sentinel2-cloud-detector-example_from_file

September 21, 2021

1 Sentinel Hub's Sentinel-2 Cloud Detector

This example notebook shows how to perform cloud classification and cloud masking on Sentinel-2 data.

In the process we will use `sentinelhub-py` Python package. The package documentation is available here.

1.1 Prerequisite

1.1.1 Imports

[ ]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline

import datetime as dt

import matplotlib.pyplot as plt
import numpy as np

from sentinelhub import SHConfig, BBox, CRS, SentinelHubRequest,␣
 ↪DataCollection, \
    MimeType, bbox_to_dimensions

from s2cloudless import S2PixelCloudDetector, CloudMaskRequest,␣
 ↪get_s2_evalscript

from plotting_utils import plot_image, plot_probabilities
```

1.1.2 Sentinel Hub credentials

These examples require Sentinel Hub OAuth client credentials. Please check `sentinelhub-py` configuration instructions on how to obtain the credentials and configure them in the package.

[ ]:
```
# In case you put the credentials into the configuration file you can leave␣
 ↪this unchanged
CLIENT_ID = "86ad3cec-1e7e-4b43-8145-1684ac658740"
CLIENT_SECRET = "ny9tEbtGx075%;Bm~3E5M_gYAFiu/j0)vQc4.4E_"
```

```
config = SHConfig()

if CLIENT_ID and CLIENT_SECRET:
    config.sh_client_id = CLIENT_ID
    config.sh_client_secret = CLIENT_SECRET
```

1.2 Download and calculate cloud masks

In the first example we will separate the process in 2 parts:

1. download satellite data
2. calculate cloud masks and probabilities from data

This will show how to perform cloud detection independetly from obtaining data.

1.2.1 Example scenes: Acatenango and Volcan Fuego (Guatemala) in December 2017

Acatenango area in Guatemala is well known for its coffee plantations. At the altitute of about 2000 m and given it's climate, it is often veiled in clouds.

First, lets define the bounding box for the area of interest:

[ ]:
```
#bbox = BBox([-90.9217, 14.4191, -90.8187, 14.5520], crs=CRS.WGS84)
bbox = BBox([-86.6953, 34.6892, -86.5923, 34.7613], crs=CRS.WGS84)
```

1.2.2 Download Sentinel-2 data

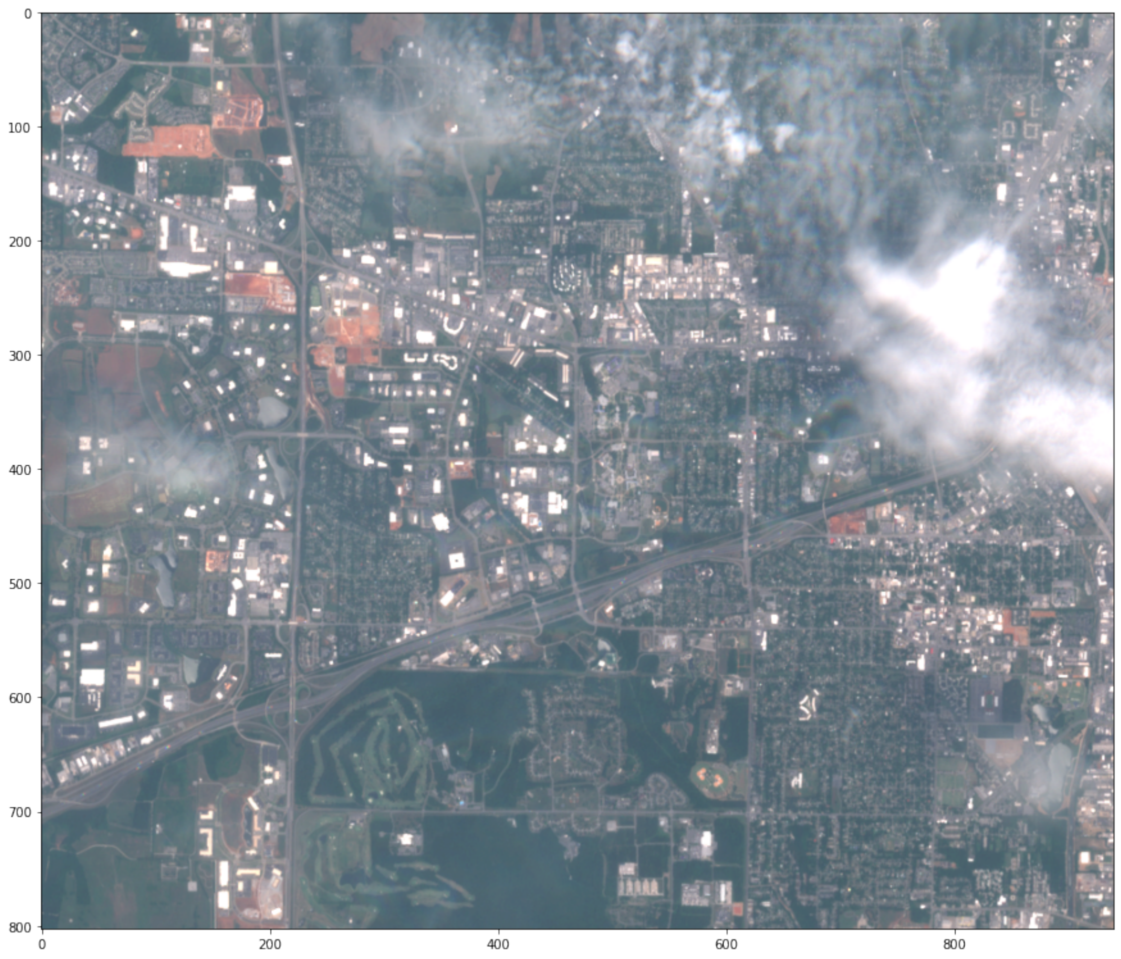
Let's download data using Sentinel Hub Process API. First let's download a true color image for which we want detect clouds. The downloaded image will be on a resolution 10 meter per pixel.

[ ]:
```
evalscript_true_color = """
    //VERSION=3

