crop_field_segmentation

May 2, 2024

1 Processing and analyzing satelite image dataset

In this notebook, we will analyze and process satelite images so we can pass them later into a convolutional neural network with the goal of predicting and finding agricultural fields in the given images

1.0.1 Imports

[96]:
```
import os, glob, random, shutil, csv
import numpy as np
import pandas as pd
from pandas.core.common import flatten
import matplotlib.pyplot as plt
import cv2
from PIL import Image
import csv
%matplotlib inline
import torch
import torch.nn.functional as F
from torch import optim, nn, utils, Tensor
from torch.utils.data import DataLoader, Dataset
from torch.utils.data.dataset import random_split
from google.colab import drive

import torch
import torch.nn.functional as F
from torch import optim, nn, utils, Tensor
from torch.utils.data import DataLoader, Dataset
from torch.utils.data.dataset import random_split

import torchvision
from torchvision.datasets import ImageFolder
from torchvision import transforms
from torchvision.io import read_image
```

1.1 Exploring our dataset

[97]:
```
drive.mount("/content/drive")
data_dir='/content/drive/MyDrive/satelite_image_nn/data'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[98]:
```
data_dir+"/train"
```

[98]:
```
'/content/drive/MyDrive/satelite_image_nn/data/train'
```

We need to get the training images paths so we can proccess them later

[99]:
```
metatada_file = data_dir + "/metadata.csv"


def get_images():

  training_image_paths = []
  test_image_paths = []

  with open(metatada_file) as file:
      csv_reader = csv.reader(file, delimiter=',')
      line_count = 0
      for row in csv_reader:

          if line_count != 0:

              if row[1] == 'train':
                image_path = data_dir + "/" + row[2]
                mask_path =  data_dir + "/" + row[3]
                training_image_paths.append((image_path, mask_path))
              elif row[1] == "test":
                image_path = data_dir + "/" + row[2]
                test_image_paths.append(image_path)

          line_count +=1

  return training_image_paths, test_image_paths


training_image_paths,test_image_paths = get_images()

print(f"we have {len(training_image_paths)} train images")
print(f"we have {len(test_image_paths)} test images")

print(training_image_paths[:5])
```

```
we have 803 train images
we have 172 test images
[('/content/drive/MyDrive/satelite_image_nn/data/train/100694_sat.jpg',
'/content/drive/MyDrive/satelite_image_nn/data/train/100694_mask.png'),
('/content/drive/MyDrive/satelite_image_nn/data/train/102122_sat.jpg',
'/content/drive/MyDrive/satelite_image_nn/data/train/102122_mask.png'),
('/content/drive/MyDrive/satelite_image_nn/data/train/10233_sat.jpg',
'/content/drive/MyDrive/satelite_image_nn/data/train/10233_mask.png'),
('/content/drive/MyDrive/satelite_image_nn/data/train/103665_sat.jpg',
'/content/drive/MyDrive/satelite_image_nn/data/train/103665_mask.png'),
('/content/drive/MyDrive/satelite_image_nn/data/train/103730_sat.jpg',
'/content/drive/MyDrive/satelite_image_nn/data/train/103730_mask.png')]
```

We are also going to create a dictionary with all of our labels, with their respected values.

[100]:
```python
labels_file = data_dir + "/class_dict.csv"

def get_classes():

  classes = {}

  with open(labels_file) as file:
      csv_reader = csv.reader(file, delimiter=',')
      line_count = 0
      for row in csv_reader:
          if line_count == 0:
              line_count +=1

              continue
          else:
              key =  (int(row[1]), int(row[2]), int(row[3]))
              classes[key] =  row[0]

              line_count +=1

  return classes

class_dict = get_classes()

classes = list(class_dict.values())

print(f"Classes are {classes}")
print(f"Label rgb values are {class_dict.keys()}")
```

```
Classes are ['urban_land', 'agriculture_land', 'rangeland', 'forest_land',
'water', 'barren_land', 'unknown']
Label rgb values are dict_keys([(0, 255, 255), (255, 255, 0), (255, 0, 255), (0,
255, 0), (0, 0, 255), (255, 255, 255), (0, 0, 0)])
```

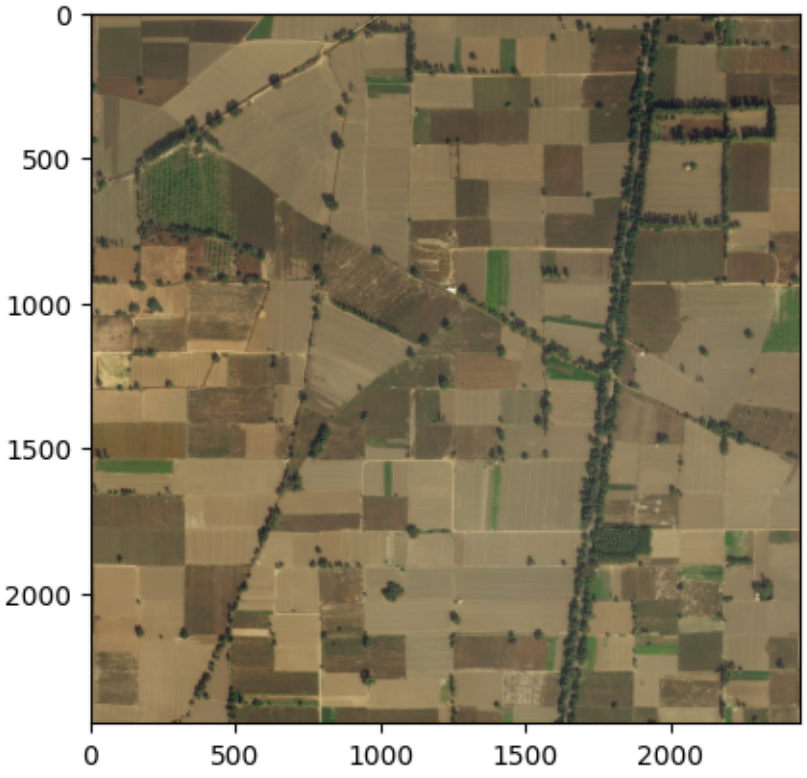
[101]:
```
classes_to_idx = {key:i for i,key in enumerate(classes)}
classes_to_idx
```

[101]:
```
{'urban_land': 0,
 'agriculture_land': 1,
 'rangeland': 2,
 'forest_land': 3,
 'water': 4,
 'barren_land': 5,
 'unknown': 6}
```

[102]:
```
satelite_image = cv2.imread(training_image_paths[0][0])
satelite_image = cv2.cvtColor(satelite_image, cv2.COLOR_BGR2RGB)

print(satelite_image.shape)
plt.imshow(satelite_image)
```

```
(2448, 2448, 3)
```

[102]:
```
<matplotlib.image.AxesImage at 0x79f67c7a1660>
```



1.2 Finding the classes of each image and saving the metadata to a new csv file

[103]:
```
def get_image_id(image_path):
  parts = image_path.split('/')

  filename = parts[-1]
  id_parts = filename.split('_')
  image_id = id_parts[0]

  return image_id
```

[104]:
```
file_path = data_dir+'/metadata_train.csv'
def save_metadata_file(file_path, class_dict, image_paths):


  if os.path.isfile(file_path):
    return

  classes = list(class_dict.values())

  file_cols = ['image_id', 'split', 'sat_image_path', 'mask_image_path'] +␣
 ↪classes


  with open(file_path, 'w', newline='') as file:

      writer = csv.writer(file)
      writer.writerow(file_cols)

      image_num = 1

      for sat_image, mask_image in image_paths:
          image = cv2.imread(mask_image)

          if image is None:
              print("Error loading image:", mask_image)
              continue

          image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
          image = cv2.resize(image, (256, 256))

          reshaped_image = image.reshape(-1, 3)
          unique_colors = np.unique(reshaped_image, axis=0)

          one_hot_classes = {class_name: 0 for class_name in classes}
```

```
            for color in unique_colors:
                color_tuple = tuple(color)

                if color_tuple in class_dict:

                    class_name = class_dict[color_tuple]
                    one_hot_classes[class_name] = 1


            one_hot_values = [0] * len(classes)
            for key,val in one_hot_classes.items():
              one_hot_values[classes_to_idx[key]] = val

            row = [get_image_id(mask_image), 'train', sat_image, mask_image] +␣
 ↪one_hot_values
            writer.writerow(row)

            if image_num % 100 == 0:
              print(f"Proessed {image_num} images")
            image_num += 1

     print("CSV file 'metadata_train.csv' created successfully.")



save_metadata_file(file_path, class_dict, training_image_paths)
```

1.3 Creating dataset for our cnn model

[105]:
```
metatada_file = data_dir + "/metadata_train.csv"
```

[106]:
```
anons_file = metatada_file
labels_df = pd.read_csv(anons_file)
labels_df.head()
```

[106]:

	image_id	split	sat_image_path	mask_image_path	urban_land
0	100694	train	/content/drive/MyDrive/satelite_image_nn/data/…	/content/drive/MyDrive/satelite_image_nn/data/…	0
1	102122	train	/content/drive/MyDrive/satelite_image_nn/data/…	/content/drive/MyDrive/satelite_image_nn/data/…	0
2	10233	train	/content/drive/MyDrive/satelite_image_nn/data/…	/content/drive/MyDrive/satelite_image_nn/data/…	1
3	103665	train	/content/drive/MyDrive/satelite_image_nn/data/…		
4	103730	train	/content/drive/MyDrive/satelite_image_nn/data/…		

```
3  /content/drive/MyDrive/satelite_image_nn/data/…           0
4  /content/drive/MyDrive/satelite_image_nn/data/…           1

   agriculture_land  rangeland  forest_land  water  barren_land  unknown
0                 1          1            0      0            0        0
1                 1          1            0      1            0        0
2                 1          0            0      1            0        0
3                 1          1            0      1            0        0
4                 1          1            0      0            0        0
```

[107]:
```
image_size = 256

train_transform = transforms.Compose([
    transforms.Resize((image_size, image_size)),
    transforms.ToTensor(),
    # transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
 ↪225])
])
```

[108]:
```
class SatelliteImagesDataset(Dataset):
    def __init__(self, annotations_file, classes, transform=None):
        self.labels_df = pd.read_csv(annotations_file)
        self.transform = transform
        self.classes = classes
        # Assuming each class is associated with a specific RGB value
        self.class_to_idx = {cls: idx for idx, cls in enumerate(self.classes)}
        self.rgb_to_class = {cls: rgb for cls, rgb in zip(self.classes,␣
 ↪class_dict.keys())}

    def __len__(self):
        return len(self.labels_df)

    def __getitem__(self, idx):
        img_path = self.labels_df.iloc[idx, 2]
        image = Image.open(img_path).convert('RGB')
        mask_path = self.labels_df.iloc[idx, 3]
        mask_image = cv2.imread(mask_path, cv2.IMREAD_COLOR)
        resized_image = cv2.resize(mask_image, (256, 256))  # Resize to 256x256
        mask_tensor = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)

        if self.transform:
            image = self.transform(image)

        mask_tensor = self.__process_mask(mask_tensor)  # Process and convert␣
 ↪mask

        return {'image': image, 'mask': torch.from_numpy(mask_tensor)}
```

```
    def __process_mask(self, mask):
      class_mask = np.zeros((mask.shape[0], mask.shape[1]), dtype=np.uint8)

      # Loop through each class and its associated RGB value
      for class_name, rgb_value in self.rgb_to_class.items():
        # Create a mask for the current RGB value
        current_mask = np.all(mask == np.array(rgb_value, dtype=np.uint8),␣
↪axis=-1)

        # Use the class index from `class_to_idx` to fill in the `class_mask`
        class_mask[current_mask] = self.class_to_idx[class_name]

      return class_mask
```

[109]:
```
class TestImagesDataset(Dataset):
    def __init__(self, annotations_file, classes, transform=None):
        print(annotations_file)
        self.labels_df = pd.read_csv(annotations_file)
        self.transform = transform
        self.classes = classes

    def __len__(self):
        return len(self.labels_df)

    def __getitem__(self, idx):
        img_path = self.labels_df.iloc[idx, 2]

        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return {'image': image}
```

[110]:
```
split_percentage = 0.8

train_image_count = int(len(training_image_paths) * split_percentage)
validation_image_count = (len(training_image_paths) - train_image_count)
test_image_count = len(test_image_paths)
print(f'Training images: {train_image_count}, Validation images:␣
↪{validation_image_count}, Testing Images: {test_image_count}')
```

```
Training images: 642, Validation images: 161, Testing Images: 172
```

[111]:
```
train_dataset = SatelliteImagesDataset(annotations_file=metatada_file,␣
 ↪classes=classes, transform=train_transform)
test_dataset = TestImagesDataset(annotations_file=data_dir + "/test_metadata.
 ↪csv", classes=classes, transform=train_transform)
```

```
/content/drive/MyDrive/satelite_image_nn/data/test_metadata.csv
```

[112]:
```
train_dataset, val_dataset = random_split(train_dataset,␣
 ↪lengths=[train_image_count, validation_image_count])

train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True,␣
 ↪num_workers=7)
val_dataloader = DataLoader(val_dataset, batch_size=16, num_workers=7)
test_dataloader = DataLoader(test_dataset, batch_size=16, num_workers=7)



print(f'Training dataset: {len(train_dataset)}')
print(f'Validation dataset: {len(val_dataset)}')
print(f'Test dataset: {len(test_dataset)}')


print(f'Classes: {train_dataset.dataset.classes}')
print(f'Transform: {train_dataset.dataset.transform}')
```

```
Training dataset: 642
Validation dataset: 161
Test dataset: 172
Classes: ['urban_land', 'agriculture_land', 'rangeland', 'forest_land', 'water',
'barren_land', 'unknown']
Transform: Compose(
    Resize(size=(256, 256), interpolation=bilinear, max_size=None,
antialias=True)
    ToTensor()
)
```

[113]:
```
train_batch = next(iter(train_dataloader))
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

[114]:
```
print(train_batch["image"].shape , train_batch["mask"].shape)
```

```
torch.Size([16, 3, 256, 256]) torch.Size([16, 256, 256])
```

2 Models

[115]:
```
!pip install lightning > /dev/null
!pip install torchinfo > /dev/null
```

[116]:
```
!pip install -U segmentation-models-pytorch > /dev/null
```

[117]:
```
import lightning as L
from lightning.pytorch.callbacks import ModelCheckpoint, EarlyStopping
from lightning.pytorch.loggers import CSVLogger

import segmentation_models_pytorch as smp
from segmentation_models_pytorch.encoders import get_preprocessing_fn
from segmentation_models_pytorch.datasets import SimpleOxfordPetDataset
import torch.nn.functional as F
```

[119]:
```
def colorize_mask(mask, color_map):
    if isinstance(mask, torch.Tensor):
        mask = mask.cpu().numpy()

    colored_mask = np.zeros((mask.shape[0], mask.shape[1], 3), dtype=np.uint8)
    for class_index, color in enumerate(color_map):
        colored_mask[mask == class_index] = color

    return colored_mask

color_map = [(0, 255, 255), (255, 255, 0), (255, 0, 255), (0, 255, 0), (0, 0,␣
 ↪255), (255, 255, 255), (0, 0, 0)]
```

[118]:
```
class SatelliteImageModel(L.LightningModule):

    def __init__(self, arch, encoder_name, in_channels, out_classes, lr,␣
 ↪pre_trained=False, optimizer="Adam"):
        '''
            Initializes an instance of the SemanticSegmentation class.

            Parameters:
            - arch (str): The architecture of the model.
            - encoder_name (str): The name of the encoder.
            - in_channels (int): The number of input channels.
            - out_classes (int): The number of output classes.
        '''
        super().__init__()

        '''
            It is important to note that we are not using a trained encoder. To␣
 ↪use a trained encoder, change
```

```
            The value in the encoder_weight parameter.
        '''
        encoder_weights = 'imagenet' if pre_trained else None
        self.model = smp.create_model(arch, encoder_name=encoder_name,␣
↪in_channels=in_channels, classes=out_classes,␣
↪encoder_weights=encoder_weights)

        '''
           These are used to store the computed true positive, false positive,␣
↪false negative and true negative 'pixels' for each image and class
        '''
        self.tp = None
        self.fp = None
        self.fn = None
        self.tn = None

        '''
            These are used to store the computed losses for each stage
        '''
        self.losses = {
            'valid': [],
            'train': [],
            'test': []
        }

        '''
            These parameters are used to normalize the input image. Models from␣
↪the segmentation_models_pytorch library
            require the input image to be normalized. The mean and standard␣
↪deviation values are used to normalize the image.
            The mean and standard deviation values are obtained from the␣
↪get_preprocessing_params function of the encoders module.
        '''
        params = smp.encoders.get_preprocessing_params(encoder_name)
        self.register_buffer("std", torch.tensor(params["std"]).view(1, 3, 1,␣
↪1))
        self.register_buffer("mean", torch.tensor(params["mean"]).view(1, 3, 1,␣
↪1))

        '''
            Dice Loss: Dice loss, also known as the Sørensen-Dice coefficient,␣
↪is a popular choice for image segmentation.
            It measures the overlap between the predicted and target␣
↪segmentation masks.
        '''
```

```
        self.loss_fn = smp.losses.DiceLoss(smp.losses.MULTICLASS_MODE,␣
↪from_logits=True)
        self.out_classes = out_classes
        self.lr = lr
        self.optimizer = optimizer

    def forward(self, image):

        # normalize image here
        image = (image - self.mean) / self.std
        mask = self.model(image)
        return mask

    def shared_step(self, batch, stage):

        image = batch["image"]

        '''
            Shape of the image should be (batch_size, num_channels, height,␣
↪width)
            if you work with grayscale images, expand channels dim to have␣
↪[batch_size, 1, height, width]
        '''
        assert image.ndim == 4

        '''
            Check that image dimensions are divisible by 32,
            encoder and decoder connected by `skip connections` and usually␣
↪encoder have 5 stages of
            downsampling by factor 2 (2 ^ 5 = 32); e.g. if we have image with␣
↪shape 65x65 we will have
            following shapes of features in encoder and decoder: 84, 42, 21,␣
↪10, 5 -> 5, 10, 20, 40, 80
            and we will get an error trying to concat these features
        '''
        h, w = image.shape[2:]
        assert h % 32 == 0 and w % 32 == 0

        mask = batch["mask"]
        mask = mask.long()

        # Shape of the mask should be [batch_size, height, width] for binary␣
↪segmentation num_classes = 1
        assert mask.ndim == 3

        assert mask.max() <= 6 and mask.min() >= 0
```

```
        logits_mask = self.forward(image)

        loss = self.loss_fn(logits_mask, mask.squeeze(1))

        pred_mask = torch.argmax(logits_mask, dim=1)

        '''
            We will compute IoU metric by two ways
              1. dataset-wise
              2. image-wise
            but for now we just compute true positive, false positive, false␣
↪negative and
            true negative 'pixels' for each image and class
            these values will be aggregated in the end of an epoch
        '''
        tp, fp, fn, tn = smp.metrics.get_stats(pred_mask.long(), mask.long(),␣
↪mode="multiclass", num_classes=self.out_classes)

        self.tp = tp
        self.fp = fp
        self.tn = tn
        self.fn = fn

        self.losses[stage].append(loss)

        return loss

    def shared_epoch_end(self, stage):
        # aggregate step metics
        tp = self.tp
        fp = self.fp
        fn = self.fn
        tn = self.tn

        '''
            Per image IoU means that we first calculate IoU score for each image
            and then compute mean over these scores
        '''
        per_image_iou = smp.metrics.iou_score(tp, fp, fn, tn,␣
↪reduction="micro-imagewise")

        '''
            Dataset IoU means that we aggregate intersection and union over␣
↪whole dataset
```

```
            and then compute IoU score. The difference between dataset_iou and␣
↪per_image_iou scores
            in this particular case will not be much, however for dataset
            with "empty" images (images without target class) a large gap could␣
↪be observed.
            Empty images influence a lot on per_image_iou and much less on␣
↪dataset_iou.
        '''
        dataset_iou = smp.metrics.iou_score(tp, fp, fn, tn, reduction="micro")

        metrics = {
            f"{stage}_loss": torch.stack(self.losses[stage]).mean(),
            f"{stage}_per_image_iou": per_image_iou,
            f"{stage}_dataset_iou": dataset_iou,

        }

        self.log_dict(metrics, prog_bar=True, logger=True)

    '''
        These are pytorch lightning hooks that are called during the training,␣
↪validation and testing steps.
        Hooks are used to perform additional operations during these steps. In␣
↪this case, we are using the hooks
        to compute the loss and metrics for each step and epoch.
    '''
    def training_step(self, batch):
        return self.shared_step(batch, "train")

    def on_train_epoch_end(self):
        return self.shared_epoch_end("train")

    def validation_step(self, batch):
        return self.shared_step(batch, "valid")

    def on_validation_epoch_end(self):
        return self.shared_epoch_end("valid")

    def test_step(self, batch):
        return self.shared_step(batch, "test")

    def on_test_epoch_end(self):
        return self.shared_epoch_end("test")

    def configure_optimizers(self):
        if self.optimizer == "Adam":
          return torch.optim.Adam(self.parameters(), lr=self.lr)
```

```
        elif self.optimizer == "AdamW":
          return torch.optim.Adam(self.parameters(), lr=self.lr)
```

2.0.1 U-Net++

[ ]:
```
unetpp = SatelliteImageModel("unetplusplus", "resnet34", in_channels=3,␣
↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="Adam")
```

```
Downloading: "https://download.pytorch.org/models/resnet34-333f7ec4.pth" to
/root/.cache/torch/hub/checkpoints/resnet34-333f7ec4.pth
100%|      | 83.3M/83.3M [00:00<00:00, 126MB/s]
```

[ ]:
```
# Early stop is a callback that is used to stop the training process when the␣
↪validation loss does not improve. In this case, we are
# using the EarlyStopping callback to stop the training process when the␣
↪validation loss does not improve for 3 epochs.
earlystop_callback = EarlyStopping('valid_loss', patience=3)

trainer = L.Trainer(max_epochs=40, logger=CSVLogger(save_dir="logs/",␣
↪name="pets_seg-model"), callbacks=[earlystop_callback])

trainer.fit(unetpp, train_dataloaders=train_dataloader,␣
↪val_dataloaders=val_dataloader)
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HPUs
WARNING: Missing logger folder: logs/pets_seg-model
WARNING:lightning.fabric.loggers.csv_logs:Missing logger folder: logs/pets_seg-
model
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:
  | Name    | Type         | Params
-----------------------------------------
0 | model   | UnetPlusPlus | 26.1 M
1 | loss_fn | DiceLoss     | 0
-----------------------------------------
26.1 M    Trainable params
```

```
0         Non-trainable params
26.1 M    Total params
104.318   Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name    | Type         | Params
-----------------------------------------
0 | model   | UnetPlusPlus | 26.1 M
1 | loss_fn | DiceLoss     | 0
-----------------------------------------
26.1 M    Trainable params
0         Non-trainable params
26.1 M    Total params
104.318   Total estimated model params size (MB)

Sanity Checking: |          | 0/? [00:00<?, ?it/s]

/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
/usr/local/lib/python3.10/dist-packages/lightning/pytorch/loops/fit_loop.py:298:
The number of training batches (41) is smaller than the logging interval
Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you
want to see logs for the training epoch.

Training: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

INFO: `Trainer.fit` stopped: `max_epochs=40` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=40` reached.
```

[ ]:

```
valid_metrics = trainer.validate(unetpp, dataloaders=val_dataloader,␣
↪verbose=False)
print(valid_metrics)
```

```
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]

Validation: |          | 0/? [00:00<?, ?it/s]

[{'valid_loss': 0.353802889585495, 'valid_per_image_iou': 0.8241180181503296,
'valid_dataset_iou': 0.8241180181503296}]
```

[ ]:
```python
# Read the metrics.csv file generated by the PyTorch Lightning logger
metrics = pd.read_csv(f"{trainer.logger.log_dir}/metrics.csv")

# Group the metrics by epoch and compute the mean loss for each epoch
df_epochs = metrics.groupby('epoch').mean()

display(df_epochs)

# Create a figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
# Set the x-axis label
ax.set_xlabel('Epochs')
# Set the y-axis label
ax.set_ylabel('Loss')
ax.plot(df_epochs['train_loss'], label="Training loss")
ax.plot(df_epochs['valid_loss'], label="Validation loss")

ax.plot(df_epochs['train_dataset_iou'], label="Training Dataset IoU")
ax.plot(df_epochs['valid_dataset_iou'], label="Validation Dataset IoU")

# Plot the training loss over epochs
# Plot the validation loss over epochs
# Set the title of the plot
ax.set_title("Training Loss")
# Add a legend to the plot
ax.legend(loc='upper right')
```

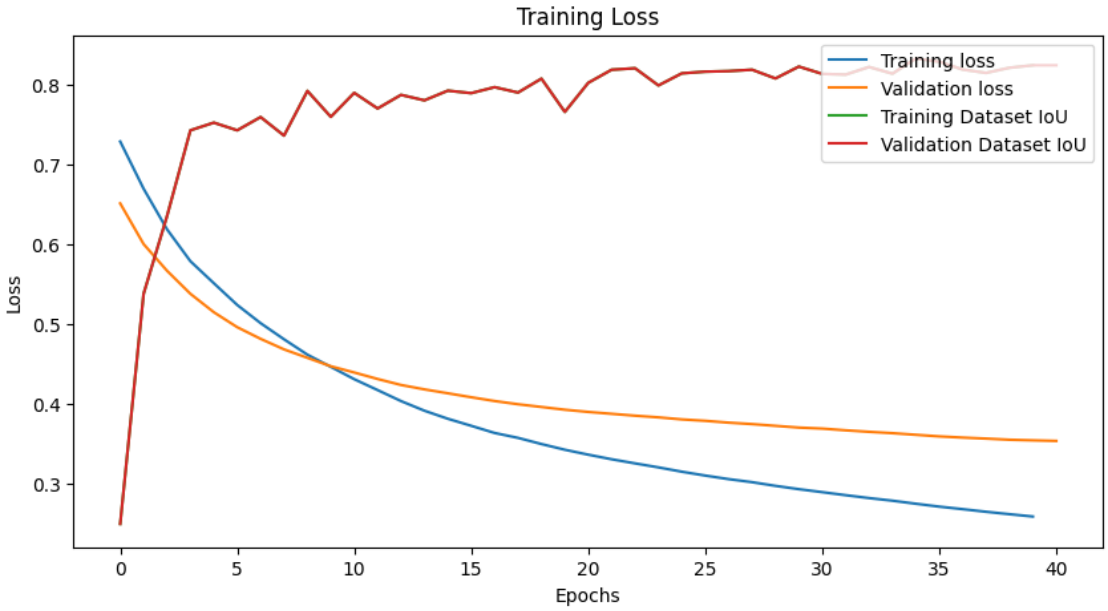
epoch	step	train_dataset_iou	train_loss	train_per_image_iou	\
0	40.0	0.249959	0.728691	0.249959	
1	81.0	0.538025	0.669530	0.538025	
2	122.0	0.635232	0.618767	0.635232	
3	163.0	0.742677	0.578786	0.742677	
4	204.0	0.752276	0.551275	0.752276	
5	245.0	0.742701	0.524043	0.742701	
6	286.0	0.759285	0.501248	0.759285	
7	327.0	0.736053	0.481074	0.736053	
8	368.0	0.791996	0.461820	0.791996	
9	409.0	0.759545	0.446535	0.759545	
10	450.0	0.789599	0.431116	0.789599	

