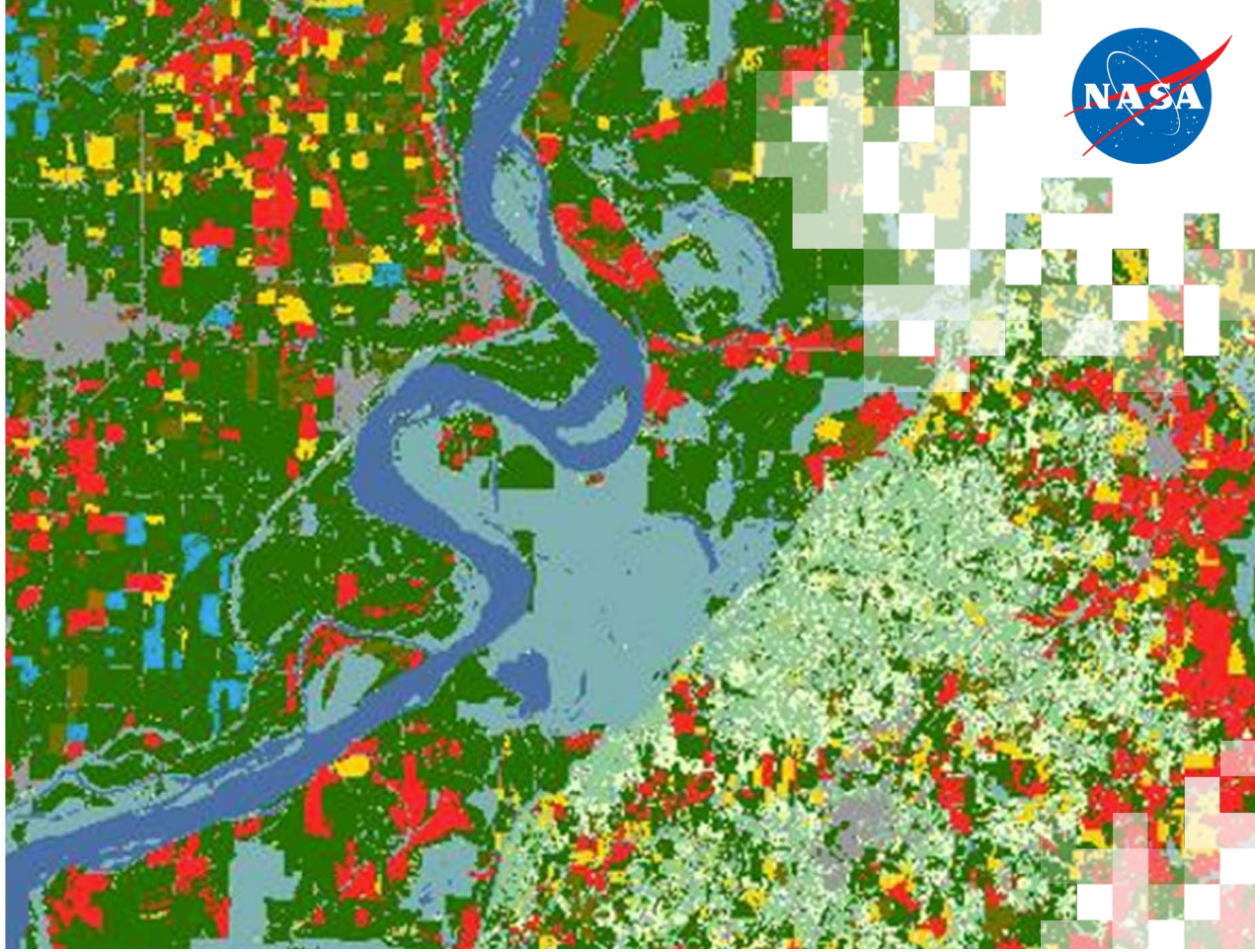
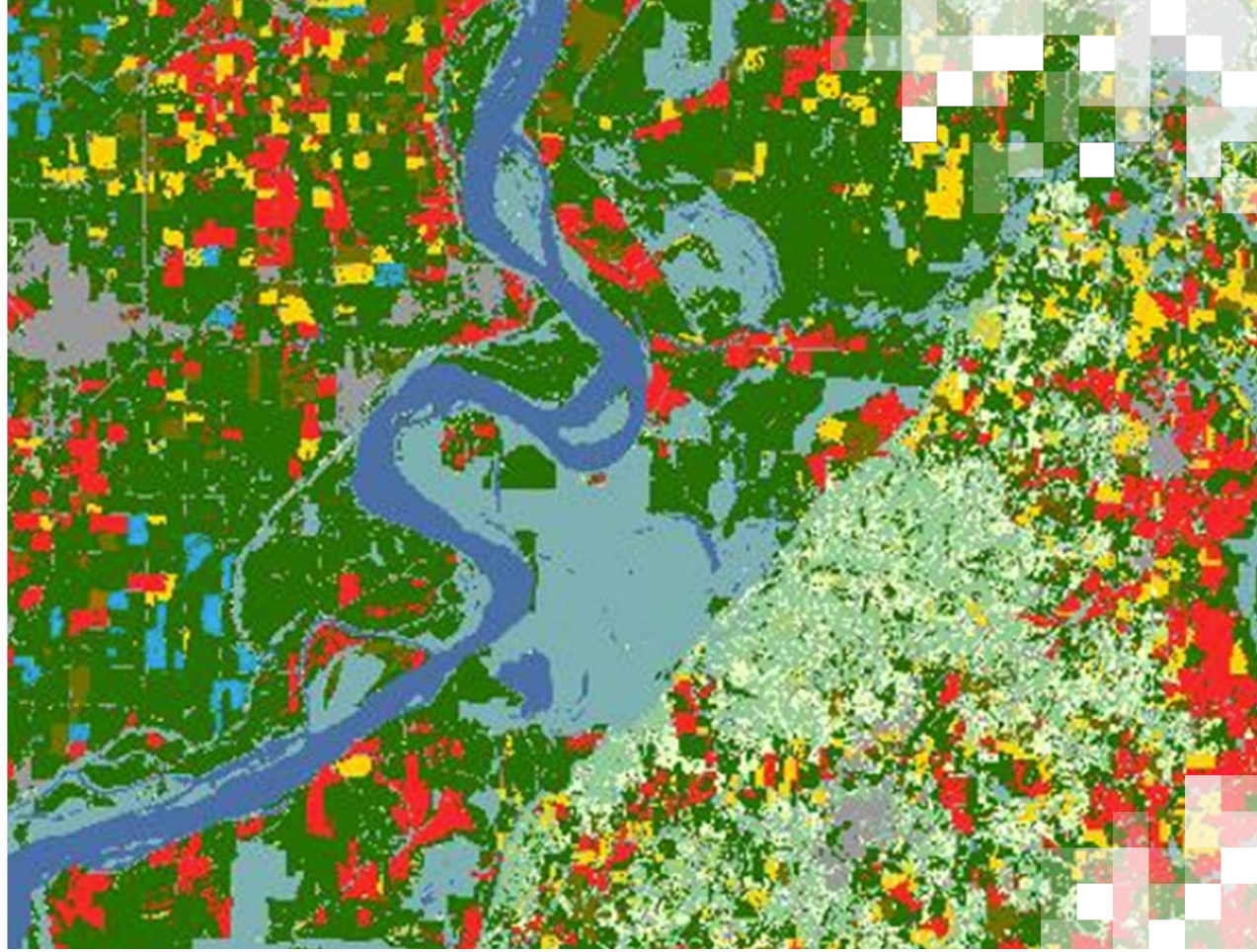


Large Scale Applications of Machine Learning using Remote Sensing for Building Agriculture Solutions

Part 2: Data Loaders for Training ML Models on Irregularly-Spaced Time-Series of Imagery

John Just (Deere & Co., Iowa State University), Erik Sorensen (Deere & Co.)

