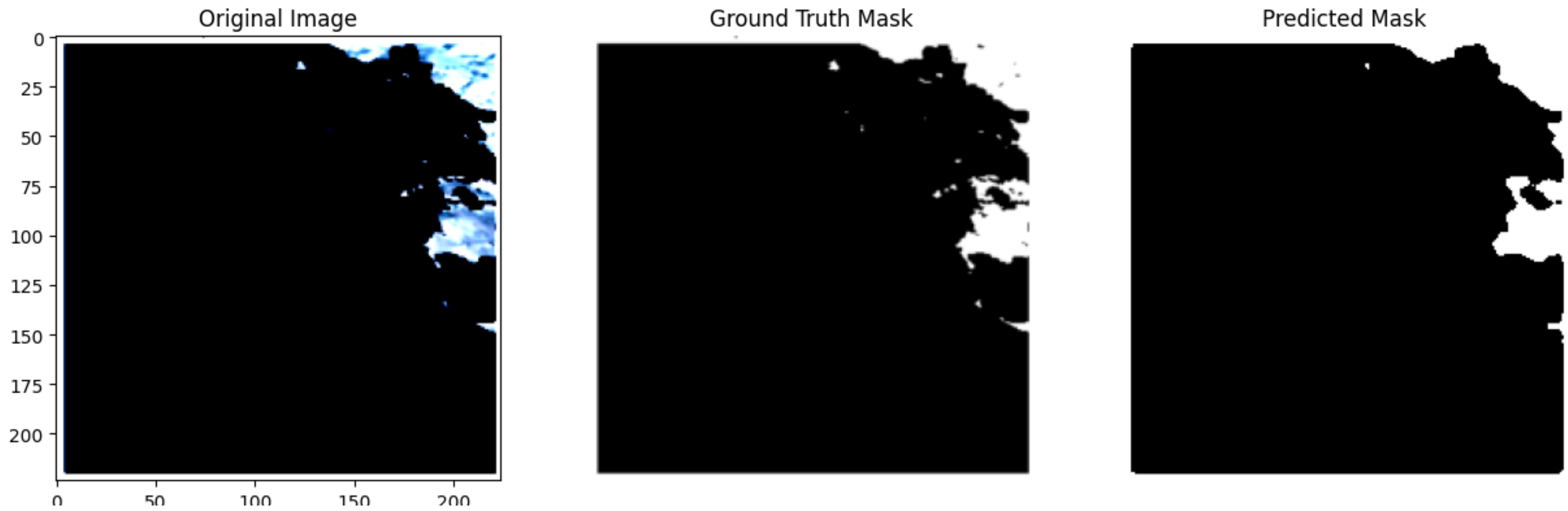
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa



WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa

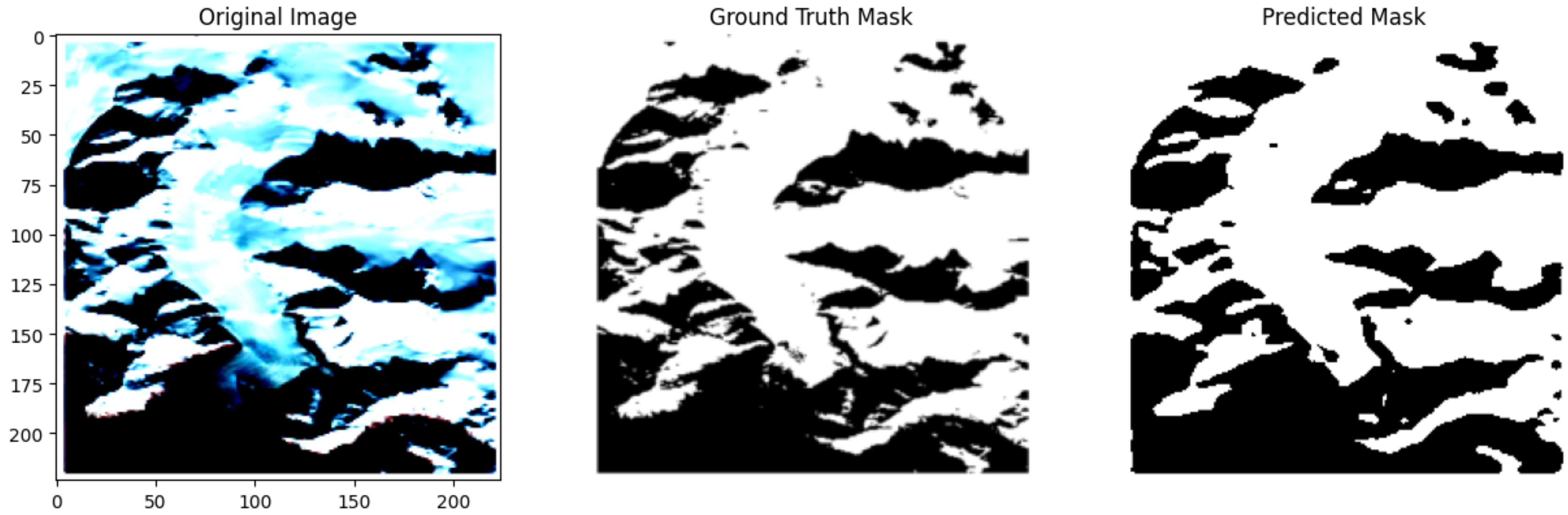


WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa

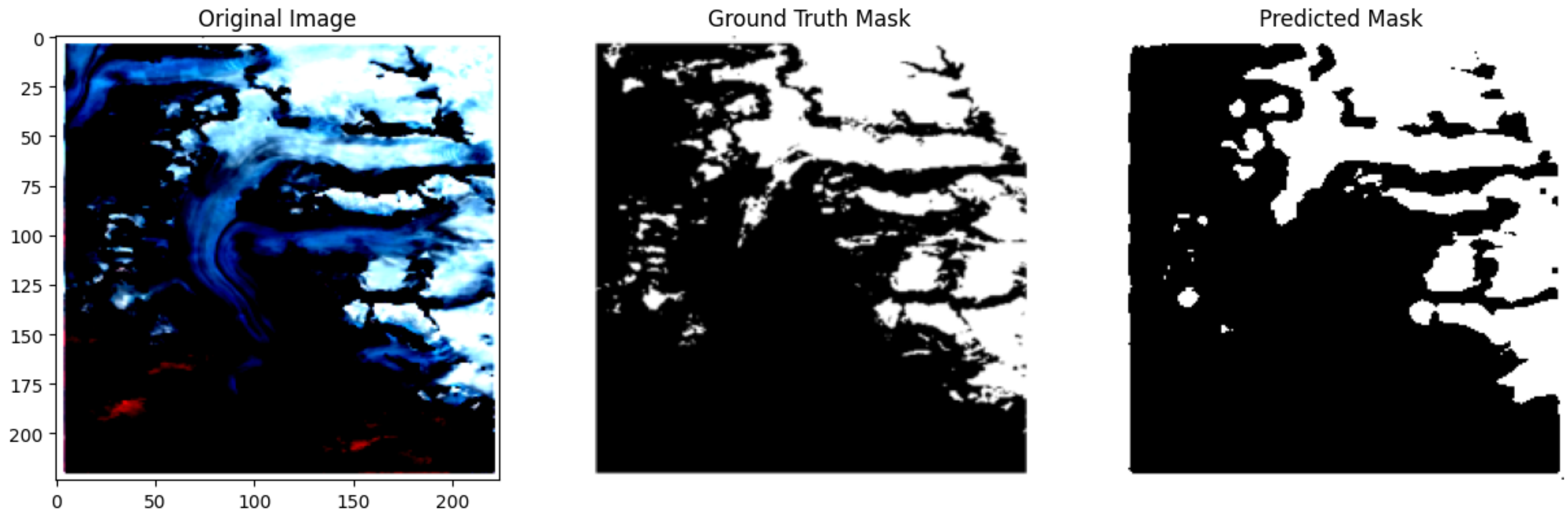


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

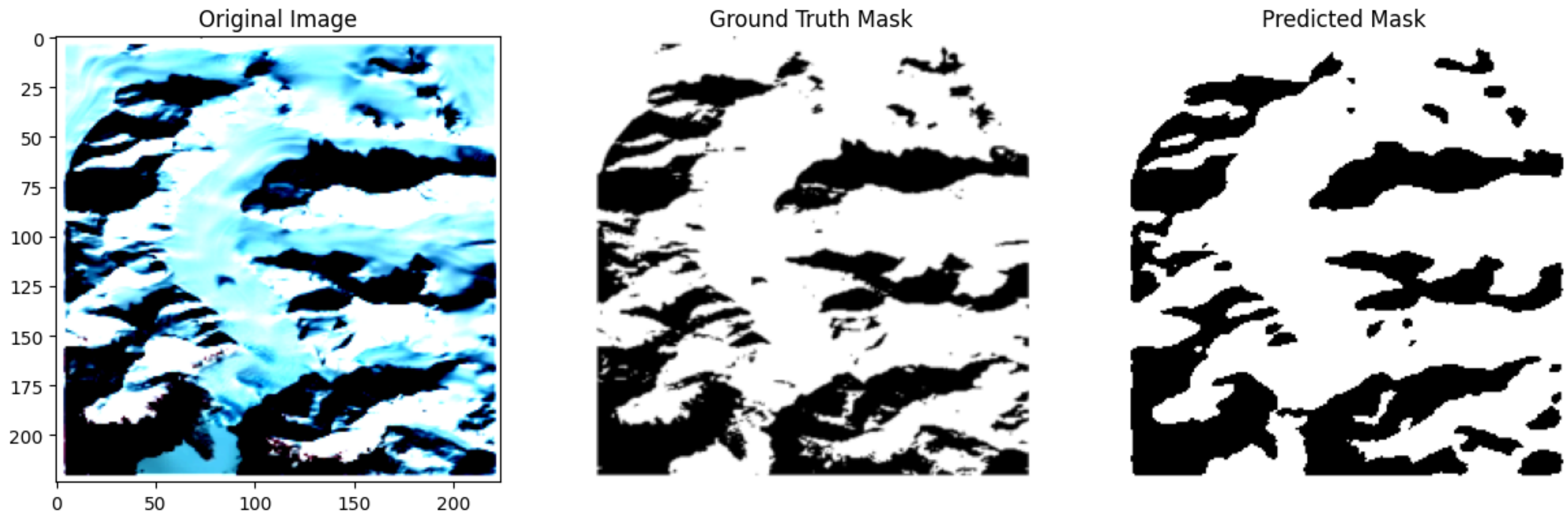

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

