Cvičení v prostředí Google Earth Engine

Ohniskové operace a maskování oblak

Dálkový průzkum Země

OBSAH

1 TEORETICKÁ ČÁST

1.1 Maskování oblak

Maskování oblak u družicových snímku je velmi užitečný proces, který vede ke zvýšení kvality družicových snímků, a hlavně ke zlepšení výsledků následující analýzy. Nejedná se o nic jiného než, že se z našich družicových snímků snažíme odstranit pixely, které reprezentují oblaka. Google Earth Engine umožňuje provádět maskování oblak na snímcích Sentinel-2 i Landsat-9.

1.2 Ohniskové operace

Ohniskové operace neboli filtrace se řadí mezi prostorové zvýraznění obrazu. Filtrace využívají kernel (pohyblivé okno), kdy v jeho centrální buňce dochází k výpočtu nové hodnoty, která se zapíše do výsledného obrazu. Kernel představuje čtvercovou matici, kdy každá buňka této matice má svou váhu. Kernel postupně prochází celým obrazem a dochází k výpočtu nových hodnot. Nová hodnota je vypočtena vynásobením hodnot pixelů a jejich vah. U okrajových pixelů k výpočtu nedochází.

Nízkofrekvenční filtry

Nízkofrekvenční filtry mají za úkol odstraňovat vliv vysokých frekvencí a zdůrazňovat frekvence nízké. Tyto filtry vyhlazují. Nízkofrekvenčních filtrů je celá řada a s některými z nich se důkladněji seznámíme v rámci praktické části cvičení.

Nízkofrekvenční filtry:

- Průměrový filtr
- Filtry s váženým středem
- Mediánová filtrace
- Módová filtrace
- Gaussův filtr
- Sieve filtrace

Vysokofrekvenční filtry

Vysokofrekvenční filtry zdůrazňují vysoké frekvence a využívají se pro zvýraznění hran. Vedou k zostření obrazu. Podrobněji si je opět představíme v rámci praktické sekce.

Vysokofrekvenční filtry:

- Laplaceovské filtry
- Sobelův filtr
- Prewittův filtr
- Ostřící filtry

2 DATA

Pro účely tohoto cvičení využijeme snímky z družic Sentinel-2 a Landsat-9. S těmito daty jsme už měli možnost pracovat v rámci předchozích cvičení v prostředí Google Earth Engine.

2.1 Sentinel-2

Sentinel-2 je mise programu Copernicus vypuštěná v roce 2015.

Sentinel-2 nese multispektrální senzor, který snímá sluneční záření odražené od povrchu Země. Data jsou získávána v 13 spektrálních pásmech ve viditelném, blízkém infračerveném a krátkovlnném infračerveném spektru.

Prostorové rozlišení snímků Sentinel-2 je 10, 20 a 60 m.

Na oběžné dráze se pohybují dvě identické družice Sentinel-2A a Sentinel-2B a díky této soustavě je časové rozlišení dat 5 dní. V mírných šířkách je časové rozpětí v případě dvou družic dokonce 2-3 dny.

Dosud jsme pracovali s daty Sentinel-2 Level 2A, které jsou po atmosférické korekci, nyní k tomu přidáme i Sentinel-2 Level 1C, což jsou snímky po radiometrických korekcích. Využijeme také datovou sadu Sentinel-2 Cloud Probability, která obsahuje informace o pravděpodobnosti výskytu oblak.

2.2 Landsat-9

Landsat-9 je mise NASA, která byla vypuštěna na oběžnou dráhu v září 2021.

Družice Landsat-9 nese dva senzory, a to OLI-2 a TIRS-2. OLI-2 (Operational Land Imager) poskytuje snímky v panchromatická a multispektrální data. TIRS-2 (Thermal InfraRed Sensor 2) získává tepelná data. Celkem družice Landsat-9 poskytuje data v 11 spektrálních pásmech.

Prostorové rozlišení snímků Landsat-9 je 15 m pro panchromatické snímky, 30 m pro multispektrální snímky a 100 m pro termální snímky.

Časové rozlišení Landsat-9 je přibližně 16 dní.

3 PRAKTICKÁ ČÁST

V praktické části tedy projdeme postup maskování oblak jak u snímků Sentinel-2 tak u snímků Landsat-9. Nakonec se zaměříme na ohniskové operace.

Pro cvičení si vytvořte vlastní repositář, do kterého si postupně uložíte skripty pro jednotlivé části cvičení. Veškeré skripty používané v rámci tohoto cvičení jsou dostupné na sdíleném repositáři: https://code.earthengine.google.com/?accept_repo=users/michaelasvehlikova/cviceni3. Doporučuji se k danému repositáři připojit, kopírovat jednotlivé bloky kódu a upravit je na základě svých potřeb.

3.1 Cloud masking Sentinel-2

Prvním úkolem v rámci dnešního cvičení bude vyzkoušet si maskování oblak a jejich stínů na snímku z družice Sentinel-2.

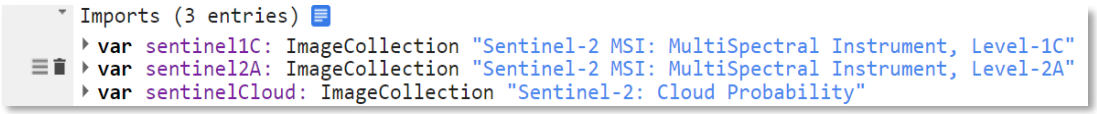
1. Import dat

Začneme nahráním potřebných datových sad do našeho pracovního prostředí. Postupně si importujeme tři produkty Sentinel-2, které využijeme pro maskování oblak.

Potřebné datové sady:

- Sentinel-2 Level-1C: COPERNICUS/S2_HARMONIZED
- Sentinel-2 Level-2A: COPERNICUS/S2_SR_HARMONIZED
- Sentinel-2 Cloud Probability: COPERNICUS/S2_CLOUD_PROBABILITY

Importované datové sady si můžeme přejmenovat dle našeho uvážení. V rámci ukázkového skriptu jsou datové sady pojmenovány **sentinel2A** (Sentinel-2 Level-2A), **sentinel1C** (Sentinel-2 Level-1C) a **sentinelCloud** (Sentinel-2 Cloud Probability)



2. Zakreslení zájmové oblasti

Dalším krokem je vyznačení zájmového území. To provedeme pomocí geometrických nástrojů v mapovém okně. Volba umístění a rozsahu území je libovolná.