    function setup() {
        return {
            input: [{
                bands: ["B02", "B03", "B04"]
            }],
            output: {
                bands: 3
            }
        };
    }

    function evaluatePixel(sample) {
        return [sample.B04, sample.B03, sample.B02];
    }
"""
```

```
request = SentinelHubRequest(
    evalscript=evalscript_true_color,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval='2021-06-01'
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.PNG)
    ],
    bbox=bbox,
    size=bbox_to_dimensions(bbox, 10),
    config=config
)

true_color_image = request.get_data()[0]

# Write true color data to file

file_color = open("hsv_color.dat", "wb")
np.save(file_color, true_color_image)
file_color.close

true_color_image.shape
```

[ ]:
```
(803, 940, 3)
```

[ ]:
```
#plot_image(true_color_image)

test_file = open("hsv_color.dat", "rb")
true_color_test = np.load(test_file)
plot_image(true_color_test)
```



Next, let's download remaining Sentinel-2 bands. `s2cloudless` detector only requires `10` out of `13` bands for cloud detection. The following utility will create an evalscript for requesting those bands.

For simplicity we will request data with reflectance values. To decrease download costs a better option is to download data in digital numbers (i.e. unsigned ints) and then rescale them with normalization factors. That is implemented in `CloudMaskRequest` class which will be shown below.

```
[ ]: evalscript = get_s2_evalscript(
         all_bands=False,
         reflectance=True
     )

     print(evalscript)
```

```
//VERSION=3
function setup() {
  return {
    input: [{
      bands: ["B01", "B02", "B04", "B05", "B08", "B8A", "B09", "B10", "B11",
"B12", "dataMask"],
      units: "reflectance"
    }],
    output: {
      bands: 11,
      sampleType: "FLOAT32"
    }
  };
}

function evaluatePixel(sample) {
  return [sample.B01, sample.B02, sample.B04, sample.B05, sample.B08,
sample.B8A, sample.B09, sample.B10, sample.B11, sample.B12, sample.dataMask];
}
```

[ ]:
```
request = SentinelHubRequest(
    evalscript=evalscript,
    input_data=[
        SentinelHubRequest.input_data(
            data_collection=DataCollection.SENTINEL2_L1C,
            time_interval='2021-06-01'
        )
    ],
    responses=[
        SentinelHubRequest.output_response('default', MimeType.TIFF)
    ],
    bbox=bbox,
    size=bbox_to_dimensions(bbox, 10),
    config=config
)

data = request.get_data()[0]

bands = data[..., :-1]
mask = data[..., -1]

file_data = open("hsv_data.dat", "wb")
np.save(file_data, bands)
file_data.close
```

[ ]:
```
<function BufferedWriter.close>
```

[ ]:
```
test_file2 = open("data_1.dat", "rb")
bands_test = np.load(test_file2)