March 12, 2024

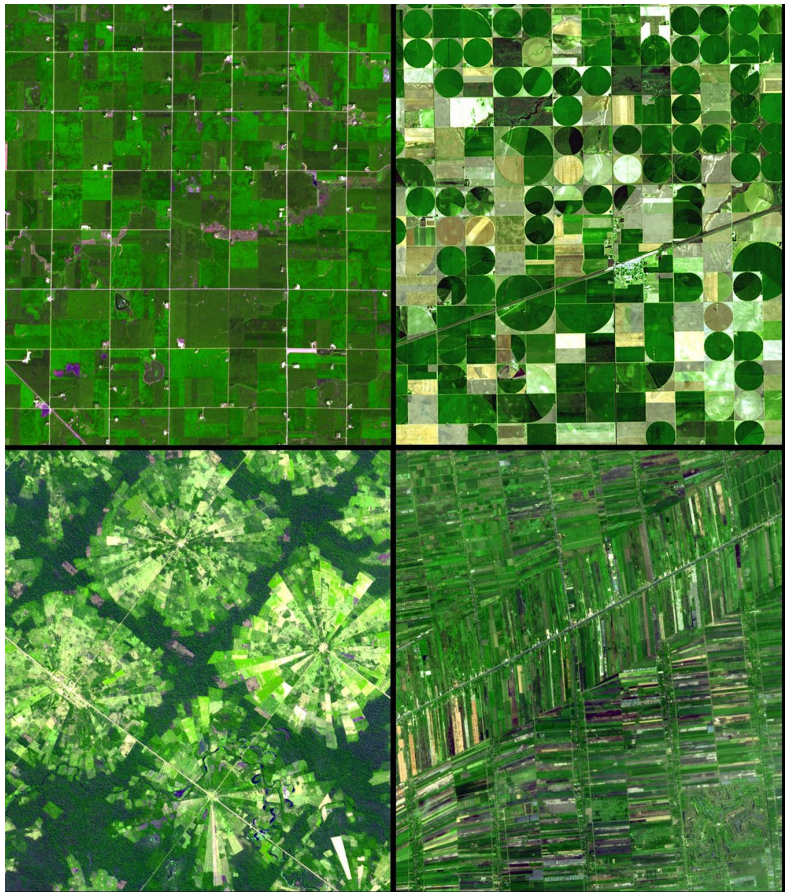


Large Scale Applications of Machine Learning using Remote Sensing for Building Agriculture Solutions

Overview

Motivation for Training

- Timely and accurate in-season crop maps at local to regional scales are crucial for agricultural decision-making and management.
- Irregularly-spaced time-series are common with optical satellite images.
- Training robust models on remote sensing data often requires very large data, but processing and training is complex.
- The Cropland Data Layer (CDL, USDA–NASS) only gives estimates of the types of crops released to the public a few months after the end of the growing season, and not their sequence or timing (e.g., for double crops).



Montage of images shows differences in field geometry and size in different parts of the world. Image credit: NASA (Instrument: Terra – ASTER)

Training Learning Objectives

By the end of this training series, participants will be able to:

- Use recommended techniques to download and process remote sensing data from Sentinel-2 and the Cropland Data Layer (CDL) at large scale (> 5GB) with cloud tools (Amazon Web Services [AWS] Simple Storage Service [S3], Databricks, Spark/Pyspark, Parquet).
- Produce interactive plots of maps, tables, time series, etc. for investigation & verification of data and models.
- Filter data from both the measured (satellite images) and target (CDL) domains to serve modeling objectives based on quality factors, land classification, area of interest (AOI) overlap, and geographical location.
- Build training pipelines in TensorFlow to train machine learning algorithms on large scale remote sensing/geospatial datasets for agricultural monitoring.
- Utilize random sampling techniques to build robustness into a predictive algorithm while avoiding information leakage across training/validation/testing splits.

Prerequisites

- Basic/general understanding of Python programming in Databricks from Part-1.
- Access to the associated data from Part-1.
- Sign up for and access Databricks Community Edition

Training Outline

Part 1

Data Preparation of Imagery for Large-Scale ML Modeling

March 05, 2024

Part 2

Data Loaders for Training ML Models on Irregularly-Spaced Time-Series of Imagery

March 12, 2024

Part 3

Training & Testing ML Models for Irregularly-Spaced Time Series of Imagery

March 19, 2024

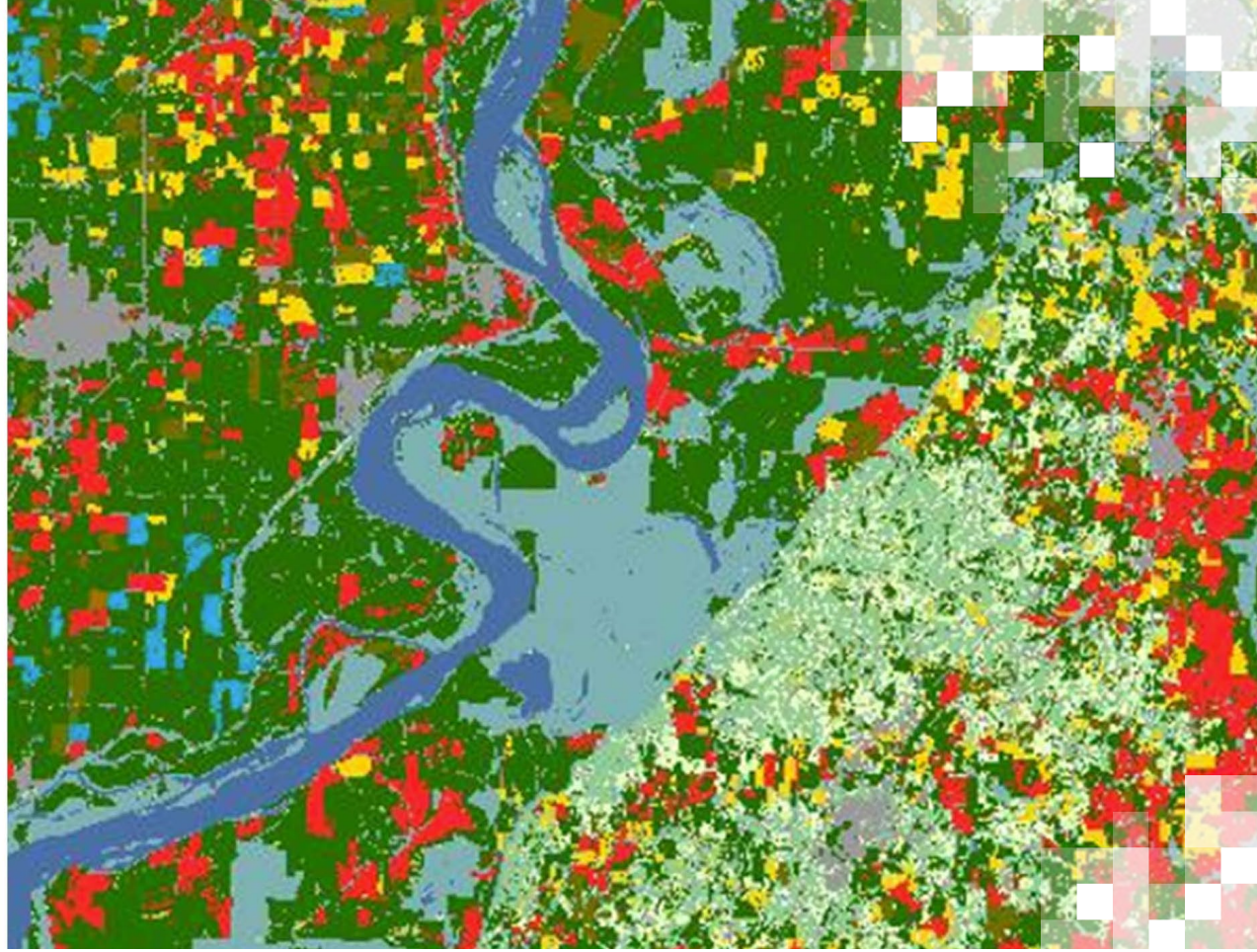
Homework

Opens March 19 – **Due April 1** – Posted on Training Webpage

A certificate of completion will be awarded to those who attend all live sessions and complete the homework assignment(s) before the given due date.

How to Ask Questions

- Please put your questions in the Questions box and we will address them at the end of the webinar.
- Feel free to enter your questions as we go. We will try to get to all the questions during the Q&A session after the webinar.
- The remainder of the questions will be answered in the Q&A document, which will be posted to the training website about a week after the training.



