README Open-Source Python Software Deep Learning Automated Detection of Severe Storm Signatures Within Geostationary Satellite Imagery

John W. Cooney

June 26, 2023

Contents

1 *Document Overview*

This document is intended to help the user install and run the open source software to detect severe storm signatures in geostationary imagery. The first thing the user will want to do is install Anaconda (Section 13.1). Anaconda allows the user to install many of the necessary Python modules in a conda environment so that all of the python modules are always compatible. This document will walk the user through the entire software setup as well as show the user how the model can be run. All code within this document is capable of being run entirely on the Google Cloud Platform (Section 9) and using Google's Virtual Machines (VMs), but Google is not necessary to run the model. Ignore any references to plotting/mapping the model results. Due to licensing restrictions, we are unable to offer that service at this time, however, we hope to offer it in future software releases.

2 *Installing Software and Creating Python Environment*

IMPORTANT NOTE!!! Software could take an hour to install. The package is setup to grab netCDF look up table files that are stored on a NASA server for faster run jobs.

There are 10 steps for installing the software. First, the user MUST have Anaconda installed (Section 13.1). Do not do the following without first installing Anaconda. This software package requires the use of the glmtools and lmatools libraries. The steps below will install these GitHub repositories for you. Please follow the steps in order for a successful installation.

1. the user must clone the project repository *git clone https://github.com/nasa/svrstormsig.git* in Terminal.
2. cd svrstormsig
3. git clone https://github.com/deeplycloudy/glmtools.git
4. cd glmtools
5. cp ../environment.yml .
6. cp ../setup.py .

7. conda env create -f environment.yml
8. conda activate svrstormsig
9. pip install -e .
10. pip install opencv-python-headless

Once the steps are complete you should be able to successfully run the software. The package can then be used when run within the svrstormsig environment. In order to activate the svrstormsig environment, type *conda activate svrstormsig* into the Terminal. This environment lets you use all of the installed libraries for this software. The package can also be used when run as a kernel within the svrstormsig environment. This kernel is created by running: python -m ipykernel install –user –name=svrstormsig. Note the difference between 1 hyphen in front of m and 2 hyphens in front of user and name. If ipykernel is not installed then run: conda install ipykernel and try again.

2.1 *Software Path Setup*

This section displays what the directory and subdirectory structure of the software should look like after install.

Directory Structure:

⟶ svrstormsig
- ⟶ data
  - ⟶ model_checkpoints
  - ⟶ region
  - ⟶ sat_projection_files
- ⟶ glmtools
  - ⟶ glmtools
  - ⟶ glmtools_docs
  - ⟶ glmtools.egg-info
- ⟶ python
  - ⟶ EDA
  - ⟶ glm_gridder

⟶ gridrad

⟶ model

⟶ new_model

⟶ visualize_results

3 *Running the Model Software*

This section will discuss how to run the software, optimal models, script breakdown, and the directory path setup.

3.1 *How to Run the Software*

First, activate your conda environment in the Terminal by typing *conda activate svrstormsig*. This will put you into the environment that has all of the required python modules. All of the software is built and run using Python code. The main script to run the code is run_all_plume_updraft_model_predict.py and is located in the new_model subdirectory. The easiest 2 methods to run the software are listed below. The first method allows the user to answer prompts in the Terminal to determine what model they want to run, the model inputs, and the dates they want to run the model over. The second method takes those prompts and puts them into a more detailed and easy to use configuration file. The configuration file can then be loaded into the software to specify the user's model run needs. See Section 3.2 for more details on the configuration file setup. The default configuration is provided at the time of software install. Note the 2 hyphens used in front of config.