3. Příprava dat

Po importu potřebných datových sad je nutné vybrat snímky, které odpovídají požadavkům. Vybereme si snímky, které pokrývají naše zájmové území v požadovaném časovém úseku pomocí metod **filterBounds()** a **filterDate()**. V rámci ukázkového skriptu je zvoleno časové rozpětí od 1. března 2023 do 1. května 2023.

Zároveň u snímku Sentinel-2 Level-1C a Sentinel-2 Level-2A vybereme potřebná pásma s využitím funkce **select()**. U Sentinel-2 Level-1C vybíráme pásma potřebná pro výpočet CDI (Cloud displacement index), těmi jsou B7, B8, B8A a B10. CDI napomáhá při detekci mraků na snímcích Sentinel-2. U Sentinel-2 Level-2A vybíráme pásma B2, B3, B4 a B5.

```
// Filtrace kolekcí dle požadovaných parametrů
var sentinel1C = sentinel1C
                 .filterBounds(oblast)
                 .filterDate("2023-03-01", "2023-05-01")
                 .select(["B7", "B8", "B8A", "B10"]);

var sentinelCloud = sentinelCloud
                 .filterDate("2023-03-01", "2023-05-01")
                 .filterBounds(oblast);

var sentinel2A = sentinel2A.filterDate("2023-03-01", "2023-05-01")
                 .filterBounds(oblast)
                 .select(["B2", "B3", "B4", "B5"]);
```

Po výběru vhodných snímků je nutné kolekce ještě ořezat. Vzhledem k tomu, že pracujeme s kolekcemi je nutné definovat funkci na ořez každého snímků v kolekci na rozsah zájmového území. Tato funkce již byla představena na předchozích cvičeních. Pro níže definovanou funkci je nutné, aby zakreslené zájmové území mělo název **oblast**.

```
// Funkce na ořez kolekce
var orezKolekce = function(snimek){
```

```
3.   return snimek.clip(oblast);
4. };
```

Nově definovanou funkci následně pomocí **map()** aplikujeme na kolekce snímků a provedeme ořez.

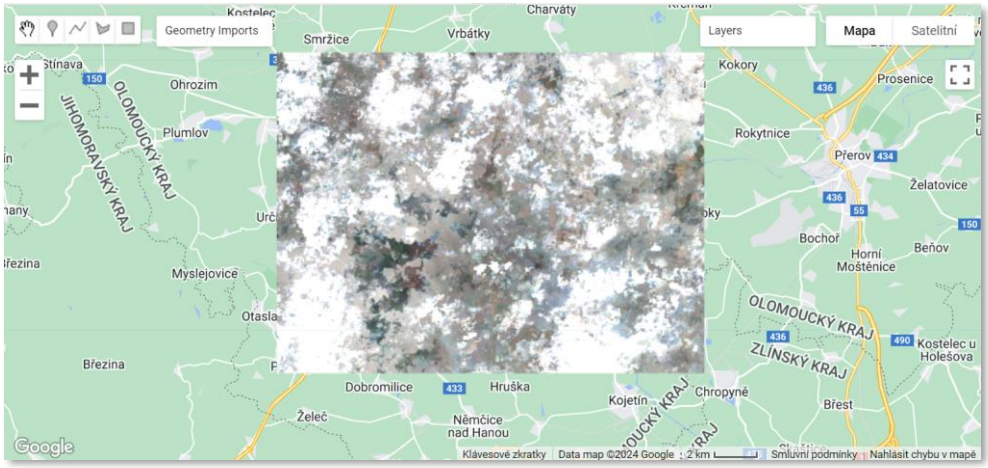
```
1. // Ořezání kolekcí na rozsah zájmového území
2. var sentinel1C = sentinel1C.map(orezKolekce);
3. var sentinel2A = sentinel2A.map(orezKolekce);
4. var sentinelCloud = sentinelCloud.map(orezKolekce);
```

4. Vizualizace snímku

Před samotným maskováním si ještě zobrazíme mediánový snímek v pravých barvách, abychom zjistili množství oblak. Pro vizualizaci využijeme kolekci snímků Sentinel-2 Level-2A. Kolekci agregujeme pomocí **median()**.

```
1. // Vizualizace mediánového snímku Sentinel-2 2A za vybrané období v RGB
2. Map.addLayer(sentinel2A.median(), {bands: ["B4", "B3", "B2"], min: 0, max: 3000}, "Mediánový
snímek před maskováním");
3. Map.centerObject(sentinel2A, 10);
```

Vzniklý mediánový snímek v rámci ukázkového skriptu je velmi oblačný. Maskování oblak v takovém případě má rozhodně smysl.



5. Spojení kolekcí

Před maskováním oblak je nutné si veškeré potřebné kolekce spojit do jedné. Definujeme si funkci, která provádí spojení dvou kolekcí (obvykle s různými pásmy) na základě jejich **system:index** atributu. Funkci definujeme pomocí klíčového slova **function** a spojení je provedeno tam, kde se hodnoty **system:index** atributu shodují. Tuto podmínku definujeme s využitím **ee.Filter.equals()**. Výstupem je nová spojená kolekce, která obsahuje pásma obou kolekcí.

```
// Funkce na spojení dvou kolekcí na základě jejich system:index atributu -> vznik spojené kolekce
function indexSpojeni(kolekce1, kolekce2, atribut) {
  var spojene = ee.ImageCollection(ee.Join.saveFirst(atribut).apply({
    primary: kolekce1,
    secondary: kolekce2,
    condition: ee.Filter.equals({
      leftField: "system:index",
      rightField: "system:index"})
  }));
  // Spojení pásem kolekcí
  return spojene.map(function(snimek) {
    return snimek.addBands(ee.Image(snimek.get(atribut)));
  });
}
```

Jako první pomocí volání funkce **indexSpojeni()** propojíme data Sentinel-2 Level-2A a Sentinel-2 Cloud Probability na základě atributu **cloud_probability**.

Následně k propojené kolekci připojíme ještě data Sentinel-2 Level-1C opět s využitím funkce **indexSpojeni()**. Tentokrát spojení provádíme na základě atributu **l1c**.

```
// Spojení Sentinel-2 cloud probability a Sentinel-2 2A
var spojeni2ACloud = indexSpojeni(sentinel2A, sentinelCloud, "cloud_probability");

// Připojení Sentinel 1C dat pro výpočet CDI
var spojeni2A1C = indexSpojeni(spojeni2ACloud, sentinel1C, "l1c");
```

6. Maskování oblak

Teď už si jen definujeme funkci, která provede maskování oblak na snímku. Následující funkci rozhodně nepřepisujte, ale pouze okopírujte do svého pracovního prostředí. Vstupním parametrem funkce je pouze snímek, který vznikl spojením jednotlivých datových sad.