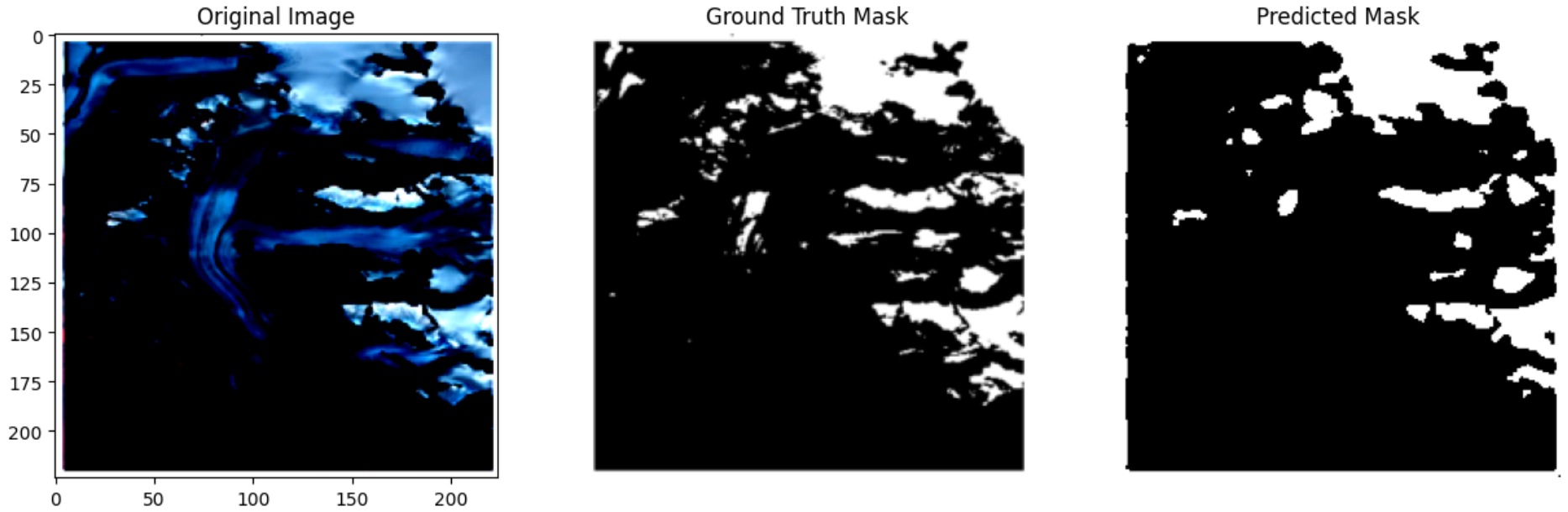



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

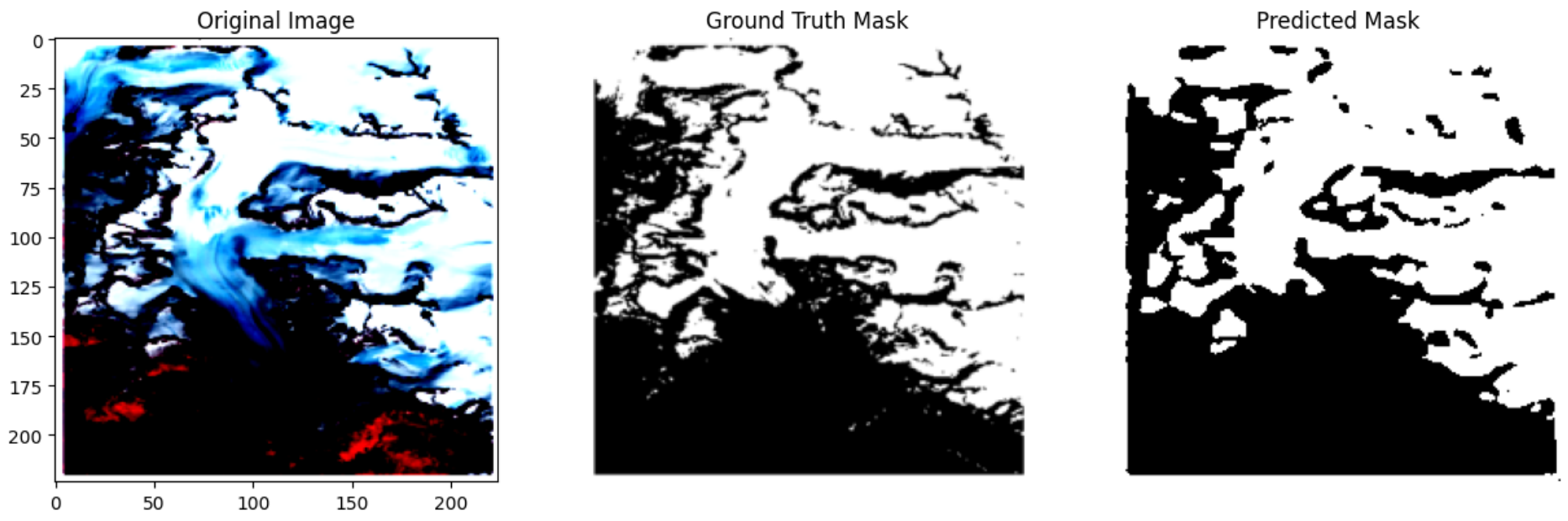



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

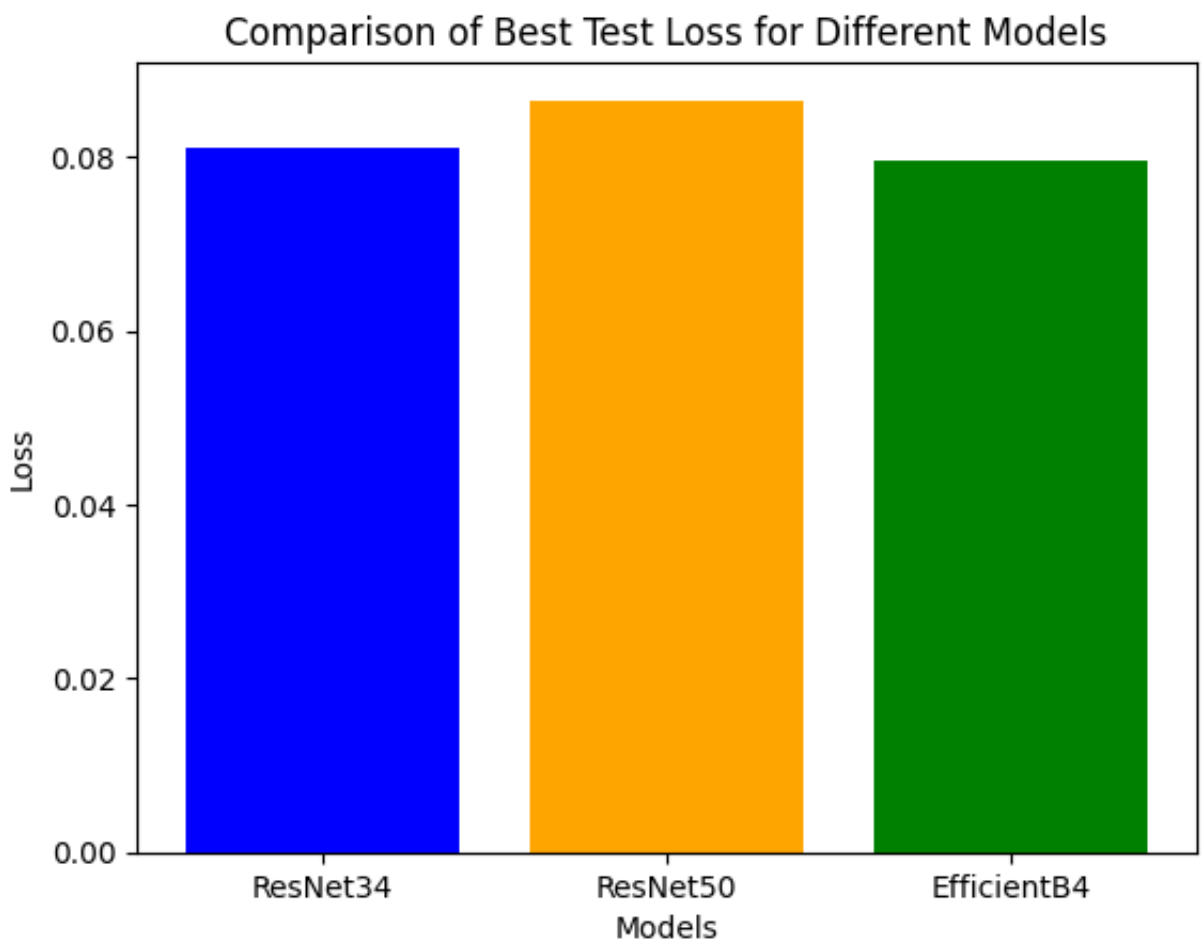



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



Comparison of All Three Models | Best Performing Model based on Loss Scores - ResNet34

```
# # Performance explanation using BarPlot:
# - ResNet34 performed best as it has sufficient capacity for the dataset's complexity while being
# - EfficientNet-B4 performed well due to its advanced design but may require more data or hyperpara
# - ResNet50 underperformed because its higher complexity likely led to overfitting on the available
loss_values = [resnet34_best_test_loss, resnet50_best_test_loss, efficientb4_best_test_loss]
models = ['ResNet34', 'ResNet50', 'EfficientB4']
plt.bar(models, loss_values, color=['blue', 'orange', 'green'])
plt.title('Comparison of Best Test Loss for Different Models')
plt.xlabel('Models')
plt.ylabel('Loss')
plt.show()
```