1. python3 path-to-run-script/run_all_plume_updraft_model_predict.py

2. python3 path-to-run-script/run_all_plume_updraft_model_predict.py –config path-to-configuration-file/modrun_config.cfg

3.2 *Configuration File*

A default configuration file, modrun_config.cfg, is installed when you clone the project git repository. This file can be changed by the user to suit their needs. Some may find using a configuration file easier than answering prompts in the Terminal window. The contents of the default configuration file are shown below. Users should ONLY change items to the right of the = sign to suit their model run needs. If anyone would like additions, clarifications, or changes, please email john.w.cooney@nasa.gov and we will address those changes as soon as possible for the next software release.

```
#########################
## Model Run Configuration Settings
##
#  Number 2 corresponds to real-time run ONLY.
#  Numbers 3-4 correspond to not real-time/archived run ONLY.
#
## IMPORTANT NOTE: ONLY change values on the right side of equals sign!!

#1)
#Is this a real-time run or not? (y/n):
real_time = n

#2)
#Question 2 corresponds to real-time run ONLY. Ignore if real_time in Q1 is n or no
#Type below the number of hours you want the model to run for:
number_of_hours = 0.01

#3)
#Question 3 corresponds to not-real-time run ONLY. Ignore if real_time in Q1 is y or yes
#Enter the model start date (ex. 2017-04-29 00:00:00):
#start_date = 2023-05-11 17:00:00
start_date = 2023-06-21 19:00:00

#4)
#Question 4 corresponds to not-real-time run ONLY. Ignore if real_time in Q1 is y or yes
#Enter the model start date (ex. 2017-04-29 00:00:00):
#end_date = 2023-05-12 03:00:00
end_date = 2023-06-22 02:30:00

#5)
#Question 5 corresponds to not-real-time run ONLY. Ignore if real_time in Q1 is y or yes
#Do you need to download the satellite data files? (y/n):
download_data = y

#6)
#If download_data is n or no or if real_time in Q1 is y or yes, ignore this question.
#Question 6 corresponds to not-real-time run ONLY.
#Enter the location of raw data files:
#DEFAULT = None, which implies to use the default file root location in the code.
raw_data_directory = None

#5)
#Type below the name of the satellite you wish to use (ex. goes-16)
satellite = goes-16

#6)
#Type the satellite scan sector you wish to use. If satellite does not have multiple
#scan sectors, ignore this. (ex. M1, M2, F, C)
sector = M2

#7)
#Type US state or country you wish to bound model output to.
#DEFAULT = None, so model is not bound to any particular state or country.
region = None

#8)
#Type longitude and latitude boundaries to confine the data to (in degrees):
#[longitude_min, latitude_min, longitude_max, latitude_max]
#(ex. [-100.0, 36.0, -86.0, 42.0])
#DEFAULT = [], so model results output are not restricted to any user
#specified lat/lon bounding box
```

```
xy_bounds = []

#9)
#Enter the desired severe weather indicator (OT or AACP):
#OT   = Overshooting Top
#AACP = Above Anvil Cirrus Plume
severe_storm_signature = AACP

#10)
#Would you like to be able to seamlessly transition between previously identified (y/n)?
#best day and night models?
#Best OT model for daytime is IR+VIS+GLM and best OT model at night is IR.
#Best AACP model for daytime is IR+VIS+GLM and best AACP model at night is IR+GLM.
optimal_params = y

#11)
#If optimal_params is n or no, what model inputs would you like to use? Keep in mind,
#visible data will is not available during night time hours and thus, no model
#results files will be created.
#Best OT model for daytime is IR+VIS+GLM
#Best OT model at night is IR.
#Best AACP model for daytime is IR+VIS+GLM
#Best AACP model at night is IR+GLM.
#(ex. IR, IR+VIS, IR+GLM, IR+IRDIFF, IR+VIS+GLM)
#NOTE: IR+IRDIFF is not available for AACP model runs
#DEFAULT = IR
model_inputs = IR

#12)
#If optimal_params is n or no, what model type would you like to run? Keep in mind,
#visible data will is not available during night time hours and thus, no model
#results files will be created.
#Mutiresunet model has been found to be the best model for both OTs and AACPs
#(ex. multiresunet, unet, attentionunet)
#NOTE: mutltiresunet is the only model type available for AACP model runs
model_type = multiresunet

#13)
#Would you like to map the model results on top of IR/VIS sandwhich images for each
#individual scene (y/n)?
#NOTE: time to plot may cause model run to be slower.
plot_data = n

#14)
#If plot_data is n or no, ignore the remaining 2 prompts.
#If plot_data is y or yes, would you like to use the previously identified best
#likelihood score threshold for your above choices (y/n)?
optimal_likelihood_response = y

#15)
#If optimal_likelihood_response is set to y or yes, ignore this.
#If optimal_likelihood_response is set to n or no, what thresholds would you like to use
#for creating the plots and identifying objects in post-processing?
#optimal_likelihood_score must be between 0 and 1.
optimal_likelihood_score = 0.4

#16)
#If severe_storm_signature is set to AACP, ignore this.
#If severe_storm_signature is set to OT, percent_omit removes the coldest and warmest X%
#when calculating the mean anvil brightness temperatures (BTDs) in post-processing.
#Default is 20 which means 20% of the warmest and coldest anvil pixel temperatures are
```

```
#removed in order to prevent the contribution of noise to the OT IR-anvil BTD calculation.
#percent_omit must be between 0 and 100.
percent_omit = 20
```

