Advanced Analysis of Satellite Imagery Using Python
Hong Xu, Kaixi Zhang
Workshop Outline

• ArcGIS is a platform for imagery
• ArcGIS has rich image analysis APIs
  - This workshop will focus on Python, specifically ArcPy in Pro 2.5 release
• Use cases
  - Annual Max NDVI Trend Analysis using time series Landsat images
  - Analyze land cover changes using linear spectral unmixing
ArcGIS Provides a Comprehensive Platform for Imagery and Remote Sensing

Management & Dissemination
- Image Analyst
- Image Server

Content
- Image Space and Mensuration
- Visualization & Exploration
- Full-Motion Video
- Oriented Imagery
- Charting
- Spectral Profile
- Temporal Profile
- Street Views

Analysis
- Indexes
- Classification
- AI / Deep Learning
- Raster Processing Functions
- Change Analysis

Map Production
- Drone Mapping
- Ortho Mapping
- Drone2Map
- Seamless Orthophotos
- Ortho Maker

Content
- All Major Sensors
- DTM Generation
- Digital Terrain Model
- Small Sats
- High-Resolution Satellites
- Community Imagery
- Radar
- Video
- Aerial Photos
- Sentinel
- Drones

Integrated and Massively Scalable
Mosaic datasets - optimal data model for managing imagery and rasters
  - 20+ satellite data: Landsat, MODIS, Sentinel, etc.
  - Reference imagery and maintain metadata
  - Define processing to be applied
  - Scalable and flexible

Multidimensional raster
  - Supports multidimensional raster analysis

Image server
  - Dynamic Mosaicking
  - On-the-fly processing
  - Persist large datasets as required
  - Scalable
  - Server as Image Services, WMS, WCS, KML
Image Analysis
Extracting Information from Imagery

Classification, deep learning, machine learning, prediction, multidimensional raster analysis

- **Image Analyst tools and ArcGIS Pro UX**
  - Complete image classification workflow
  - Interface with deep learning toolkits to create training data
  - Inferencing to identify, label, or classify imagery

- **Raster functions** - fast on-the-fly and persisted image processing
- **Raster analytics** - scale up and distribute analysis for large datasets
ArcGIS Python Environment for Image Processing and Analysis

• **Python packages**
  - ArcPy
    - ArcPy.ia (Image Analyst)
    - ArcGIS API for Python

• **Code snippets and demos use**
  - ArcPy in Pro 2.5
  - ArcGIS Notebooks
Processing One Image

import arcpy
arcpy.CheckOutExtension("Image Analyst")
# create a Raster
scene = arcpy.Raster(r"C:\Landsat\LC080428342013236LGN80\LC080428342013236LGN80_MTL.txt\Multispectral")
# Display
sceneRGB = arcpy.ia.Render(scene, rendering_rule=["bands": [4, 3, 2], 'Number of Standard Deviation: 2'])
# Compute an NDVI
sceneNDVI = arcpy.ia.NDVI(scene, [5, 4])
sceneNDVI_render = arcpy.ia.Render(sceneNDVI, rendering_rule=["min": -1, 'max': 1], colormap="NDVI")
Process a Collection of Images

- Landsat time series imagery
  - 16-day cycle, 40 years
  - 30m/10m
  - Blue, Green, Red, NIR, SWIR-1, SWRI-2, TIR, Pan

- To discover
  - What is the water occurrence rate in past 10 years?
  - Did deforestation happen in this area?
  - What are the land cover changes?
  - What are the pattern and trend?
  - Etc.
Workflow of Processing a Collection of Images Using Python

1. Raster Collection
2. Filter
3. Map
4. Reduce
Use Case 1: Annual Max NDVI Trend Analysis using time series Landsat images

- Input: a folder contains 173 Landsat scenes
- Output: a time series Annual Maximum NDVI and a trend raster
Raster Collection

RasterCollection (rasters, {attribute_dict})
Raster Collection

- Created from
  - List of rasters
    ```python
    import arcpy
    # from a list of rasters
    in_rasters = [r"D:\data\L7_2000.tif", r"D:\data\L7_2001.tif", r"D:\data\L7_2002.tif"]
    rc=arcpy.ia.RasterCollection(in_rasters, {"Variable": "landsat7",
    ```

- mosaic dataset, Image service, multidimensional raster
  ```python
  # create from a mosaic dataset
  rc=arcpy.ia.RasterCollection(r"D:\data\Landsat7.gdb\CA_1990_2000")
  # create from an Image service
  rc=arcpy.ia.RasterCollection("https://localhost/server/rest/services/CA/ImageServer")
  # create from multidimensional raster
  rc=arcpy.ia.RasterCollection(r"D:\data\CA_raster.crf")
  ```

Filter

- **Filter items by**
  - Spatial extent
  - Time extent
  - Geometry
  - Where clause and attributes
  - Raster properties

```python
filtered_rc = rc.filterByTime(end_time = '2009-01-01T00:00:00', time_field_name = 'AcquisitionDate')
filtered_rc = rc.filterByGeometry(line)
filtered_rc = rc.filterByAttribute("SensorName", "CONTAINS", "Landsat")
filtered_rc = rc.filterByCalendarRange(calendar_field="DAY_OF_YEAR", start=1, end=5)

filtered_rc = rc.filterByGeometry(study_area).filterByTime(time_field_name = 'AcquisitionDate',
start_time = '1998-01-01 00:00:00', end_time = '1995-12-31 00:00:00')
```
Map