11	491.0	0.770023	0.417477	0.770023
12	532.0	0.787061	0.403693	0.787061
13	573.0	0.780217	0.391644	0.780217
14	614.0	0.792340	0.381566	0.792340
15	655.0	0.789232	0.372740	0.789232
16	696.0	0.796713	0.363775	0.796713
17	737.0	0.789965	0.357628	0.789965
18	778.0	0.807416	0.349928	0.807416
19	819.0	0.765731	0.342789	0.765731
20	860.0	0.802148	0.336614	0.802148
21	901.0	0.818550	0.330857	0.818550
22	942.0	0.820419	0.325707	0.820419
23	983.0	0.798883	0.320704	0.798883
24	1024.0	0.814045	0.315285	0.814045
25	1065.0	0.816131	0.310451	0.816131
26	1106.0	0.816836	0.305985	0.816836
27	1147.0	0.818575	0.302181	0.818575
28	1188.0	0.807615	0.297567	0.807615
29	1229.0	0.822495	0.293414	0.822495
30	1270.0	0.813342	0.289678	0.813342
31	1311.0	0.812440	0.285890	0.812440
32	1352.0	0.821988	0.282264	0.821988
33	1393.0	0.813618	0.278981	0.813618
34	1434.0	0.832662	0.275176	0.832662
35	1475.0	0.828597	0.271542	0.828597
36	1516.0	0.818802	0.268391	0.818802
37	1557.0	0.814572	0.265113	0.814572
38	1598.0	0.821026	0.262082	0.821026
39	1639.0	0.824118	0.259163	0.824118
40	1640.0	NaN	NaN	NaN

epoch	valid_dataset_iou	valid_loss	valid_per_image_iou
0	0.249959	0.651202	0.249959
1	0.538025	0.600256	0.538025
2	0.635232	0.566868	0.635232
3	0.742677	0.538013	0.742677
4	0.752276	0.514971	0.752276
5	0.742701	0.496213	0.742701
6	0.759285	0.481610	0.759285
7	0.736053	0.468435	0.736053
8	0.791996	0.457843	0.791996
9	0.759545	0.447293	0.759545
10	0.789599	0.439496	0.789599
11	0.770023	0.431348	0.770023
12	0.787061	0.423854	0.787061
13	0.780217	0.418363	0.780217
14	0.792340	0.413469	0.792340

15	0.789232	0.408495	0.789232
16	0.796713	0.403755	0.796713
17	0.789965	0.399734	0.789965
18	0.807416	0.396318	0.807416
19	0.765731	0.392886	0.765731
20	0.802148	0.390110	0.802148
21	0.818550	0.387761	0.818550
22	0.820419	0.385404	0.820419
23	0.798883	0.383326	0.798883
24	0.814045	0.380757	0.814045
25	0.816131	0.378982	0.816131
26	0.816836	0.376691	0.816836
27	0.818575	0.374896	0.818575
28	0.807615	0.372704	0.807615
29	0.822495	0.370517	0.822495
30	0.813342	0.369262	0.813342
31	0.812440	0.367101	0.812440
32	0.821988	0.365215	0.821988
33	0.813618	0.363534	0.813618
34	0.832662	0.361495	0.832662
35	0.828597	0.359485	0.828597
36	0.818802	0.357990	0.818802
37	0.814572	0.356738	0.814572
38	0.821026	0.355294	0.821026
39	0.824118	0.354530	0.824118
40	0.824118	0.353803	0.824118

[ ]: <matplotlib.legend.Legend at 0x7f15dccc3e80>



[ ]:
```
batch = next(iter(val_dataloader))
with torch.no_grad():
    unetpp.eval()
    logits = unetpp(batch["image"])
pr_masks = logits.sigmoid()

for image, gt_mask, pr_mask in zip(batch["image"], batch["mask"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Ground truth
    plt.subplot(1, 3, 2)
    plt.imshow(colorize_mask(gt_mask, color_map))
    plt.title("Ground truth")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

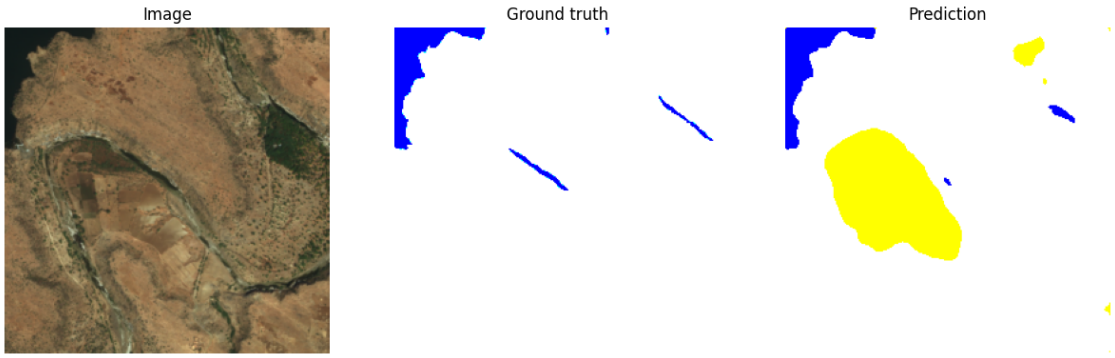



Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

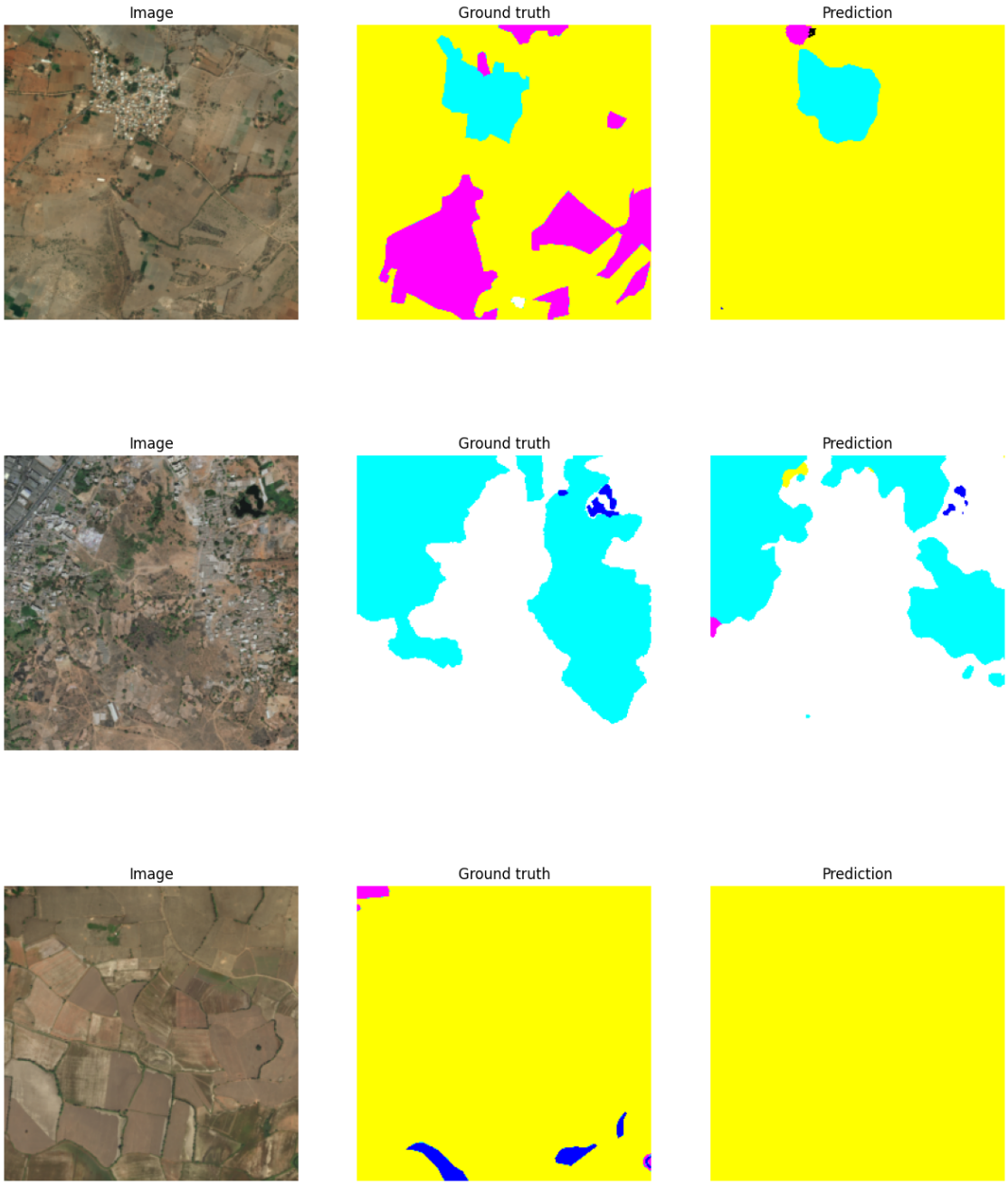


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

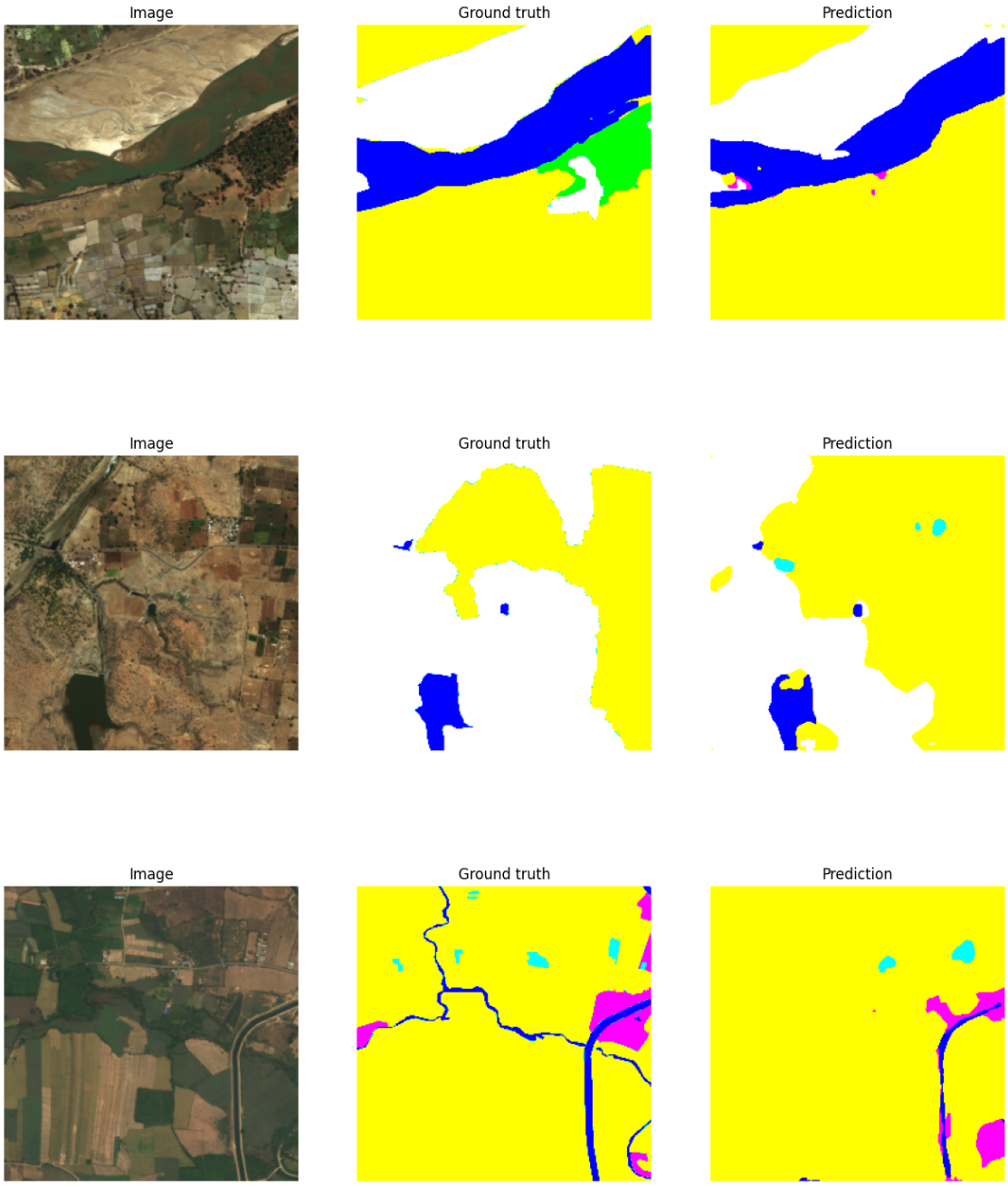
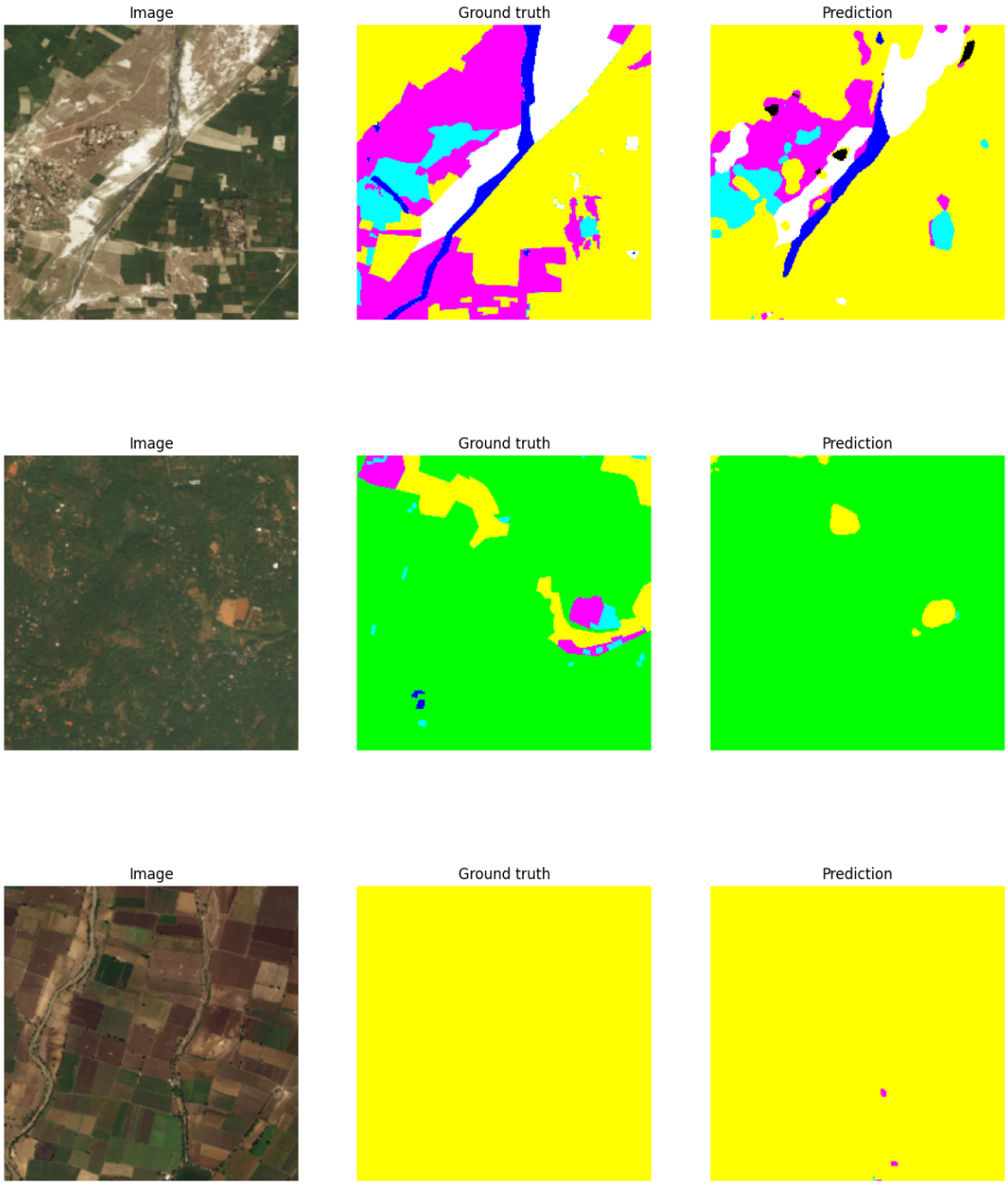


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction



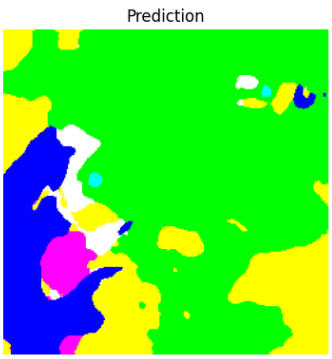
Save Model

```
[ ]: torch.save(unetpp.state_dict(), '/content/drive/MyDrive/satelite_image_nn/
     ↪models/unet++_resnet34.pth')
```

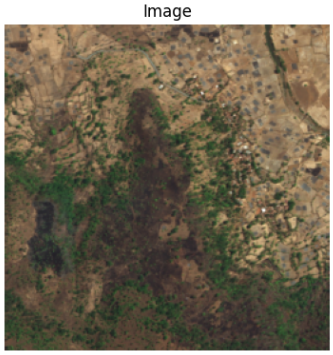
Test Model

[120]:
```
model1 = SatelliteImageModel("unetplusplus", "resnet34", in_channels=3,␣
 ↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="Adam")
model1.load_state_dict(torch.load('/content/drive/MyDrive/satelite_image_nn/
 ↪models/unet++_resnet34.pth'))

batch = next(iter(test_dataloader))
with torch.no_grad():
    model1.eval()
    logits = model1(batch["image"])
pr_masks = logits.sigmoid()

for image, pr_mask in zip(batch["image"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



Image



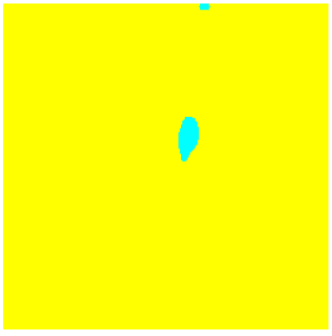
Prediction

Image

Prediction

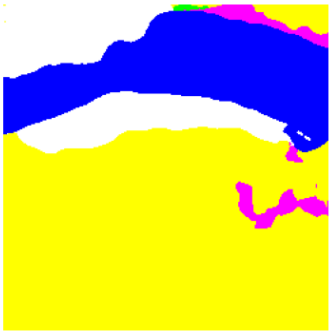
Image

Prediction

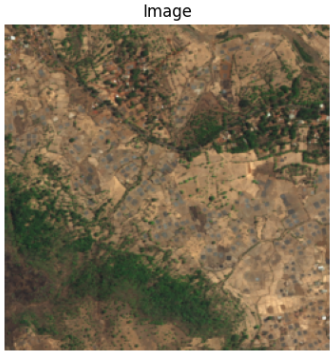


Image

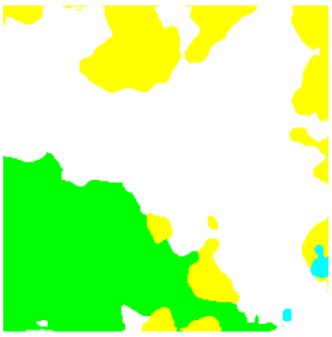
Prediction



Image

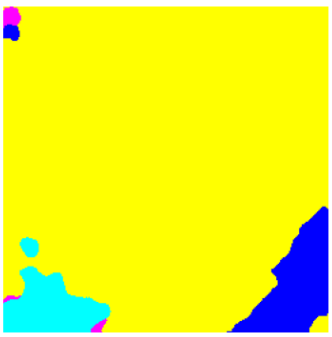


Prediction



Image

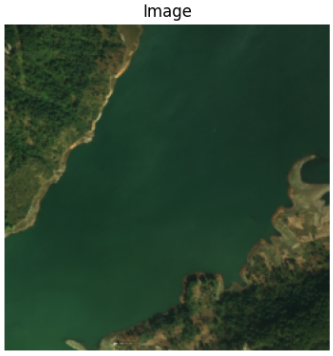
Prediction



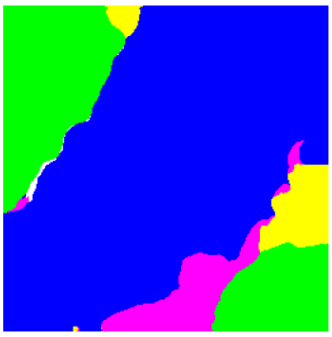
Image

Prediction

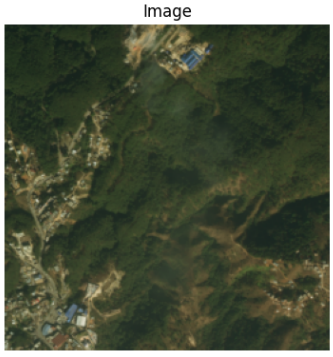
Image



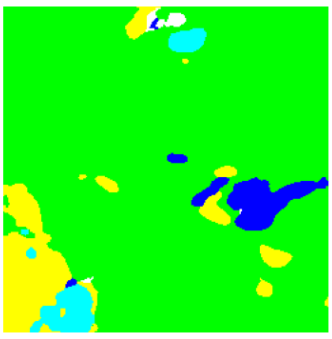
Prediction



Image



Prediction

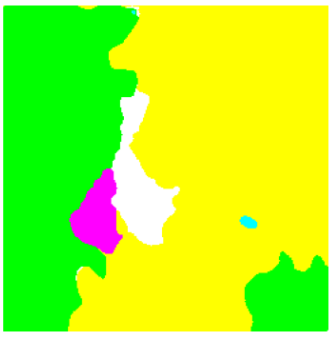


Image

Prediction

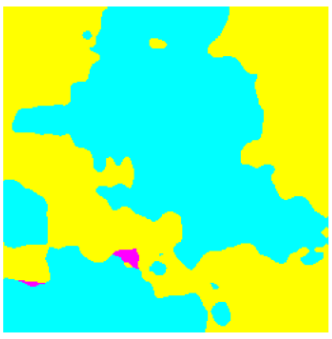
Image

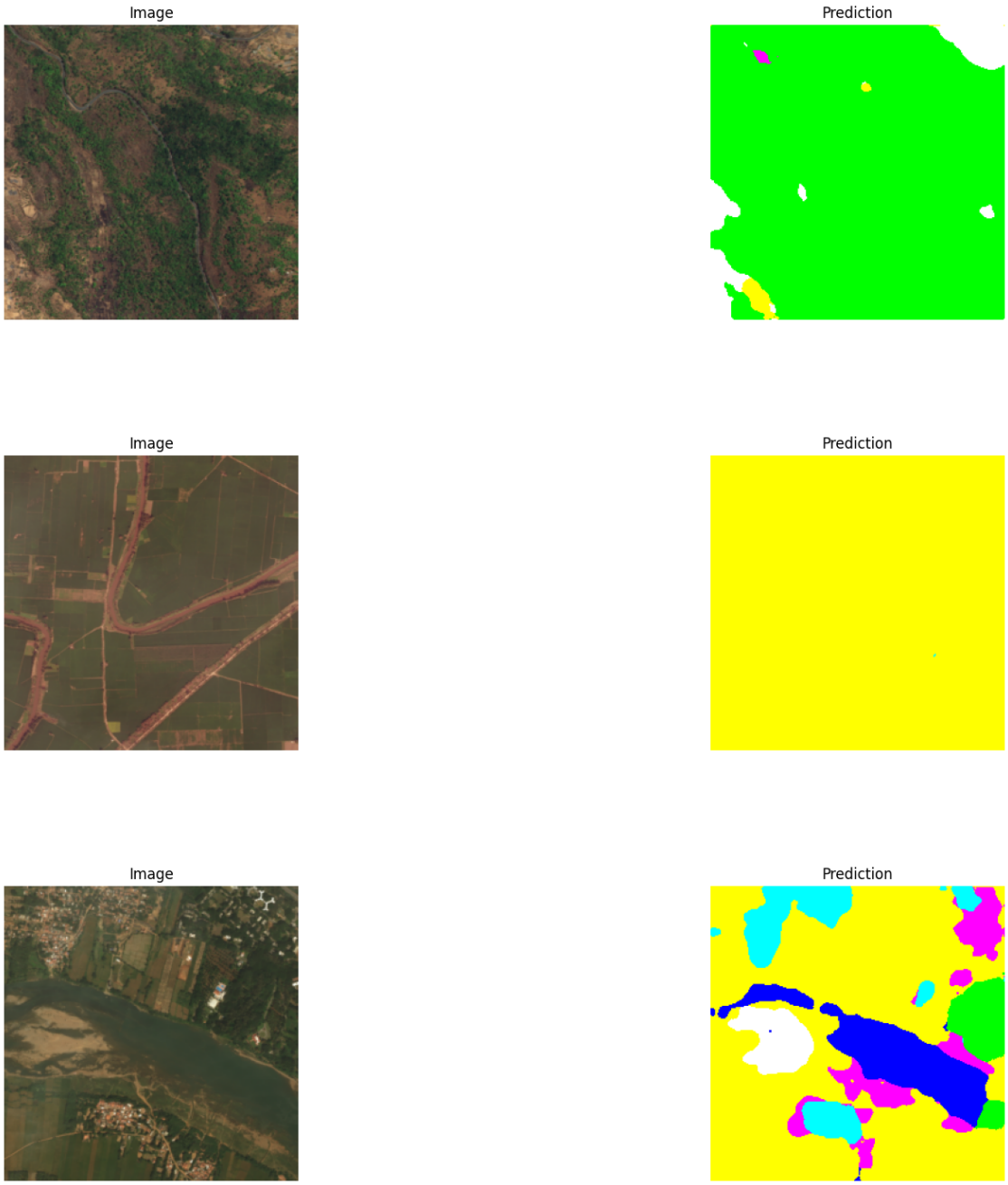
Prediction



Image

Prediction





2.0.2 Deeplabv3

```
[ ]: deeplabv3 = SatelliteImageModel("deeplabv3", "resnet152", in_channels=3,␣
      ↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="Adam")
```

```
Downloading: "https://download.pytorch.org/models/resnet152-b121ed2d.pth" to
/root/.cache/torch/hub/checkpoints/resnet152-b121ed2d.pth
100%|      | 230M/230M [00:01<00:00, 148MB/s]
```