Jako první je proveden výpočet CDI indexu pomocí algoritmu **ee.Algorithms.Sentinel2.CDI()**, což je funkce implementována přímo v Google Earth Engine pro práci s daty Sentinel-2.

Dále funkce vybere pásma probability (pravděpodobnost mraků) a B10 (cirrusové mraky).

Následuje identifikace oblačných oblastí na základě stanovených podmínek. Výsledná maska vzniká na základě splnění jedné nebo obou následujících podmínek:

- Hodnota probability je větší než 65 % a CDI je menší než -0,5.
- Hodnota pásma B10 je větší než 0.01.

Kromě oblak je nutné identifikovat i jejich stíny pomocí atributu **MEAN_SOLAR_AZIMUTH_ANGLE**.

Výstupem naší funkce je snímek s maskovanými oblaky.

```
// Funkce na maskování oblak ze snímku
function maskovatSnimek(snimek) {
  // Výpočet CDI indexu z pásem Sentinel-2 1C kolekce
  var cdi = ee.Algorithms.Sentinel2.CDI(snimek);
  var s2c = snimek.select("probability");
  var cirrus = snimek.select("B10").multiply(0.0001);

  /*
  Předpoklad, že pixely s hodnotou probability atributu>65 % a CDI<-0,5 -> low-to-mid mraky
  Pokud je hodnota cirrus pásma>0.01 -> cirrusové mraky
  Konečným výsledkem masky je splnění jedné nebo obou podmínek
  */

  var jeMrak = s2c.gt(65).and(cdi.lt(-0.5)).or(cirrus.gt(0.01));

  // Převzorkování na rozlišení 20 m
  jeMrak = jeMrak.focal_min(3).focal_max(16);
  jeMrak = jeMrak.reproject({crs: cdi.projection(), scale: 20});

  //Předpověď stínů z mraků, které jsme nalezly
  var stinAzimut = ee.Number(90)
      .subtract(ee.Number(snimek.get("MEAN_SOLAR_AZIMUTH_ANGLE")));

  jeMrak = jeMrak.directionalDistanceTransform(stinAzimut, 50);
  jeMrak = jeMrak.reproject({crs: cdi.projection(), scale: 100});

  jeMrak = jeMrak.select("distance").mask();
  return snimek.select("B2", "B3", "B4").updateMask(jeMrak.not());
}

```

Funkci na maskování oblak aplikujeme na celou spojenou kolekci snímků pomocí **map()**. Bude tedy provedeno maskování oblak na každém snímku v kolekci.

```
// Použití maskovací funkce na jednotlivé snímky v kolekci
var maskovane = ee.ImageCollection(spojeni2A1C.map(maskovatSnimek));
```

7. Vizualizace maskovaného snímku

Samozřejmě je opět nemožné zobrazit každý snímek v kolekci, a proto maskovanou kolekci snímku agregujeme s využitím **reduce().** Jako parametr funkci **reduce()** zadáváme způsob agregace. Opět využíváme agregaci na základě mediánu, abychom mohli porovnat snímek před a po maskování oblak. Kromě způsobu agregace specifikujeme funkci **reduce()** také velikost dlaždic snímku pro výpočet, která může být v rozsahu 0.1-16. Volíme raději vyšší hodnoty velikosti dlaždic. Příliš malé dlaždice mohou vést k tomu, že se maskovaný snímek nakonec nevykreslí kvůli velké výpočetní náročnosti.

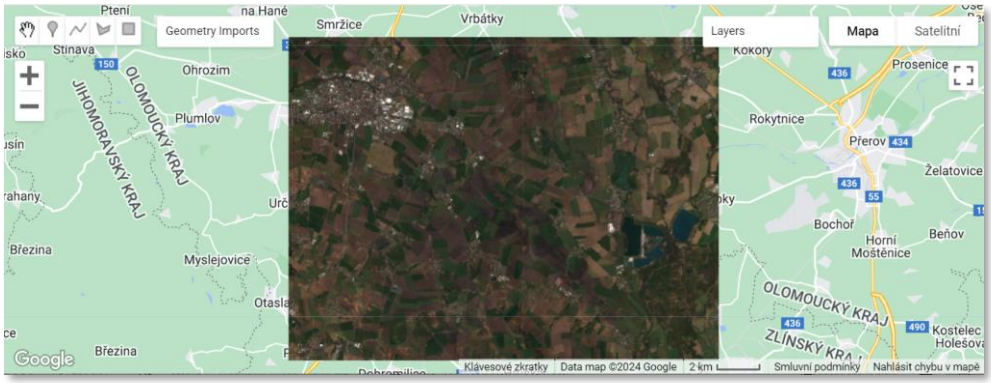
Mediánový snímek po maskování si pomocí **Map.addLayer()** zobrazíme v mapovém okně.

```
// Vytvoření mediánového snímku z jednotlivých maskovaných snímků – pozor na tilescale (0.1-16): příliš malé dlaždice mohou vést k User memory limit chybové hlášce
var median = maskovane.reduce(ee.Reducer.median(), 12);

```

```
4. // Zobrazení našeho výsledného maskovaného snímku
5. Map.addLayer(median, {bands: ["B4_median", "B3_median", "B2_median"], min: 0, max: 3000},
"Mediánový snímek po maskování");
```

I při volbě vyšších hodnot velikosti dlaždic může vykreslení mediánového snímku po maskování zabrat delší dobu. V případě, že se objeví chybová hláška *User Memory limit exceeded* je nutné zvětšit velikost dlaždic. V rámci ukázkového skriptu proběhlo maskování snímků úspěšně a rozdíl mezi mediánovým snímkem před a po maskování je očividný.