#bands.shape, mask.shape
```

1.2.3 Initialize the cloud detector and make classification

We can specify the following arguments in the initialization of a `S2PixelCloudDetector`:

- `threshold` - cloud probability threshold value. All pixels with cloud probability above threshold value are masked as cloudy pixels. Default is `0.4`.
- `average_over` - Size of the disk in pixels for performing convolution (averaging probability over pixels). For this resolution `4` is appropriate.
- `dilation_size` - Size of the disk in pixels for performing dilation. For this resolution `2` is appropriate.
- `all_bands` - Flag specifying that input images will consists of all 13 Sentinel-2 bands. It has to be set to `True` if we would download all bands. If you define a layer that would return only 10 bands, then this parameter should be set to `False`.

[ ]:
```
cloud_detector = S2PixelCloudDetector(
    threshold=0.4,
    average_over=4,
    dilation_size=2,
    all_bands=False
)
```

Run the classification There are two possibilities: * `get_cloud_probability_maps` will return cloud probability map * `get_cloud_masks` will return binary cloud masks

[ ]:
```
%%time

cloud_prob = cloud_detector.get_cloud_probability_maps(bands_test)
```

```
CPU times: user 4min 54s, sys: 1.14 s, total: 4min 56s
Wall time: 1min 31s
```

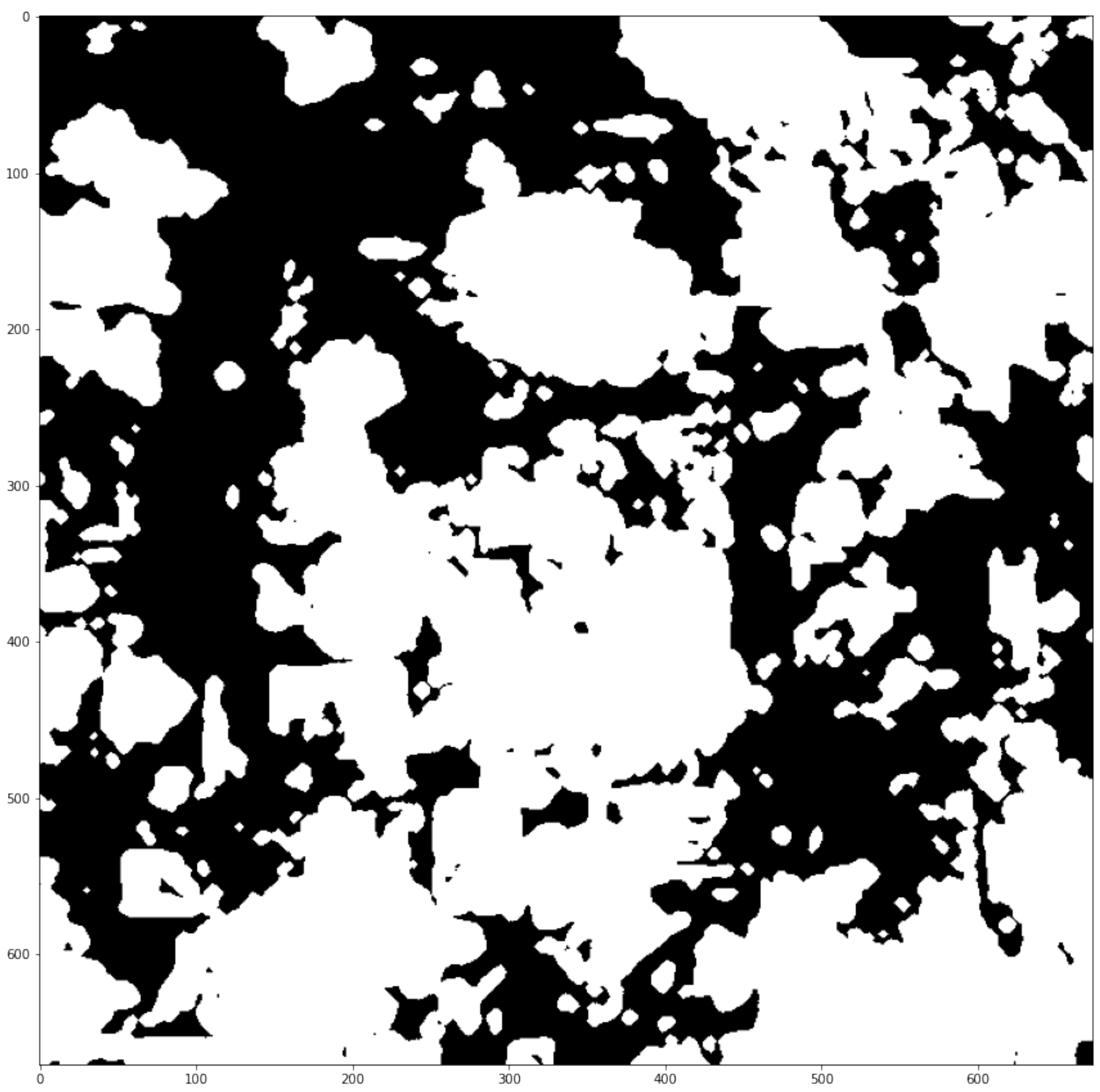
[ ]:
```
%%time