Large Scale Applications of Machine Learning using Remote Sensing for Building Agriculture Solutions

Part 2: Data Loaders for Training ML Models on Irregularly-Spaced Time-Series of Imagery

Part 2 – Trainers

John Just

Principal Data Scientist

John Deere

Erik Sorensen

Senior Data Scientist

John Deere

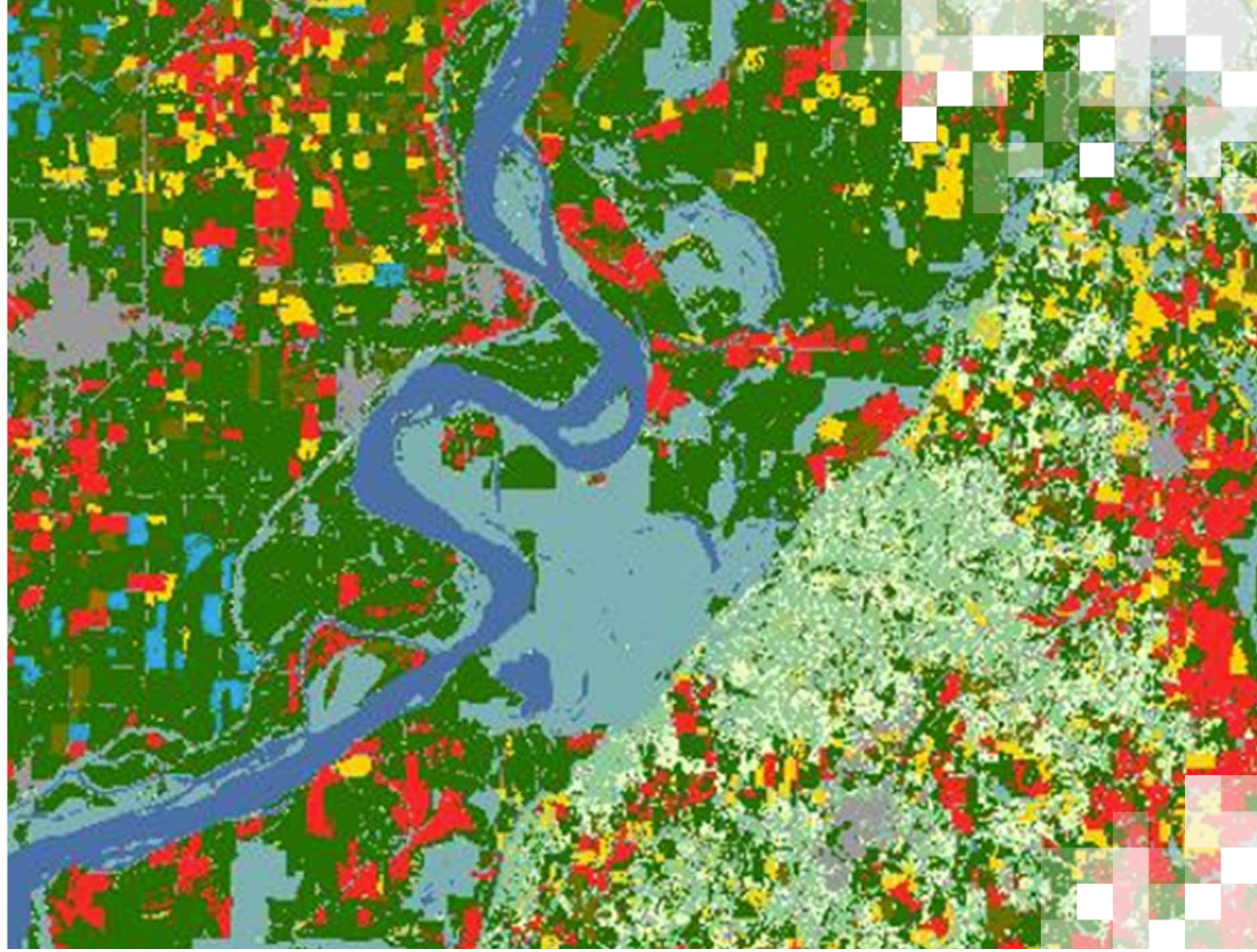
Part 2 Objectives

By the end of Part 2, participants will be able use Python in Databricks Community Edition to:

- Split the dataset into train/val/test to avoid data leakage across time/space.
- Build a specialized/optimized queue (TensorFlow data loader):
  - Extract/read Parquet files into time-series of pixels (with random selection).
  - Filter any data we might want to ignore (e.g., double cropping).
  - Apply augmentations (e.g., random selections of dates for each pixel).
  - Bucketize irregularly spaced time-series per pixel into regular time intervals.
  - Normalize values to aid convergence during optimization.
- Optimize data pipeline for speed using parallelization and view benefits.

Review of Prior Knowledge

- Satellite imagery has high time-series irregularity due to things like cloud cover.
- The Cropland Data Layer (CDL) is a model that provides labels of which crops are growing in the contiguous US and is released at the end of the year.
- Storing data as byte-strings in Parquet files allows efficient storage of large time-series data.



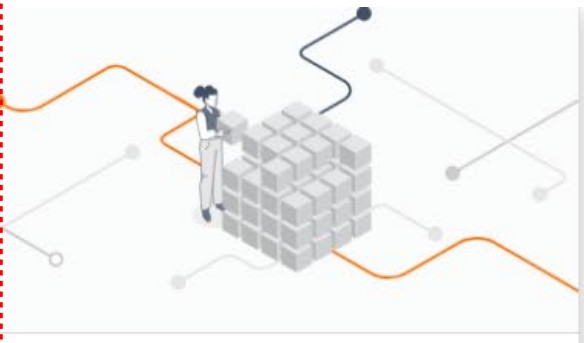
Part 2 Section 1:

About TensorFlow & Data Flow

TensorFlow

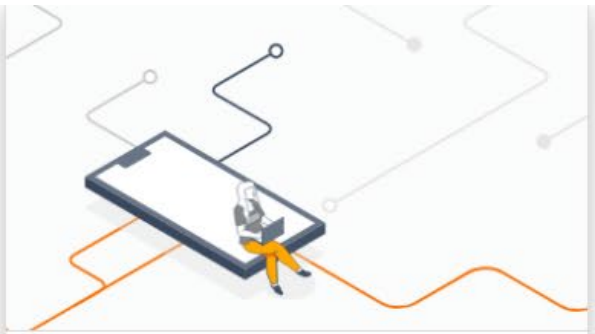
Open-Source library from Google for creating scalable end-to-end machine learning pipelines: Tensorflow.org

Part-2 (this part) | Part-3 (next time) | Other capabilities (not used in demo)



Prepare data

Use TensorFlow tools to process and load data.



Build ML models

Use pre-trained models or create custom ones.

Deploy models

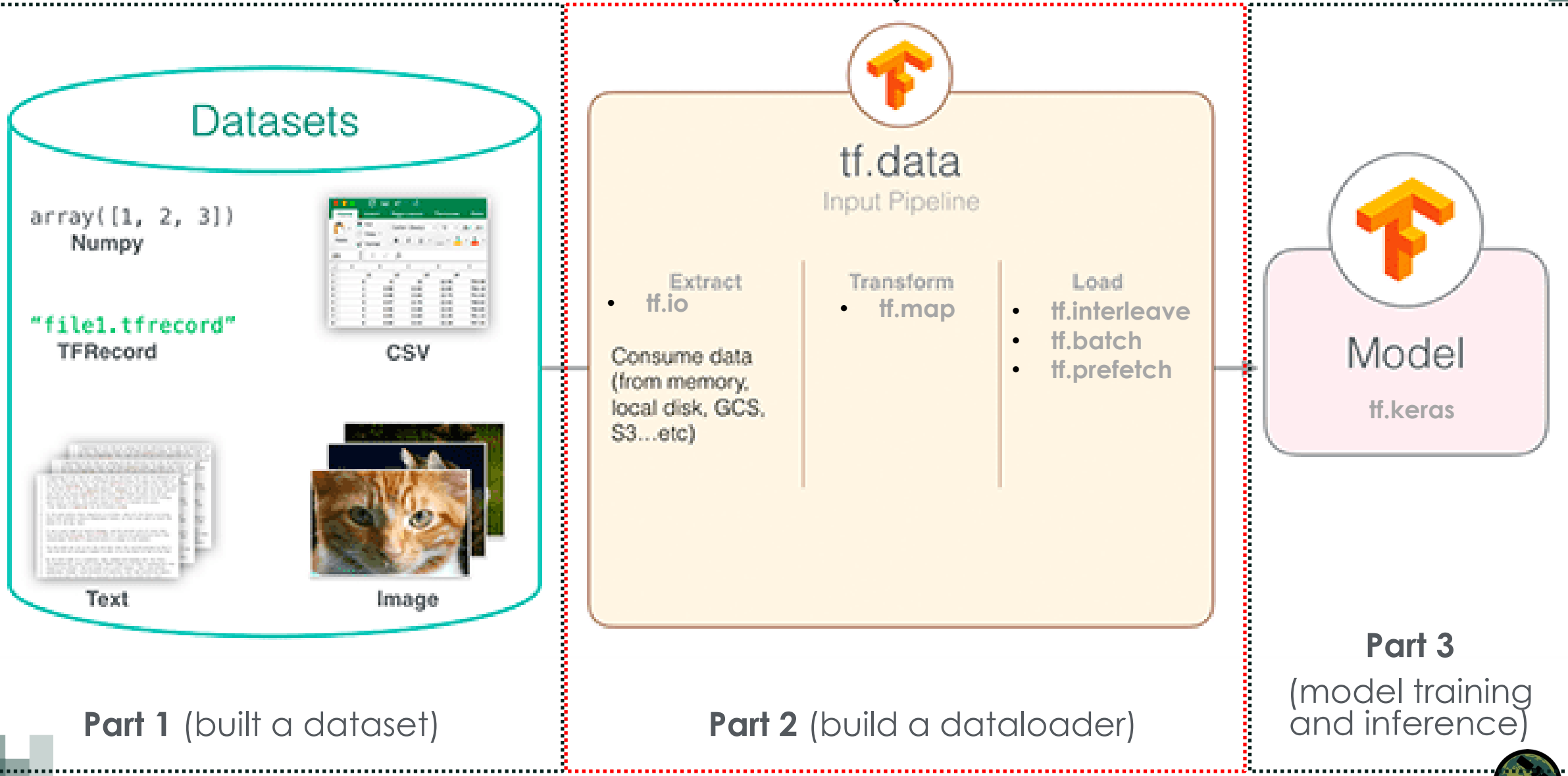
Run on-prem, on-device, in the browser, or in the cloud.