[ ]:
```
# Early stop is a callback that is used to stop the training process when the␣
 ↪validation loss does not improve. In this case, we are
# using the EarlyStopping callback to stop the training process when the␣
 ↪validation loss does not improve for 3 epochs.
earlystop_callback = EarlyStopping('valid_loss', patience=3)

trainer = L.Trainer(max_epochs=20, logger=CSVLogger(save_dir="logs/",␣
 ↪name="pets_seg-model"), callbacks=[earlystop_callback])

trainer.fit(deeplabv3, train_dataloaders=train_dataloader,␣
 ↪val_dataloaders=val_dataloader)
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:
  | Name    | Type      | Params
-------------------------------------
0 | model   | DeepLabV3 | 74.3 M
1 | loss_fn | DiceLoss  | 0
-------------------------------------
74.3 M    Trainable params
0         Non-trainable params
74.3 M    Total params
297.084   Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name    | Type      | Params
-------------------------------------
0 | model   | DeepLabV3 | 74.3 M
1 | loss_fn | DiceLoss  | 0
-------------------------------------
74.3 M    Trainable params
0         Non-trainable params
74.3 M    Total params
297.084   Total estimated model params size (MB)
```

```
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
```

```
/usr/local/lib/python3.10/dist-packages/lightning/pytorch/loops/fit_loop.py:298:
The number of training batches (41) is smaller than the logging interval
Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you
want to see logs for the training epoch.

Training: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

INFO: `Trainer.fit` stopped: `max_epochs=20` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=20` reached.
```

[ ]:
```python
valid_metrics = trainer.validate(deeplabv3, dataloaders=val_dataloader,␣
  ↪verbose=False)
print(valid_metrics)
```

```
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
```

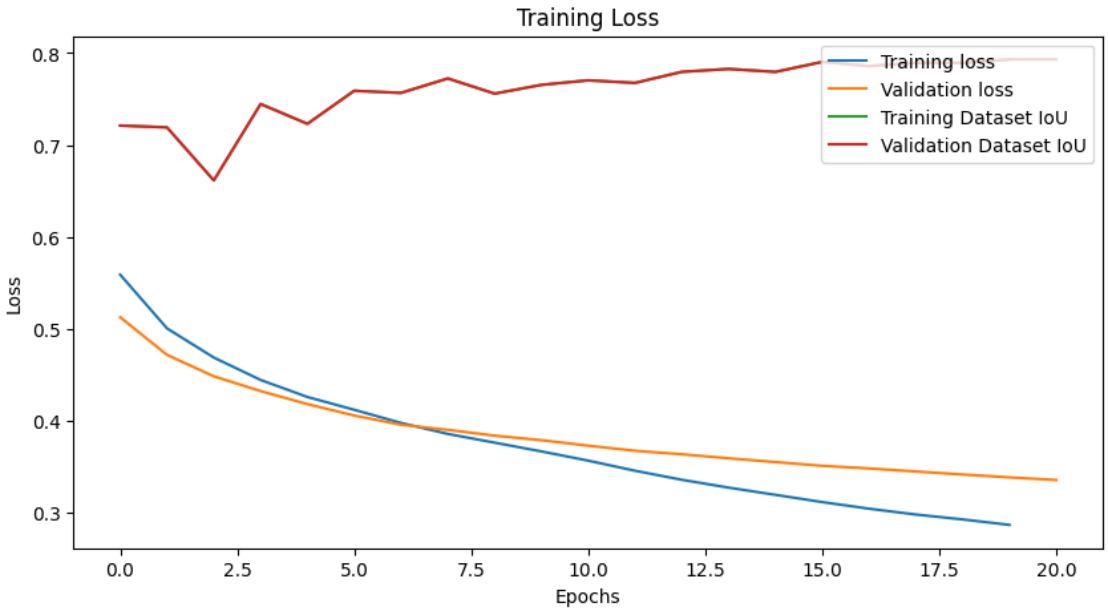
```
[{'valid_loss': 0.3357509970664978, 'valid_per_image_iou': 0.7930751442909241,
'valid_dataset_iou': 0.7930751442909241}]
```

[ ]:
```python
# Read the metrics.csv file generated by the PyTorch Lightning logger
metrics = pd.read_csv(f"{trainer.logger.log_dir}/metrics.csv")

# Group the metrics by epoch and compute the mean loss for each epoch
df_epochs = metrics.groupby('epoch').mean()

display(df_epochs)

# Create a figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
# Set the x-axis label
ax.set_xlabel('Epochs')
# Set the y-axis label
ax.set_ylabel('Loss')
ax.plot(df_epochs['train_loss'], label="Training loss")
ax.plot(df_epochs['valid_loss'], label="Validation loss")

ax.plot(df_epochs['train_dataset_iou'], label="Training Dataset IoU")
ax.plot(df_epochs['valid_dataset_iou'], label="Validation Dataset IoU")

# Plot the training loss over epochs
# Plot the validation loss over epochs
# Set the title of the plot
ax.set_title("Training Loss")
# Add a legend to the plot
ax.legend(loc='upper right')
```

epoch	step	train_dataset_iou	train_loss	train_per_image_iou	\
0	40.0	0.721167	0.559003	0.721167	
1	81.0	0.719180	0.500624	0.719180	
2	122.0	0.661537	0.468898	0.661537	
3	163.0	0.744626	0.444529	0.744626	
4	204.0	0.723067	0.426021	0.723067	
5	245.0	0.759002	0.412097	0.759002	
6	286.0	0.756715	0.397800	0.756715	
7	327.0	0.772393	0.385821	0.772393	
8	368.0	0.755891	0.376355	0.755891	
9	409.0	0.765422	0.366810	0.765422	
10	450.0	0.770358	0.356780	0.770358	
11	491.0	0.767565	0.345801	0.767565	
12	532.0	0.779612	0.335969	0.779612	
13	573.0	0.782759	0.327422	0.782759	

```
14     614.0           0.779540    0.319583            0.779540
15     655.0           0.790405    0.311705            0.790405
16     696.0           0.785892    0.304518            0.785892
17     737.0           0.789477    0.298166            0.789477
18     778.0           0.789232    0.292853            0.789232
19     819.0           0.793075    0.286920            0.793075
20     820.0                NaN         NaN                 NaN

       valid_dataset_iou  valid_loss  valid_per_image_iou
epoch
0               0.721167    0.512732             0.721167
1               0.719180    0.471846             0.719180
2               0.661537    0.448505             0.661537
3               0.744626    0.432460             0.744626
4               0.723067    0.418262             0.723067
5               0.759002    0.405837             0.759002
6               0.756715    0.395773             0.756715
7               0.772393    0.390298             0.772393
8               0.755891    0.383951             0.755891
9               0.765422    0.379007             0.765422
10              0.770358    0.373050             0.770358
11              0.767565    0.367518             0.767565
12              0.779612    0.363697             0.779612
13              0.782759    0.359333             0.782759
14              0.779540    0.355215             0.779540
15              0.790405    0.351147             0.790405
16              0.785892    0.348299             0.785892
17              0.789477    0.344993             0.789477
18              0.789232    0.341712             0.789232
19              0.793075    0.338584             0.793075
20              0.793075    0.335751             0.793075
```

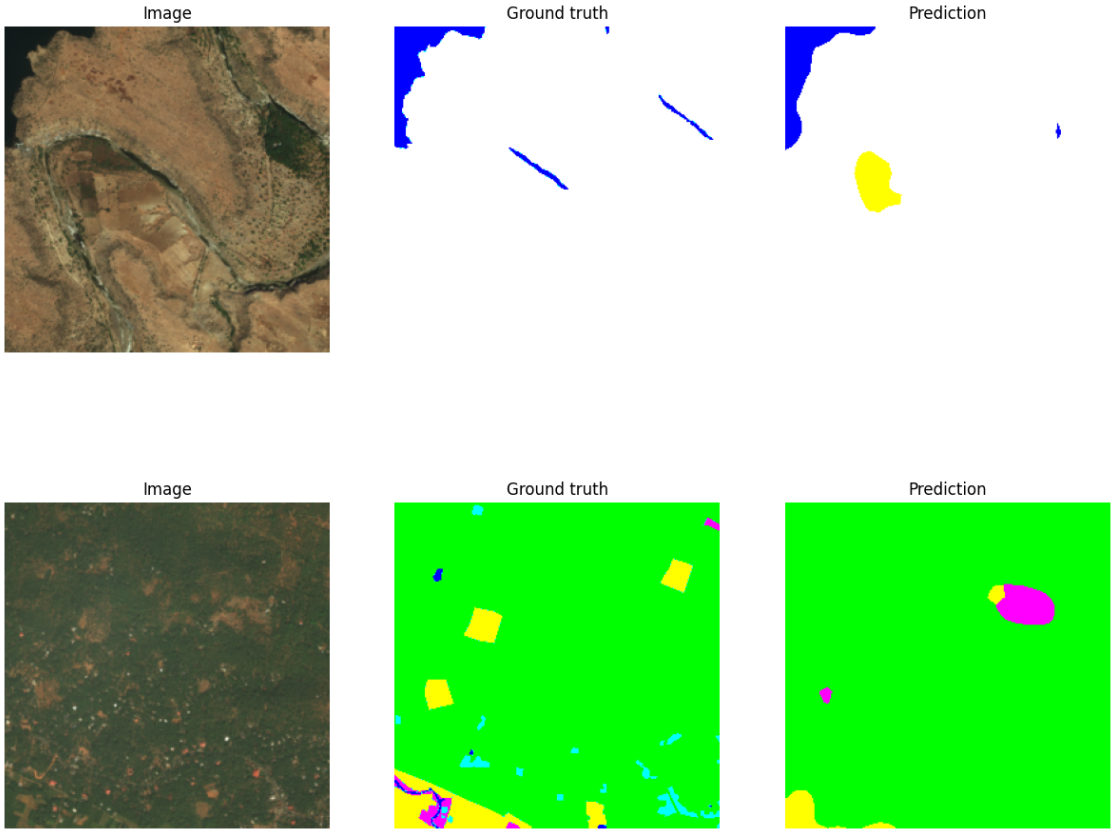
[ ]: <matplotlib.legend.Legend at 0x7f15d01afb20>



[ ]:
```
batch = next(iter(val_dataloader))
with torch.no_grad():
    deeplabv3.eval()
    logits = deeplabv3(batch["image"])
pr_masks = logits.sigmoid()

for image, gt_mask, pr_mask in zip(batch["image"], batch["mask"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Ground truth
    plt.subplot(1, 3, 2)
    plt.imshow(colorize_mask(gt_mask, color_map))
    plt.title("Ground truth")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
```

```
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



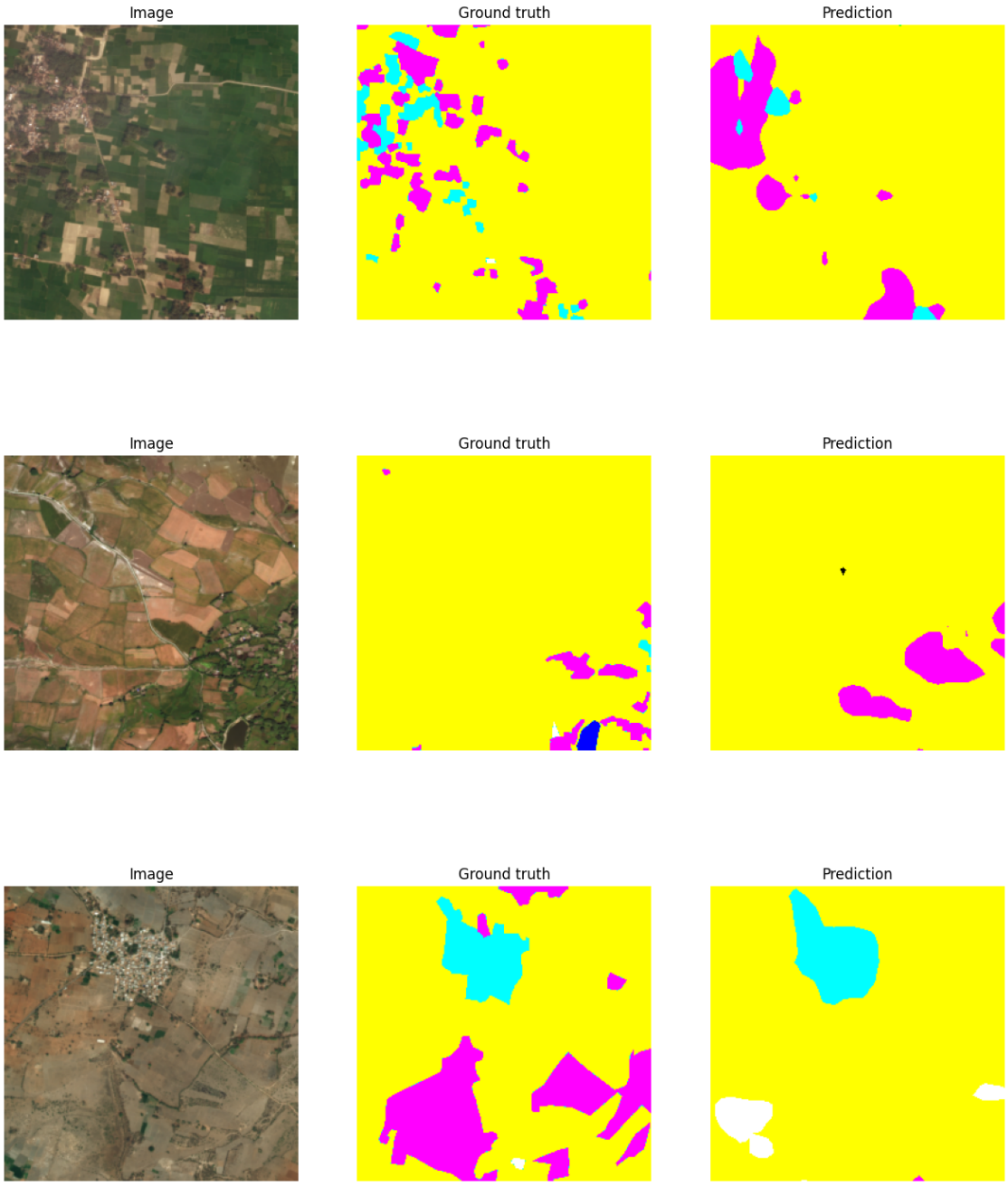


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

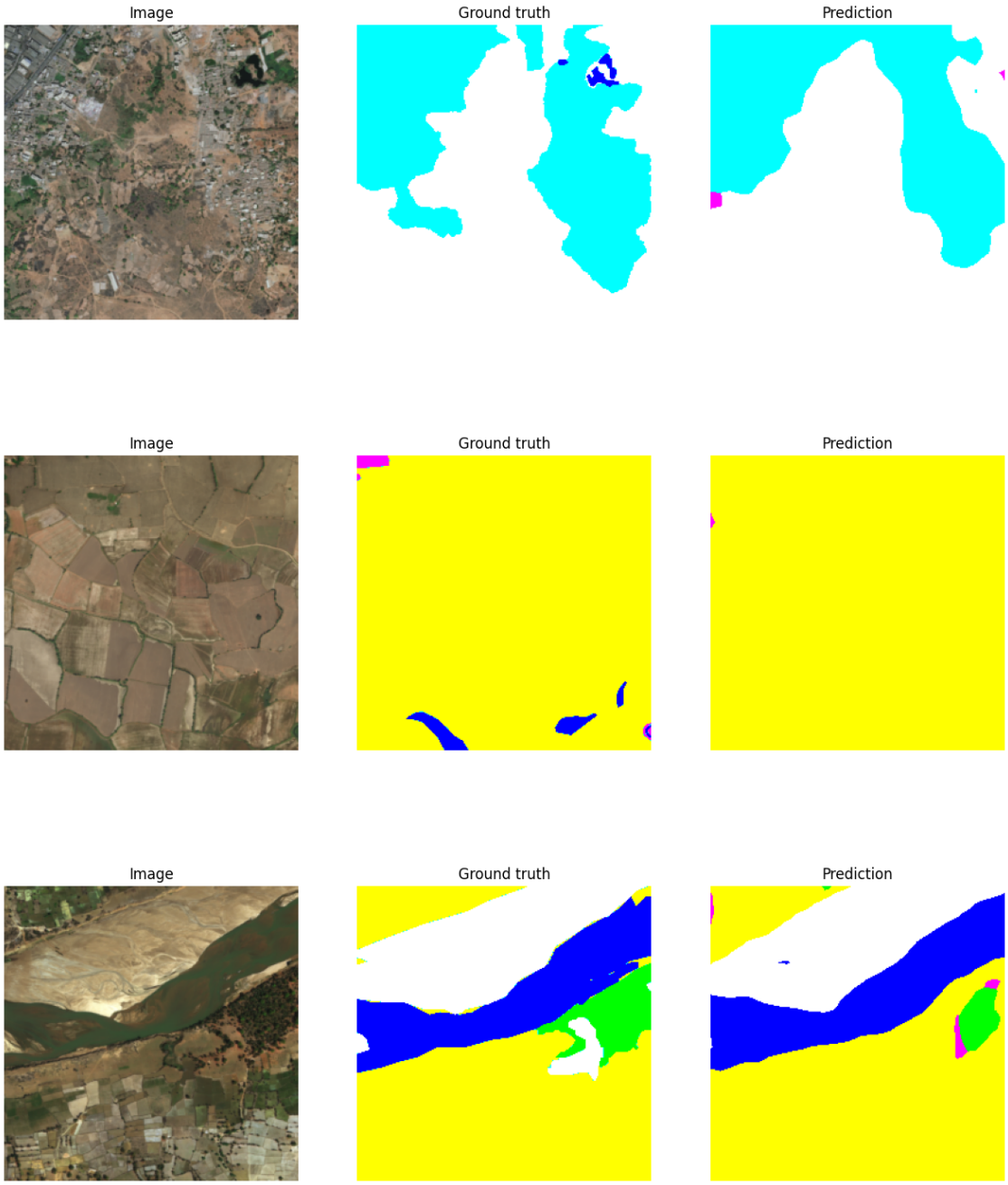


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

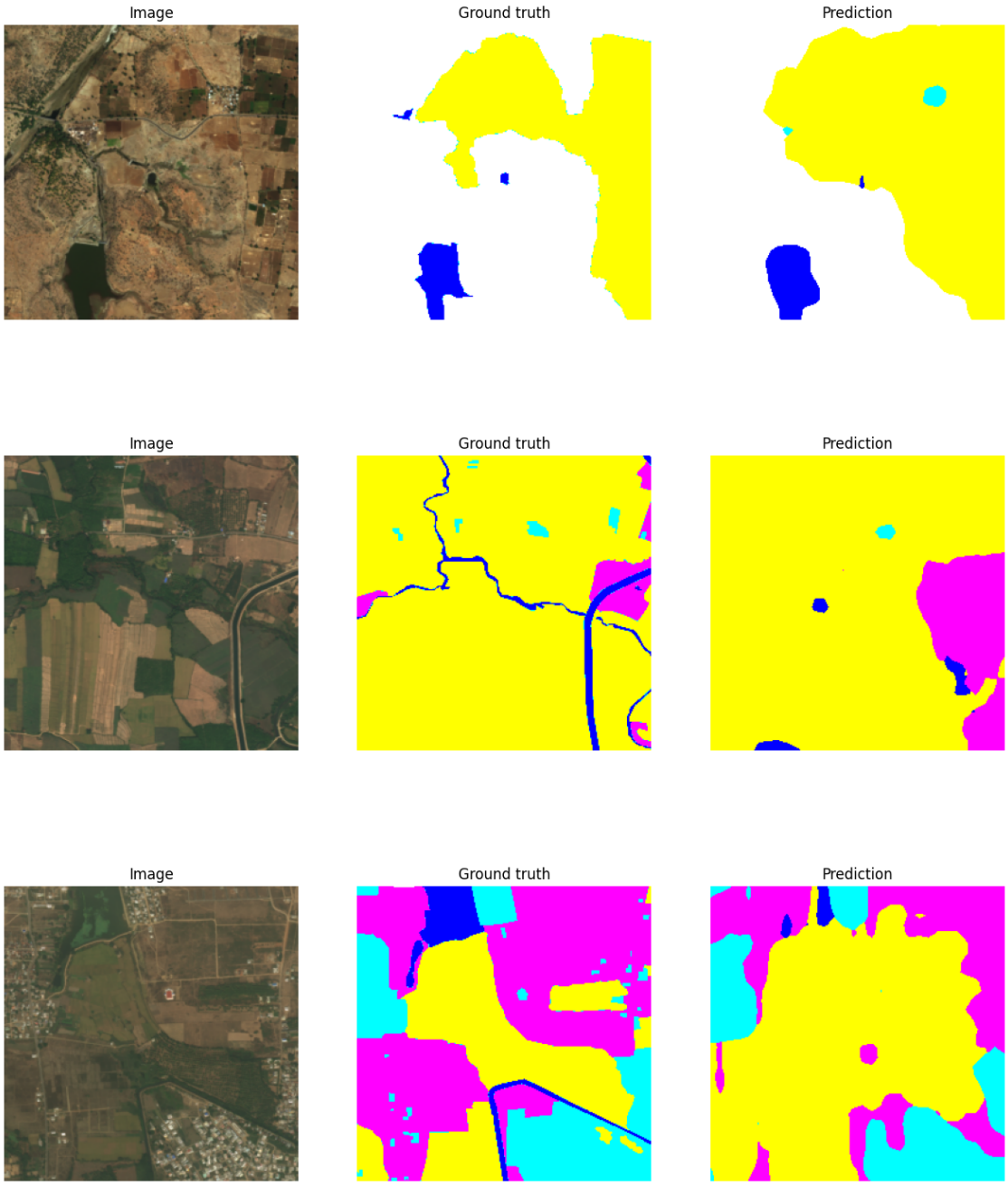


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

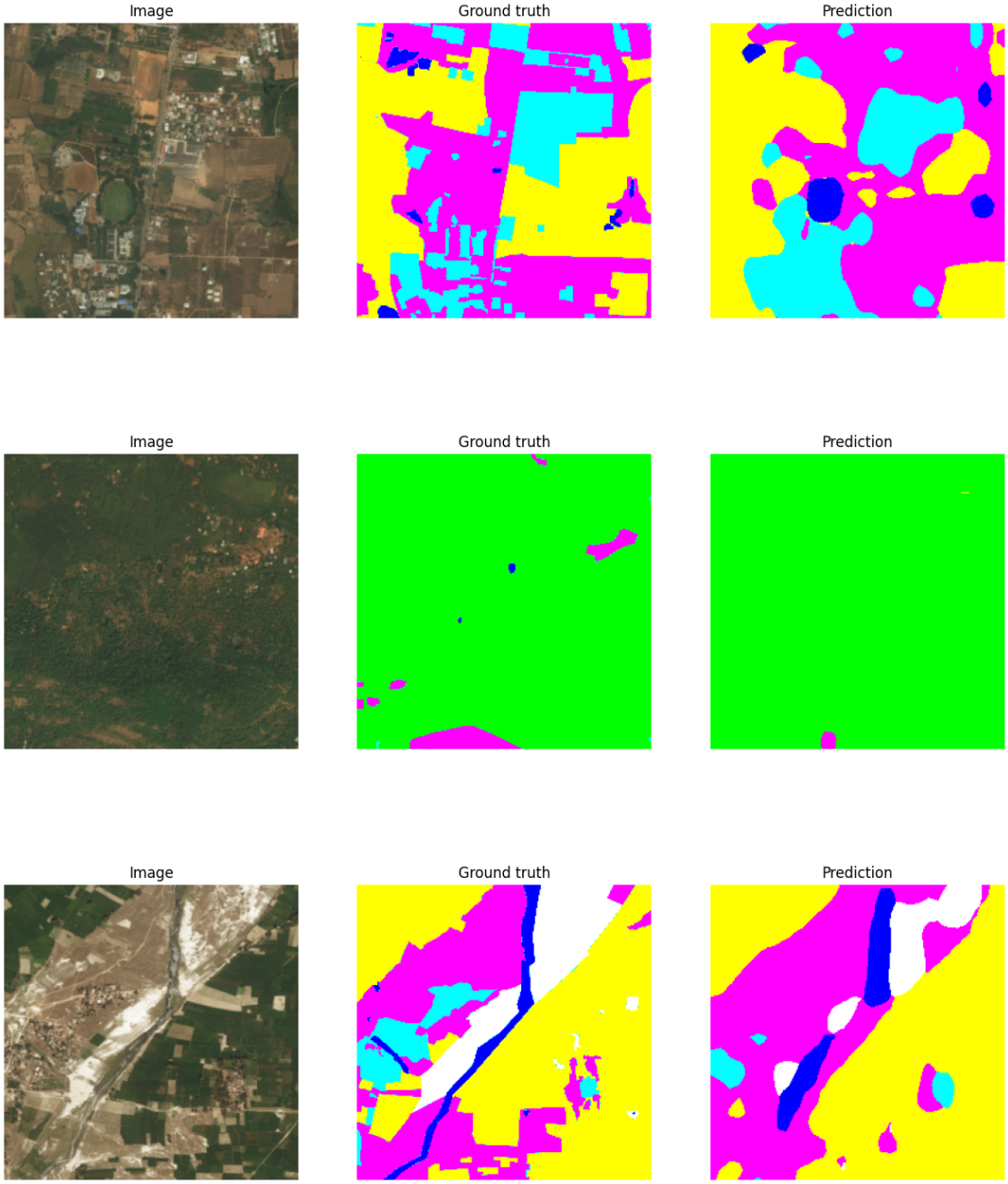
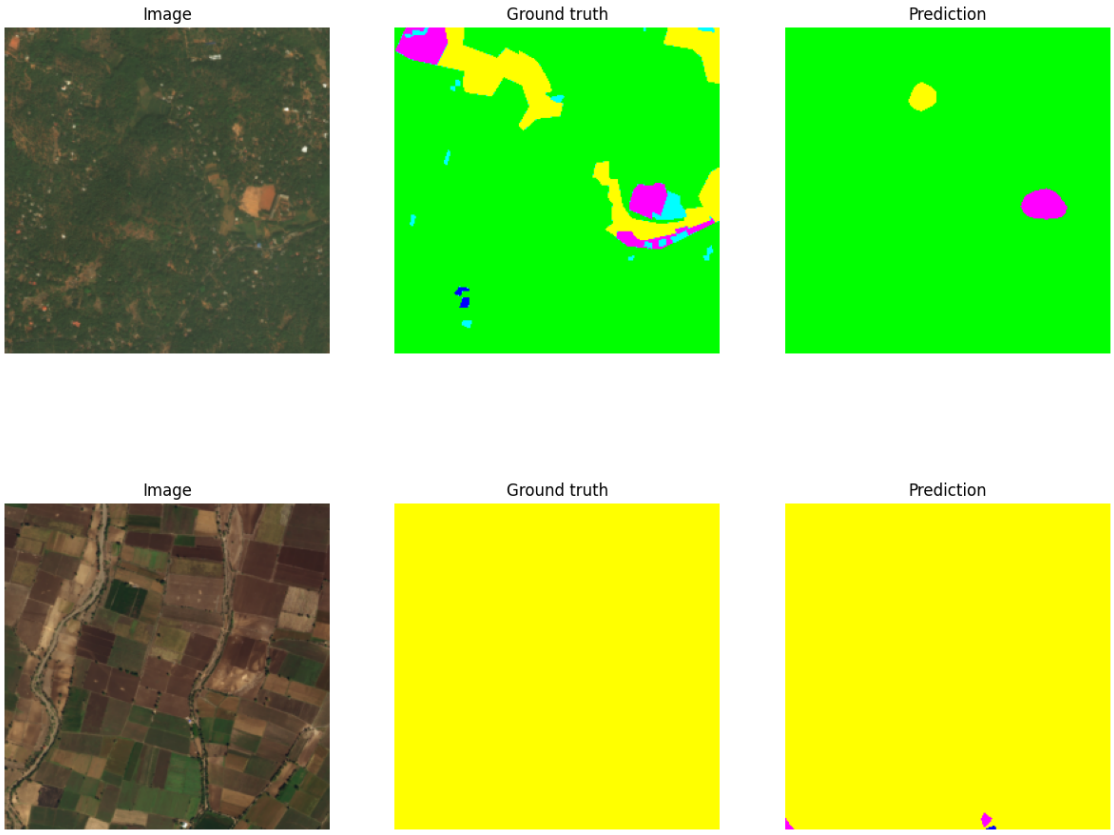


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction



Save Model

[ ]:
```
torch.save(deeplabv3.state_dict(), "/content/drive/MyDrive/satelite_image_nn/
↪models/deeplabv3_resnet152.pth")
```

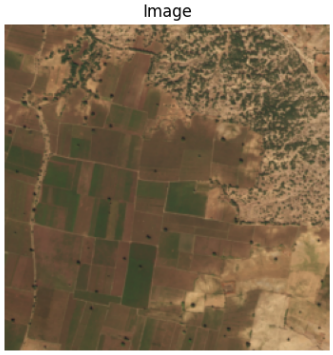
Test Model

[122]:
```
model2 = SatelliteImageModel("deeplabv3", "resnet152", in_channels=3,␣
↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="Adam")
model2.load_state_dict(torch.load('/content/drive/MyDrive/satelite_image_nn/
↪models/deeplabv3_resnet152.pth'))

batch = next(iter(test_dataloader))
with torch.no_grad():
    model2.eval()
    logits = model2(batch["image"])
pr_masks = logits.sigmoid()

for image, pr_mask in zip(batch["image"], pr_masks):
    plt.figure(figsize=(15, 5))
```