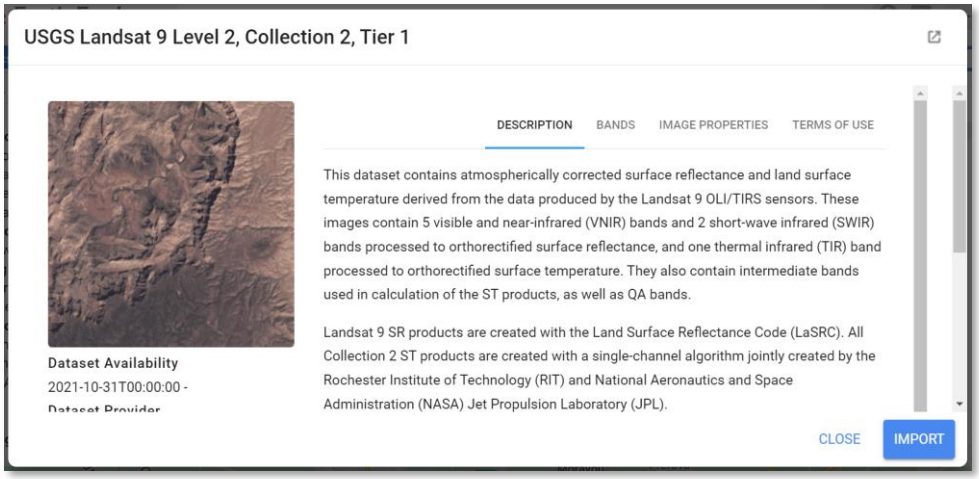


3.2 Cloud masking Landsat-9

Snímky Landsat-9 často vykazují vysokou úroveň oblačnosti. Proto je na místě ukázat si způsob maskování oblak také u Landsat-9. Postup maskování snímků u Landsat-9 je značně jednodušší, než je tomu u snímků ze Sentinel-2.

1. Import dat

Pro potřeby této sekce je nutné si importovat datovou sadu Landsat-9 Level 2, Collection 2, Tier 1 s kódem LANDSAT/LC09/C02/T1_L2.



2. Zakreslení zájmové oblasti

Následně zakreslíme naši zájmovou oblast. Její umístění a rozsah je opět libovolný. Lze pracovat se stejným územím jako v první části dnešního cvičení.

3. Příprava dat

Nahranou kolekci Landsat-9 vyfiltrujeme na základě našich požadavků. Pro účely ukázkového skriptu bylo vybráno časové období od 1. února 2023 do 1. května 2023. Nezapomeneme kolekci vyfiltrovat i na základě rozsahu zájmového území.

```
// Filtrace kolekce snímků Landsat 9
var snimkyLandsat = landsat
                    .filterBounds(oblast)
                    .filterDate("2023-02-01", "2023-05-01");
```

Opět definujeme funkci na ořez kolekce snímků.

```
// Funkce na ořez kolekce
var orezKolekce = function(snimek){
  return snimek.clip(oblast);
};
```

Vyfiltrovanou kolekci snímků ořežeme.

```
// Ořezání kolekce na rozsah zájmové oblasti
var snimkyLandsat = snimkyLandsat.map(orezKolekce);
```

4. Vizualizace snímku

Před maskováním si naši kolekci zobrazíme v mapovém okně pomocí **Map.addLayer()**. Využijeme možnost zobrazení mediánového snímku pomocí metody **median().**

Pozor na označení pásem potřebných pro RGB snímek. Označení pásem Landsat-9 není stejné jako u snímků Sentinel-2 v první části cvičení. Pásma potřebná pro RGB snímek

u kolekce snímků Landsat-9 jsou SR_B4, SR_B3 a SR_B2. Liší se také rozsah minimálních a maximálních hodnot využitých pro vizualizaci.

```
1. // Zobrazení mediánového snímku v RGB
2. Map.addLayer(snimkyLandsat.median(), {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 7000, max: 18000}, "Mediánový snímek RGB – před maskováním");
3. Map.centerObject(snimkyLandsat, 10);
```

Zobrazený mediánový snímek se vyznačuje značnou oblačností. V případě, že se vám povede najít kolekci snímků bez oblačnosti, tak si pohrajte s časovým rozpětím.

5. Maskování oblak

Definujeme si funkci na maskování oblak ze snímku Landsat-9. Vstupním parametrem naší funkce bude snímek, který chceme maskovat.

První budeme definovat masku **qaMaska**, která využívá pásmo QA_PIXEL, které obsahuje informace o kvalitě pixelů v obraze. Bitové hodnoty reprezentují různé jevy jako například mraky nebo stíny mraků. Pomocí bitového operátoru **bitwiseAnd**() tyto nežádoucí jevy odstraníme.

Druhá maska, kterou definujeme, je **saturaceMaska**, která se týká saturace pixelů a využívá pásmo QA_RADSAT. Vybírá pixely s hodnotou 0, tedy ty, které nejsou saturované (nedochází k přesycení detektoru) a odstraňuje ty, které jsou saturované.

Jednotlivé masky následně aplikujeme na snímek a výsledkem následující funkce bude maskovaný snímek, ze kterého byly odstraněny nežádoucí pixely.

```
1. // Funkce na maskování oblak ze snímku Landsat-9
2. function maskaL9(snimek) {
3.   // Bit 0 - Fill
4.   // Bit 1 - Dilated Cloud
5.   // Bit 2 - Cirrus
6.   // Bit 3 - Cloud
7.   // Bit 4 - Cloud Shadow
8.   // Vytvoření masek na odstranění nežádoucích prvků
```

```
 9.    var qaMaska = snimek.select("QA_PIXEL").bitwiseAnd(parseInt("11111", 2)).eq(0);
10.    var saturaceMaska = snimek.select("QA_RADSAT").eq(0);
11.
12.    // Přidání upravených pásem
13.    return snimek
14.        .updateMask(qaMaska)
15.        .updateMask(saturaceMaska);
16. }
```

Funkci na maskování oblak pak mapujeme na kolekci snímků.

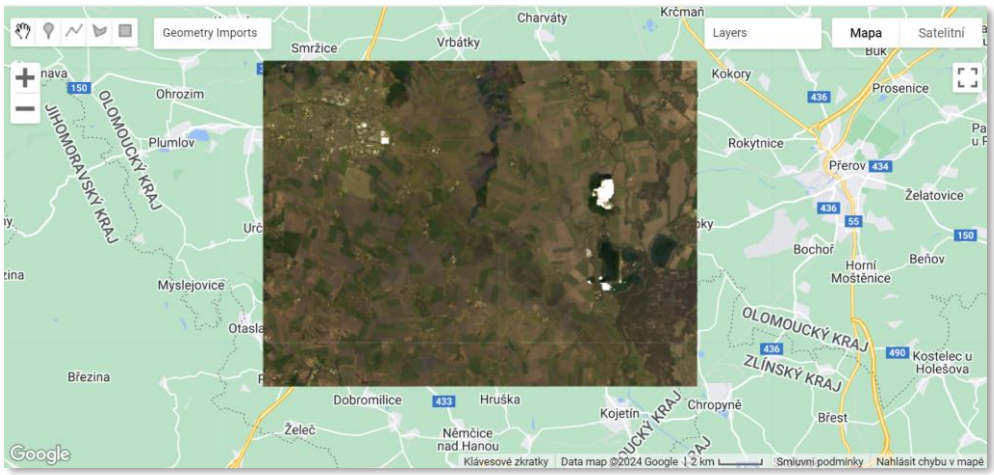
```
1. // Aplikace funkce na maskování oblak na naši kolekci
2. var maskovaneSnimky = snimkyLandsat.map(maskaL9);
```

6. Vizualizace maskovaného snímku

Na závěr maskování si zobrazíme mediánový snímek po maskování oblak a porovnáme s mediánovým snímkem před maskováním.

```
1. // Vizualizace mediánového snímku po maskování oblak
2. Map.addLayer(maskovaneSnimky.median(), {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 7000, max:
18000}, "Mediánový snímek L8 RGB – po maskování" );
```

V rámci ukázkového skriptu proběhlo maskování úspěšně. Z mediánového snímku po maskování je viditelné, že maskování oblak není v některých místech zcela dokonalé. Výsledek maskování je však mnohem vhodnější a využitelnější pro další analýzu než snímek před maskováním.