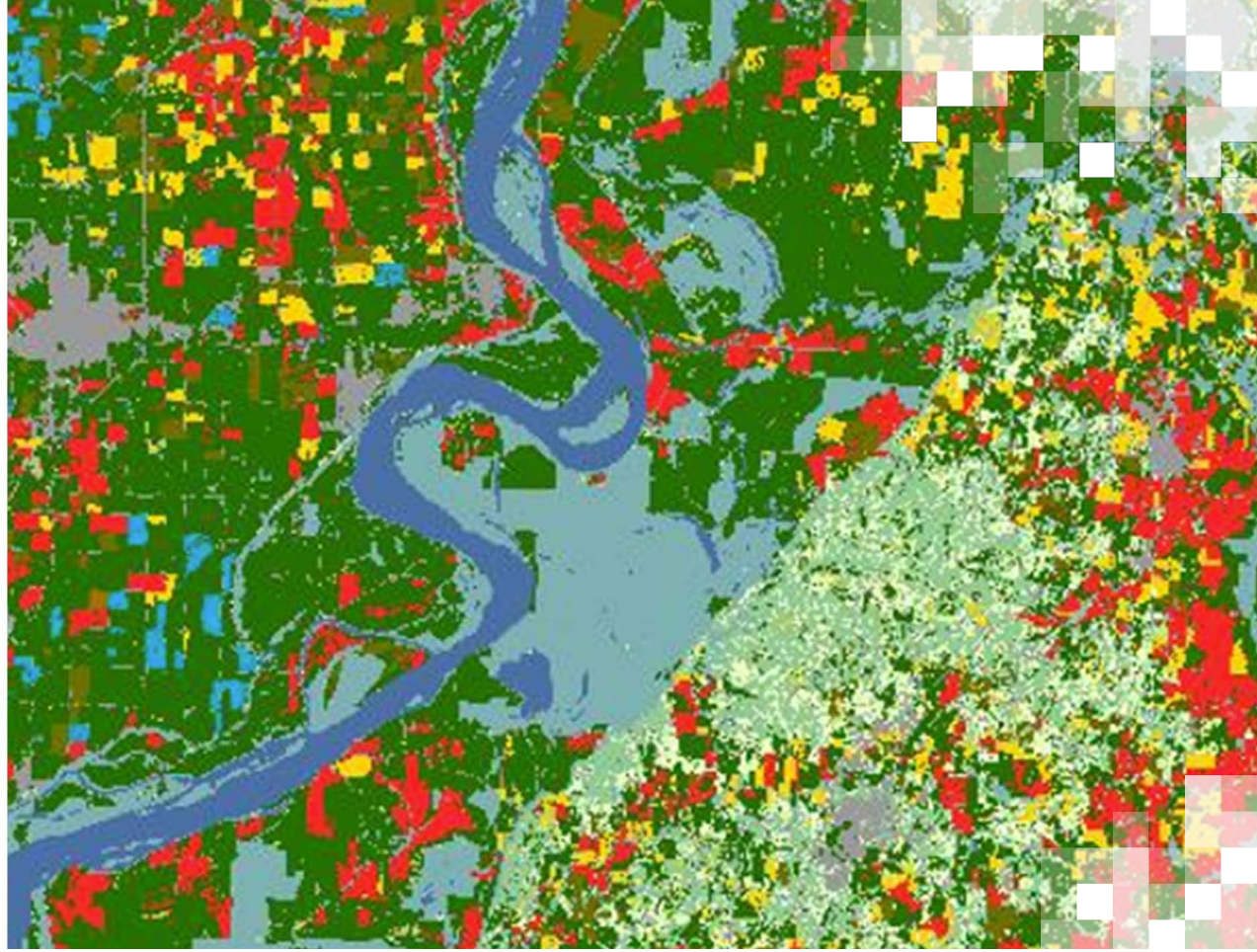
Implement MLOps

Run models in production and keep them performing.

Data Flow

We are here





Part 2 Section 2:

TensorFlow Data Pipelines

The Data Pipeline Overview

Several nice explanations for Tensorflow (TF) data pipelines exist, e.g., Stanford's CS230 (from Andrew Ng), and directly from the TF docs. According to the CS230 suggestions, one good order for operations is as follows:

1. Get a list of Parquet file path locations
2. Read in each file's contents as TensorFlow dataset. Could be a generator (lazy file loading) or in memory (rows in a table). For this demo, we load in the data in-memory due to performance on DB Community
3. .filter()
   - Filters out any data we do not want during training (e.g., double cropping labels)
4. .shuffle()
   - Randomly shuffles the order of the data to diversify the samples in each batch and ultimately find more robust gradient updates
5. .map(parse function)
   - Applied to each pixel/year to sample, bin image values, and prepare along with labels for modeling
6. .batch training data and label outputs
   - Loads "N" (batch size) number of examples from map each call for data
7. .prefetch
   - Does all the above in background processes (parallel queue "producer") from the main process ("consumer", where model training occurs) to minimize wait time for new training batches

Example TensorFlow Dataloader Code

```
train_ds = train_files_ds \
        .filter(filter_double_croppings) \
        .shuffle(BATCH_SIZE*10) \
        .map(lambda x: parse(x, norm=True), num_parallel_calls = tf.data.AUTOTUNE) \
        .batch(BATCH_SIZE) \
        .prefetch(1) \
```

NOTE: num_parallel_calls controls how many CPUs are used for parallelization of the mapping function

.map(function)

Map is where the bulk of the work is performed in the dataloader. It can parallelize the application of a [customized] function to the input data. In our case, we apply a **parsing** function that:

- Reads in each row from the Parquet files,
- Converts data features from raw bytes into tf.tensors using tf.io.decode_raw()
  - Reading data as bytes guarantees the data is loaded properly,
- Bucketizes data (randomly selects 1 image if more than 1 available per interval) and forms into a (num buckets)*(num bands) array,
- Performs normalization (important for the optimization process),
- One-hot-encodes the labels (loss function expects the labels to be in this format), and
- Returns tuple of data features and labels.

Reading Byte Strings with Tensorflow.IO

- Recall from Part 1
  - Time-series of images were grouped by each CDL pixel/year
  - These time-series were converted into byte strings for more efficient storage
- Most datatypes (e.g., images) involve decoding. TF has functions to decode the byte strings using tf.io.decode_raw()
- decode_raw() just needs the expected **datatype** of the data (e.g., float32, string) and it will convert the byte string back into the original multidimensional array
  - Array shape: [num_images, num_bands]