```
    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

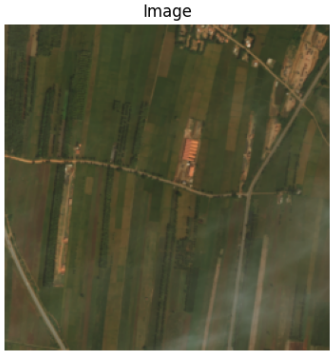
```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

Image

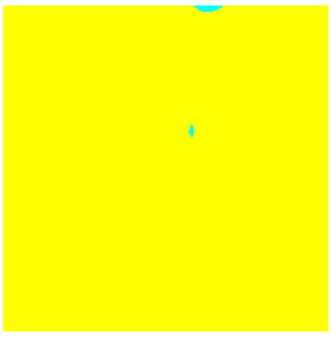


Prediction

Image



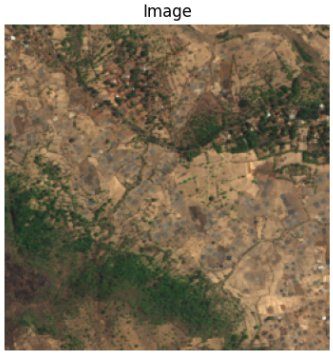
Prediction



Image

Prediction

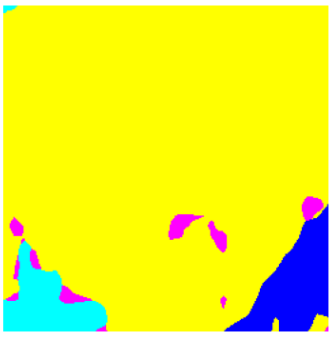
Image



Prediction

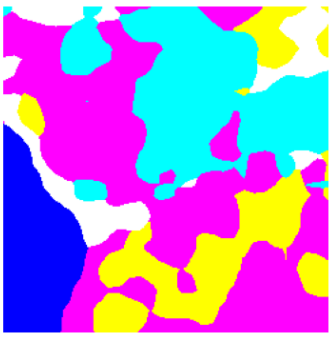
Image

Prediction



Image

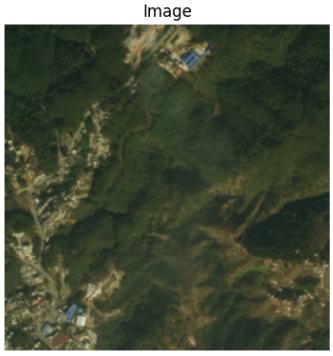
Prediction



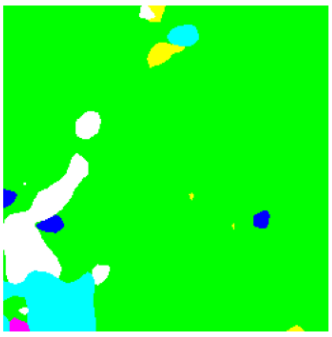
Image

Prediction

Image

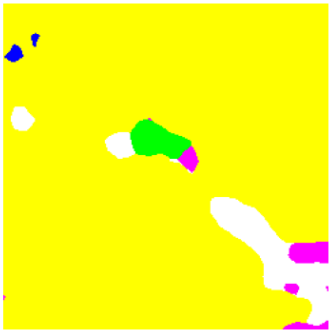


Prediction



Image

Prediction



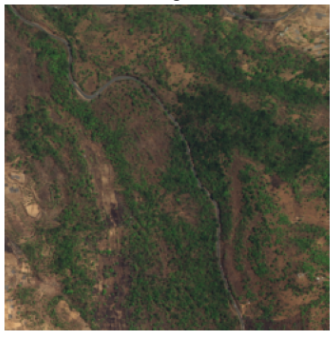
Image

Prediction

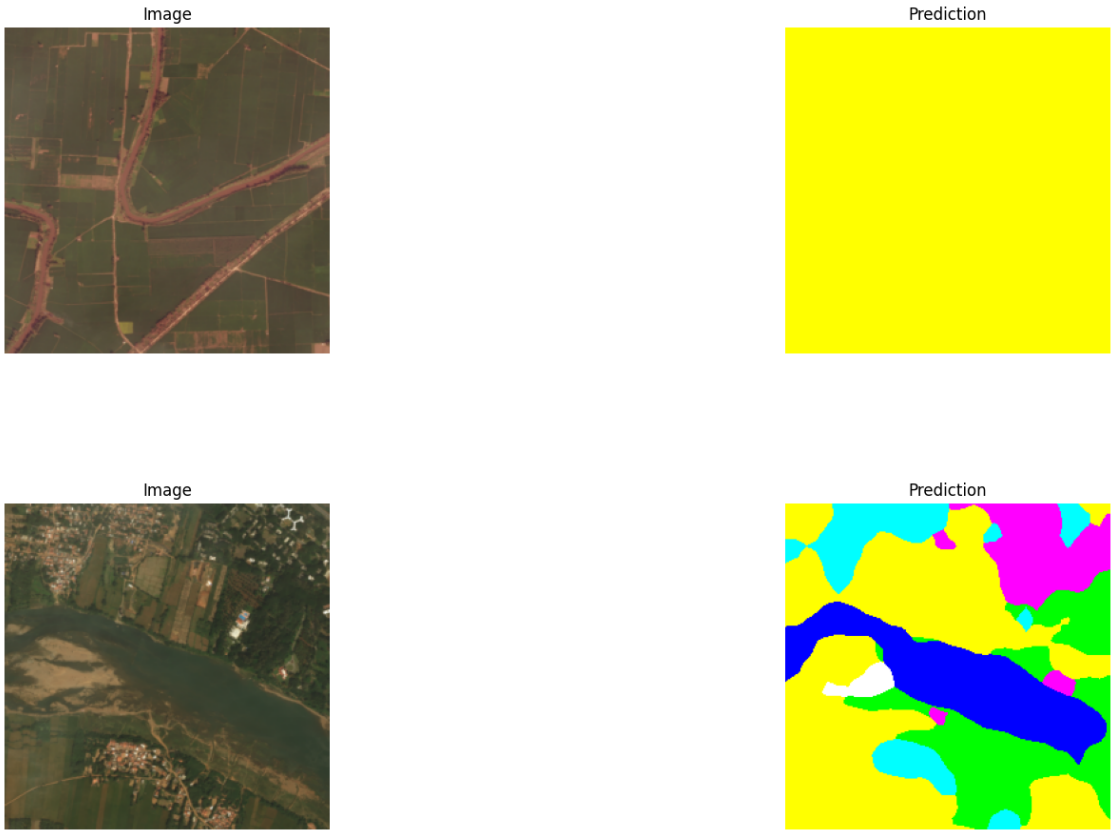
Image

Prediction

Image



Prediction



2.0.3 U-Net

[ ]:
```
unet = SatelliteImageModel("unet", "resnet50", in_channels=3, out_classes=7, lr␣
↪= 0.0001, pre_trained=True, optimizer="Adam")
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to
/root/.cache/torch/hub/checkpoints/resnet50-19c8e357.pth
100%|      | 97.8M/97.8M [00:00<00:00, 136MB/s]
```

[ ]:
```
# Early stop is a callback that is used to stop the training process when the␣
↪validation loss does not improve. In this case, we are
# using the EarlyStopping callback to stop the training process when the␣
↪validation loss does not improve for 3 epochs.
earlystop_callback = EarlyStopping('valid_loss', patience=3)

trainer = L.Trainer(max_epochs=20, logger=CSVLogger(save_dir="logs/",␣
↪name="pets_seg-model"), callbacks=[earlystop_callback])

trainer.fit(unet, train_dataloaders=train_dataloader,␣
↪val_dataloaders=val_dataloader)
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:
  | Name    | Type     | Params
-------------------------------------
0 | model   | Unet     | 32.5 M
1 | loss_fn | DiceLoss | 0
-------------------------------------
32.5 M    Trainable params
0         Non-trainable params
32.5 M    Total params
130.088   Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name    | Type     | Params
-------------------------------------
0 | model   | Unet     | 32.5 M
1 | loss_fn | DiceLoss | 0
-------------------------------------
32.5 M    Trainable params
0         Non-trainable params
32.5 M    Total params
130.088   Total estimated model params size (MB)

Sanity Checking: |          | 0/? [00:00<?, ?it/s]

/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
/usr/local/lib/python3.10/dist-packages/lightning/pytorch/loops/fit_loop.py:298:
The number of training batches (41) is smaller than the logging interval
Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you
want to see logs for the training epoch.

Training: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
INFO: `Trainer.fit` stopped: `max_epochs=20` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=20` reached.
```

[ ]:
```
valid_metrics = trainer.validate(unet, dataloaders=val_dataloader,␣
  ↪verbose=False)
print(valid_metrics)
```

```
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
[{'valid_loss': 0.38910654187202454, 'valid_per_image_iou': 0.8128405809402466,
'valid_dataset_iou': 0.8128405809402466}]
```

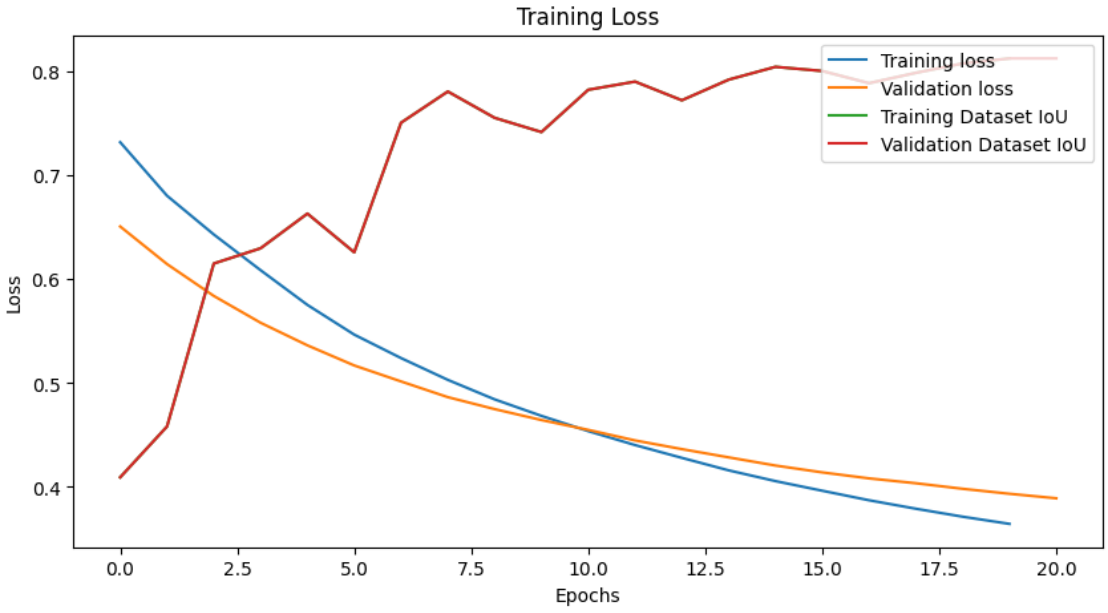
[ ]:
```
# Read the metrics.csv file generated by the PyTorch Lightning logger
metrics = pd.read_csv(f"{trainer.logger.log_dir}/metrics.csv")

# Group the metrics by epoch and compute the mean loss for each epoch
```

```
df_epochs = metrics.groupby('epoch').mean()

display(df_epochs)

# Create a figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
# Set the x-axis label
ax.set_xlabel('Epochs')
# Set the y-axis label
ax.set_ylabel('Loss')
ax.plot(df_epochs['train_loss'], label="Training loss")
ax.plot(df_epochs['valid_loss'], label="Validation loss")

ax.plot(df_epochs['train_dataset_iou'], label="Training Dataset IoU")
ax.plot(df_epochs['valid_dataset_iou'], label="Validation Dataset IoU")

# Plot the training loss over epochs
# Plot the validation loss over epochs
# Set the title of the plot
ax.set_title("Training Loss")
# Add a legend to the plot
ax.legend(loc='upper right')
```

epoch	step	train_dataset_iou	train_loss	train_per_image_iou	\
0	40.0	0.409331	0.732141	0.409331	
1	81.0	0.458268	0.680603	0.458268	
2	122.0	0.615281	0.643246	0.615281	
3	163.0	0.629965	0.608824	0.629965	
4	204.0	0.663371	0.575401	0.663371	
5	245.0	0.626103	0.546778	0.626103	
6	286.0	0.750918	0.524185	0.750918	
7	327.0	0.780991	0.503190	0.780991	
8	368.0	0.755632	0.484399	0.755632	
9	409.0	0.741936	0.468510	0.741936	
10	450.0	0.782662	0.453803	0.782662	
11	491.0	0.790528	0.440384	0.790528	
12	532.0	0.772537	0.428052	0.772537	
13	573.0	0.792413	0.416017	0.792413	
14	614.0	0.804753	0.405651	0.804753	
15	655.0	0.800761	0.396383	0.800761	
16	696.0	0.789012	0.387202	0.789012	
17	737.0	0.798982	0.379056	0.798982	
18	778.0	0.807989	0.371366	0.807989	
19	819.0	0.812841	0.364449	0.812841	
20	820.0	NaN	NaN	NaN	

epoch	valid_dataset_iou	valid_loss	valid_per_image_iou
0	0.409331	0.650944	0.409331
1	0.458268	0.614751	0.458268
2	0.615281	0.584047	0.615281
3	0.629965	0.558207	0.629965
4	0.663371	0.536439	0.663371
5	0.626103	0.517071	0.626103
6	0.750918	0.501680	0.750918
7	0.780991	0.486537	0.780991
8	0.755632	0.475016	0.755632
9	0.741936	0.464479	0.741936
10	0.782662	0.455018	0.782662
11	0.790528	0.444841	0.790528
12	0.772537	0.436476	0.772537
13	0.792413	0.428529	0.792413
14	0.804753	0.420658	0.804753
15	0.800761	0.414056	0.800761
16	0.789012	0.408273	0.789012
17	0.798982	0.403599	0.798982
18	0.807989	0.398252	0.807989
19	0.812841	0.393453	0.812841
20	0.812841	0.389107	0.812841

[ ]: <matplotlib.legend.Legend at 0x7f15d0fadc30>



[ ]:
```
batch = next(iter(val_dataloader))
with torch.no_grad():
    unet.eval()
    logits = unet(batch["image"])
pr_masks = logits.sigmoid()

for image, gt_mask, pr_mask in zip(batch["image"], batch["mask"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Ground truth
    plt.subplot(1, 3, 2)
    plt.imshow(colorize_mask(gt_mask, color_map))
    plt.title("Ground truth")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

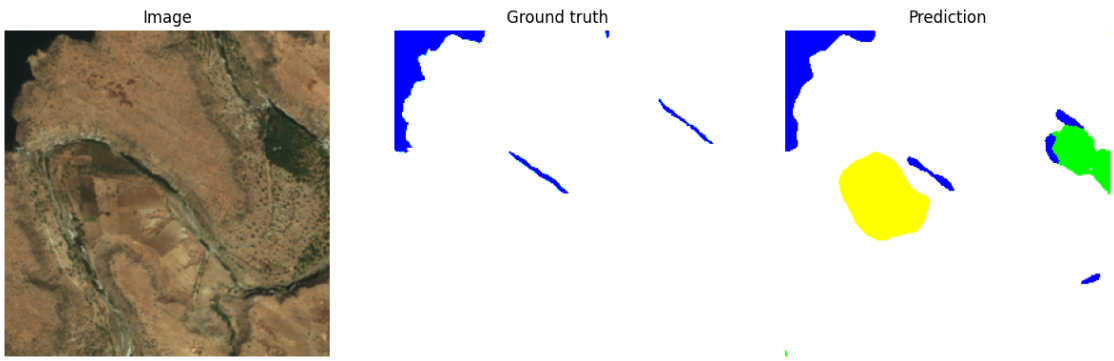



Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

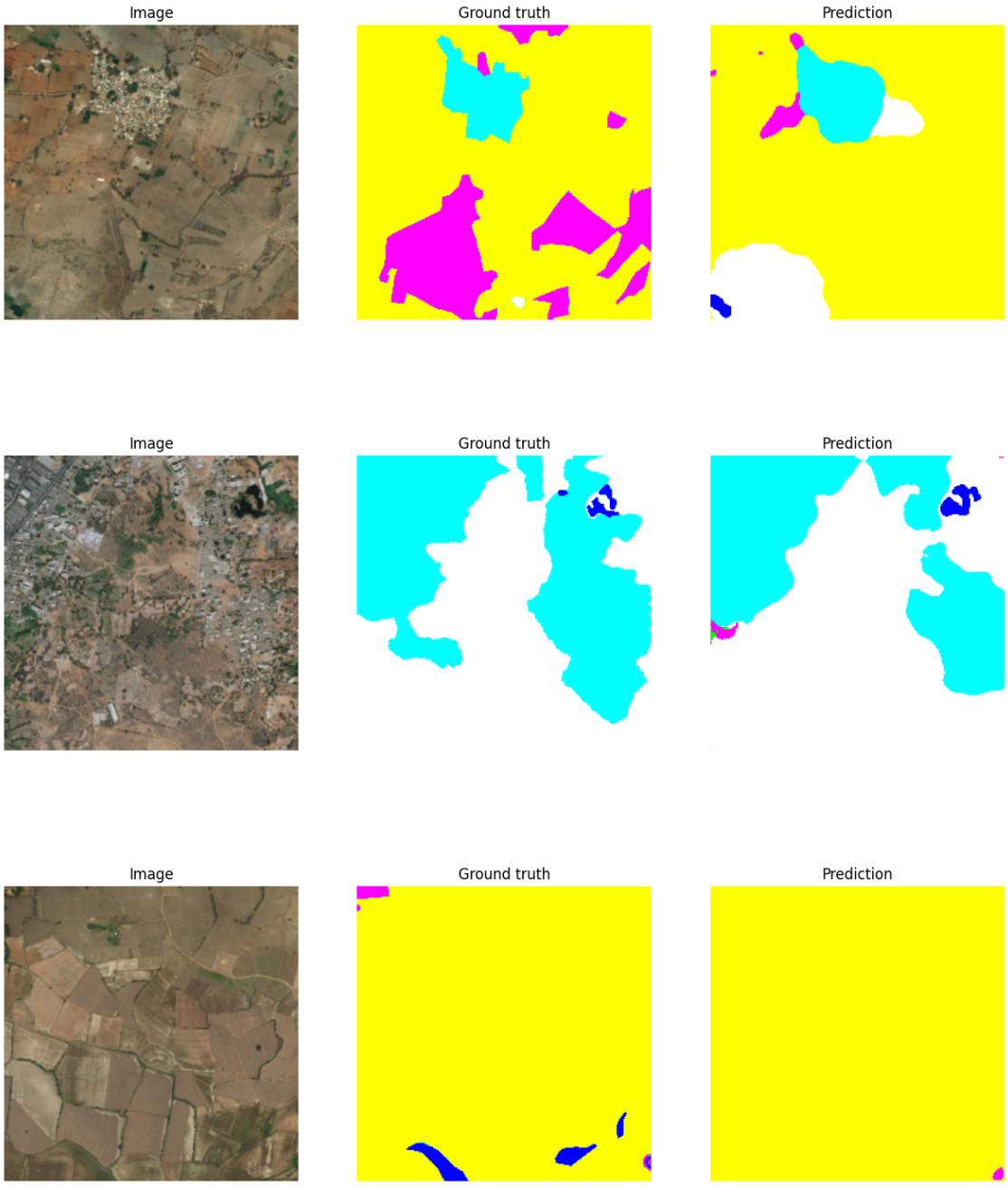


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

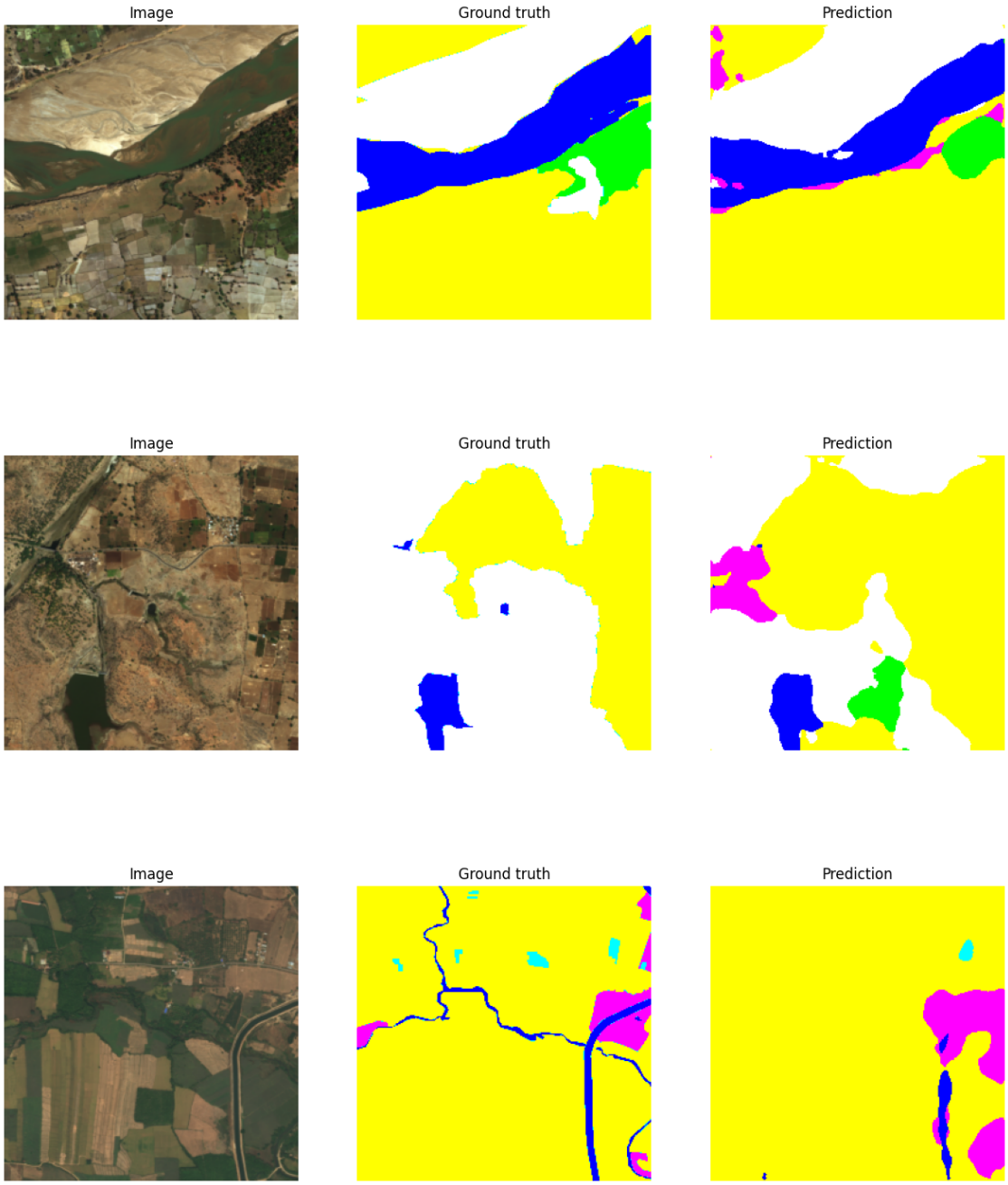
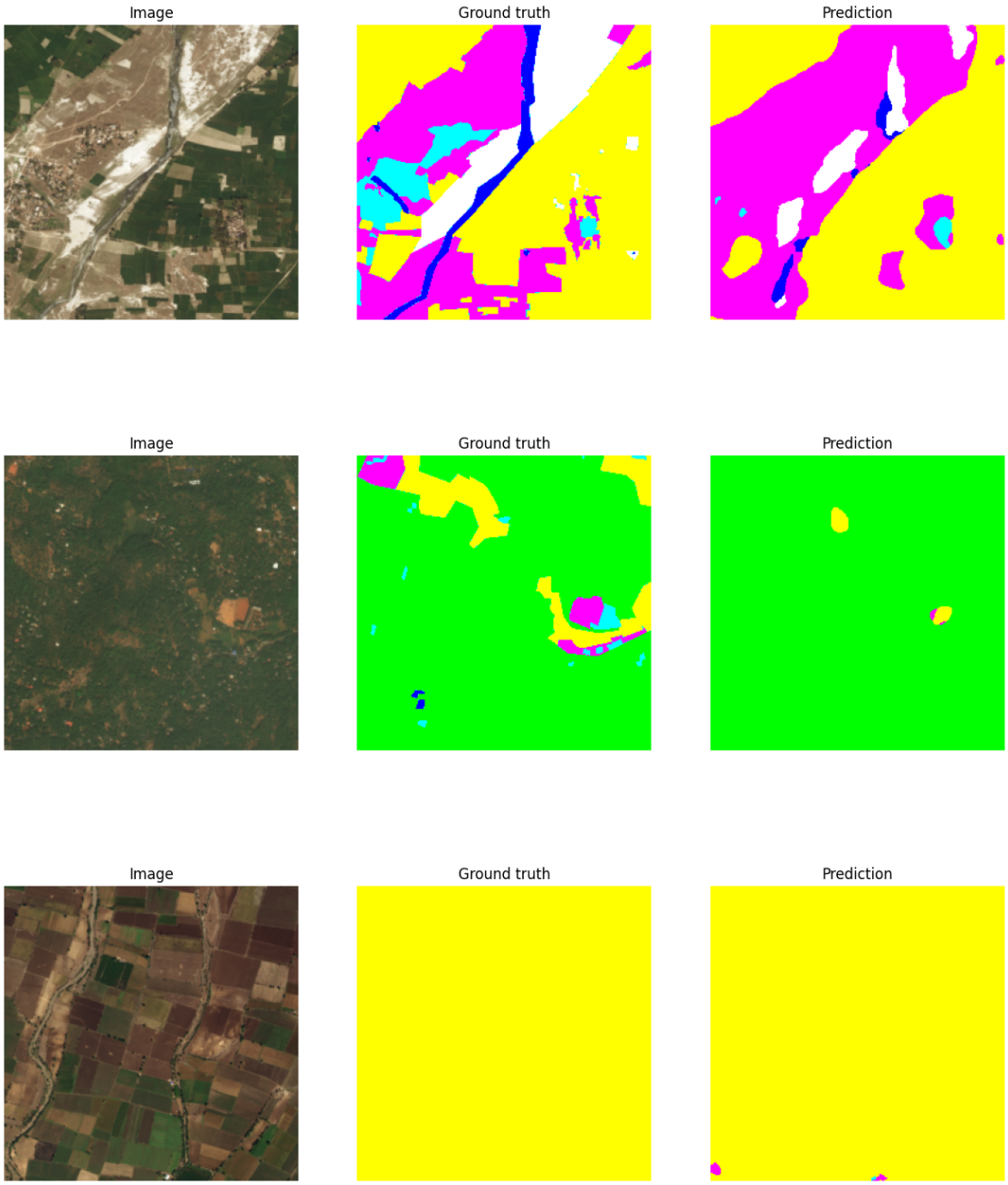