3.3 Ohniskové operace

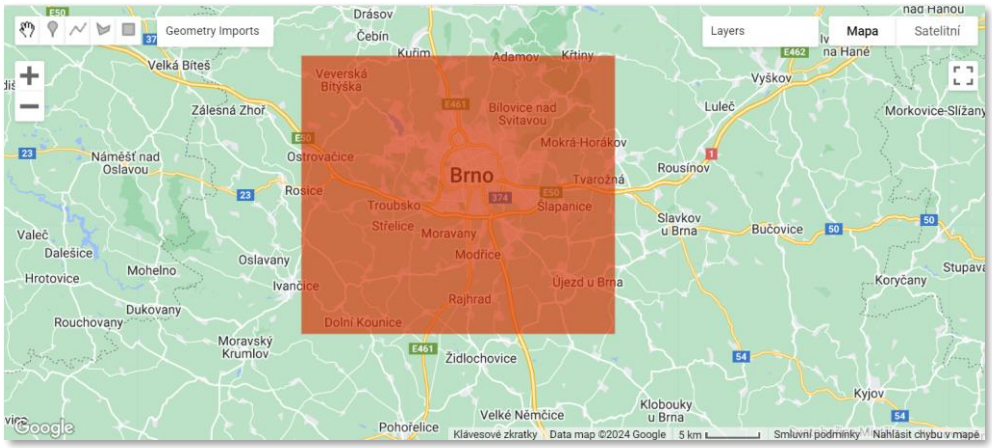
V druhé části cvičení si představíme ohniskové operace v prostředí Google Earth Engine. Projdeme si jak nízkofrekvenční filtry, tak i filtry vysokofrekvenční.

1. Import dat

Importujeme si do svého pracovního prostředí datovou sadu LANDSAT/LC09/C02/T1_L2, se kterou jsme pracovali při maskování oblak Landsat-9.

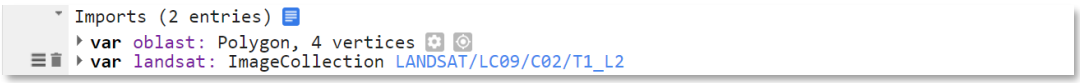
2. Zakreslení zájmové oblasti

Dále zakreslíme libovolné zájmové území. Klidně můžete ponechat stejné zájmové území jako v předchozí části cvičení.



3. Příprava dat

Naimportovanou datovou sadu Landsat-9 a zakreslenou zájmovou oblast si můžete přejmenovat dle svého uvážení. V rámci ukázkového skriptu jsou snímky Landsat-9 označeny jako **landsat** a zájmová oblast jako **oblast**.



Je nutné vyfiltrovat celou kolekci snímků Landsat-9 na rozsah zájmové oblasti ve vybraném časovém období pomocí metod **filterBounds()** a **filterDate()**. Zároveň z naší vyfiltrované kolekce vybereme snímek s nejmenší oblačností, a proto kolekci vzestupně seřadíme díky funkci **sort()** podle hodnot atributu CLOUD_COVER, který nese informaci o oblačnosti. Následně s využitím metody **first()** vybereme první snímek ze seřazené kolekce. Náš vybraný snímek ořežeme na rozsah zájmové oblasti s metodou **clip()**.

```
1. // Filtrace snímku zájmové oblasti ve vybraném období s nejmenší oblačností
 2. var snimekLandsat = landsat
 3.     .filterBounds(oblast)
 4.     .filterDate("2023-05-01","2023-9-30")
 5.     .sort("CLOUD_COVER")
```

```
6.      .first()
7.      .clip(oblast);
```

4. Vizualizace snímku

Snímek si ještě zobrazíme v pravých barvách v mapovém okně pomocí **Map.addLayer()**. Pro vizualizaci v pravých barvách využijeme pásma SR_B4, SR_B3 a SR_B2. Rozsah hodnot pro vizualizaci nastavíme na 7000-15000.

Snímek přiblížíme díky **Map.centerObject()**.

```
1. // Vizualizace snímku v RGB
2. Map.addLayer(snimekLandsat, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 7000, max: 15000},
"Snímek RGB");
3.
4. // Přiblížení zájmové oblasti
5. Map.centerObject(oblast, 10);
```

V případě, že by se snímek zdál přesvětlený či moc tmavý, tak si vyzkoušejte upravit rozsah vizualizačních hodnot přes ozubené kolečko u našeho snímku v záložce *Layers* v pravém horním rohu mapového okna.

3.3.1 Nízkofrekvenční filtry

Na zkušebních datech si jako první vyzkoušíme nízkofrekvenční filtry, které mají za cíl odstranit vliv vysokých frekvencí. Postupně si představíme vyhlazovací kernel, Gaussův filtr a mediánový filtr.

1. Vyhlazovací kernel

Vyhlazovací kernel definujeme pomocí **ee.Kernel.square()**. Kernelu zadáváme poloměr a jednotky poloměru. Jestli si jako jednotky stanovíme pixely, tak poloměr představuje počet pixelů od středu. Pokud máme v nastavení **normalize:true** jako v ukázce,

tak se váhy v kernelu rovnají jedné a každá buňka kernelu má stejnou váhu. Nová hodnota centrálního pixelu je vypočítaná jako průměr hodnot okolních pixelů.

```
// Definice vyhlazovacího kernelu
var kernel = ee.Kernel.square({
    radius: 1,
    units: "pixels",
    normalize: true
});
print(kernel);
```

Vyhlazovací kernel aplikujeme na náš ořezaný snímek pomocí **convolve()**. Vyhlazovací kernel je použit na každé pásmo snímku.