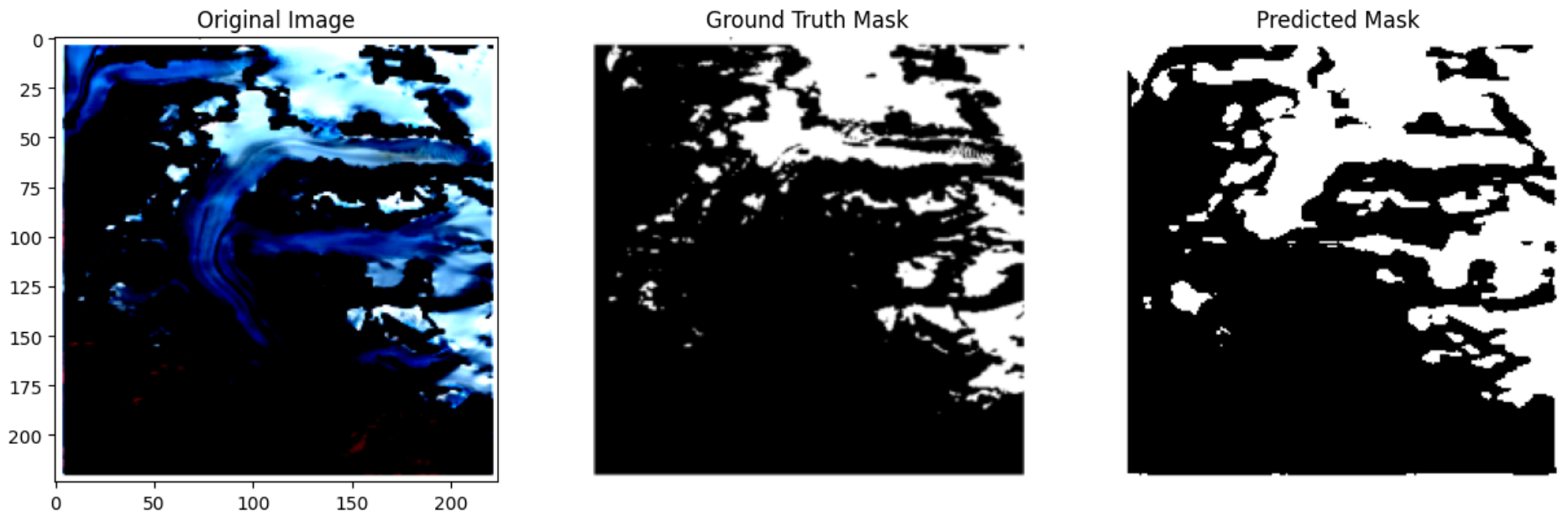

```
# printing the inidvidual values for clear comparison, as explained in the previous cell, resnet34 s
resnet34_best_test_loss, resnet50_best_test_loss, efficientb4_best_test_loss
```

```
(0.08100565274556477, 0.08649138609568278, 0.07955124974250793)
```