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction



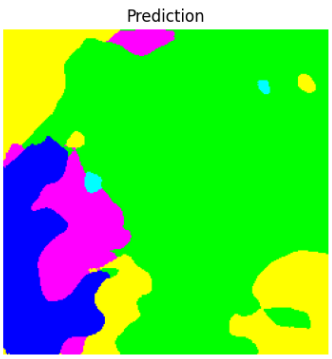
Save Model

```
[ ]: torch.save(unet.state_dict(), "/content/drive/MyDrive/satelite_image_nn/models/
     ↪unet_resnet50.pth")
```

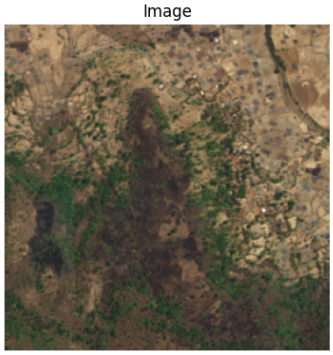
Test Model

[124]:
```
model3 = SatelliteImageModel("unet", "resnet50", in_channels=3, out_classes=7,␣
 ↪lr = 0.0001, pre_trained=True, optimizer="Adam")
model3.load_state_dict(torch.load('/content/drive/MyDrive/satelite_image_nn/
 ↪models/unet_resnet50.pth'))

batch = next(iter(test_dataloader))
with torch.no_grad():
    model3.eval()
    logits = model3(batch["image"])
pr_masks = logits.sigmoid()

for image, pr_mask in zip(batch["image"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



Image

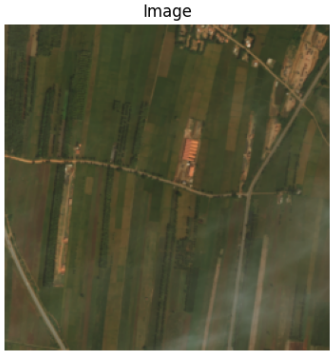


Prediction

Image

Prediction

Image

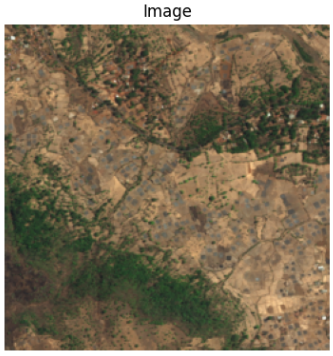


Prediction

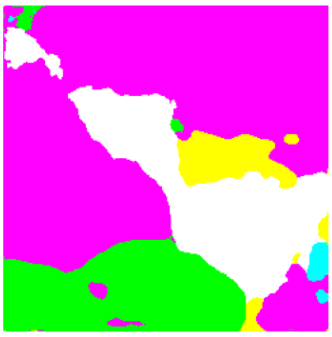
Image

Prediction

Image



Prediction

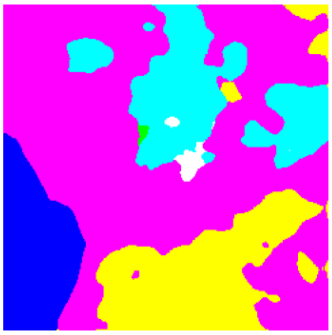


Image

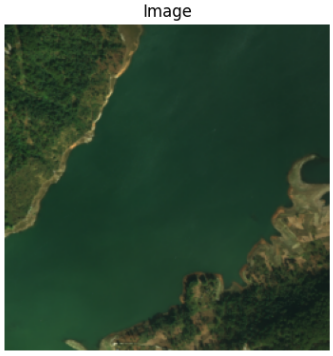
Prediction

Image

Prediction

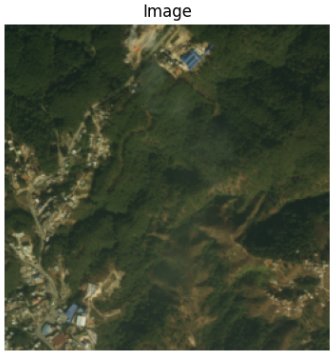


Image

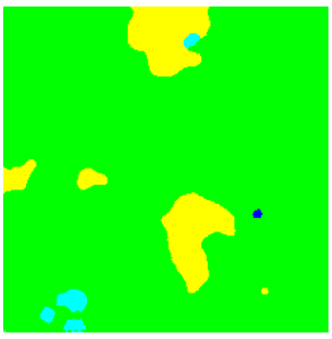


Prediction

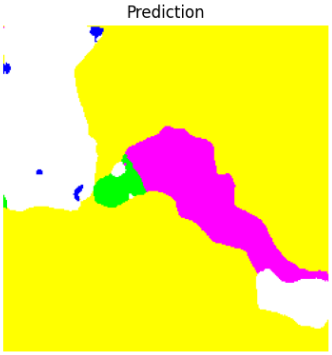
Image

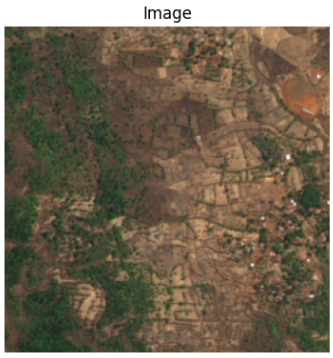


Prediction

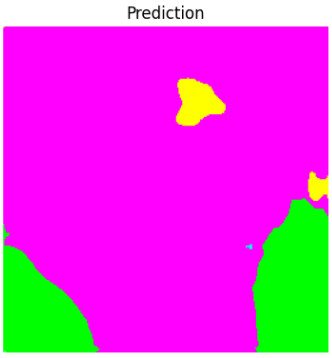



Image

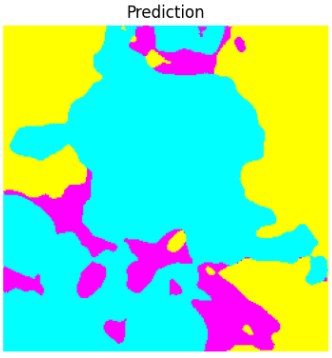

Prediction

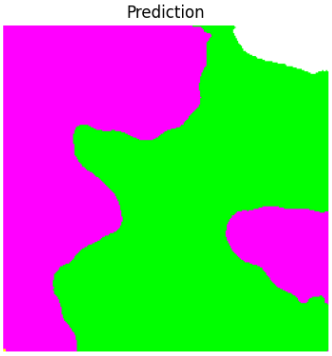


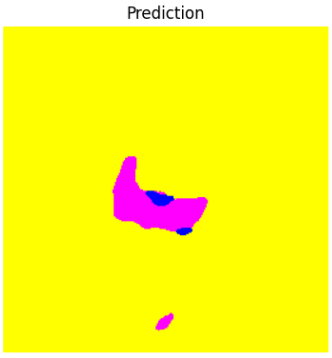
Image

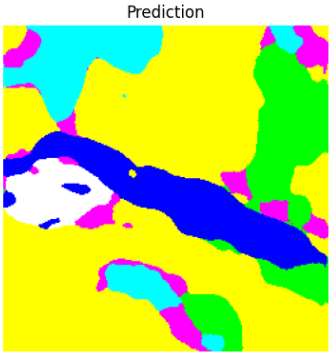

Prediction


Image


Prediction







2.0.4 DeepLabV3+

[ ]:
```
deeplabv3plus = SatelliteImageModel("deeplabv3plus", "resnet34", in_channels=3,␣
↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="Adam")
```

[ ]:
```
# Early stop is a callback that is used to stop the training process when the␣
↪validation loss does not improve. In this case, we are
```

```
# using the EarlyStopping callback to stop the training process when the␣
 ↪validation loss does not improve for 3 epochs.
earlystop_callback = EarlyStopping('valid_loss', patience=3)

trainer = L.Trainer(max_epochs=20, logger=CSVLogger(save_dir="logs/",␣
 ↪name="pets_seg-model"), callbacks=[earlystop_callback])

trainer.fit(deeplabv3plus, train_dataloaders=train_dataloader,␣
 ↪val_dataloaders=val_dataloader)
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:
  | Name    | Type          | Params
-----------------------------------------
0 | model   | DeepLabV3Plus | 22.4 M
1 | loss_fn | DiceLoss      | 0
-----------------------------------------
22.4 M    Trainable params
0         Non-trainable params
22.4 M    Total params
89.756    Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name    | Type          | Params
-----------------------------------------
0 | model   | DeepLabV3Plus | 22.4 M
1 | loss_fn | DiceLoss      | 0
-----------------------------------------
22.4 M    Trainable params
0         Non-trainable params
22.4 M    Total params
89.756    Total estimated model params size (MB)
```

```
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
```

```
/usr/local/lib/python3.10/dist-packages/lightning/pytorch/loops/fit_loop.py:298:
The number of training batches (41) is smaller than the logging interval
```

```
Trainer(log_every_n_steps=50). Set a lower value for log_every_n_steps if you
want to see logs for the training epoch.

Training: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

INFO: `Trainer.fit` stopped: `max_epochs=20` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=20` reached.
```

[ ]:

```
valid_metrics = trainer.validate(deeplabv3plus, dataloaders=val_dataloader,␣
  ↪verbose=False)
print(valid_metrics)
```

```
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]

Validation: |          | 0/? [00:00<?, ?it/s]
```

```
[{'valid_loss': 0.3771437406539917, 'valid_per_image_iou': 0.7889636754989624,
'valid_dataset_iou': 0.7889636754989624}]
```

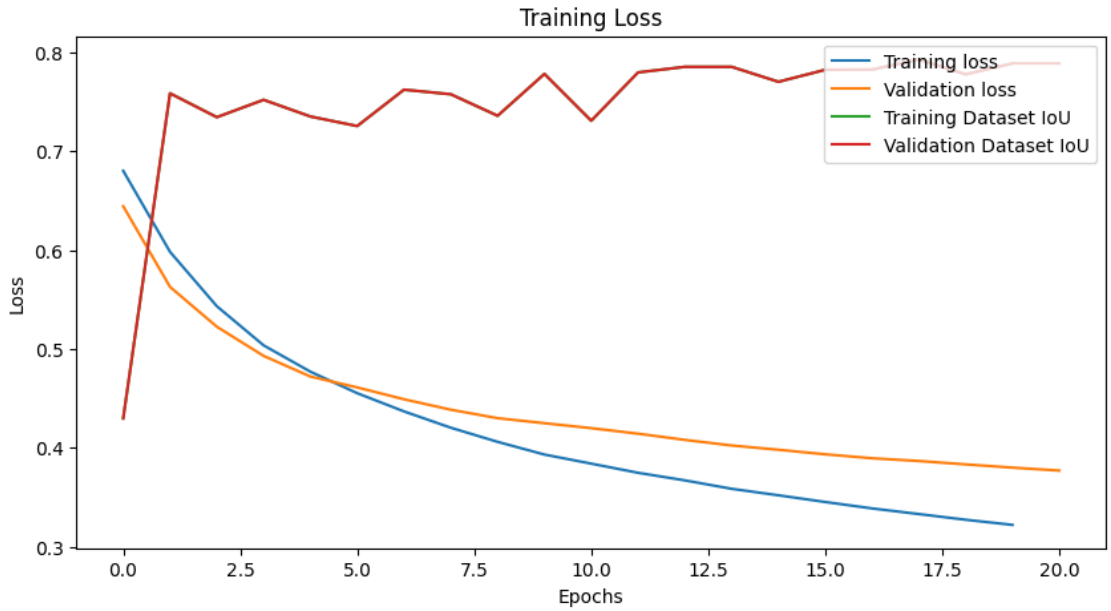
[ ]:
```python
# Read the metrics.csv file generated by the PyTorch Lightning logger
metrics = pd.read_csv(f"{trainer.logger.log_dir}/metrics.csv")

# Group the metrics by epoch and compute the mean loss for each epoch
df_epochs = metrics.groupby('epoch').mean()

display(df_epochs)

# Create a figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
# Set the x-axis label
ax.set_xlabel('Epochs')
# Set the y-axis label
ax.set_ylabel('Loss')
ax.plot(df_epochs['train_loss'], label="Training loss")
ax.plot(df_epochs['valid_loss'], label="Validation loss")

ax.plot(df_epochs['train_dataset_iou'], label="Training Dataset IoU")
ax.plot(df_epochs['valid_dataset_iou'], label="Validation Dataset IoU")

# Plot the training loss over epochs
# Plot the validation loss over epochs
# Set the title of the plot
ax.set_title("Training Loss")
# Add a legend to the plot
ax.legend(loc='upper right')
```

epoch	step	train_dataset_iou	train_loss	train_per_image_iou	\
0	40.0	0.429918	0.680332	0.429918	
1	81.0	0.758718	0.598368	0.758718	
2	122.0	0.734560	0.543411	0.734560	
3	163.0	0.752136	0.503793	0.752136	
4	204.0	0.735272	0.477157	0.735272	
5	245.0	0.725608	0.455408	0.725608	
6	286.0	0.762407	0.436979	0.762407	
7	327.0	0.757775	0.420468	0.757775	
8	368.0	0.736007	0.406191	0.736007	
9	409.0	0.778357	0.393327	0.778357	
10	450.0	0.730986	0.384156	0.730986	
11	491.0	0.779854	0.374896	0.779854	
12	532.0	0.785503	0.367224	0.785503	
13	573.0	0.785503	0.358694	0.785503	
14	614.0	0.770382	0.352123	0.770382	
15	655.0	0.782516	0.345341	0.782516	

16	696.0	0.782710	0.338854	0.782710
17	737.0	0.792977	0.333145	0.792977
18	778.0	0.777922	0.327384	0.777922
19	819.0	0.788964	0.322200	0.788964
20	820.0	NaN	NaN	NaN

epoch	valid_dataset_iou	valid_loss	valid_per_image_iou
0	0.429918	0.644510	0.429918
1	0.758718	0.562924	0.758718
2	0.734560	0.522667	0.734560
3	0.752136	0.493149	0.752136
4	0.735272	0.472250	0.735272
5	0.725608	0.461232	0.725608
6	0.762407	0.449247	0.762407
7	0.757775	0.438700	0.757775
8	0.736007	0.430127	0.736007
9	0.778357	0.425068	0.778357
10	0.730986	0.420035	0.730986
11	0.779854	0.414456	0.779854
12	0.785503	0.408180	0.785503
13	0.785503	0.402513	0.785503
14	0.770382	0.398181	0.770382
15	0.782516	0.393585	0.782516
16	0.782710	0.389582	0.782710
17	0.792977	0.386814	0.792977
18	0.777922	0.383393	0.777922
19	0.788964	0.380114	0.788964
20	0.788964	0.377144	0.788964

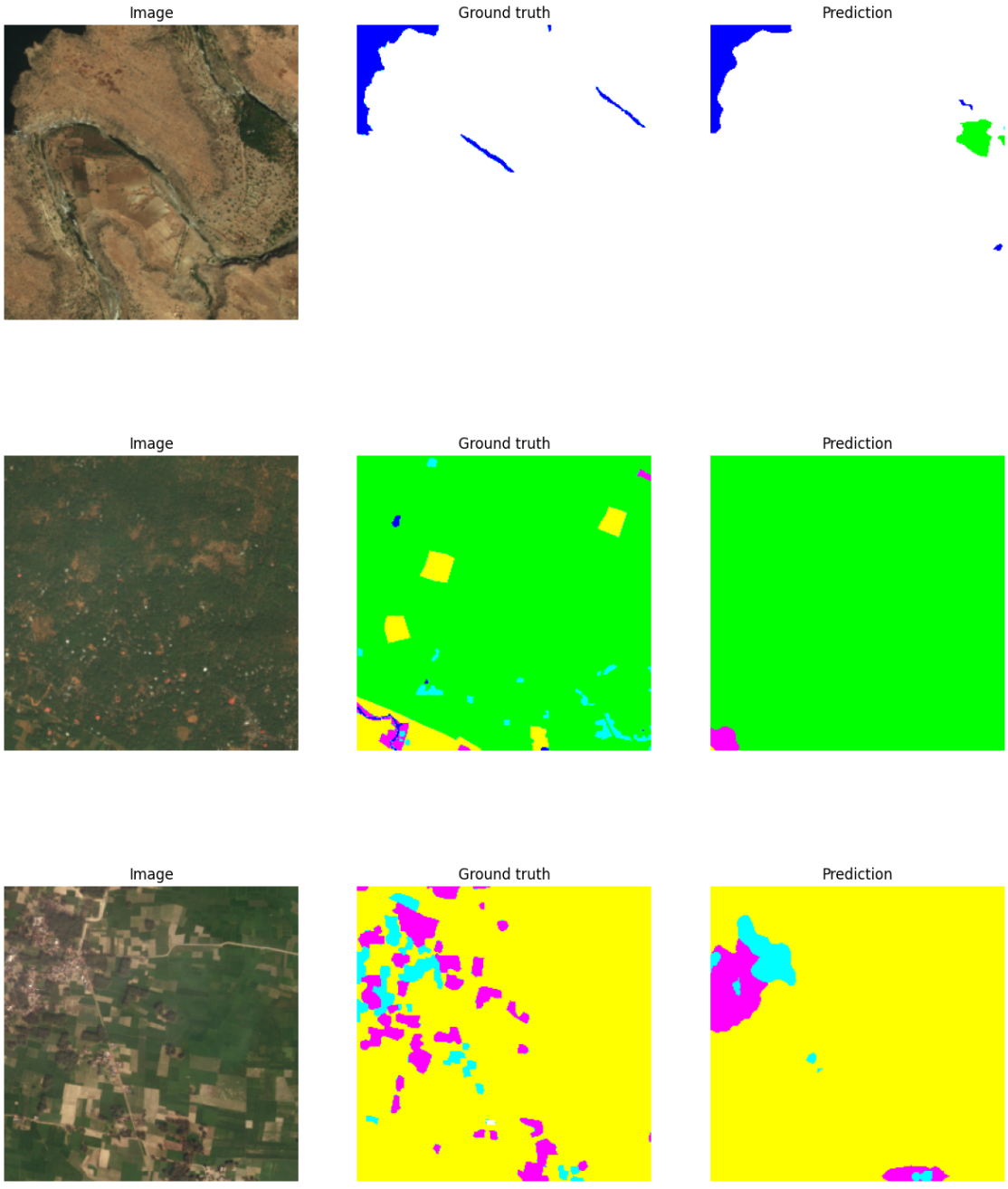
[ ]: <matplotlib.legend.Legend at 0x7f15b41e2da0>



[ ]:
```
batch = next(iter(val_dataloader))
with torch.no_grad():
    deeplabv3plus.eval()
    logits = deeplabv3plus(batch["image"])
pr_masks = logits.sigmoid()

for image, gt_mask, pr_mask in zip(batch["image"], batch["mask"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Ground truth
    plt.subplot(1, 3, 2)
    plt.imshow(colorize_mask(gt_mask, color_map))
    plt.title("Ground truth")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
```