cloud_mask = cloud_detector.get_cloud_masks(bands_test)
```

```
CPU times: user 4min 33s, sys: 1.01 s, total: 4min 34s
Wall time: 1min 22s
```

1.2.4 Visualize the results

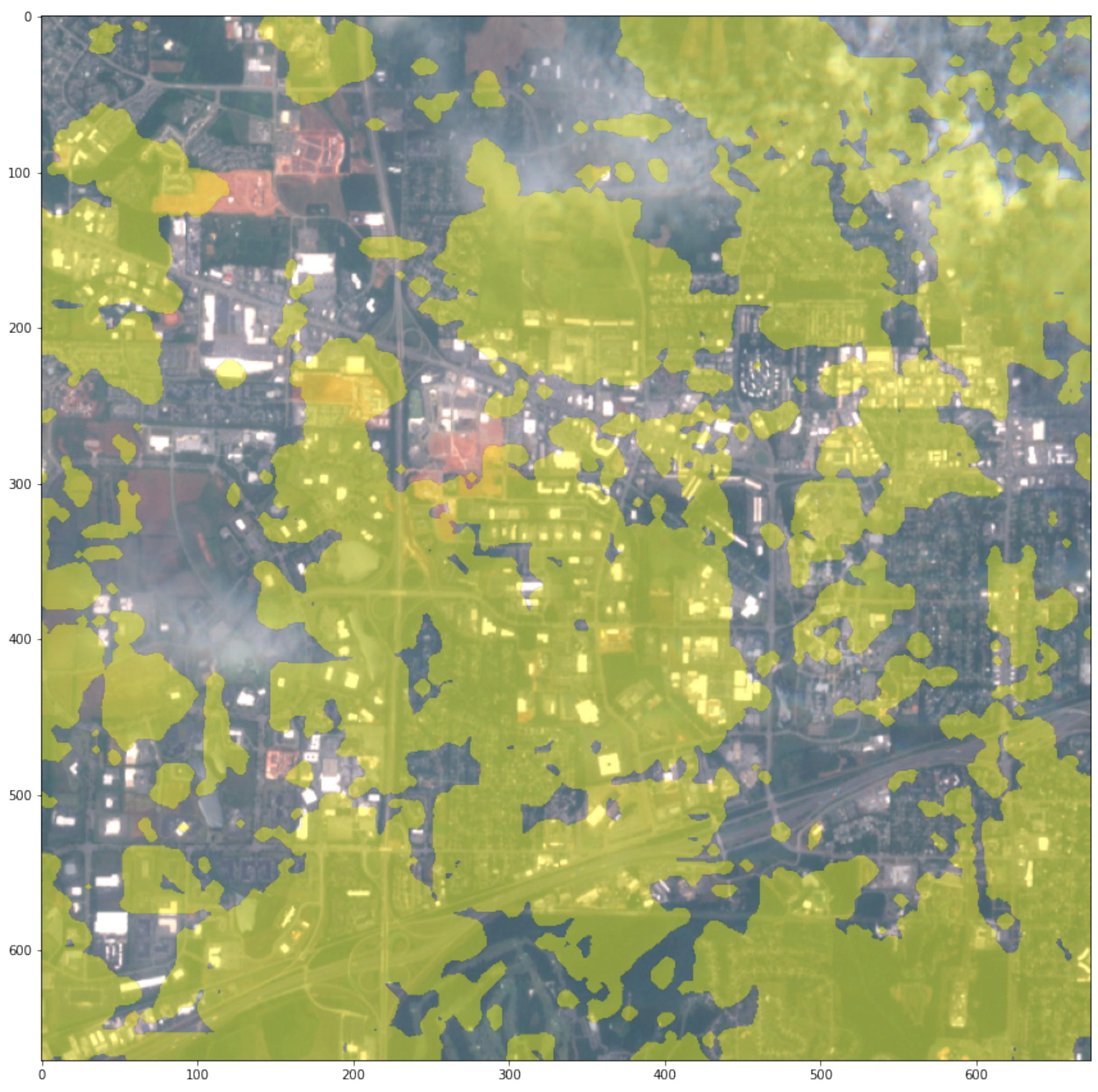
We have a binary cloud mask:

[ ]:
```
plot_image(mask=cloud_mask)
```



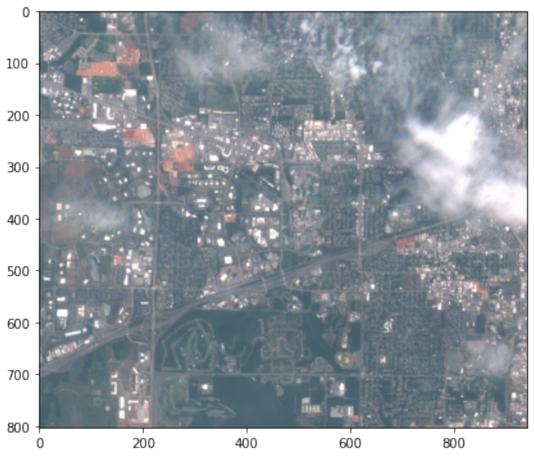
Let's plot a true color image overlaid with the cloud mask:

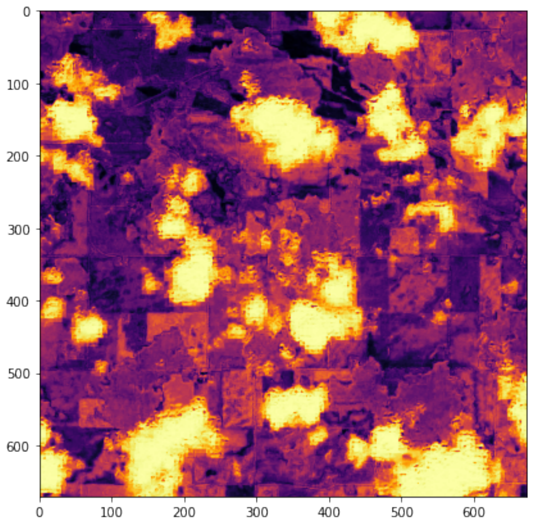
[ ]:
```
plot_image(image=true_color_image, mask=cloud_mask)
```



Besides that we also have pseudo-probability scores telling us how likely it is that certain pixel is cloudy:

[ ]:
```
plot_probabilities(true_color_image, cloud_prob)
```





1.3 Use `CloudMaskRequest` to produce cloud masks

`CloudMaskRequest` class combines download procedure with cloud detection. It works in a way that it processes all images from a given time interval and not just a single one. The process is optimized for performance and download costs.

The main input of `CloudMaskRequest`in an instance of cloud detector object. Additionally, we have to specify parameters defining location, time interval, etc. This time we'll download all Sentinel-2 bands only to have all RBG bands for visualization.

[ ]:
```
cloud_detector = S2PixelCloudDetector(
    threshold=0.4,
    average_over=1,
    dilation_size=1,
    all_bands=True
)

cloud_mask_request = CloudMaskRequest(
    cloud_detector,
    bbox=bbox,
    time=('2021-06-01', '2021-06-30'),
    size=bbox_to_dimensions(bbox, 60),
    time_difference=dt.timedelta(hours=2),
    config=config
)
```

So far no data has been downloaded or processed. The class only made a request to Sentinel Hub Catalog API to obtain information when data is available. Those are the following timestamps:

[ ]:
```
cloud_mask_request.get_timestamps()
```