3.3 *Optimal Models*

See Table 2 for the model types and inputs along with the optimal model thresholds for each model type allowed.

3.3.1 *OT Optimal Model From Testing*

During the day, the OT detection model that performed best during our tests was the IR+VIS+GLM MultiResUnet model. Table 2 shows the optimal threshold for this model is 0.40. See Section 4.4 for more details regarding optimal model thresholds.

During the night, the OT detection model that performed best during our tests was the IR MultiResUnet model. Table 2 shows the optimal threshold for this model is 0.40. See Section 4.4 for more details regarding optimal model thresholds.

3.3.2 *AACP Optimal Model From Testing*

During the day, the AACP detection model that performed best during our tests was the IR+VIS+GLM MultiResUnet model. Table 2 shows the optimal threshold for this model is 0.30. See Section 4.4 for more details regarding optimal model thresholds.

During the night, the AACP detection model that performed best during our tests was the IR+GLM MultiResUnet model. Table 2 shows the optimal threshold for this model is 0.60. See Section 4.4 for more details regarding optimal model thresholds.

3.4 *Model Checkpoint Files*

The best model checkpoint files are included during the install process. These files were output during our model training and from runs that performed well in tests. They are located in the 'data/model_checkpoints/' directory of the install. These files contain the information that the model needs in order to make detections for AACPs or OTs.

3.5 *Script Breakdown*

The following shows the order of operations for the software and the scripts that are called to carry that out.

1. `run_all_plume_updraft_model_predict.py` is the MAIN function. This function prompts the user with questions regarding what parameters they want to use to run the model and the dates/region to run the model over. A configuration file can also be loaded into this function which answers the questions for the user ahead of time.

2. (a) `run_tf_1_channel_plume_updraft_day_predict` takes 1 model input channel (typically IR only) and calls all of the subroutines to pre-process and run the model.

   (b) `run_tf_2_channel_plume_updraft_day_predict` takes 2 model input channels (IR+VIS or IR+GLM) and calls all of the subroutines to pre-process and run the model.

   (c) `run_tf_3_channel_plume_updraft_day_predict` takes 3 model input channels (IR+VIS+GLM) and calls all of the subroutines to pre-process and run the model.

3. (a) `run_download_goes_ir_vis_l1b_glm_l2_data` used in real-time model runs. Downloads the latest GOES ABI and GLM data from Google Cloud's public data storage. See Section 5 for more details.

   (b) `run_download_goes_ir_vis_l1b_glm_l2_data_parallel` used in archived model runs. This model uses parallel threads in order to speed up GOES ABI and GLM data downloads from Google Cloud's public data storage. See Section 5 for more details.

4. (a) `run_create_image_from_three_modalities` used in real-time model runs. If the user specifies they want output maps, this function will open a new Thread to start creating the image in the background while the rest of the model process runs. Once the model is finished running and the output file is available, the parallel Thread will plot the model results onto an IR/VIS sandwich image. Creates "combined" netCDF file of the data. See Section 6 for more details.