Výsledný vyhlazený snímek si zobrazíme v pravých barvách.

```
// Aplikace filtru na snímek a jeho vizualizace v RGB
var vyhlazenySnimek = snimekLandsat.convolve(kernel);
Map.addLayer(vyhlazenySnimek, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 7000, max: 15000},
"Vyhlazený snímek RGB");

```

2. Gaussův filtr

Pro vytvoření Gaussova filtru existuje funkce **ee.Kernel.gaussian()**, které opět specifikujeme velikost poloměru kernelu a jeho jednotky.

Váhy v našem kernelu si můžeme do konzole vytisknout pomocí **print()**. V případě tohoto filtru mají nejvyšší váhu buňky nejblíže středu kernelu.

```
// Definice Gausssova filtru na vyhlazení
var gaussuvFiltr = ee.Kernel.gaussian({
    radius: 1,
    units: "pixels",
});
print(gaussuvFiltr);
```

Gaussův filtr aplikujeme na náš snímek s využitím **convolve()** a poté náš vyhlazený snímek zobrazíme v pravých barvách v mapovém okně.

```
1. // Aplikace Gaussova filtru na snímek a jeho vizualizace v RGB
2. var gaussSnimek = snimekLandsat.convolve(gaussuvFiltr);
3.  Map.addLayer(gaussSnimek, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 7000, max: 15000},
"Snímek RGB – Gaussův filtr");
```

Gaussův filtr zachovává linie lépe než výše zmíněný čtvercový vyhlazovací kernel.

3. Mediánový filtr

Pokud pracujeme s průměrnými hodnotami jsou výsledné hodnoty pixelů ovlivněny extrémními hodnotami, proto může být vhodnější volbou využití mediánové filtru. Z představených nízkofrekvenčních filtrů by měl mediánový filtr nejlépe zachovávat hrany.

Mediánový filtr aplikujeme na snímek metodou **reduceNeighborhood()**. Funkci upřesníme způsob výpočtu nových hodnot pixelů. Vzhledem k tomu, že našim cílem je aplikovat na snímek mediánový filtr, tak nové hodnoty budou vypočteny pomocí **ee.Reducer.median().** Metoda **reduceNeighborhood()** vyžaduje také specifikaci velikosti kernelu a k tomu využijeme kernel definovaný pro účely vyhlazovacího kernelu.

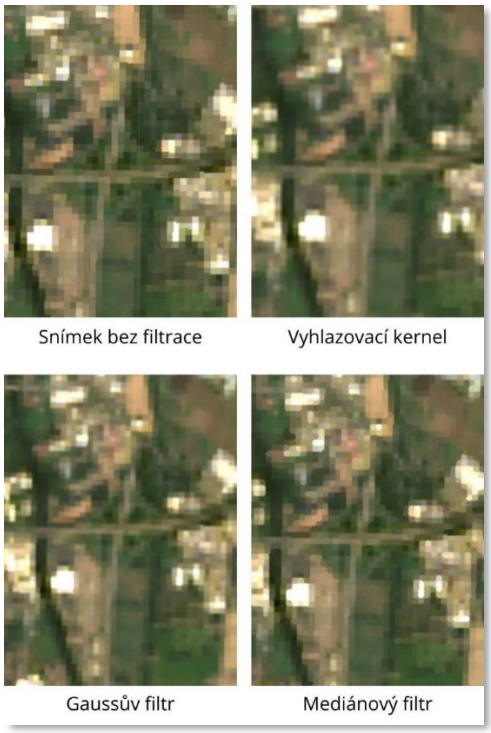
```
1. // Aplikace mediánového filtru na snímek
2. var medianSnimek = snimekLandsat.reduceNeighborhood({
3.     reducer: ee.Reducer.median(),
4.     kernel: kernel
5. });
```

Snímek po mediánové filtraci si zobrazíme v mapovém okně.

```
1. // Vizualizace snímku s aplikovaným mediánovým filtrem
2.  Map.addLayer(medianSnimek, {bands: ["SR_B4_median", "SR_B3_median", "SR_B2_median"], min:
7000, max: 15000}, "Snímek RGB – Mediánový filtr");
```

4. Porovnání výsledků

Po provedení všech kroků můžeme porovnat výsledky jednotlivých filtrací.



Nebojte se vyzkoušet si změnit velikost poloměru kernelů a tím změnit rozsah sousedství. Pokud zvolíme příliš velký poloměr, může se stát, že odstraníme značnou část detailů a dojde k příliš velkému vyhlazení. Také si můžete zkusit, jak ovlivní volba jednotek stupeň vyhlazení. Nebojte se experimentovat pro dosažení požadovaných výsledků.

5. Post-klasifikační úpravy

V předchozím cvičení, které se týkalo klasifikací bylo zmíněno, že textura pepř a sůl na klasifikovaných snímcích může být eliminována díky vyhlazovacím filtrům. Pokud máte k dispozici skript s řízenou a neřízenou klasifikací, tak si vyzkoušejte aplikovat představené filtry na své klasifikované snímky. Opět neplatí obecné pravidlo pro to, jaký filtr použít, tudíž je nejlepší cestou experimentovat.

3.3.2 Vysokofrekvenční filtry

Vysokofrekvenční filtry na rozdíl od nízkofrekvenčních filtrů zachovávají vysoké frekvence a vedou ke zvýraznění hran. Společně si vyzkoušíme Laplaceův, Prewittův, a nakonec Sobelův filtr.

1. Laplaceův filtr

U Laplaceova filtru dává záporný součet vah jednotlivých buněk kernelu hodnotu váhy středového pixelu. Součet všech vah v kernelu je tedy roven nule. Tento filtr zdůrazňuje odstupy středové buňky od okolí.

V Google Earth Engine je implementována funkce **ee.Kernel.laplacian8()** na definici Laplaceova filtru. Jedná se o kernel s velikostí 3x3 pixely.

V Google Earth Engine existuje i funkce **ee.Kernel.laplacian4()**, která místo 8-sousedství využívá 4-sousedství. To znamená, že váhy jsou rozděleny pouze do buněk, které sdílí hranu se středovou buňkou.