```
[ ]: [datetime.datetime(2021, 6, 1, 16, 43, 50, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 3, 16, 33, 56, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 6, 16, 43, 52, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 8, 16, 33, 56, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 11, 16, 43, 52, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 13, 16, 33, 55, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 16, 16, 43, 51, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 18, 16, 33, 56, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 21, 16, 43, 51, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 23, 16, 33, 56, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 26, 16, 43, 52, tzinfo=tzutc()),
      datetime.datetime(2021, 6, 28, 16, 33, 56, tzinfo=tzutc())]
```

The following will trigger download and cloud masking process:

```
[ ]: %%time

     cloud_masks = cloud_mask_request.get_cloud_masks()

     cloud_masks.shape
```

```
CPU times: user 2min 19s, sys: 1.63 s, total: 2min 21s
Wall time: 1min 3s
```

```
[ ]: (12, 134, 157)
```

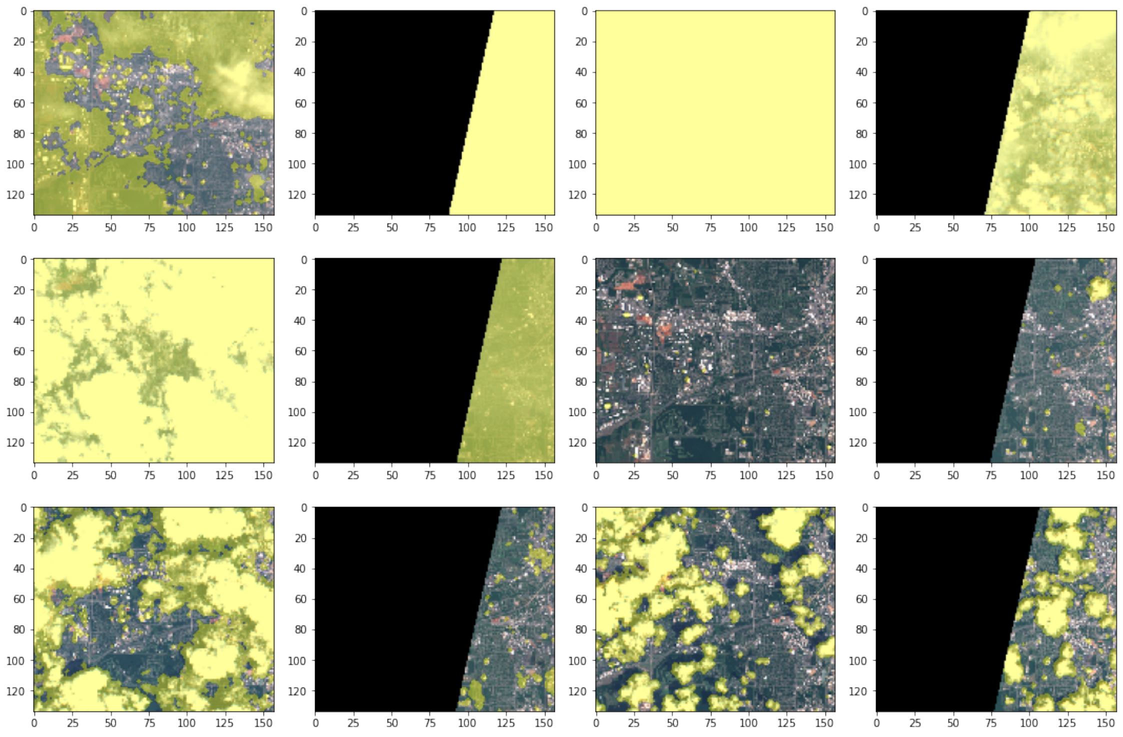
Let's extract RGB bands:

```
[ ]: true_color_images = cloud_mask_request.get_data()[..., [3, 2, 1]]

     true_color_images.shape
```

```
[ ]: (12, 134, 157, 3)
```

Let's plot cloud masks together with images:

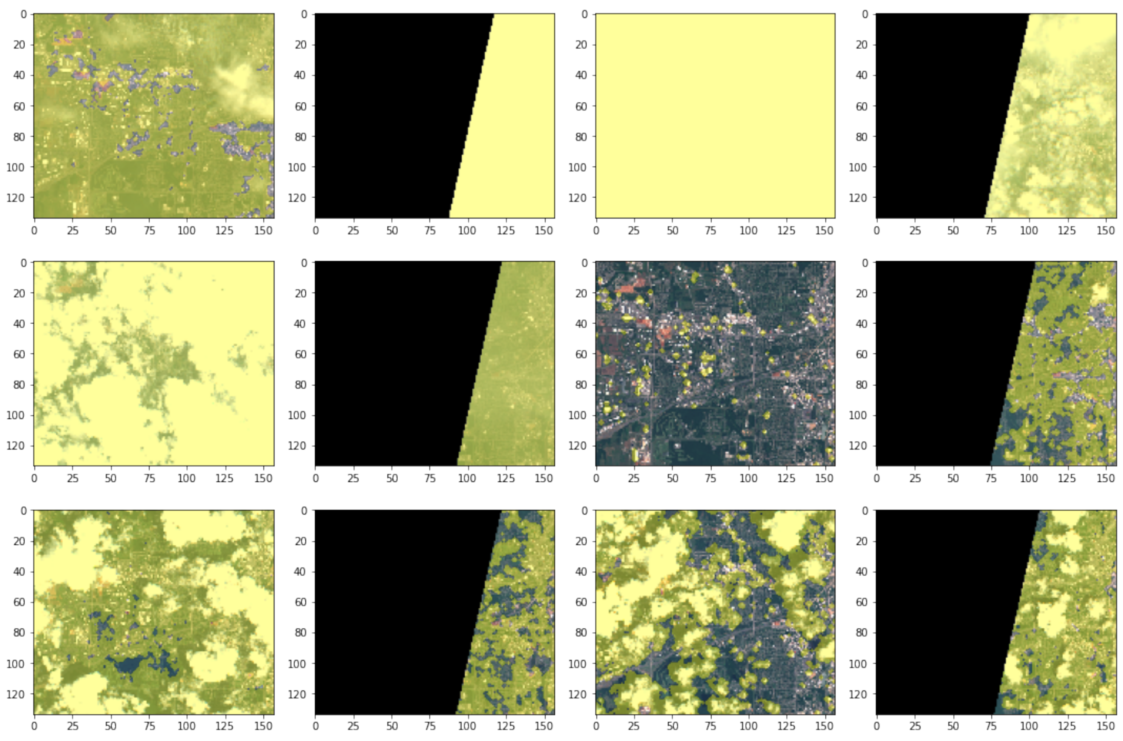
```
[ ]: fig = plt.figure(figsize=(15, 10))
     n_cols = 4
     n_rows = int(np.ceil(len(true_color_images) / n_cols))

     for idx, (image, mask) in enumerate(zip(true_color_images, cloud_masks)):
         ax = fig.add_subplot(n_rows, n_cols, idx + 1)
         plot_image(image, mask, ax=ax, factor=3.5)

     plt.tight_layout()
```



If the default probability threshold of `0.4` doesn't suit us we can override it to compute new binary cloud masks with a different threshold:

```
[ ]: cloud_masks_different_threshold = cloud_mask_request.
      ↪get_cloud_masks(threshold=0.2)
```

```
[ ]: fig = plt.figure(figsize=(15, 10))
     n_cols = 4
     n_rows = int(np.ceil(len(true_color_images) / n_cols))

     for idx, (image, mask) in enumerate(zip(true_color_images,␣
      ↪cloud_masks_different_threshold)):
         ax = fig.add_subplot(n_rows, n_cols, idx + 1)
         plot_image(image, mask, ax=ax, factor=3.5)

     plt.tight_layout()
```


0
20
40
60
80
100
120
0 25 50 75 100 125 150

```
[ ]:
```

```
[ ]:
```