(b) `run_create_image_from_three_modalities2` used in archived model runs. This model uses parallel threads in order to speed up archived runs. This function interpolates IR and GLM data onto the VIS data grid to create "combined" netCDF file of the data. See Section 6 for more details.

5. `create_vis_ir_numpy_arrays_from_netcdf_files2` creates IR, VIS, and GLM numpy files for a specific date and satellite scan sectors. The numpy arrays are stored (1, number of latitude pixels, number of longitude pixels). This function uses the combined netCDF files (created from combine_ir_glm_vis) to normalize the IR, VIS, and GLM data (0-1) which is then output to numpy arrays that are loaded as input into the model. See Section 7.1 for more details. **IMPORTANT NOTE!!!** IR data values are set to -1 in regions that are not able to be scanned by the satellite. These regions come about due to the gridding process for CONUS and FULL disk scans. They are set to -1 in order to distinguish that there should never be any model detection at those points. When running the model, we can easily find those points and make sure the results likelihoods are set to 0 (not an OT or AACP).

6. (a) `tf_1_channel_plume_updraft_day_predict` takes 1 model input channel and runs the specified model. Writes model output to numpy file and also opens the combined netCDF file and writes model results there too.

   (b) `tf_2_channel_plume_updraft_day_predict` takes 2 model input channels and runs the specified model. Writes model output to numpy file and also opens the combined netCDF file and writes model results there too.

   (c) `tf_3_channel_plume_updraft_day_predict` takes 3 model input channels and runs the specified model. Writes model output to numpy file and also opens the combined netCDF file and writes model results there too.

7. `write_plot_model_result_predictions` is used in archived runs to map the model results for each individual scene as well as a time aggregated map of the model results, SPC reports, and minimum brightness temperatures.

3.6 *Model Input/Output Files and Locations*

The software creates files that are used as input into the model and also outputs multiple files and, optionally, an image file. This list of input and output files as well as their default locations are listed below.

1. If downloading files, the user can specify the download directory location, however, the default location is '../../../data/goes-data/'. The subdirectory is then the 4 digit year + 2 digit month + 2 digit day.

2. The first file created is termed in this document as the combined netCDF file. It is created by the combine_ir_glm_vis function, as described in Section 6. An example file header is provided in Section 13.2.1. This file contains satellite information which can be used for mapping the data and model input variables. This file will also be rewritten later to add the contents of the model detections. These files can be read into the McIDAS-V software for visualization. The files are stored in '../../../data/goes-data/combined_nc_dir/'. The subdirectory is then the 4 digit year + 2 digit month + 2 digit day.

3. Numpy files for each model input variable are created for a single time/scan. Depending on the model run chosen, numpy arrays for IR, VIS, and GLM are used as model inputs into the the machine learning algorithm. All of the numpy arrays contain normalized values ranging between 0 and 1. These files are stored in '../../../data/labelled/'. See Section 7 for more details.

4. csv file containing IR/VIS/GLM file names with dates and times. This is created in order to keep organized which numpy file goes with which date and raw data file. These files are stored in '../../../data/labelled/'. See Section 7 for more details.

5. Numpy model results file stores the model likelihoods of a detection for each point in the domain. Values range from 0 to 1, with values closer to 1 implying the model is more confident in a detection. These root directory to these files is '../../../data/aacp_results/'. The subdirectories to the file are then the model type (day_night_optimal, IR, IR+GLM, etc.), what the model was trying to detect (updraft or plume), whether it was a real-time or not real-time model, the date that the model was run, the date of interest, and the scan sector.

6. As mentioned in item (2) in this list, the model results are also output to the combined netCDF file. The combined netCDF is opened and a new variable containing the model output, similar to that seen in the numpy output files is written. Some example variable names are ir_ot, ir_glm_ot, ir_glm_aacp, and ir_vis_glm_aacp. This data was written to the combined netCDF files in order to provide an easier user experience and also be able to visualize the results in the McIDAS-V software.