Hodnoty vah v kernelu si lze zobrazit v konzoli pomocí **print()**.

```
1. // Definice Laplaceova filtru
2. var laplaceFiltr = ee.Kernel.laplacian8();
3.
4. // Vytištění filtru/kernelu
5. print('Laplaceův filtr:', laplaceFiltr);
```

Po vytištění kernelu do konzole můžeme ověřit, že váha centrální buňky odpovídá zápornému součtu okolních buněk.

```
Laplaceův filtr:
▼ Kernel.laplacian8
    type: Kernel.laplacian8
  ▶ center: [1,1]
    weights:
      [1.0, 1.0, 1.0]
      [1.0, -8.0, 1.0]
      [1.0, 1.0, 1.0]
```

Po definici Laplaceova filtru už jej můžeme použít na snímek pomocí **convolve()** a následně si filtrovaný snímek zobrazit.

```
1. // Aplikace Laplaceova filtru na snímek a vizualizace snímku
2. var laplaceSnimek = snimekLandsat.convolve(laplaceFiltr);
3. Map.addLayer(laplaceSnimek, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 3000, max: 15000},
"Snímek RGB - Laplaceův filtr");
```

2. Sobelův filtr

U Sobelova filtru jsou váhy lineárně uspořádané a jejich součet odpovídá nule. Tento filtr se využívá pro detekci horizontálních a vertikálních hran.

Pro tento filtr je v Google Earth Engine implementována funkce s názvem **ee.Kernel.sobel()**.

Do konzole si necháme vypsat váhy v kernelu a ověříme, že je výše uvedené tvrzení pravdivé.

```
1. // Definice Sobelova vertikálního filtru
2. var sobelVertikalni = ee.Kernel.sobel();
3.
4. // Vytištění filtru/kernelu
```

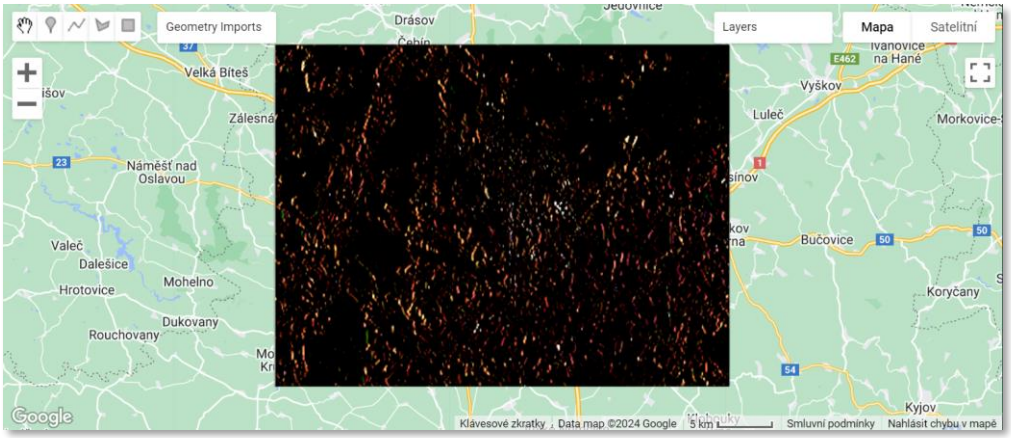
```
5. print("Sobelův filtr vertikální:", sobelVertikalni);
```

```
Sobelův filtr vertikální:
▾Kernel.sobel
    type: Kernel.sobel
  ▸ center: [1,1]
    weights:
      [-1.0, 0.0, 1.0]
      [-2.0, 0.0, 2.0]
      [-1.0, 0.0, 1.0]
```

Sobelův filtr aplikujeme na snímek s pomocí metody **convolve()** a snímek vizualizujeme v mapovém okně.

```
1. // Aplikace Sobelova filtru na snímek a vizualizace snímku
2. var sobelSnimekV = snimekLandsat.convolve(sobelFiltr);
3.  Map.addLayer(sobelSnimekV, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 3000, max: 15000},
"Snímek RGB - Sobelův filtr - vertikální");
```

Ze snímku je viditelné, že došlo ke zvýraznění vertikálních hran.



Pro detekci hran v horizontálním směru je nutné náš kernel rotovat, aby došlo ke změně vah v buňkách. Pro rotaci využijeme funkci **rotate()** a jako parametr dáváme číslo 1, což znamená, že dojde k rotaci o 90 stupňů.

Pro ověření úspěšnosti rotace si váhy kernelu vypíšeme do konzole.

```
1. // Definice Sobelova horizontálního filtru
2. var sobelHorizontalni = sobelVertikalni.rotate(1);
3.
4. // Vytištění filtru/kernelu
5. print("Sobelův filtr horizontální:", sobelHorizontalni);
```

```
Sobelův filtr horizontální:
▼ Kernel.rotate
    type: Kernel.rotate
  ▶ center: [1,1]
  ▶ kernel: Kernel.sobel
    rotations: 1
    weights:
      [-1.0, -2.0, -1.0]
      [0.0, 0.0, 0.0]
      [1.0, 2.0, 1.0]
```

Sobelův filtr na detekci horizontálních hran použijeme na snímek a poté snímek zobrazíme v mapovém okně.

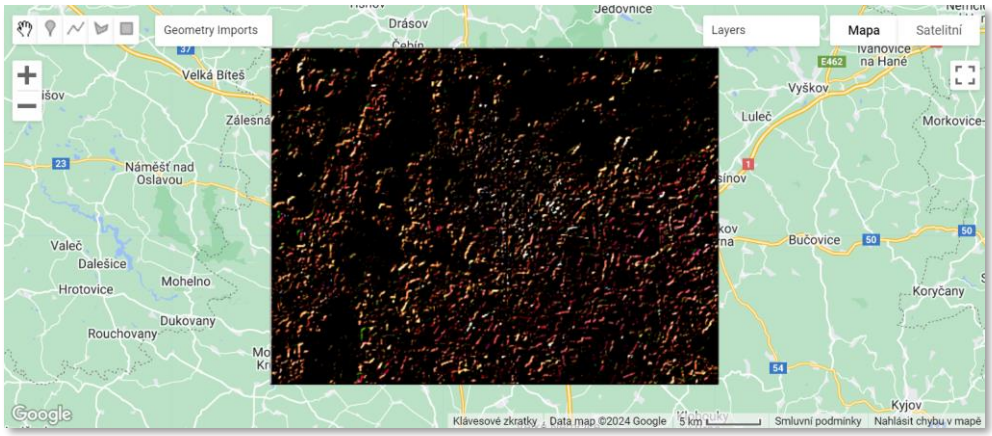
```
1. // Aplikace Sobelova horizontálního filtru na snímek a vizualizace snímku
2. var sobelSnimekH = snimekLandsat.convolve(sobelHorizontalni);
3. Map.addLayer(sobelSnimekH, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 3000, max: 15000},
"Snímek RGB - Sobelův filtr - horizontální");
```

Ze snímku je patrné, že se provedla detekce horizontálních hran.

Výsledné snímky s detekcí horizontálních a vertikálních hran spojíme do jednoho. To jednoduše provedeme metodou **add()**.