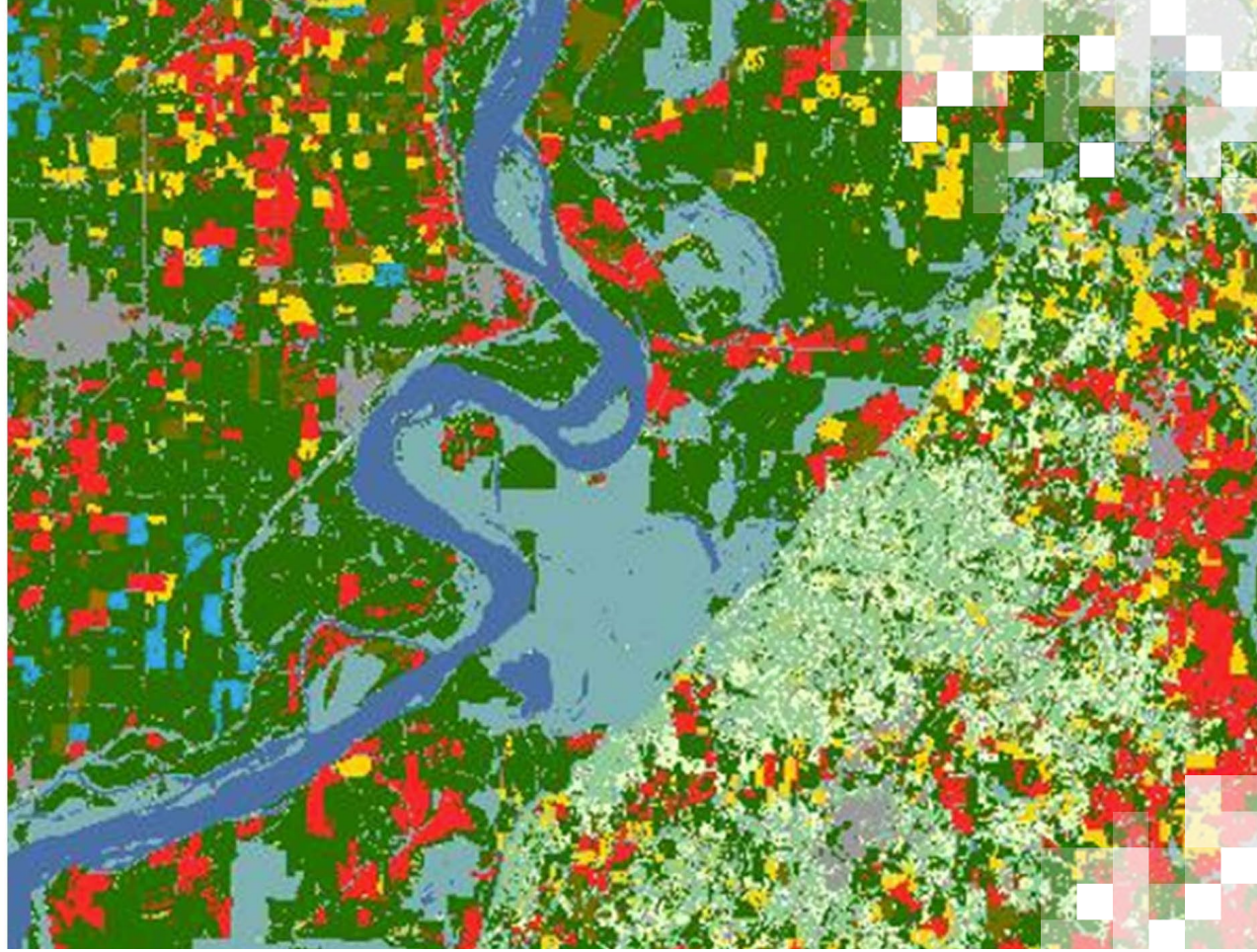
Byte String Input

lon	lat	# scenes	bands	tiles	img dates	scl_vals	bbox	year	CDL
-89.2973	36.85473	38	AUECmAOEBM	MTZTQkZfMCw	Re5GAkYgRjRG	BQUKBQUFCgoF	549309,	2019	Corn
-89.3087	36.95602	71	AEYBMQHnAq4	MTZTQkZfMCw	Re5F7kYCRgJGI	BQUFBQUFBQUF	549309,	2019	Corn
-89.3206	36.88018	36	ATACXAOZBCw	MTZTQkZfMCw	Re5GAkYgRjRG	BQUKBQUFCgUF	549309,	2019	Corn
-89.3322	36.86472	33	ARcB4QKxA4M	MTZTQkZfMCw	Re5GIEY0RjIGP	BQoFBQUKBQUF	549309,	2019	Corn
-89.3382	36.79661	37	AQQBugJiA1ID	MTZTQkZfMCw	Re5GAkYgRjRG	AgIKBQUFCgoFB	549309,	2019	Corn
-89.3394	36.84096	37	ACgBcwHOArk	MTZTQkZfMCw	Re5GIEY0RjIGP	BQoFBQUKCgUF	549309,	2019	Corn
-89.3493	36.9019	37	AO4B5gJSAvgD	MTZTQkZfMCw	Re5GAkYgRjRG	BQUKBQUFCgUF	549309,	2019	Corn
-89.3552	36.95054	72	Ak8DYAQ4Bbg	MTZTQkZfMCw	Re5F7kYCRgJGI	BQUFBQUFBQUF	549309,	2019	Corn
-89.3577	36.74937	37	ANIAmQFTAfo	MTZTQkZfMCw	Re5GAkYgRjRG	BQUKBQUFCgoF	549309,	2019	Corn
-89.3632	36.97514	70	ACsBSwIVAIUC	MTZTQkZfMCw	Re5F7kYCRgJGI	BQUFBQUFBQUF	549309,	2019	Corn

tf.io.decode_raw(byte_string, tf.int32)

Decoded Multi-Dimensional Data

decoded band vals	decoded tiles	decoded img dates	decoded scl_vals
321,664,900,1228,1415	16SBF_0,16SBF_0,16SE	2019-01-06,2019-01-26,201	5,5,10,5,5,5,10,10,
70,305,487,686,843,98	16SBF_0,16SBG_0,16S	2019-01-06,2019-01-06,201	5,5,5,5,5,5,5,5,5,5,
304,604,921,1068,147	16SBF_0,16SBF_0,16SE	2019-01-06,2019-01-26,201	5,5,10,5,5,5,10,5,5
279,481,689,899,1075,	16SBF_0,16SBF_0,16SE	2019-01-06,2019-02-25,201	5,10,5,5,5,10,5,5,5
260,442,610,850,999,1	16SBF_0,16SBF_0,16SE	2019-01-06,2019-01-26,201	2,2,10,5,5,5,10,10,
40,371,462,697,744,81	16SBF_0,16SBF_0,16SE	2019-01-06,2019-02-25,201	5,10,5,5,5,10,10,5,
238,486,594,760,872,9	16SBF_0,16SBF_0,16SE	2019-01-06,2019-01-26,201	5,5,10,5,5,5,10,5,5
591,864,1080,1464,174	16SBF_0,16SBG_0,16S	2019-01-06,2019-01-06,201	5,5,5,5,5,5,5,5,5,5,
210,153,339,506,738,9	16SBF_0,16SBF_0,16SE	2019-01-06,2019-01-26,201	5,5,10,5,5,5,10,10,
43,331,533,597,755,11	16SBF_0,16SBG_0,16S	2019-01-06,2019-01-06,201	5,5,5,5,5,5,5,5,5,



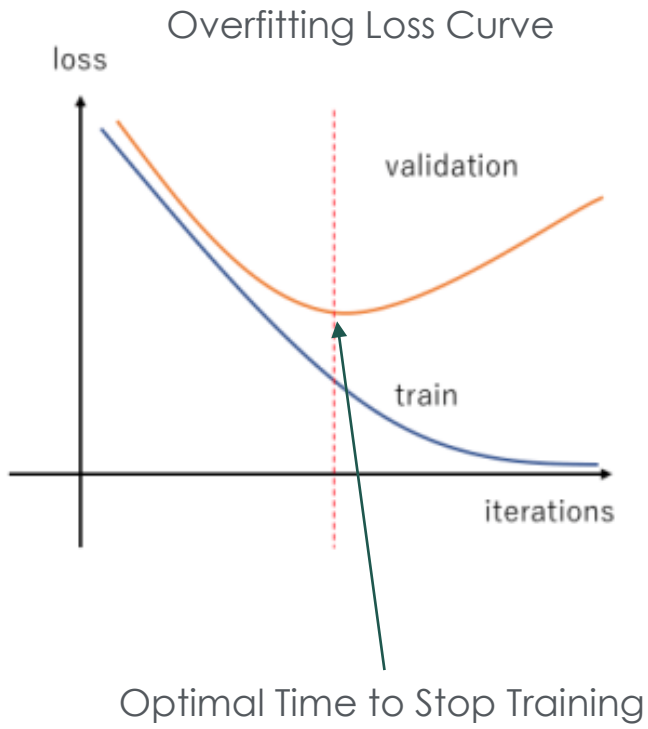
Part 2 Section 3:

Preparing Dataset for Model Training

Train/Val/Test With Time-Series Data

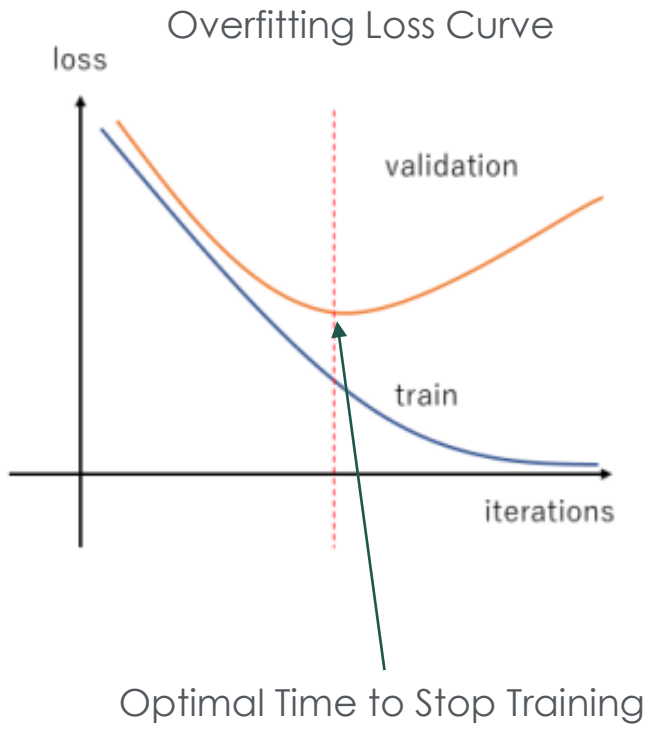
Before training a model, we need to set aside some data to use for validation & testing (not used for model training).