7. Optional output of a PNG file containing the model detections on a gridded PNG image of the model input data (IR and VIS, if VIS is available). These files are stored in ../../../data/aacp_results_imgs/'. The subdirectories to the file are then the model type (day_night_optimal, IR, IR+GLM, etc.), what the model was trying to detect (updraft or plume), whether it was a real-time or not real-time model, the date that the model was run, the date of interest, and the scan sector.

4 *Model Characteristics*

4.1 *Model Training*

For training the model, we used 1-minute mesoscale domain sectors (M1 or M2) GOES-16 data. Table 1 shows the dates and mesoscale regions used. The variable 'ftype' corresponds to whether or not the date was input into the model as a training (train) or validation (val) dataset. The independent model testing (test) dataset were not input into the model at any time.

4.2 *Model Types*

See Table 2 for the model types and inputs allowed for each model type.

4.2.1 *OTs*

For detecting OTs, we set up the software to handle a variety of satellite inputs and model types. Model inputs for OTs available include: IR only, IR+GLM, IR+VIS, IR+IRDIFF, and IR+VIS+GLM. For each of these input we tested 3 distinct model architectures: unet, multiresunet, and attentionunet. Since VIS data are not available at night, we found optimal models for daytime and nighttime.

Table 1: Dates and times of convective events included in this study.

Date	Time Range (UTC)	Mesoscale Sector	ftype
30 April 2019	1800-2359	1	train
01 May 2019	0000-0043	1	train
05-06 May 2019	2200-0055	1	train
05-06 May 2019	1930-0102	2	train
07-08 May 2019	1800-0100	1	train
17-18 May 2019	1700-0122	2	val
20-21 May 2019	1800-0104	1	train
26-27 May 2019	1800-0056	2	test
13-14 May 2020	1800-0105	2	train

During the day, the OT detection model that performed best during our tests was the IR+VIS+GLM multiresunet model.

During the night, the OT detection model that performed best during our tests was the IR multiresunet model.

4.2.2 *AACPs*

For detecting AACPs, we set up the software to handle a variety of satellite inputs but only 1 model type (multiresunet) is available. Model inputs for OTs available include: IR only, IR+GLM, IR+VIS, and IR+VIS+GLM. Since VIS data are not available at night, we found optimal models for daytime and nighttime.

During the day, the AACP detection model that performed best during our tests was the IR+VIS+GLM multiresunet model.

During the night, the AACP detection model that performed best during our tests was the IR+GLM multiresunet model.

4.3 *Allowed Data Ranges*

For more details on allowed data ranges, refer to create_vis_ir_numpy_arrays_from_netcdf_files2.py or Section 7.2 of this document.

IR BT range is 180 K to 230 K. VIS reflectance range is 0 to 1. GLM flash extent density range is 0 to 20. IRDIFF range is -20 K to 10 K.

4.4 *Optimal Model Likelihood Scores & Thresholds*

For each pixel in the domain, the model outputs the 'confidence' that a pixel is part of the desired severe storm signature. These scores range from 0, not confident, to 1, very confident. Higher scores indicate a higher likelihood that the object you are trying to detect is in fact that object. If you are trying to detect OTs, for instance, a pixel with a score of 0.8 indicates that that pixel is very likely part of an OT. Using Receiver Operating Characteristic (ROC) curves and Intersection Over Union (IOU) statistical metrics in test cases independent from model training, we have found thresholds that we believe work best for limiting the number of false detections while still identifying the vast majority of severe storm signature events (See ROC curve figure). Now, it is important to note that these thresholds may not be best for each individual case and thus need to be adjusted based on the case the user is reviewing. While running the model, the user has the option to change the thresholds to suit their needs and findings. The optimal thresholds found for each model type and inputs are shown in Table 2.

5 *Downloading IR, VIS, and GLM data from Google Cloud*

The GOES data can be downloaded from Google Cloud (see Section 9 for more information about the Google Cloud Platform). The IR, VIS, and GLM data can be downloaded using the run_download_goes_ir_vis_l1b_glm_l2_data.py or run_download_goes_ir_vis_l1b_glm_l2_data_parallel.py functions. Data can be dowloaded in 'real-time' (1 VIS file, 1 IR file, and 15 GLM files) closest to the time of day the script is run. 15 GLM files are downloaded because data files are created every 20 seconds and we want enough data to use with the 1-minute IR and VIS data.