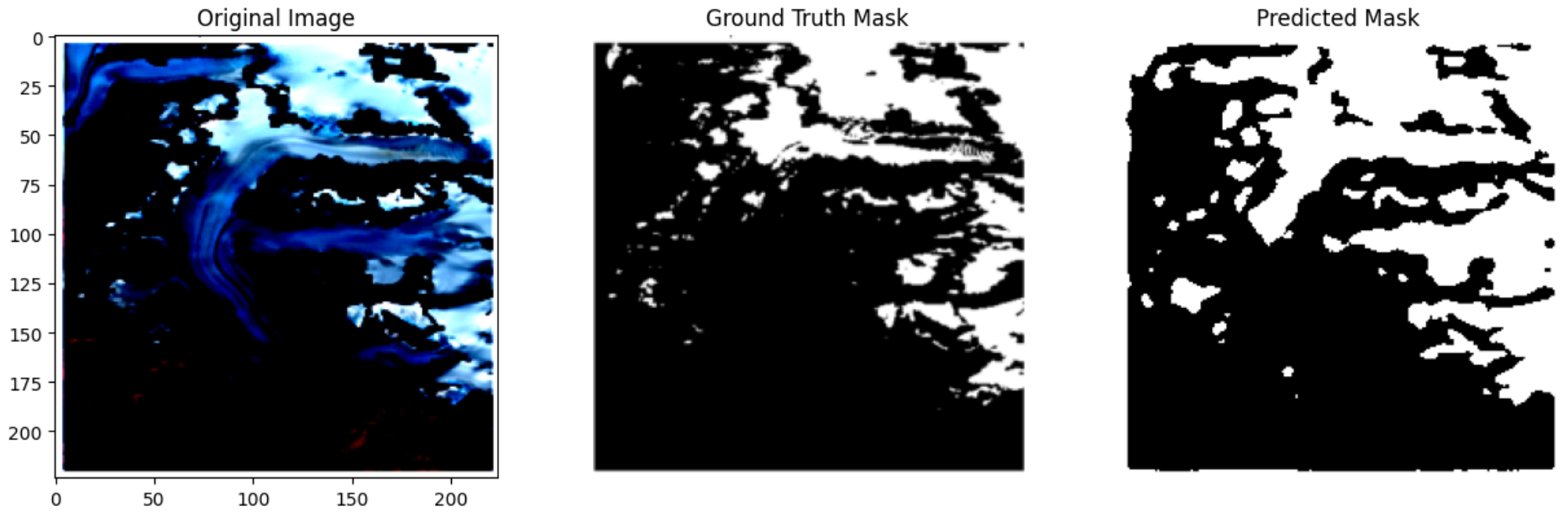
```
# comparing the predicted output of the best weights saved for all the three models on one sample te
# On closer obervation, it can be viewed that ResNet34 segments minor regions better followed by ef
test_dataset = SegmentationDataset([image_paths[81]], [mask_paths[81]])
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=True)

for encoder_name in ["resnet34", "resnet50", "efficientnet-b4"]:
  print(f'For encoder - {encoder_name}')
  model = smp.Unet(
    encoder_name=encoder_name,
    encoder_weights="imagenet",
    in_channels=3,
    classes=1
  )
  model = model.to(device)
  model.load_state_dict(torch.load(f'Best_model_{encoder_name}.pth'))
  test_image_plot(test_loader, model)
```

```
For encoder - resnet34
<ipython-input-27-62e9b54a6186>:15: FutureWarning: You are using `torch.load` with `weights_only=False` (
  model.load_state_dict(torch.load(f'Best_model_{encoder_name}.pth'))
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```

```
For encoder - resnet50
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```




```
For encoder - efficientnet-b4
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floa
```



Deployment Considerations

```
Here, the first step for deploying any model would be exporting it as .pth file, which is storing it
which as seen in the above code, I have already saved the models.

I have created a global loss variable, which keeps track of the lowest loss achieved across all cros
saved the best one, and everytime new minimum loss would be obtained, model for that step using torc
will be saved.

Then there are two types of deplpoyment possible:

(a) Cloud-based Deployment:

Possible Cloud Platforms: AWS(Sage Maker), Azure, and Google Cloud
I prefer Google Cloud because of its integration with BigQuery for data analytics, and support for (
the primary source of my dataset.

Steps to Deploy on Google Cloud Platform:

1) Set up your project and environment.
2) Create an image classification dataset, and import images.
3) Train an AutoML image classification model.
4) Evaluate and analyze model performance.
5) Deploy a model to an endpoint, and send a prediction.
6) Clean up your project.

I got these steps from this tutorial: https://cloud.google.com/vertex-ai/docs/tutorials/image-class

Alternatively, we can also choose it deploy it as an Docker:

A sample docker file would look something like this:

FROM pytorch/pytorch:latest

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt

CMD ["python", "inference_server.py"]

(b) Deploy as an API:

We may use Flask/Django to create an UI, and connect the backend with either our cloud platform or c
The reason I am suggesting Flask/Django is because both are Python-based and hence easy to integrate
For large-scale deployment, I would Django because of its additional security constraints, whereas f
Flask would be the easy-to-go option.

(c) Deploy on Edge-Device:

We may deploy the model on edge-devices like Raspberry-Pi or Jetson, though I would suggest using Je
because previously I have worked with both and from my experience I can say that raspberry-pi's dies
neural networks, while Jetsons provide performance similar to small GPUs.
```