- The Validation dataset is mainly used to
  - avoid overfitting
  - tune hyperparameters
- The Test dataset is primarily used to gauge how model may perform in production/real-world performance (test)
- A common split of the dataset is as follows:
    - Train: 80%
    - Validation: 10%
    - Test: 10%



Train/Val/Test For Crop-Type Prediction

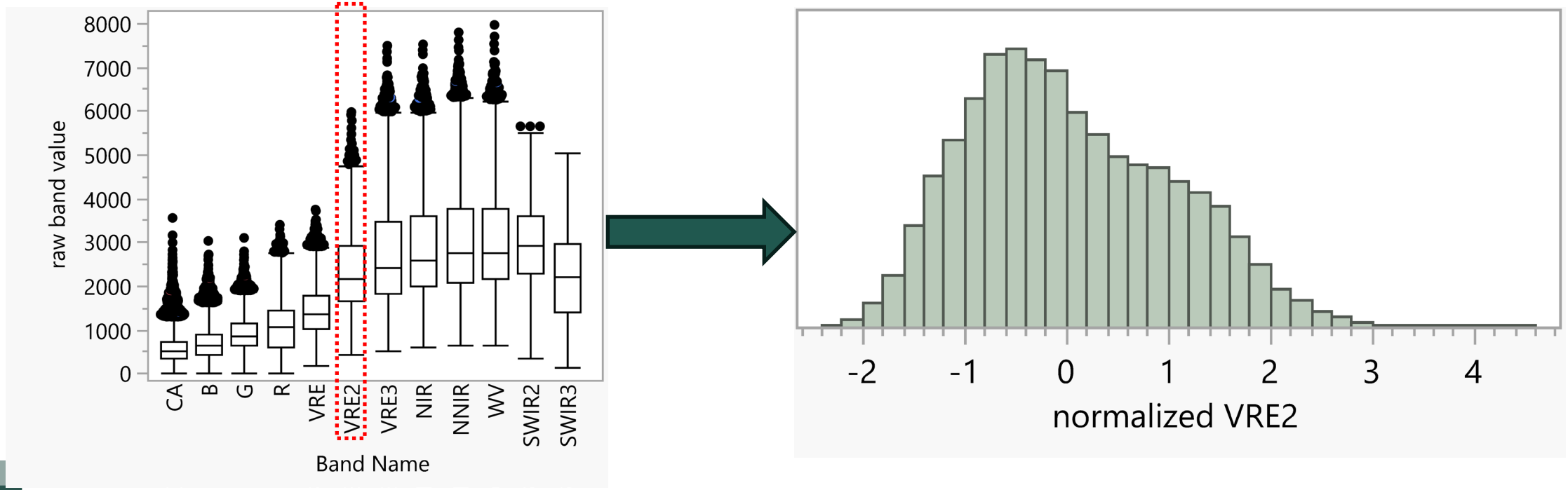
- We must be careful to not "leak" any information across time/space from the training set into the validation or test sets, else our model's performance may look better than it may perform in reality.
- **For our task of predicting CDL labels using imagery, we perform the train/val/test splits using the year the images were taken. This ensures no information is shared between the data splits.**
  - **2021 train, 2020 validation, 2019 test**



Normalization

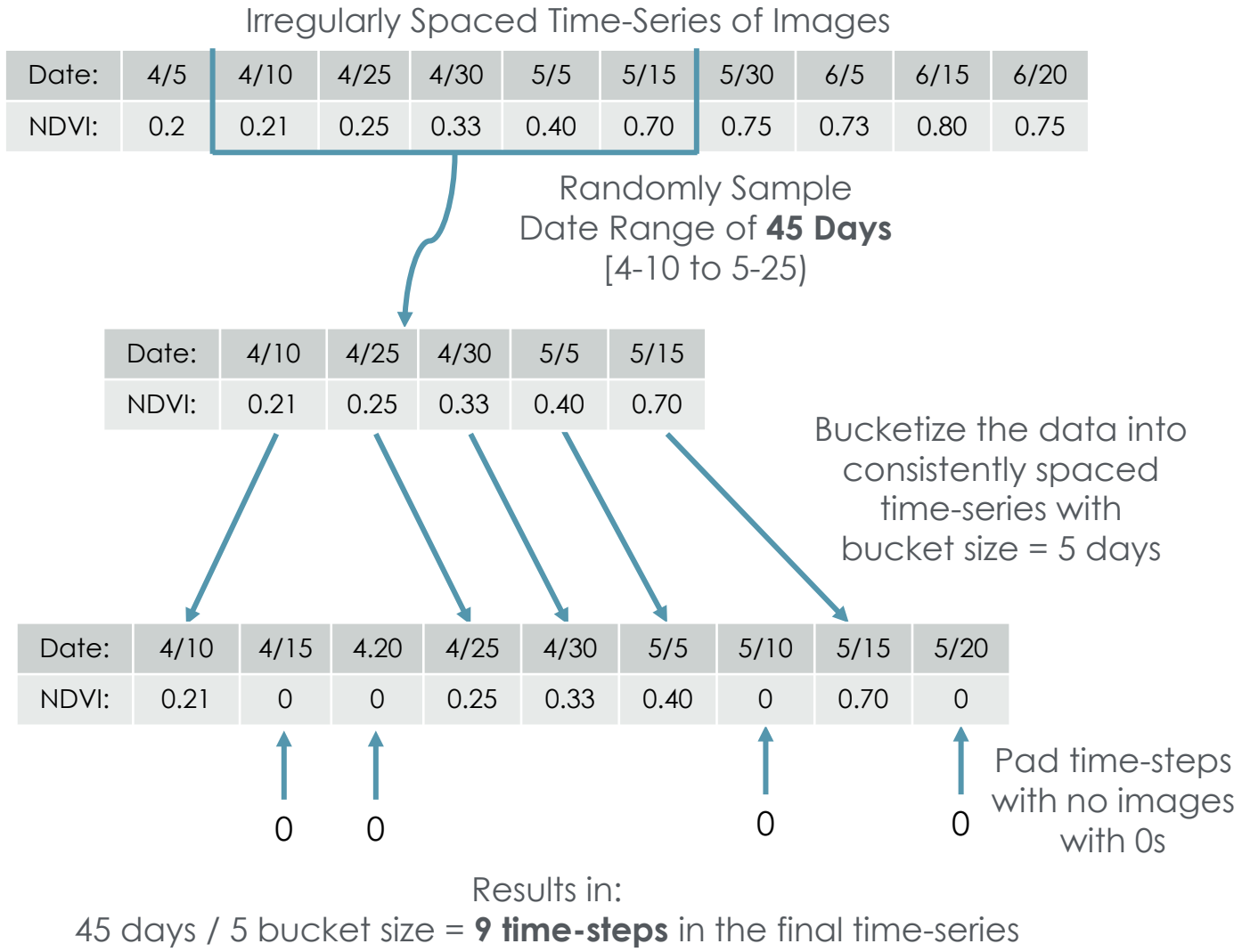
- The inputs to our model are the 12 Sentinel-2 bands, which each have a different range of values. This is not conducive to typical optimization procedures.
- Normalization is the process of making each feature have a mean of 0 and std of 1
  - This step is typical for optimization procedures in statistical modeling or convergence issues can result.