```
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



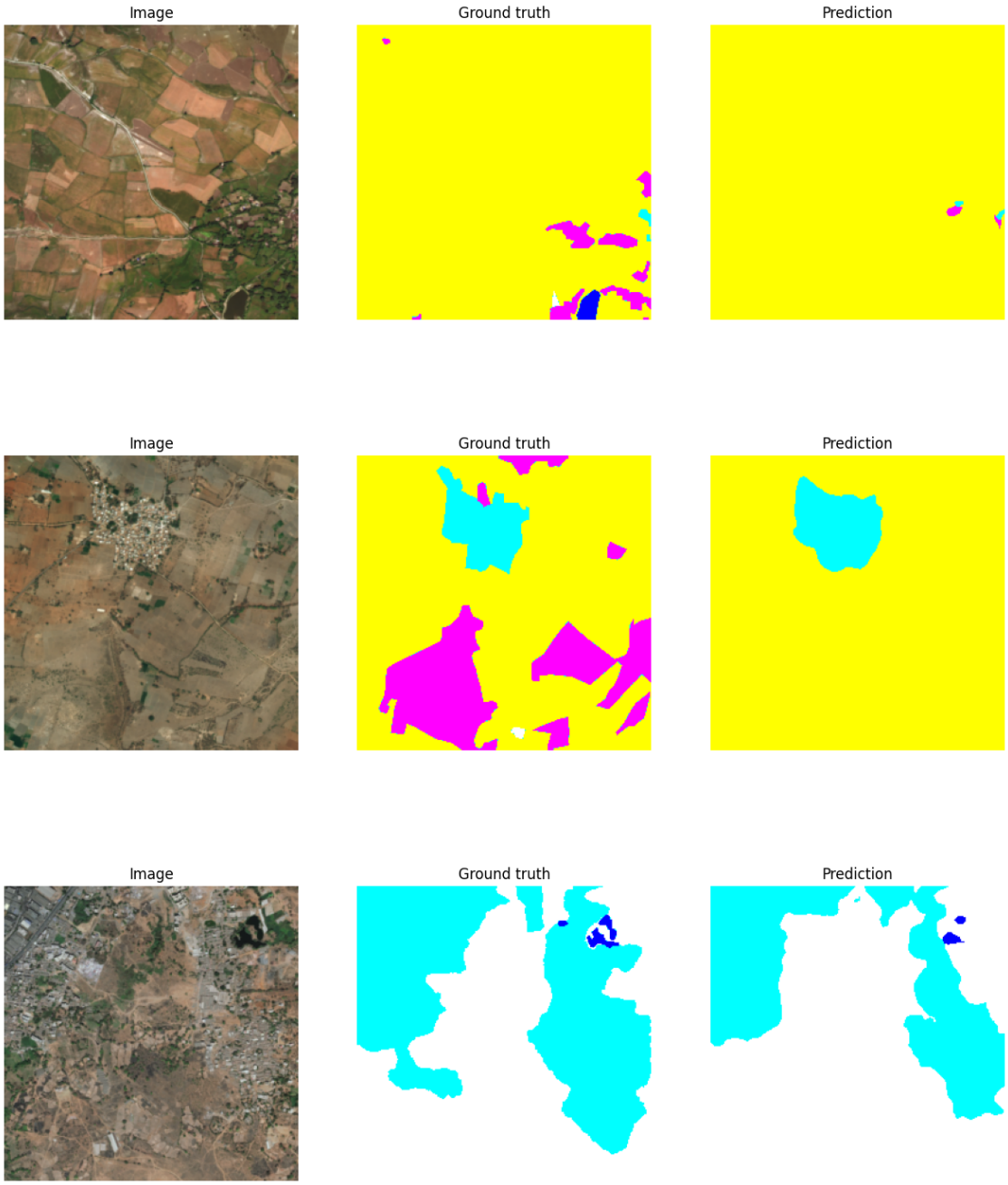


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

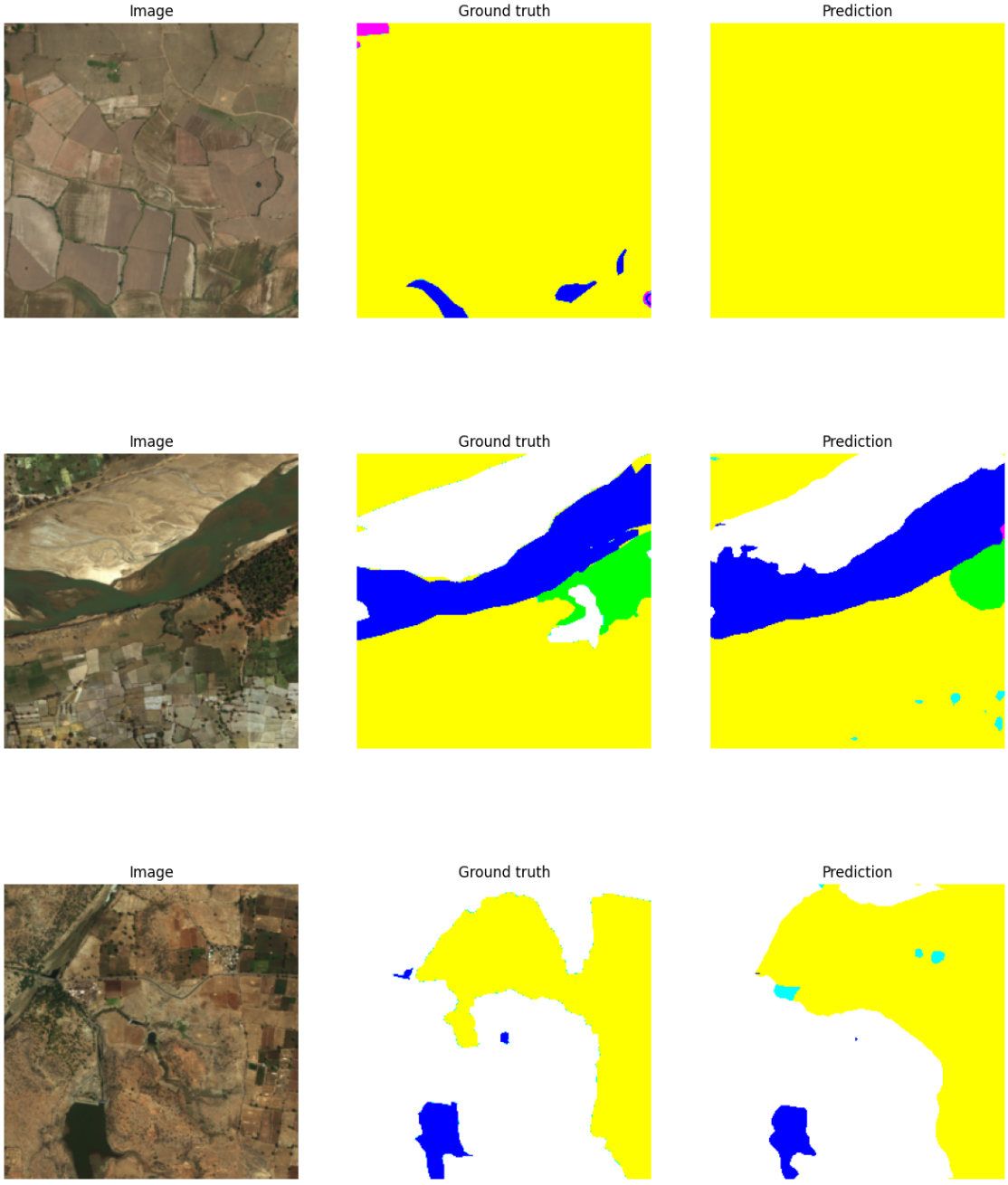


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

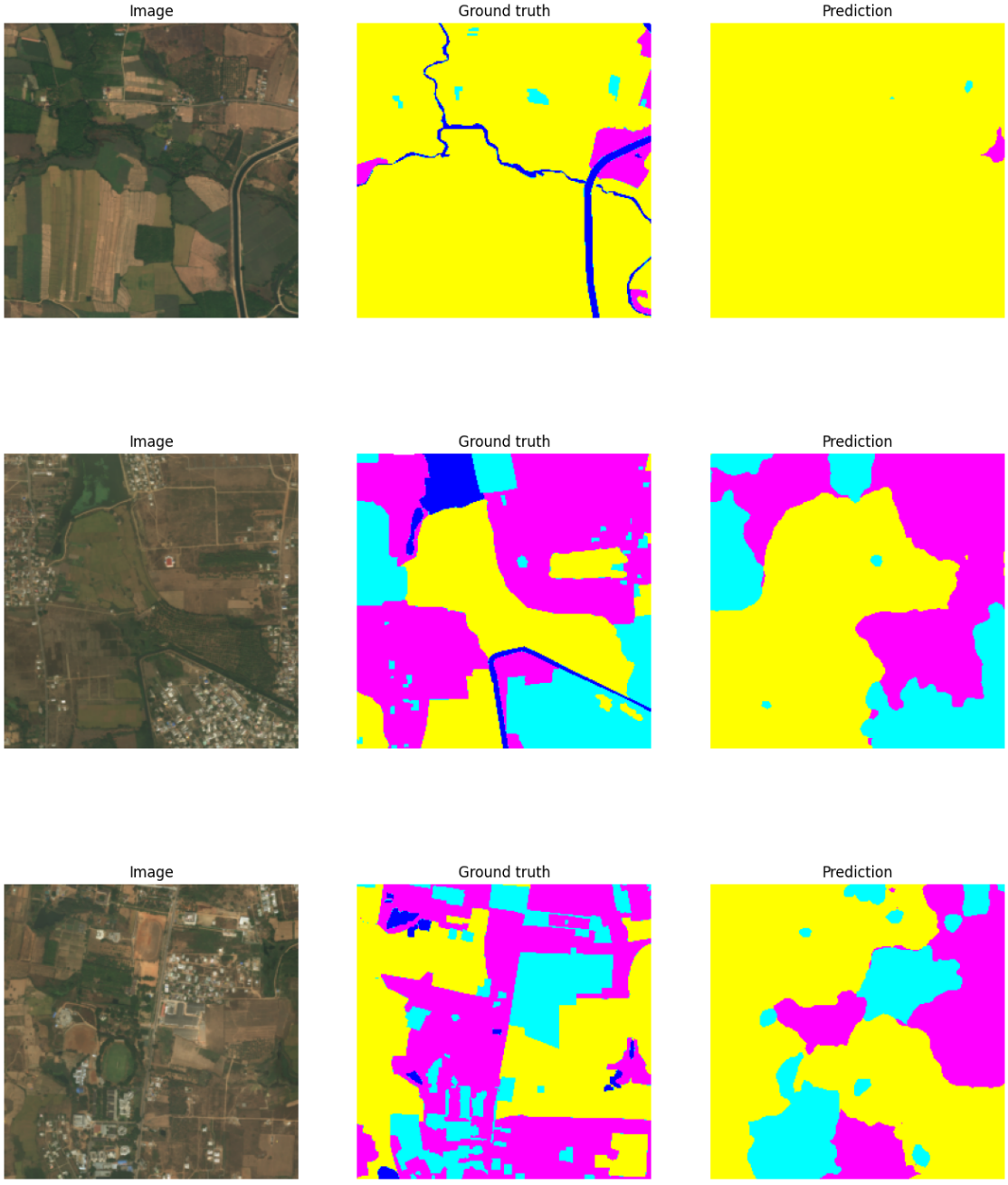


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

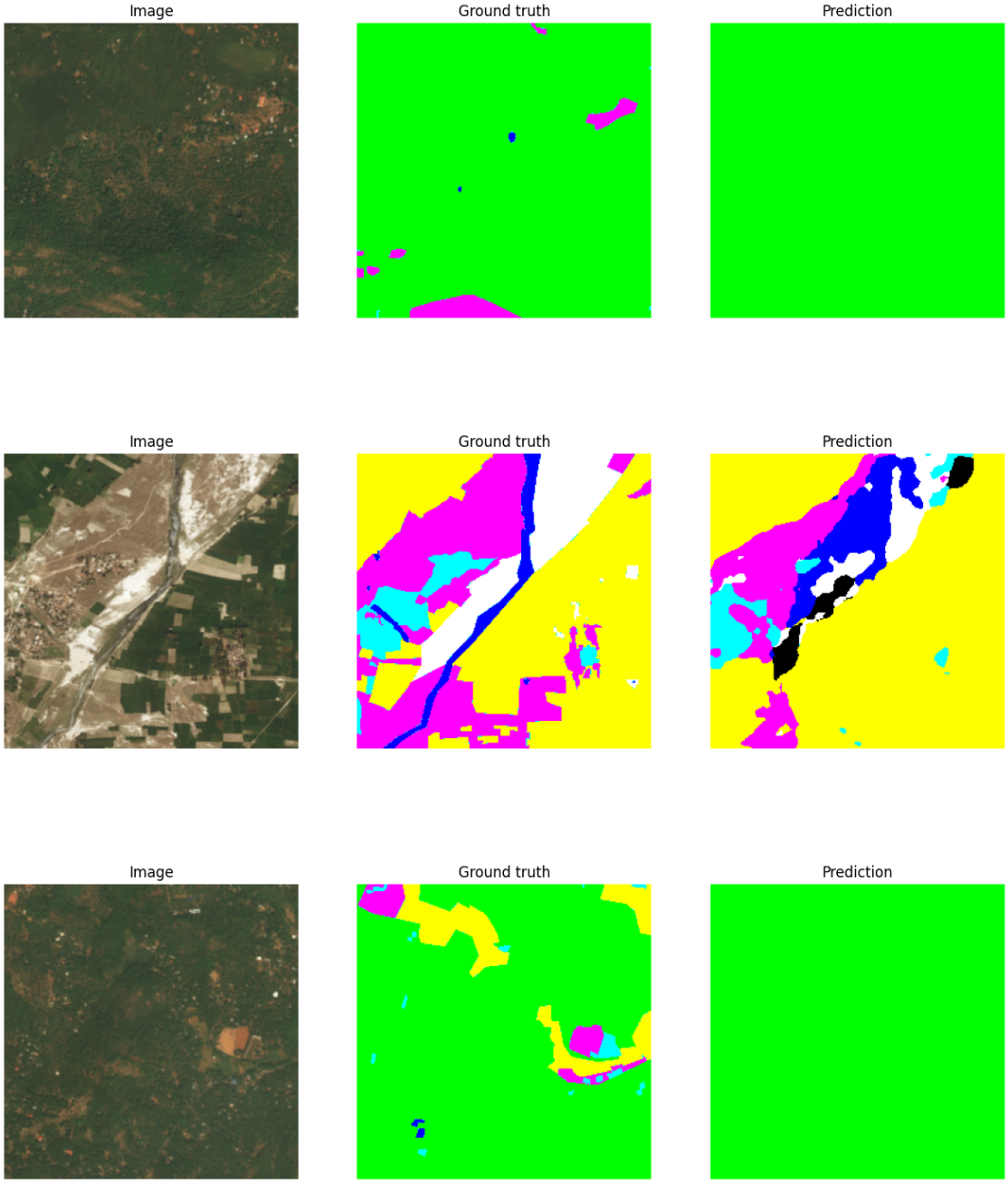
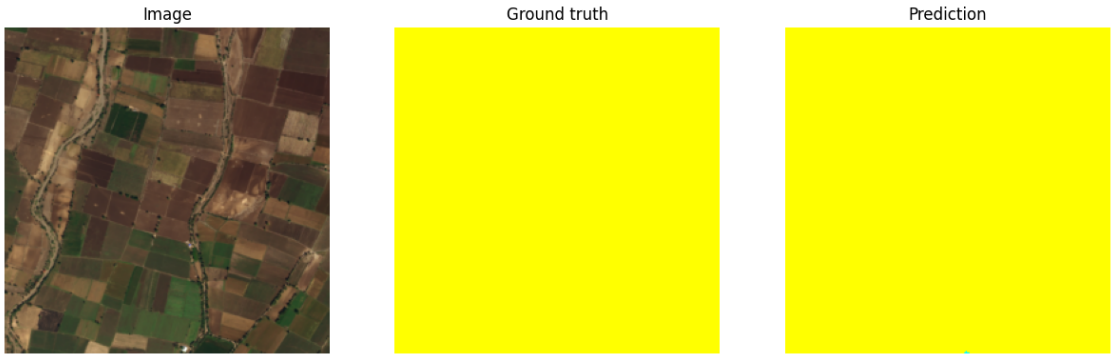


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction



Save Model

[ ]:
```
torch.save(deeplabv3plus.state_dict(), "/content/drive/MyDrive/
 ↪satelite_image_nn/models/deeplabv3plus_resnet34.pth")
```

Test Model

[125]:
```
model4 = SatelliteImageModel("deeplabv3plus", "resnet34", in_channels=3,␣
 ↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="Adam")
model4.load_state_dict(torch.load('/content/drive/MyDrive/satelite_image_nn/
 ↪models/deeplabv3plus_resnet34.pth'))

batch = next(iter(test_dataloader))
with torch.no_grad():
    model4.eval()
    logits = model4(batch["image"])
pr_masks = logits.sigmoid()

for image, pr_mask in zip(batch["image"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")
```

```
    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

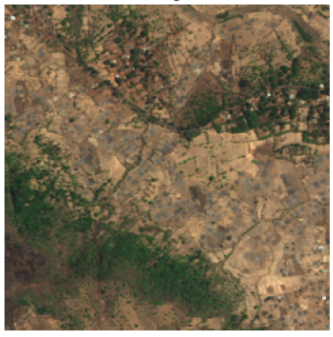
Image

Prediction

Image

Prediction

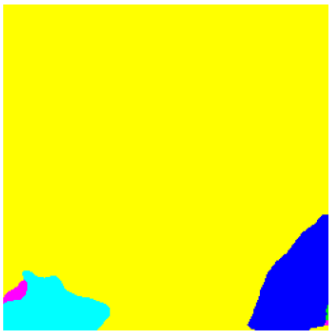
Image



Prediction

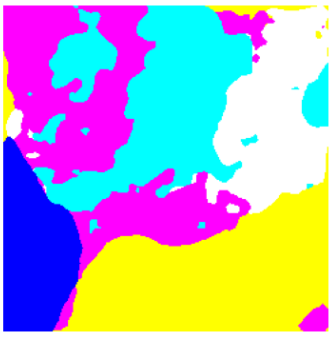
Image

Prediction



Image

Prediction



Image

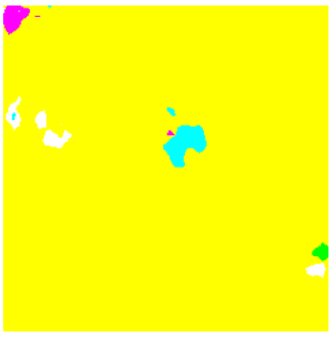
Prediction

Image

Prediction

Image

Prediction



Image

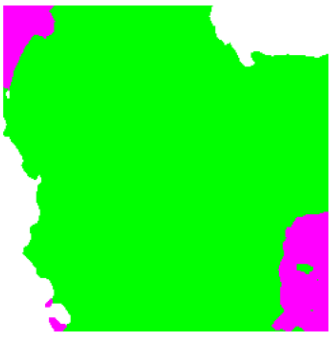
Prediction

Image

Prediction

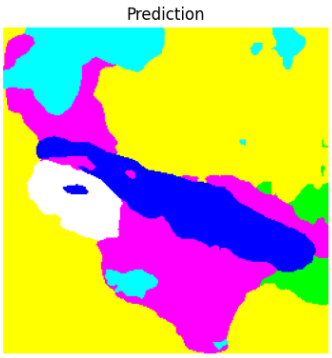
Image

Prediction



Image

Prediction



2.0.5 U-Net Resnet152

[ ]:
```
unet_resnet150 = SatelliteImageModel("unet", "resnet152", in_channels=3,␣
 ↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="Adam")
```

[ ]:
```
# Early stop is a callback that is used to stop the training process when the␣
 ↪validation loss does not improve. In this case, we are
# using the EarlyStopping callback to stop the training process when the␣
 ↪validation loss does not improve for 3 epochs.
earlystop_callback = EarlyStopping('valid_loss', patience=3)

trainer = L.Trainer(max_epochs=20, logger=CSVLogger(save_dir="logs/",␣
 ↪name="pets_seg-model"), callbacks=[earlystop_callback])

trainer.fit(unet_resnet150, train_dataloaders=train_dataloader,␣
 ↪val_dataloaders=val_dataloader)
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:
  | Name    | Type      | Params
-------------------------------------
0 | model   | Unet      | 67.2 M
```

```
1 | loss_fn | DiceLoss | 0
-----------------------------------
67.2 M    Trainable params
0         Non-trainable params
67.2 M    Total params
268.631   Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name    | Type     | Params
-----------------------------------
0 | model   | Unet     | 67.2 M
1 | loss_fn | DiceLoss | 0
-----------------------------------
67.2 M    Trainable params
0         Non-trainable params
67.2 M    Total params
268.631   Total estimated model params size (MB)
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]

INFO: `Trainer.fit` stopped: `max_epochs=20` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=20` reached.
```

[ ]:
```
valid_metrics = trainer.validate(unet_resnet150, dataloaders=val_dataloader,␣
 ↪verbose=False)
print(valid_metrics)
```

```
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]

Validation: |          | 0/? [00:00<?, ?it/s]

[{'valid_loss': 0.40100064873695374, 'valid_per_image_iou': 0.8046537041664124,
'valid_dataset_iou': 0.8046537041664124}]
```

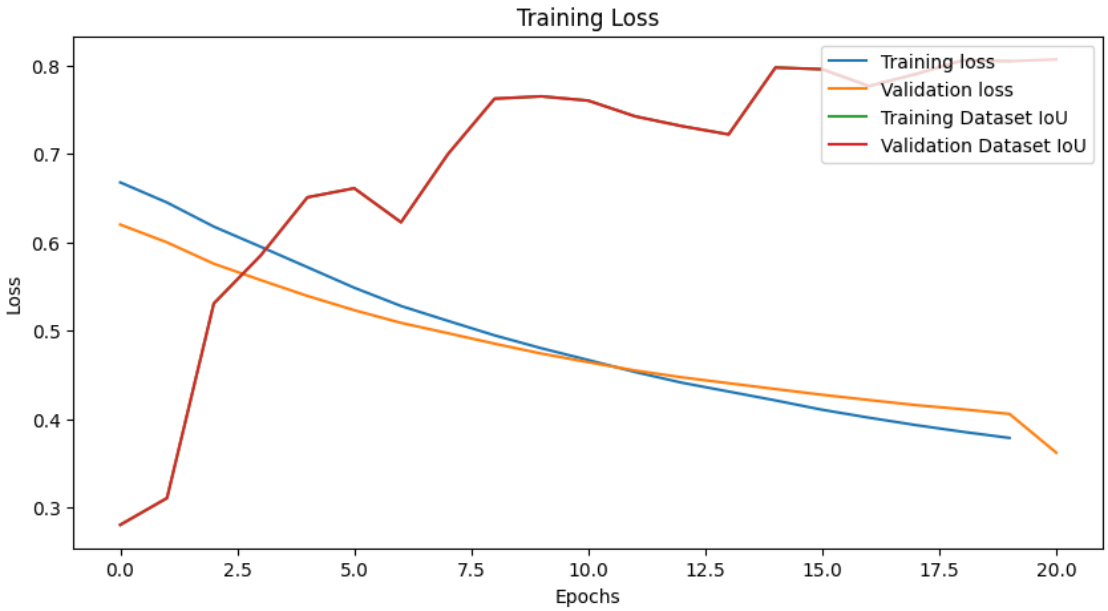
[ ]:
```
# Read the metrics.csv file generated by the PyTorch Lightning logger
metrics = pd.read_csv(f"{trainer.logger.log_dir}/metrics.csv")

# Group the metrics by epoch and compute the mean loss for each epoch
df_epochs = metrics.groupby('epoch').mean()

display(df_epochs)

# Create a figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
# Set the x-axis label
ax.set_xlabel('Epochs')
# Set the y-axis label
ax.set_ylabel('Loss')
ax.plot(df_epochs['train_loss'], label="Training loss")
ax.plot(df_epochs['valid_loss'], label="Validation loss")

ax.plot(df_epochs['train_dataset_iou'], label="Training Dataset IoU")
ax.plot(df_epochs['valid_dataset_iou'], label="Validation Dataset IoU")

# Plot the training loss over epochs
# Plot the validation loss over epochs
# Set the title of the plot
ax.set_title("Training Loss")
# Add a legend to the plot
ax.legend(loc='upper right')
```

```
        step  train_dataset_iou  train_loss  train_per_image_iou  \
epoch
0       40.0           0.280313    0.667606             0.280313
```

1	81.0	0.310628	0.644833	0.310628
2	122.0	0.530661	0.617570	0.530661
3	163.0	0.584852	0.594824	0.584852
4	204.0	0.650698	0.571716	0.650698
5	245.0	0.660968	0.548488	0.660968
6	286.0	0.622520	0.527837	0.622520
7	327.0	0.699387	0.511051	0.699387
8	368.0	0.762289	0.494595	0.762289
9	409.0	0.764899	0.480031	0.764899
10	450.0	0.760064	0.466741	0.760064
11	491.0	0.742307	0.453196	0.742307
12	532.0	0.731215	0.441098	0.731215
13	573.0	0.721868	0.431101	0.721868
14	614.0	0.797452	0.421055	0.797452
15	655.0	0.795556	0.410474	0.795556
16	696.0	0.776477	0.401513	0.776477
17	737.0	0.790088	0.393157	0.790088
18	778.0	0.805648	0.385584	0.805648
19	819.0	0.804654	0.378626	0.804654
20	820.0	NaN	NaN	NaN

epoch	valid_dataset_iou	valid_loss	valid_per_image_iou
0	0.280313	0.619893	0.280313
1	0.310628	0.599768	0.310628
2	0.530661	0.575631	0.530661
3	0.584852	0.557139	0.584852
4	0.650698	0.539201	0.650698
5	0.660968	0.523100	0.660968
6	0.622520	0.508754	0.622520
7	0.699387	0.497103	0.699387
8	0.762289	0.485209	0.762289
9	0.764899	0.474011	0.764899
10	0.760064	0.464293	0.760064
11	0.742307	0.454892	0.742307
12	0.731215	0.447171	0.731215
13	0.721868	0.440509	0.721868
14	0.797452	0.433920	0.797452
15	0.795556	0.427429	0.795556
16	0.776477	0.421501	0.776477
17	0.790088	0.415761	0.790088
18	0.805648	0.411056	0.805648
19	0.804654	0.405793	0.804654
20	0.806608	0.361974	0.806608

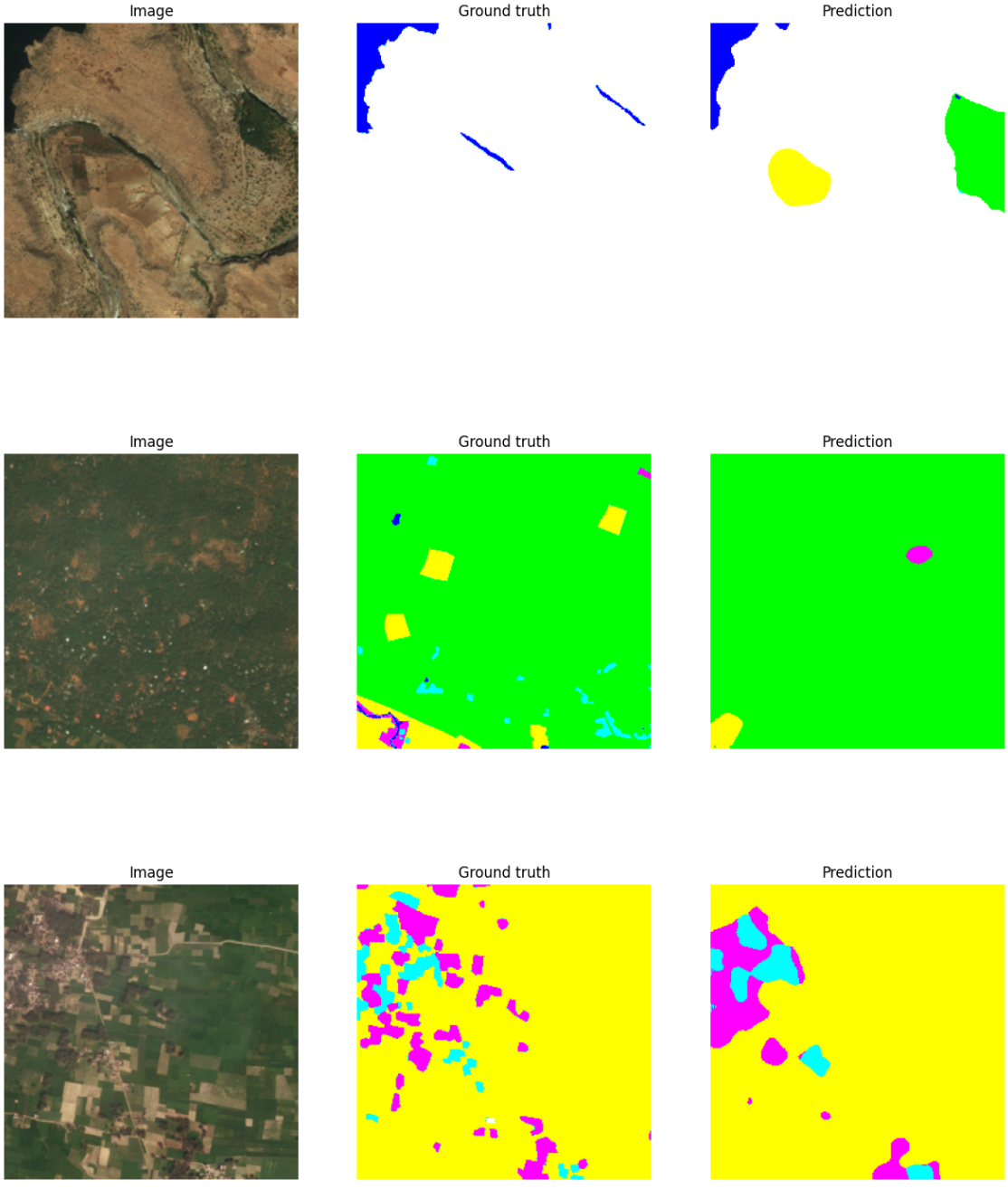
[ ]: <matplotlib.legend.Legend at 0x7f15b45ec4f0>



[ ]:
```
batch = next(iter(val_dataloader))
with torch.no_grad():
    unet_resnet150.eval()
    logits = unet_resnet150(batch["image"])
pr_masks = logits.sigmoid()

for image, gt_mask, pr_mask in zip(batch["image"], batch["mask"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Ground truth
    plt.subplot(1, 3, 2)
    plt.imshow(colorize_mask(gt_mask, color_map))
    plt.title("Ground truth")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
```