Steps to make my model real-time:

```
1) Create data-fetching code for automatically retrieving latest images from GoogleEarthEngine for a
Here I have manually given the time-frame but instead of that I can give a specific range and retrie
Though some additional problems may arise: it is not necessary that everytime a file would arrive or
files for specific time periods, so I will have to sort all the files in the system as per datetime
keeping a fixed number.
2) Run that data-fetching code on cron-job, it is a linux tool, which runs a described pipleine evel
3) Then run the trained model in eval model to generate segmented masks.
4) Present the output to a user using some GUI, or store the segmented masks on some permanent stora



One more additional feature can be added:
1) Few features of the images may change over-time, hence our model need to capture those
new characteristics, and for that we should incorporate continual learning/increemental learning met
in our model.
Challenges in Continual Learning:
Catastrophic Forgetting: The phenomenon where a model forgets previously learned knowledge when trai
Scalability: Handling an ever-increasing amount of data without requiring large storage or computati
Data Distribution Shift: Adapting to changes in the underlying data distribution over time (non-stat
Efficient Memory Usage: Ensuring the model doesn't require storing all previous data, which might be
Methods to Incorporate Continual Learning:
Regularization-based methods (EWC, MAS).
Replay-based methods (Experience Replay, Generative Replay).
Dynamic architecture methods (Progressive Networks, PathNet).
Meta-learning methods (MAML, Reptile).
Neuro-inspired methods (Self-Organizing Maps, Hebbian Learning).
Knowledge distillation methods for transferring knowledge across task
```

Ethical Consideration

```
# Fairness:
Bias in Training Data: Here the model is trained on just the images from the data obtained over the a
from certain regions or specific types of glaciers, it could fail to generalize well to other areas o
or certain geographical regions, the model might underperform when applied to glaciers in less-repres
outcomes. Hence we need to make sure that the training datasets are diverse, representing a wide rang
identify any biases that may affect the model’s performance

# Societal Impact:

The use of machine learning models in glacier segmentation has significant societal implications, par
change and environmental monitoring. As glaciers play a critical role in the Earth's water cycle, acc
glacier dynamics can help predict future water availability, potential natural disasters, and even cl
the societal benefits are clear, several ethical issues need to be considered.

# Accuracy vs. Accountability:
If the model is used to make important environmental or policy decisions (e.g., regarding glacier mar
prevention), errors in the segmentation model could have large-scale consequences. The ethical dilemm
and transparency while avoiding overreliance on the AI system for high-stakes decision-making. Hence
continuous model monitoring, and human-in-the-loop mechanisms to ensure that the model’s predictions

# StakeHolder Effects:
Various stakeholders are involved in or affected by the development and deployment of glacier segment
organizations, policy makers, local communities, and industries (e.g., agriculture, tourism, or hydro

The information derived from glacier segmentation models may be used for resource management or envir
shared responsibly, it could lead to mismanagement of local resources or environmental degradation, e
local communities are included in the decision-making process, and any changes informed by model resu

Machine learning systems for environmental monitoring are often developed and deployed by large resea
interest of marginalized communities. These groups might lack access to the technology or the benefit
collaborative partnerships to ensure that smaller organizations, local governments, and vulnerable co

# Transparency and Accountability:
Machine learning models, especially deep learning models, are often seen as "black boxes" because it
glacier segmentation  model misidentifies certain areas as glaciers or non-glaciers, it is important
to improve transparency, allowing  stakeholders to understand how the model works and how decisions a

To implement this, softwares like SHAP/LIME can be used.

#Data Ownership and Consent:
In cases where drone-based or sensor-based data is used to capture information on specific glaciers,
and how is it shared?
Without proper consent, data could be used in ways that violate privacy rights or lead to the exploit
Here, for this project all the rights are reserved with GoogleEarthEngine.
```

```
Start coding or generate with AI.
```