X_normalized = (X – mean_of_features) / std_of_features



Bucketing Irregularly-Spaced Time-Series Data

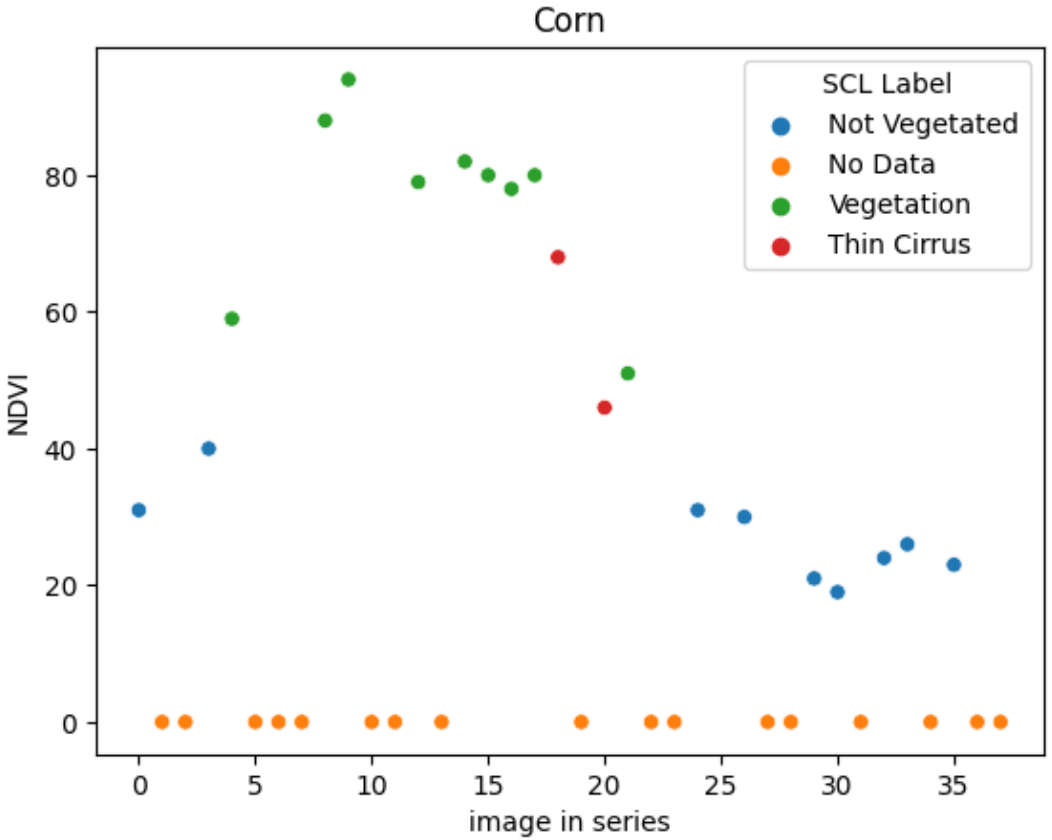
- Most model architectures require standard input size for model training, so having time-series with differing number of images won't work.
  - e.g., 1D-CNN (Convolutional Neural Network)
- We need to standardize the length of the input time-series.
- We do this by taking images within a **randomly sampled date range**, **bucketing** the time-series with a constant size for each bucket (e.g., 5 days), and **padding** any buckets without imagery with 0 values.
  - The random sampling also helps the model's generalization performance
- This ensures each time-series is the same length for model training.
- The final data loader uses a time-range of 120 days.



Results of Bucketing The Data (Discretizing in Time Dimension)

Example:

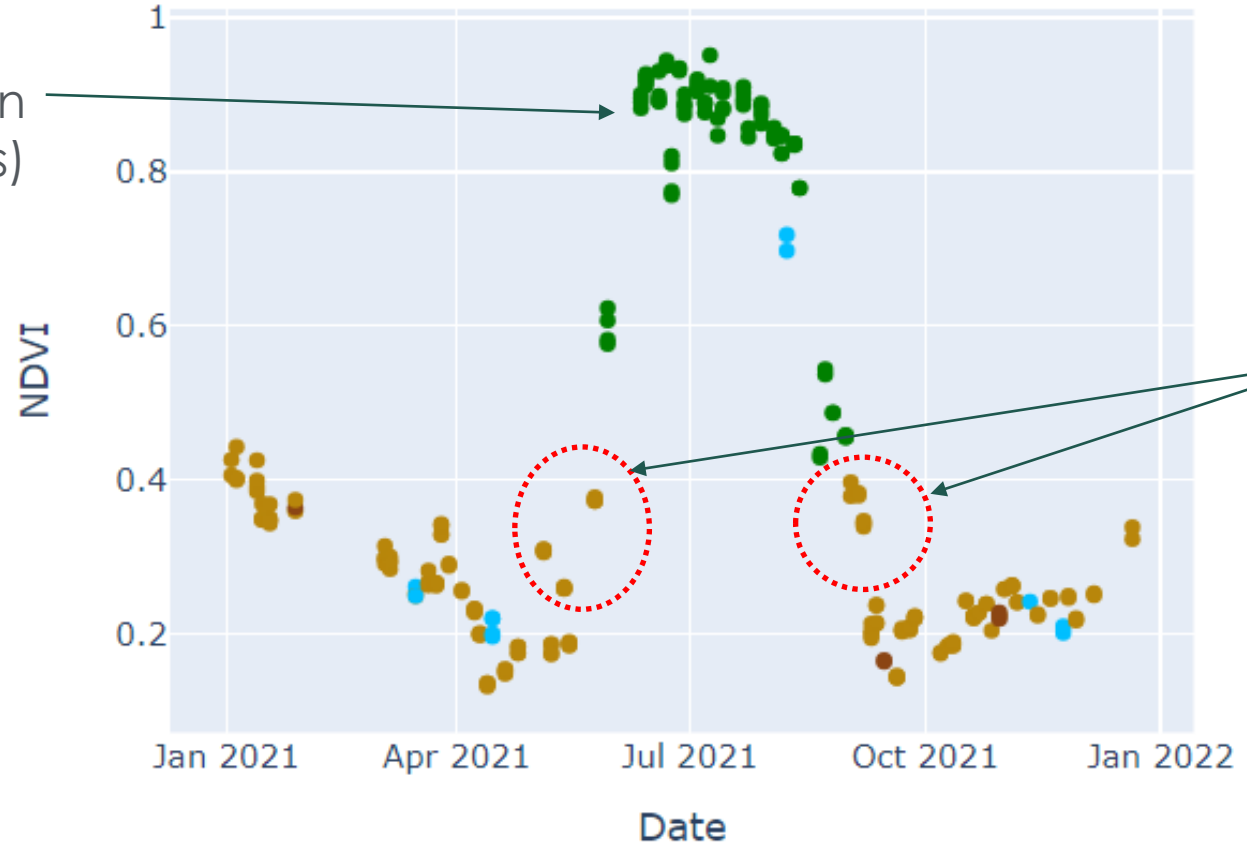
- 38 images total
- 180-day timeframe/span
- Discrete time interval "bucket" size of 5-days
- Buckets/intervals with no image are given a default "0" value.
  - Akin to substituting a "mean" value for each band when all data is normalized.



Scene Classification Layer Filtering

Of all the categories, the Scene Classification (SCL) layer is most reliable at identifying when something is vegetated (i.e., high precision). Thus, we can use this particular output from the SCL to signal when it's a reasonable time to predict crop type (e.g., at least one image from the last 2 (approx. 10 days) in the time-series must have been vegetated).

Vegetation (green points)



These areas during initial crop growth and late senescence have vegetation but are not picked up by the SCL classifier as vegetation. Thus, we need to require vegetation within a certain timeframe of latest image to make a prediction.

Modifying Target Variables to Align with Training Goals

- Limit the target space to align with our goals to reduce problem complexity.
  - E.g., focus on vegetated timeframes.
- In this example, we want to identify
  - 4 major crop types: **Corn, Soybeans, Cotton, and Rice**
  - 1 label for other cultivated crops (general crops)
  - 1 label for uncultivated areas (e.g., urban areas)
  - 0 for any area without a crop growing that is otherwise cultivated
- Reduces the target space of the CDL from 100+ labels to 6 labels.
  - This can be customized depending on your own training goals.

Target Variable Groups Summary

Primary CDL Labels
Corn
Soybeans
Cotton
Rice

"Other Crops" CDL Labels	
Other Hay/Non-Alfalfa	Winter Wheat
Pop or Orn Corn	Alfalfa
Peanuts	Potatoes
Sorghum	Peas
Oats	Herbs
Peaches	Rye
Clover/Wildflowers	Cantaloupes
Pecans	Sunflower
Sod/Grass Seed	Watermelons
Other Crops	Sweet Corn
Dry Beans	Sweet Potatoes

Uncultivated CDL Labels
All other labels

Final Training Label List
Corn
Soybeans
Cotton
Rice
Cultivated
Uncultivated
No Crops Growing