```
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



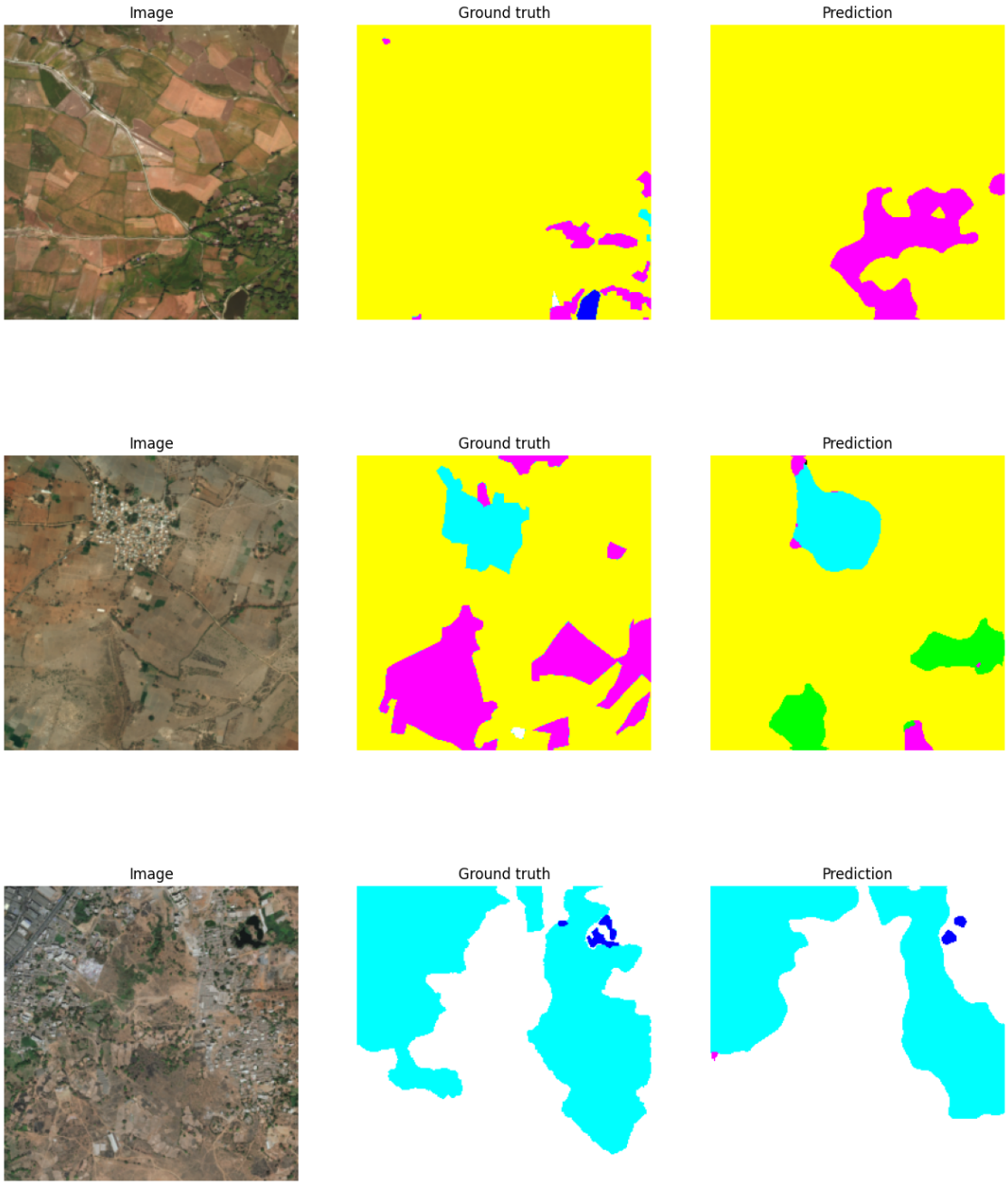


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

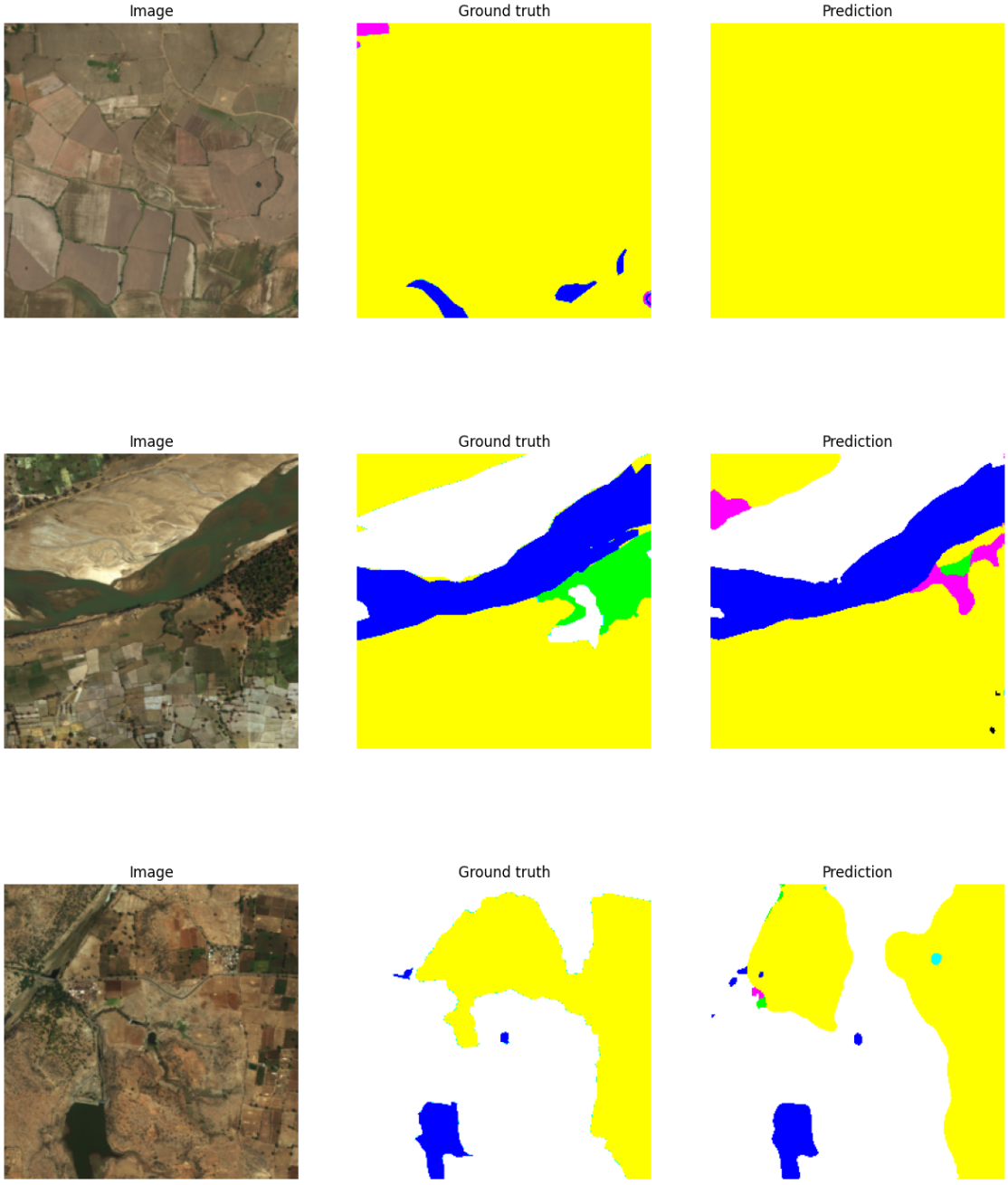


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

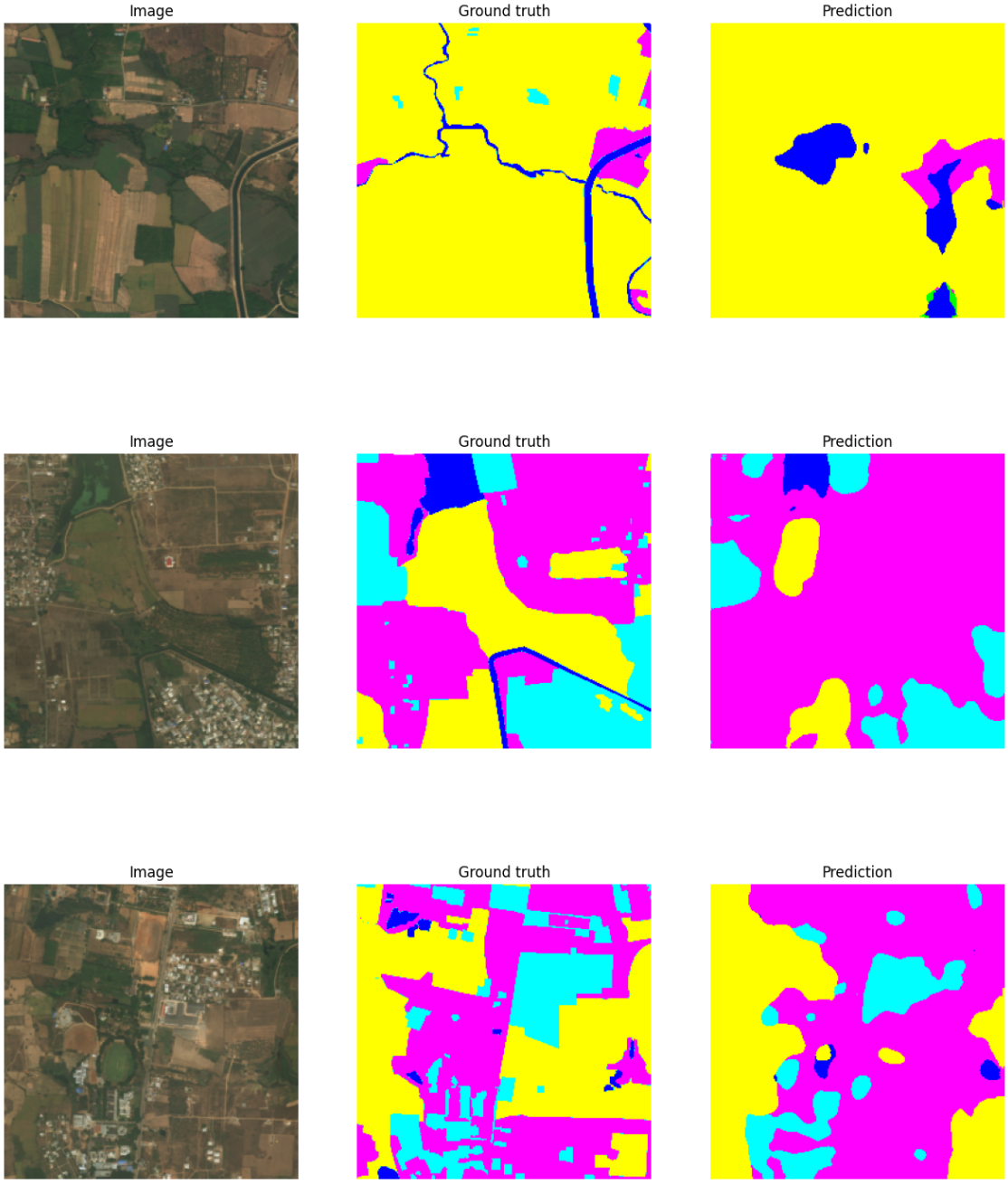


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

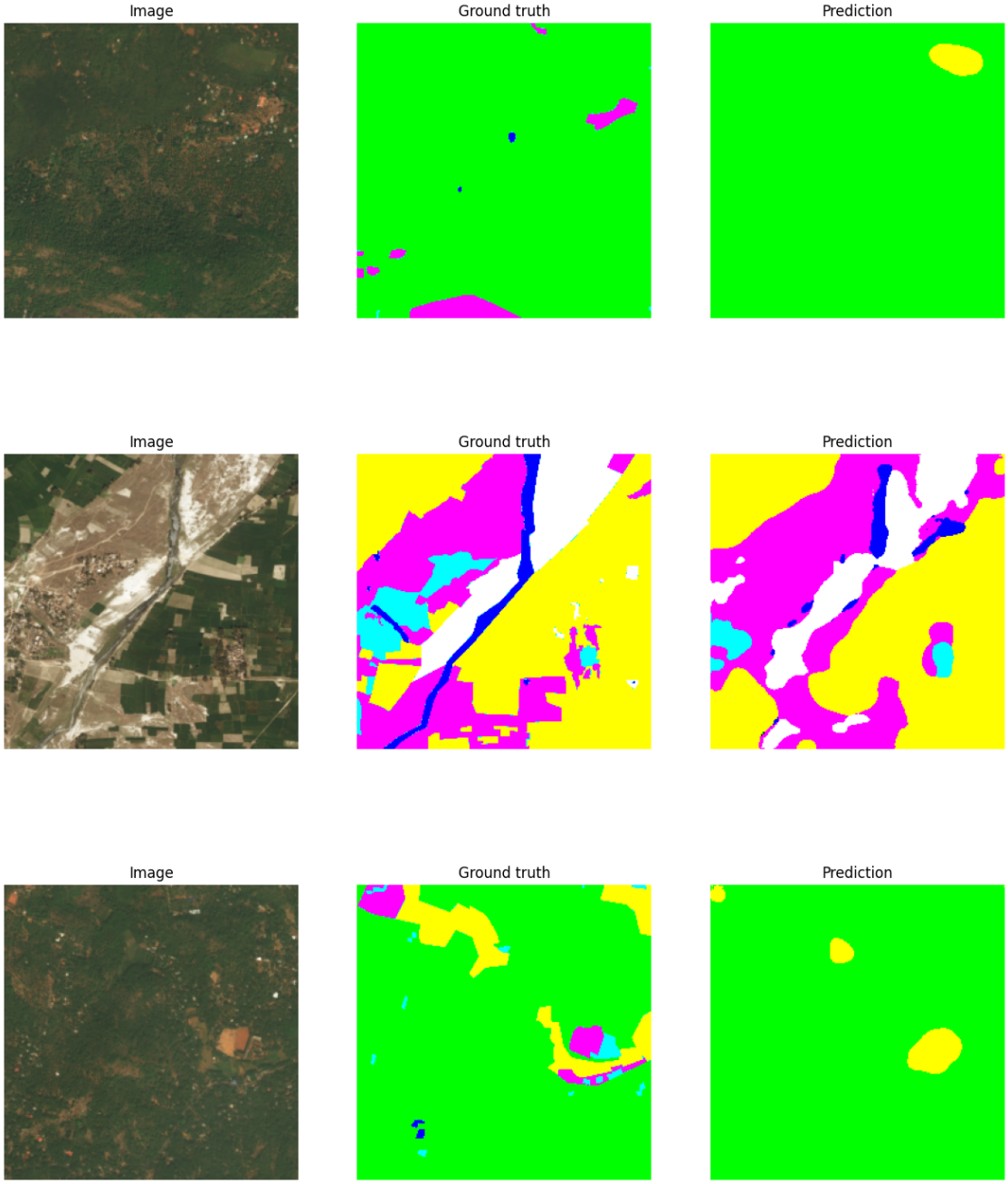
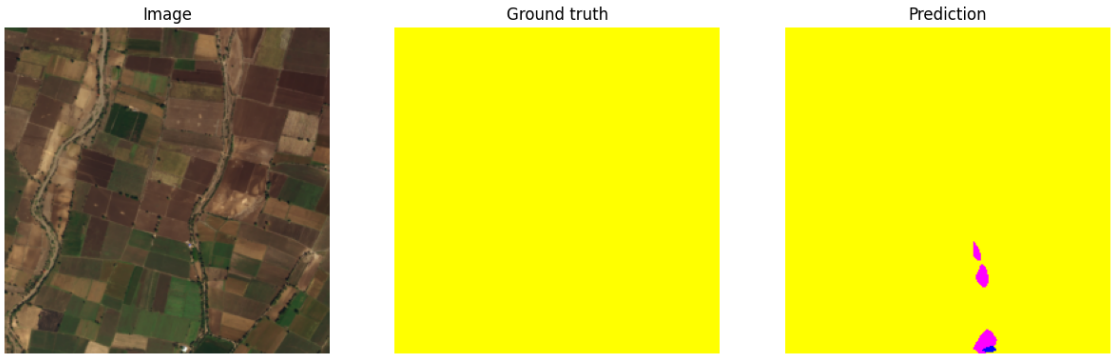


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction



[ ]:
```
torch.save(unet_resnet150.state_dict(), "/content/drive/MyDrive/
 ↪satelite_image_nn/models/unet_resnet152.pth")
```

Test Model

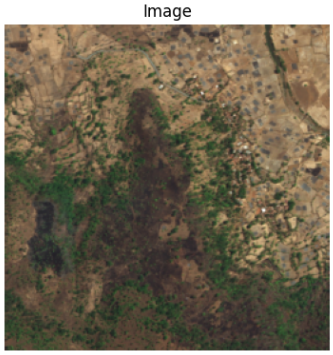
[126]:
```
model5 = SatelliteImageModel("unet", "resnet152", in_channels=3, out_classes=7,␣
 ↪lr = 0.0001, pre_trained=True, optimizer="Adam")
model5.load_state_dict(torch.load('/content/drive/MyDrive/satelite_image_nn/
 ↪models/unet_resnet152.pth'))

batch = next(iter(test_dataloader))
with torch.no_grad():
    model5.eval()
    logits = model5(batch["image"])
pr_masks = logits.sigmoid()

for image, pr_mask in zip(batch["image"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

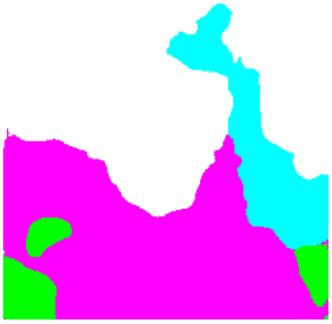
Image

Prediction

Image



Prediction



Image

Prediction

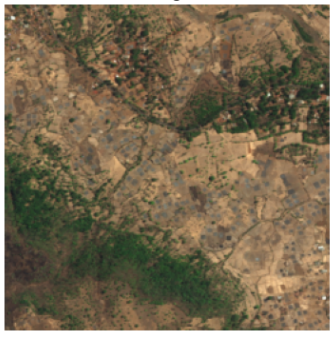
Image

Prediction

Image

Prediction

Image



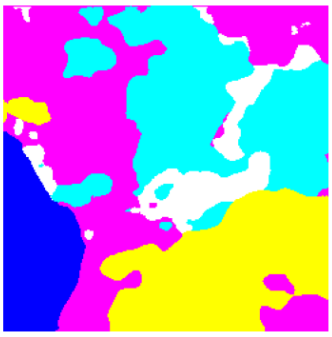
Prediction

Image

Prediction

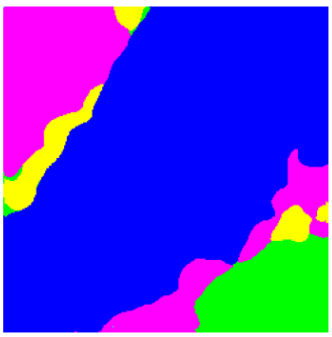
Image

Prediction



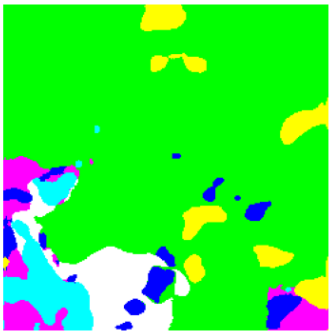
Image

Prediction



Image

Prediction



Image

Prediction

Image

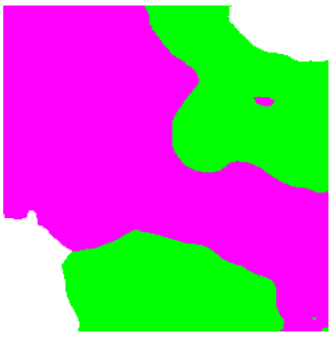
Prediction

Image

Prediction

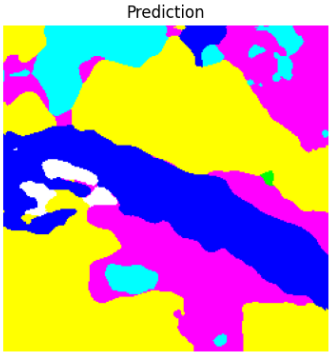
Image

Prediction



Image

Prediction



2.0.6 U-Net Resnet50 AdamW

[ ]:
```
unet_resnet50_adamw = SatelliteImageModel("unet", "resnet50", in_channels=3,␣
 ↪out_classes=7, lr = 0.0001, pre_trained=True, optimizer="AdamW")
```

[ ]:
```
# Early stop is a callback that is used to stop the training process when the␣
 ↪validation loss does not improve. In this case, we are
# using the EarlyStopping callback to stop the training process when the␣
 ↪validation loss does not improve for 3 epochs.
earlystop_callback = EarlyStopping('valid_loss', patience=3)

trainer = L.Trainer(max_epochs=20, logger=CSVLogger(save_dir="logs/",␣
 ↪name="pets_seg-model"), callbacks=[earlystop_callback])

trainer.fit(unet_resnet50_adamw, train_dataloaders=train_dataloader,␣
 ↪val_dataloaders=val_dataloader)
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU
cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
INFO:
  | Name    | Type     | Params
-------------------------------------
0 | model   | Unet     | 32.5 M
```

```
1 | loss_fn | DiceLoss | 0
-------------------------------------
32.5 M    Trainable params
0         Non-trainable params
32.5 M    Total params
130.088   Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name    | Type     | Params
-------------------------------------
0 | model   | Unet     | 32.5 M
1 | loss_fn | DiceLoss | 0
-------------------------------------
32.5 M    Trainable params
0         Non-trainable params
32.5 M    Total params
130.088   Total estimated model params size (MB)

Sanity Checking: |          | 0/? [00:00<?, ?it/s]

Training: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]

Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
INFO: `Trainer.fit` stopped: `max_epochs=20` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped:
`max_epochs=20` reached.
```

[ ]:
```
valid_metrics = trainer.validate(unet_resnet50_adamw,␣
  ↪dataloaders=val_dataloader, verbose=False)
print(valid_metrics)
```

```
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
[0]
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
```

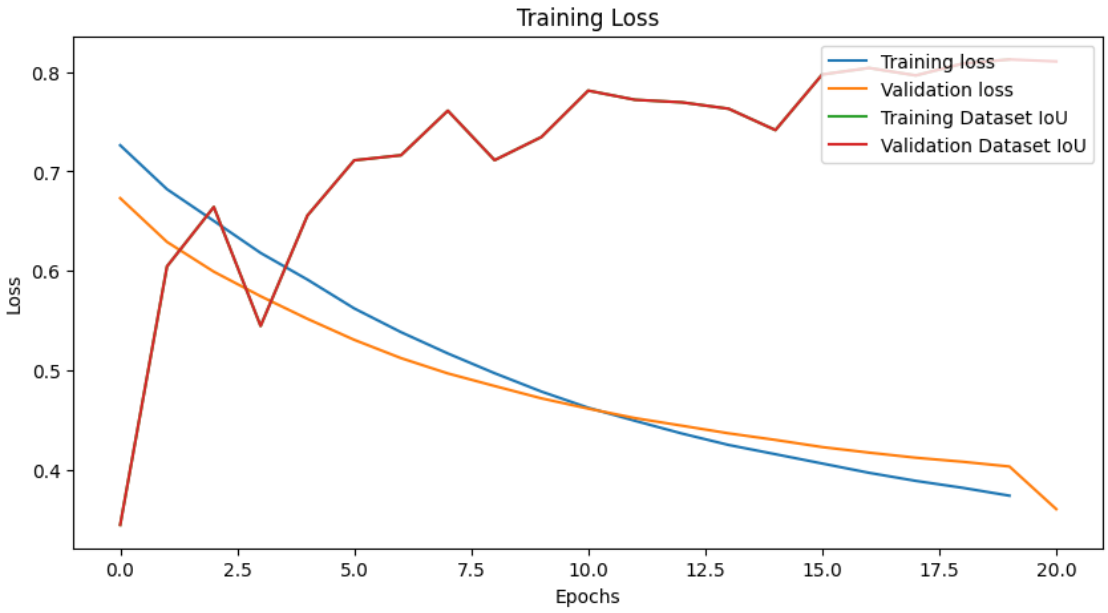
```
[{'valid_loss': 0.39905327558517456, 'valid_per_image_iou': 0.8125147223472595,
'valid_dataset_iou': 0.8125147223472595}]
```

[ ]:
```
# Read the metrics.csv file generated by the PyTorch Lightning logger
metrics = pd.read_csv(f"{trainer.logger.log_dir}/metrics.csv")

# Group the metrics by epoch and compute the mean loss for each epoch
df_epochs = metrics.groupby('epoch').mean()

display(df_epochs)

# Create a figure and axis for plotting
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
# Set the x-axis label
ax.set_xlabel('Epochs')
# Set the y-axis label
ax.set_ylabel('Loss')
ax.plot(df_epochs['train_loss'], label="Training loss")
ax.plot(df_epochs['valid_loss'], label="Validation loss")

ax.plot(df_epochs['train_dataset_iou'], label="Training Dataset IoU")
ax.plot(df_epochs['valid_dataset_iou'], label="Validation Dataset IoU")