Souhrnný snímek vizualizujeme do mapového okna pomocí **Map.addLayer()**.

```
1. // Spojení snímků s vertikálním a horizontálním zvýrazněním hran
2. var kombinaceSobel = sobelSnimekV.add(sobelSnimekH);
3.
4. // Vykreslení snímku s detekovanými hranami
5. Map.addLayer(kombinaceSobel, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 3000, max: 15000},
"Snímek RGB - Sobelův filtr - kombinace");
```



3. Prewittův filtr

Poslední filtr, který si vyzkoušíme je Prewittův filtr. U Prewittova filtru jsou váhy v krajních buňkách kernelu vyrovnané.

Prewittův filtr lze v Google Earth Engine definovat pomocí funkce **ee.Kernel.prewitt()**. Vzniká opět kernel s velikostí 3x3 pixely.

Po definici si do konzole necháme vypsat váhy v jednotlivých buňkách kernelu.

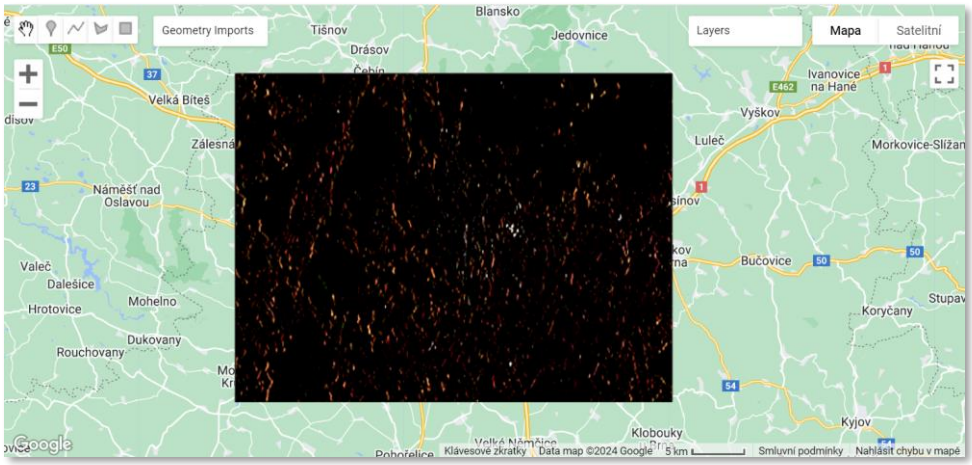
```
1. // Definice Prewittova filtru
2. var prewittFiltr = ee.Kernel.prewitt();
3.
4. // Vytištění filtru/kernelu
5. print("Prewittův filtr:", prewittFiltr);
```

```
Prewittův filtr:
▼ Kernel.prewitt
    type: Kernel.prewitt
  ▶ center: [1,1]
    weights:
      [1.0, 0.0, -1.0]
      [1.0, 0.0, -1.0]
      [1.0, 0.0, -1.0]
```

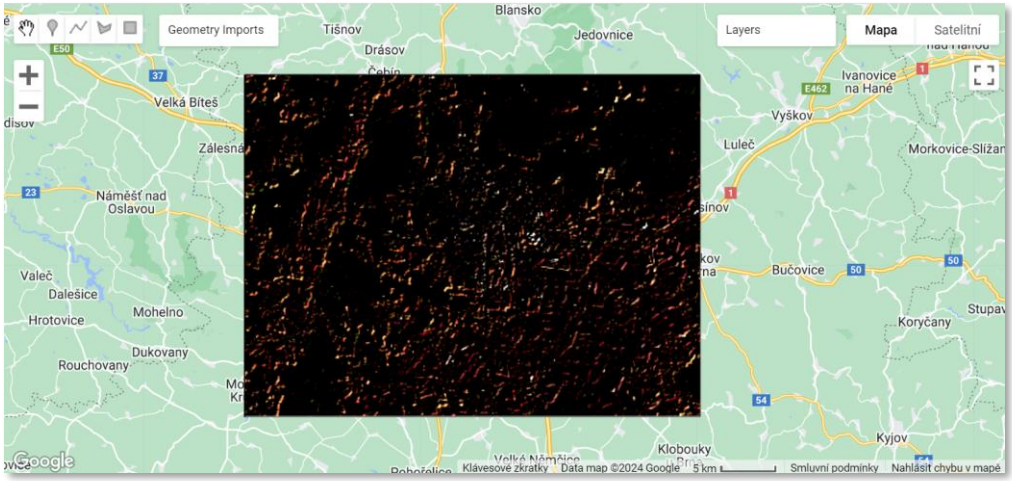
Následně už nám nic nebrání v tom použít filtr na snímek a zobrazit si jej.

```
1. // Aplikace Prewittova filtru na snímek a jeho vizualizace
2. var prewittSnimek = snimekLandsat.convolve(prewittFiltr);
3. Map.addLayer(prewittSnimek, {bands: ["SR_B4", "SR_B3", "SR_B2"], min: 3000, max: 15000},
"Snímek RGB - Prewittův filtr");
```

Ze snímku vyplývá, že byly opět detekovány hrany pouze v jednom směru, a to ve vertikálním.



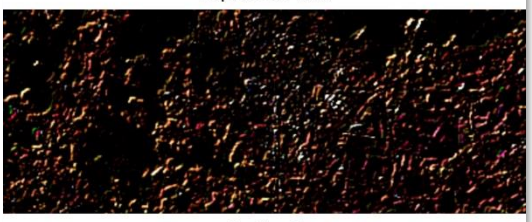
V případě, že chceme detekovat hrany v horizontálním směru a vytvořit snímek, který zobrazuje všechny hrany, tak postupujeme stejně jako v případě Sobelova filtru.



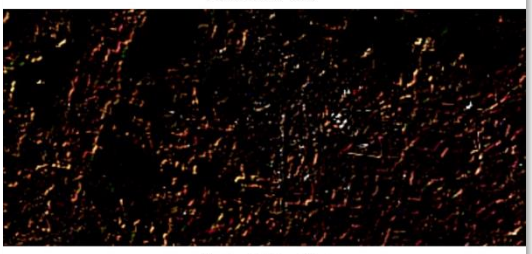
4. Porovnání výsledků

Z představených vysokofrekvenčních filtrů se z hlediska jednoduchosti provedení jeví nejlépe Laplaceův filtr. Opět je vhodné experimentovat a dojít k nejvhodnějšímu řešení pro náš konkrétní případ.

Laplaceův filtr



Sobelův filtr



Prewittův filtr