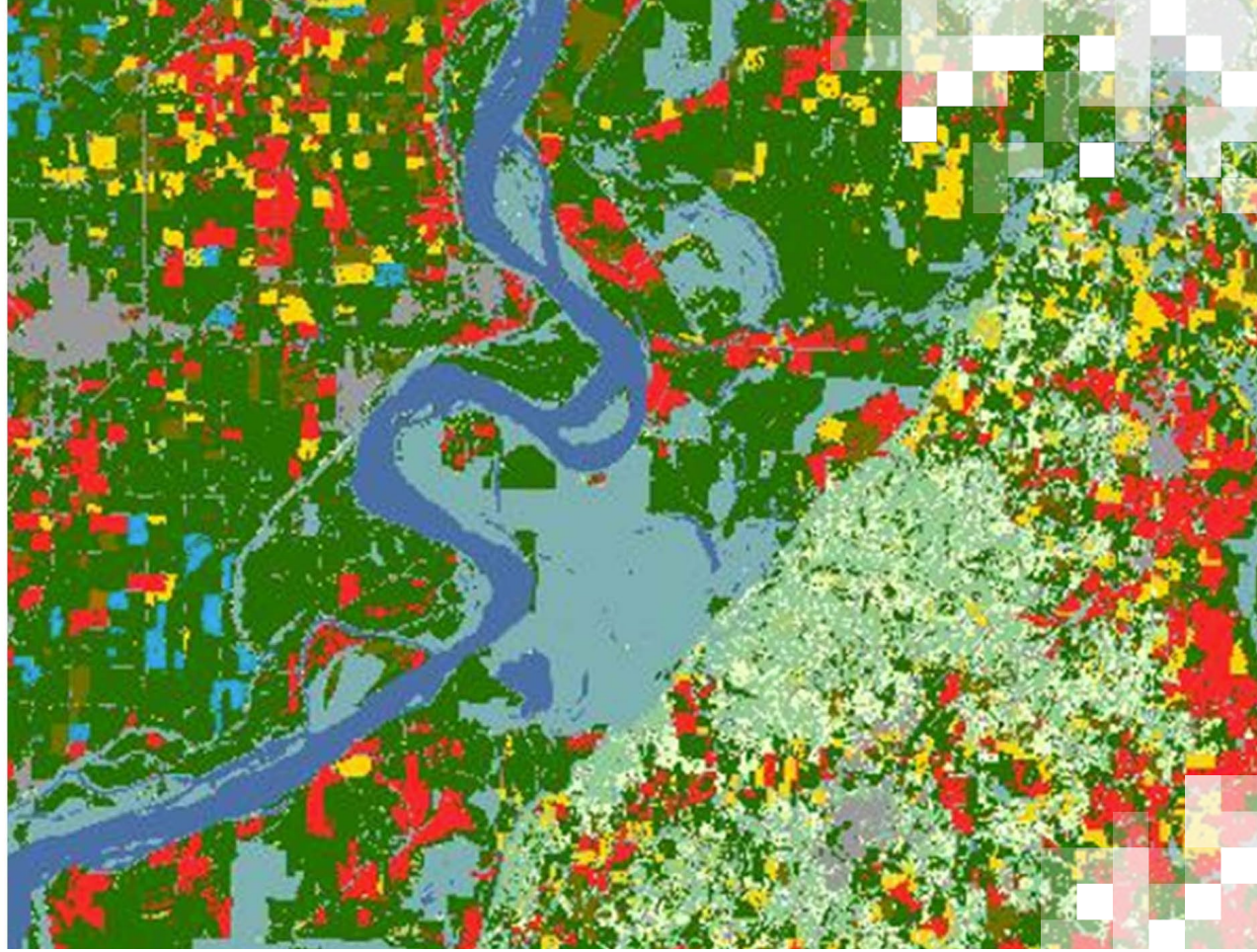
One-hot-encoded Labels

- One-hot-encoding is a method of transforming the label space into 0s and 1s.
- The index where the 1 value is indicates the label type (e.g., a 1 in the **first** index indicates **Cultivated**)
- Therefore, the length of the one-hot-encoded label is the number of labels to classify. In our case, the length is 6.
- This is necessary if using a Softmax activation function with Categorical Crossentropy loss.
  - We will get into the details of these in Part 3.

Our Labels
Uncultivated
Cultivated
No Crops Growing
Corn
Soybeans
Cotton
Rice

One-hot-encoded Labels

No Crops Growing	Uncultivated	Cultivated	Corn	Soybeans	Cotton	Rice
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0



Part 2:
Summary

Summary

- Created a customized and highly efficient queue to load data via **TensorFlow datasets** using the **Parquet** files from Part-1 as inputs.
- Map, shuffle, batch, and prefetch to optimize the performance of the TensorFlow dataset with parallelization.
  - Use modules like **tf.io.decode_raw()** to convert the **byte strings** back to useable data.
- Examined some pre-processing steps:
  - Split the data into train/val/test splits properly to avoid **"data-leakage"** and track if the model is **overfitting**.
  - **Normalized** the data inputs for improved model training stability.
  - **Bucketed** the irregularly-spaced time-series imagery to prepare for model training.
  - Modified the target variable to **align with training goals**.

Looking Ahead to Part 3

- Training a 1D-CNN (One-Dimensional – Convolutional Neural Network) to predict crop type from Sentinel-2 imagery.
- Monitoring model performance in Databricks using Tensorboard.
- Testing and Visualizing model results.

Homework and Certificates

- **Homework:**
  - One homework assignment
  - Opens on March 19
  - Access from the training webpage
  - Answers must be submitted via Google Forms
  - **Due by April 1**

- **Certificate of Completion:**
  - Attend all three live webinars (attendance is recorded automatically)
  - Complete the homework assignment by the deadline
  - You will receive a certificate via email approximately two months after completion of the course.

Contact Information

Trainers:

- John Just (John Deere)
  - JustJohnP@JohnDeere.com
- Erik Sorensen
  - SorensenErik@JohnDeere.com
- Sean McCartney
  - Sean.McCartney@nasa.gov

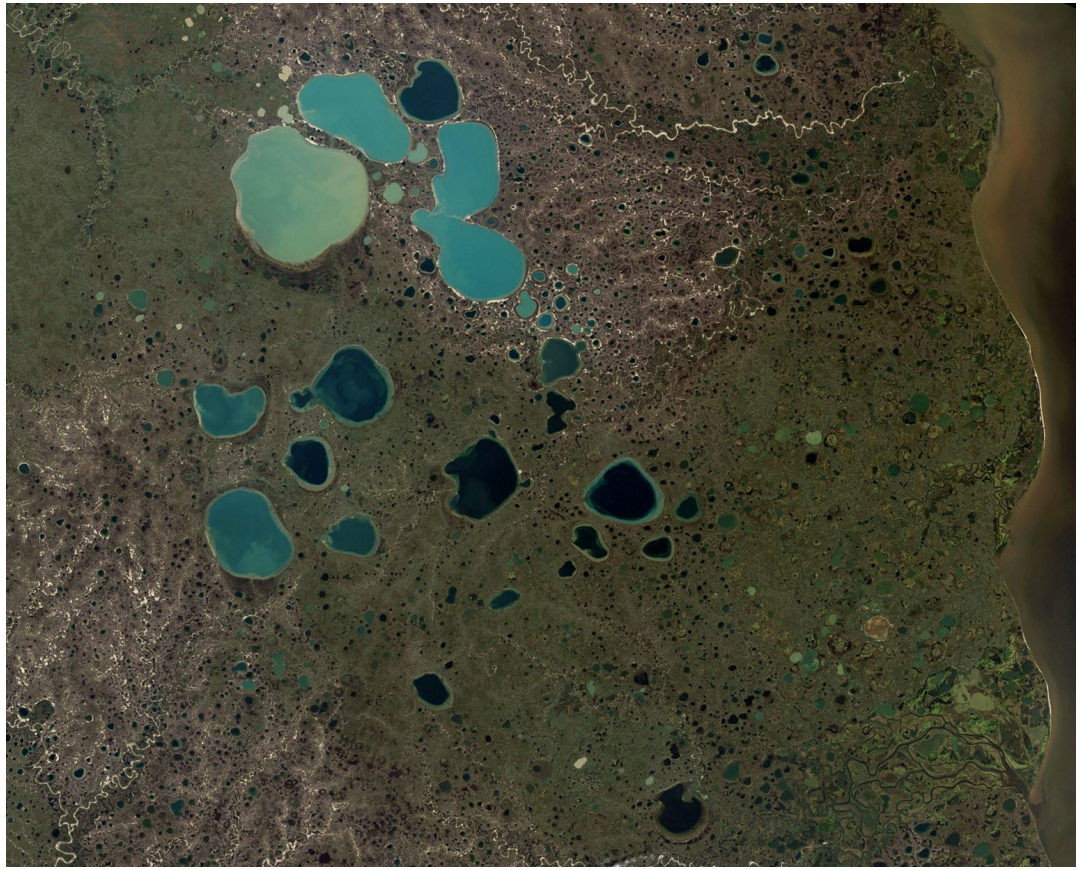
- ARSET Website
- Follow us on X (formerly Twitter)!
  - @NASAARSET
- ARSET YouTube

Visit our Sister Programs:

- DEVELOP
- SERVIR

Questions?

- Please enter your questions in the Q&A box. We will answer them in the order they were received.
- We will post the Q&A to the training website following the conclusion of the webinar.



https://earthobservatory.nasa.gov/images/6034/pothole-lakes-in-siberia


APPLIED REMOTE SENSING TRAINING PROGRAM
TRAIN - EMPOWER - ADVANCE

Thank You!