- Map a function for each item in the RasterCollection
  - \( N \rightarrow N \)
- `RasterCollection.map(func)`

```python
def MyFunc(item):
    in_raster = item['Raster']
    processed_raster = process_function(in_raster…)
    return {'raster': processed_raster, …}

processed_rc = rc.map(MyFunc)
```

```python
def calcNDVI(item):
    ndvi = arcpy.ia.NDVI(item['Raster'], nir_band_id = 4, red_band_id = 3)
    return {'raster': ndvi, 'Name': item['Name'], 'AcquisitionDate': item['AcquisitionDate']}

ndvi_rc = filtered_rc.map(calcNDVI)
```
Support many built-in functions

Band index functions
- BandArithmetic, Cig, Cire, ClayMinerals, EVI, FerrousMinerals, GEMI, GNDVI, GVITM, IronOxide, MSAVI, MTVI2, NDVI, NDVire, NDWI, PVI, RTVICore, SAVI, SRre, Sultan, TSAVI, VARI

Classification

Pixel stretch functions
- Stretch, convolution, Pansharpen, and etc.

Analysis functions
- TasseledCap, Threshold, HeatIndex, WindChill, Apply and etc.
  arcpy.ia.Apply (in_raster, raster_function, {raster_function_arguments})

- more…

Custom functions
Reduce

- Reduce by computing pixel values from items
  - N→1
- Supported methods
  - Min, Max, Mean, Sum, Median, Majority
  - Mosaic (Frist, Last)

```python
# get max NDVI in each year
annual_max_ndvi_list = []
for year in range(1990, 2000):
    # choose all items in the given year
    ndvi_in_one_year = ndvi_rc.filterByCalendarRange(calendar_field = "YEAR", start = year, time_field_name = "AcquisitionDate")
    # get max NDVI
    annual_max_ndvi_list.append(ndvi_in_one_year.max)
annual_max_ndvi_rc = arcpy.ia.RasterCollection(annual_max_ndvi_list, "years": [yr for yr in range(1990, 2000)])
```
Multidimensional Analysis

- Convert to a multidimensional raster
  - RasterCollection::ToMultidimensionalRaster

- Perform analysis using functions
  - Aggregate
  - Anomaly
  - Trend
  - Multidimensional algebra
  - ...

```python
# convert raster collection to a multidimensional raster
ndvi_md_raster = ndvi_rc.ToMultidimensionalRaster(variable_field_name='Variable', dimension_field_names='StdTime')

# get max NDVI per year
annual_max_ndvi = arcpy.sa.Aggregate(ndvi_md_raster, dimension_name = "StdTime", raster_function = "MaxIgnoreNoData",
                                      aggregation_definition = "yearly")

# apply trend
max_ndvi_linear_trend = arcpy.sa.Apply(annual_max_ndvi, "TrendAnalysis",
                                       raster_function_arguments = {"DimensionName": "StdTime", "RegressionType": 0})
```
Annual Max NDVI Trend Analysis using time series Landsat images

Kaixi
Analyze land cover changes using linear spectral unmixing

Kaixi
Use Case 2: Analyze Land Cover Change Using Linear Unmixing

- Input: a mosaic dataset contains 1087 Images
- Processing: linear unmixing function
- Output: time series fraction + statistics output
Mosaicking a Collection of Images

- **RasterCollection.mosaic** *(mosaic_method)*
  - rc.mosaic('LAST')
  - rc.mosaic('FIRST')

- Code snippet: mosaic RGB band of most recent images

```python
# sort the collection from earliest to latest
sorted_rc = rc.sort('StdTime')
# choose Blue, Green, Red bands from each item in the collection
rgb_collection = sorted_rc.selectBands([2,3,4])
# mosaic
mosaic_raster = rgb_collection.mosaic('LAST')
# save
mosaic_raster.save(r"c:\data\mosaic_output.crf")
```
Processing Settings

- **Output formats**
  - CRF (.crf) : output one item or multiple items as a multidimensional raster
  - TIFF (.tif) : Output single item

- **Output compression**
  - `arcpy.env.compression = "LERC 0.01"`  # for controlled accuracy
  - `arcpy.env.compression = "LERC 0"`    # lossless
  - `arcpy.env.compression = "LZ77 75"`  # lossless
  - `arcpy.env.compression = "JPEG 75"`   # lossy

- **Parallel processing factor**
  - `arcpy.env.parallelprocessingfactor = "6"`
  - `arcpy.env.parallelprocessingfactor = "90%"`
Help References


- Same capabilities and APIs for *ArcGIS for Python* will be available in ArcGIS 10.8.1
Understanding Urban Expansion Using Nightlight Imagery

- Yearly nightlight images (1992-2013)
- Trend function
- Render as RGB

```python
# Fit Linear Regression Model
nightlight_linear_trend = arcpy.t.A Apply(nightlights_1992_2013, "TrendAnalysis",
  raster_function_arguments = {"DimensionName": "stdTime", "RegressionType": 0})

# check the dimension value of the trend output
nightlight_linear_trend.getDimensionValues("NightlightData", "stdTime")
```