# Plot the training loss over epochs
# Plot the validation loss over epochs
# Set the title of the plot
ax.set_title("Training Loss")
# Add a legend to the plot
ax.legend(loc='upper right')
```

epoch	step	train_dataset_iou	train_loss	train_per_image_iou	\
0	40.0	0.344714	0.726241	0.344714	
1	81.0	0.604524	0.682087	0.604524	
2	122.0	0.664216	0.650243	0.664216	
3	163.0	0.544659	0.618044	0.544659	
4	204.0	0.655639	0.591262	0.655639	
5	245.0	0.711145	0.562285	0.711145	
6	286.0	0.716231	0.538408	0.716231	
7	327.0	0.761034	0.517017	0.761034	
8	368.0	0.711301	0.497167	0.711301	
9	409.0	0.734560	0.478707	0.734560	
10	450.0	0.781184	0.462458	0.781184	
11	491.0	0.771986	0.449292	0.771986	
12	532.0	0.769426	0.436565	0.769426	
13	573.0	0.762976	0.425042	0.762976	
14	614.0	0.741520	0.415757	0.741520	
15	655.0	0.797427	0.406400	0.797427	
16	696.0	0.803958	0.397053	0.803958	
17	737.0	0.796442	0.388930	0.796442	
18	778.0	0.808364	0.381961	0.808364	
19	819.0	0.812515	0.374050	0.812515	
20	820.0	NaN	NaN	NaN	

epoch	valid_dataset_iou	valid_loss	valid_per_image_iou
0	0.344714	0.673088	0.344714
1	0.604524	0.629107	0.604524
2	0.664216	0.599196	0.664216
3	0.544659	0.574524	0.544659
4	0.655639	0.551816	0.655639
5	0.711145	0.530708	0.711145
6	0.716231	0.512315	0.716231
7	0.761034	0.496948	0.761034
8	0.711301	0.484289	0.711301
9	0.734560	0.471889	0.734560
10	0.781184	0.461561	0.781184
11	0.771986	0.452026	0.771986
12	0.769426	0.444415	0.769426
13	0.762976	0.436734	0.762976
14	0.741520	0.430108	0.741520
15	0.797427	0.422909	0.797427
16	0.803958	0.417343	0.803958
17	0.796442	0.412211	0.796442
18	0.808364	0.408156	0.808364
19	0.812515	0.403379	0.812515
20	0.810539	0.360545	0.810539

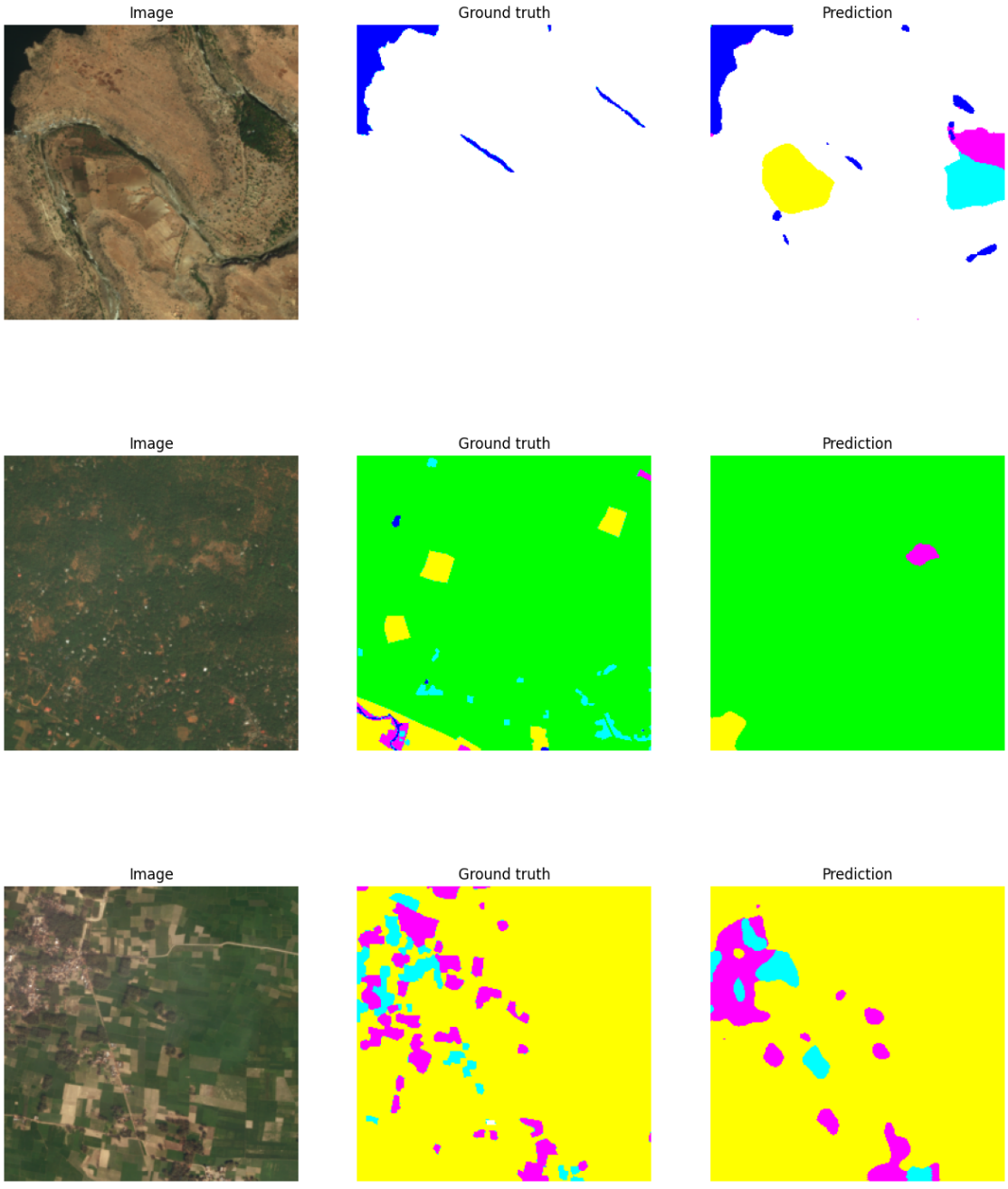
[ ]: <matplotlib.legend.Legend at 0x7f15b4cb2d70>



[ ]:
```
batch = next(iter(val_dataloader))
with torch.no_grad():
    unet_resnet50_adamw.eval()
    logits = unet_resnet50_adamw(batch["image"])
pr_masks = logits.sigmoid()

for image, gt_mask, pr_mask in zip(batch["image"], batch["mask"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Ground truth
    plt.subplot(1, 3, 2)
    plt.imshow(colorize_mask(gt_mask, color_map))
    plt.title("Ground truth")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
 ↪class
```

```
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```



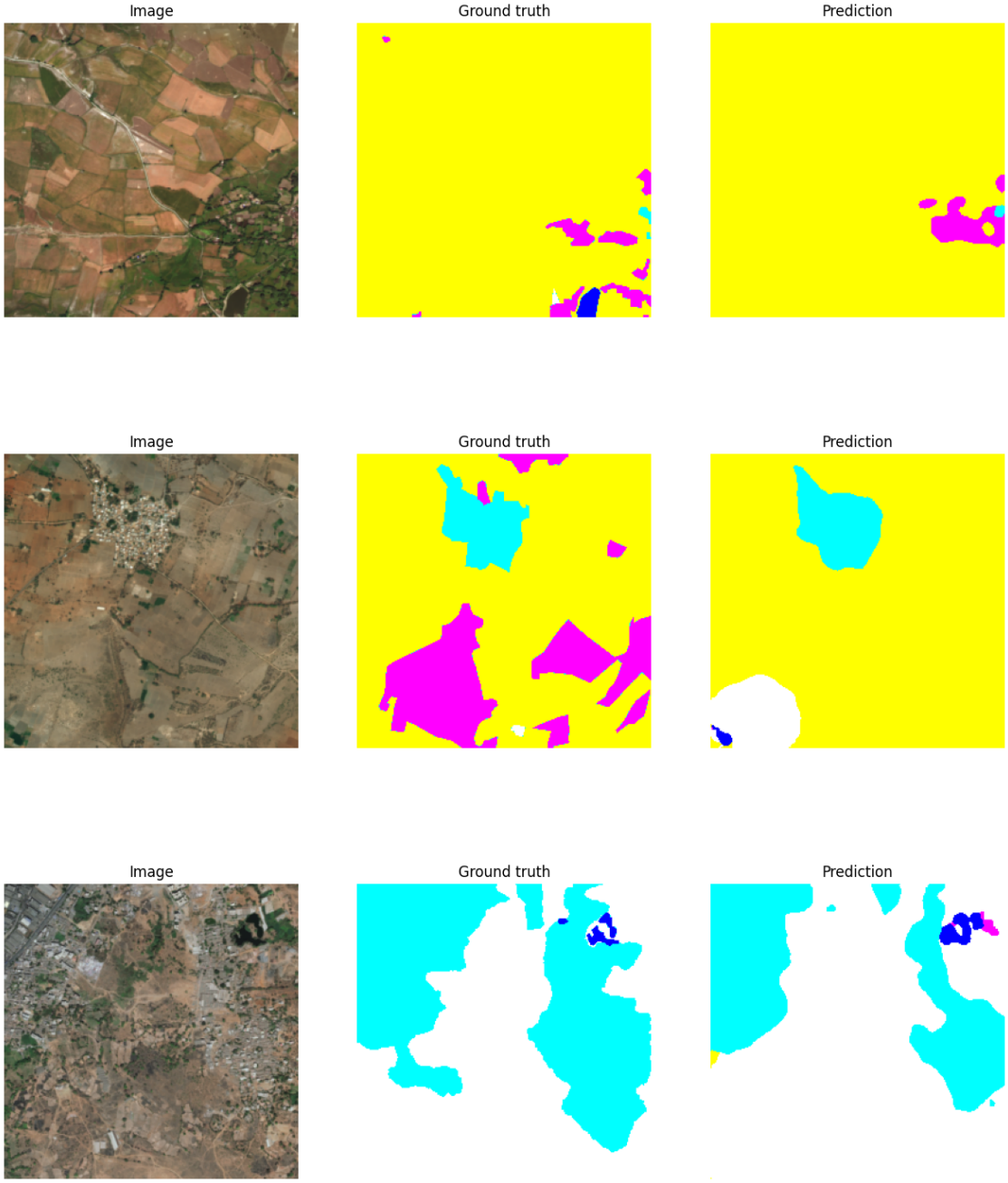


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

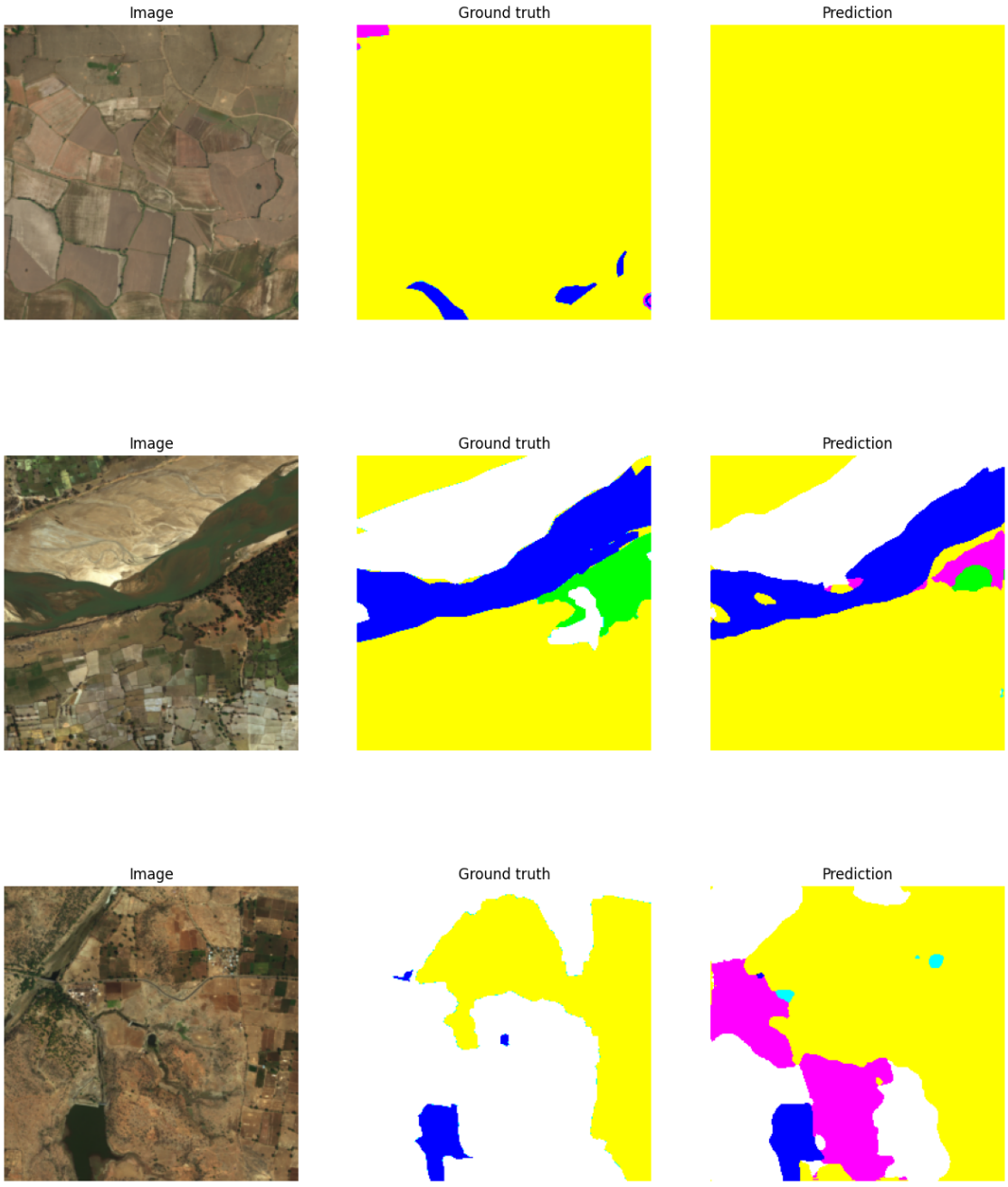


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

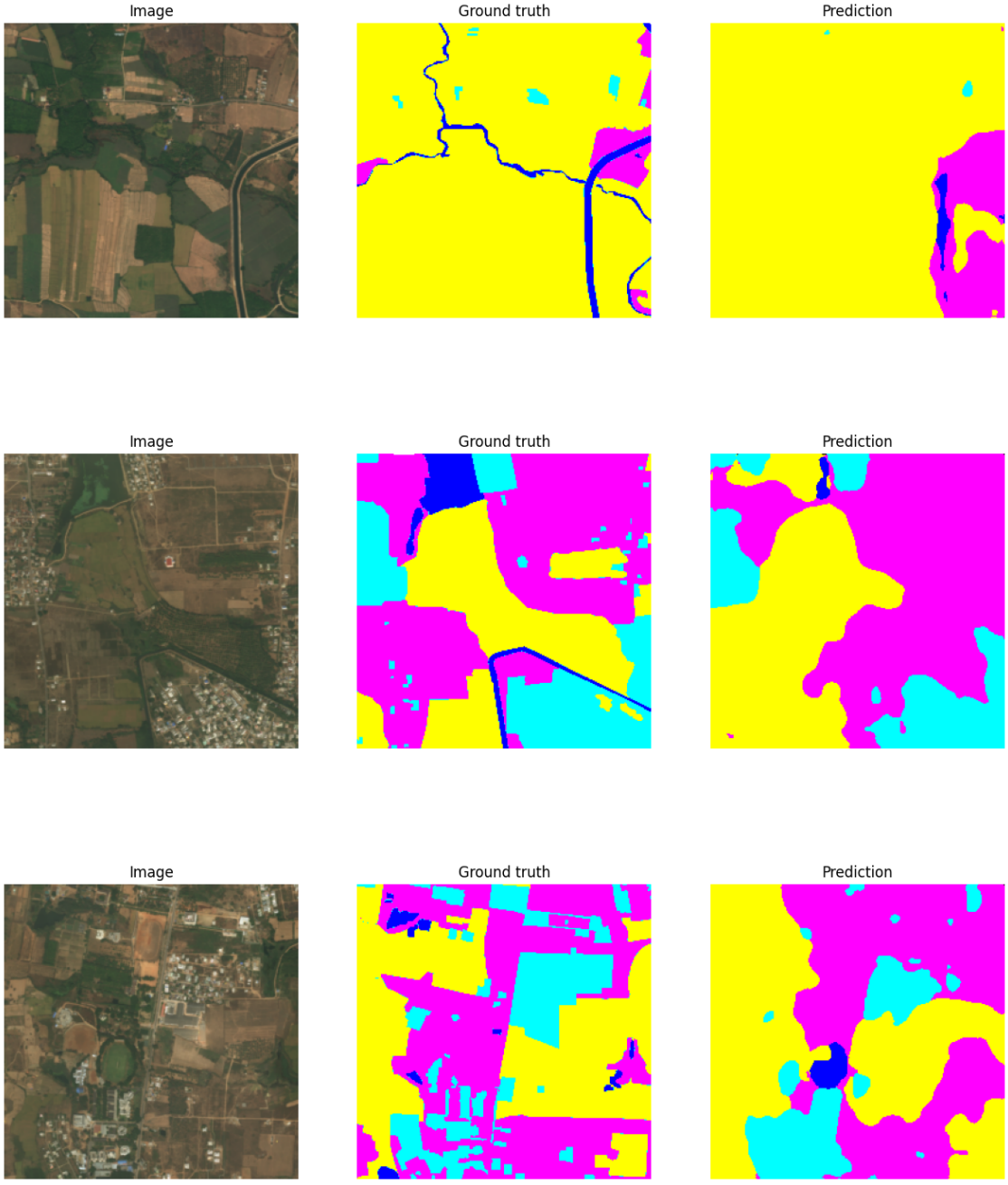


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction

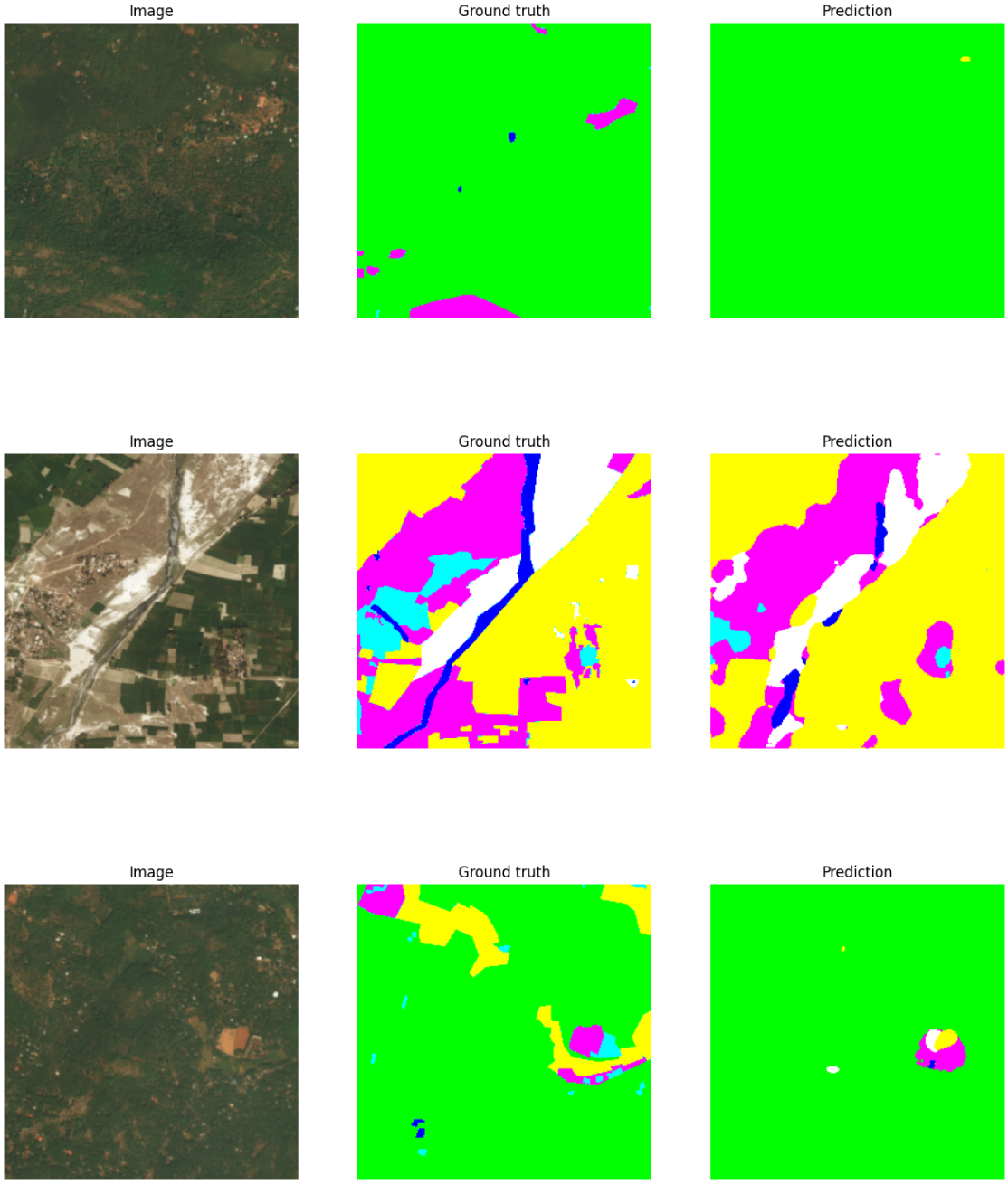
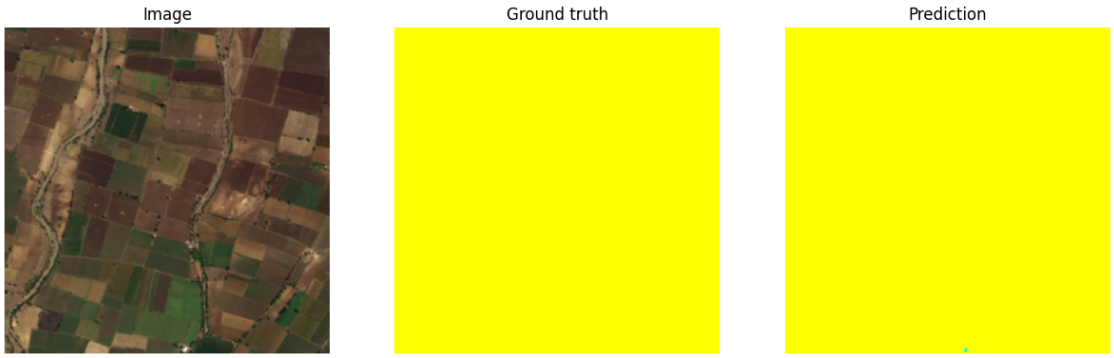


Image
Ground truth
Prediction
Image
Ground truth
Prediction
Image
Ground truth
Prediction



Save Model

[ ]:
```
torch.save(unet_resnet50_adamw.state_dict(), "/content/drive/MyDrive/
↪satelite_image_nn/models/unet_resnet50_adamw.pth")
```

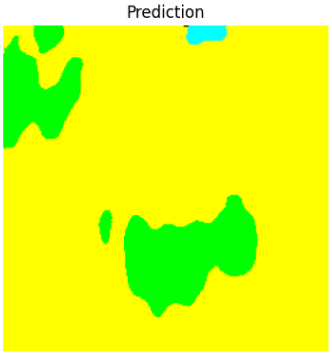
Test Model

[127]:
```
model6 = SatelliteImageModel("unet", "resnet50", in_channels=3, out_classes=7,␣
↪lr = 0.0001, pre_trained=True, optimizer="AdamW")
model6.load_state_dict(torch.load('/content/drive/MyDrive/satelite_image_nn/
↪models/unet_resnet50_adamw.pth'))

batch = next(iter(test_dataloader))
with torch.no_grad():
    model6.eval()
    logits = model6(batch["image"])
pr_masks = logits.sigmoid()

for image, pr_mask in zip(batch["image"], pr_masks):
    plt.figure(figsize=(15, 5))

    # Image
    plt.subplot(1, 3, 1)
    plt.imshow(image.numpy().transpose(1, 2, 0))
    plt.title("Image")
    plt.axis("off")

    # Prediction
    plt.subplot(1, 3, 3)
    pr_mask = pr_mask.argmax(0)  # Convert class probabilities to most likely␣
↪class
    plt.imshow(colorize_mask(pr_mask, color_map))
    plt.title("Prediction")
    plt.axis("off")
```

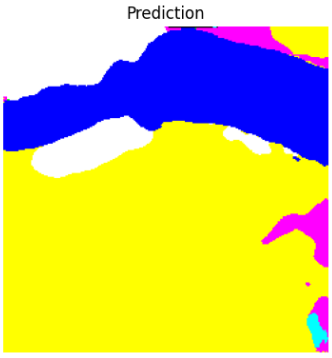
```
    plt.show()
```

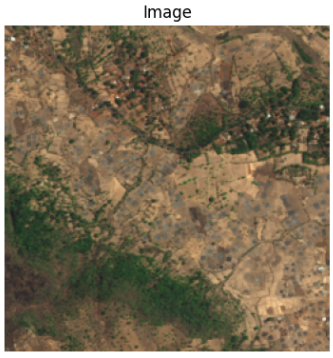
```
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork()
was called. os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
```


Image

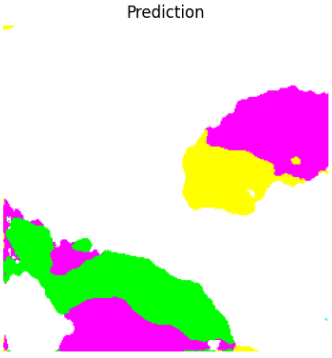

Prediction


Image


Prediction



Image


Prediction

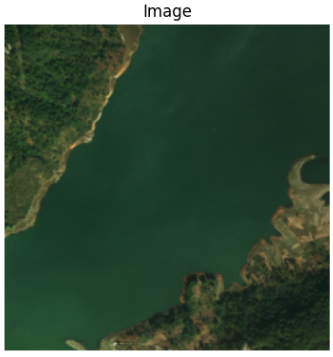
Image

Prediction

Image

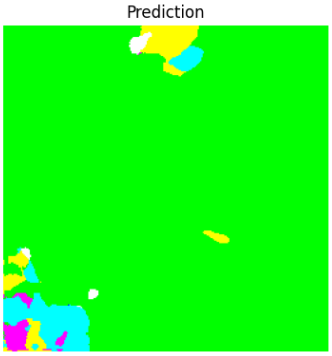
Prediction

Image

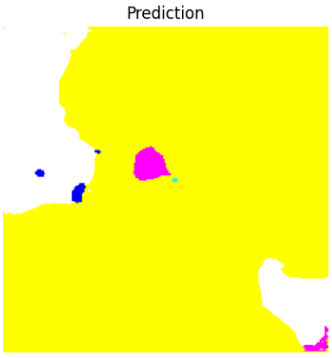


Prediction

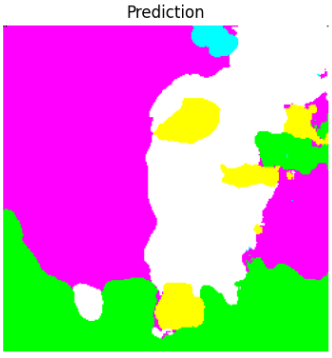

Image


Prediction


Image


Prediction


Image

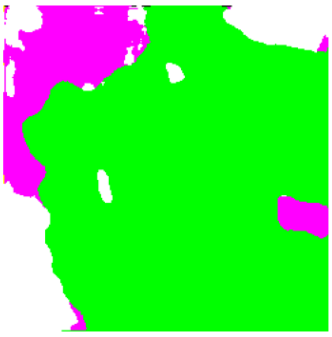

Prediction

Image

Prediction

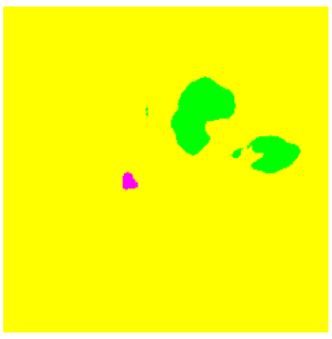
Image

Prediction

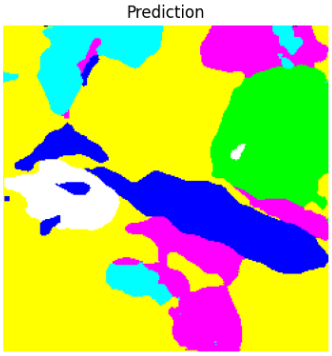


Image

Prediction




Image


Prediction