If you do not have any issues downloading the files from GCP to local storage then you can ignore this paragraph. Downloading from a Google VM should be pretty straightforward because

Table 2: Model types and optimal thresholds found from independent test cases.

Severe Storm Signature	Model Type	Model Inputs	Optimal Model Threshold
OT	MultiResUnet	IR	0.40
OT	MultiResUnet	IR+GLM	0.65
OT	MultiResUnet	IR+VIS	0.25
OT	MultiResUnet	IR+IRDIFF	0.35
OT	MultiResUnet	IR+VIS+GLM	0.40
OT	Unet	IR	0.45
OT	Unet	IR+GLM	0.45
OT	Unet	IR+VIS	0.35
OT	Unet	IR+IRDIFF	0.45
OT	Unet	IR+VIS+GLM	0.45
OT	AttentionUnet	IR	?????
OT	AttentionUnet	IR+GLM	?????
OT	AttentionUnet	IR+VIS	0.65
OT	AttentionUnet	IR+IRDIFF	?????
OT	AttentionUnet	IR+VIS+GLM	0.20
AACP	MultiResUnet	IR	0.80
AACP	MultiResUnet	IR+GLM	0.60
AACP	MultiResUnet	IR+VIS	0.40
AACP	MultiResUnet	IR+VIS+GLM	0.30

the Google Credentials should already exist on the VM. An issue can arise when downloading files to your local computer, however. The issue may be that you do not have your Google Credentials set up in your .bashrc or .zshrc file. To fix the issue, you need to create your service account key json file (ex. *gcloud iam service-accounts keys create /Users/user-name/google-cloud-sdk/google-service-account-file.json –iam-account=project@appspot.gserviceaccount.com*) into your Terminal. The path and file name (.json path and name after create) can be whatever and wherever you want. The example above is where I put the file and what I named it. Then I added *export GOOGLE_APPLICATION_CREDENTIALS=/Users/user-name/google-cloud-sdk/google-service-account-file.json* to my .bashrc file. This ultimately fixed the issue.

6 *Output netCDF Files*

6.1 *Overview*

The pipeline necessary to create a layered image and combined netCDF file has four parts: 1) Main run script, 2) GLM data gridding (optional), 3) Combining IR, VIS, and GLM data into netCDF file, and 4) image creation. These functions are named run_create_image_from_three_modalities (or run_create_image_from_three_modalities2 for parallel jobs in archived model runs), glm_gridder, combine_ir_glm_vis, and img_from_three_modalities2, respectively. The main function (1) runs the other three subroutines. These "combined" netCDF files can be loaded into the McIDAS-V software.

6.2 *Script Breakdown*

1. `run_create_image_from_three_modalities` is the MAIN PROGRAM. Another program that can act as the main program is `run_create_image_from_three_modalities2`. This program runs the netCDF file creations in parallel threads which is useful for archived model runs. The programs import and call all of the following functions. User specifies the input directory (inroot) to the GOES IR/VIS and GLM raw data directories and the function creates the GLM gridded file and loops over all of the GOES IR and VIS files (in alphabetical order.). Optional keywords for GLM data included in netCDF file creation (no_write_glm keyword). The default functionality is to not write or plot the GLM data (saves time and

space and not needed for IR/VIS sandwich images). Optional keywords for VIS data included in netCDF file creation (no_write_vis keyword). Setting the keyword to True allows the job to be run on the native IR resolution which is lower than VIS. Ultimately, this saves time and space. The default functionality is to not write or plot the VIS data if VIS data are not required.

2. `glm_gridder` grids and reformats the GLM lightning flash data and writes output to a netCDF file. Grids ONLY files within ±2.5 minutes of GOES IR/VIS start scan time as default. The 'twindow' keyword can specify time window of files surrounding GOES IR/VIS start scan time. Example file ncdump header is provided in Section 13.2.1. This file is read in combine_ir_glm_vis.py.

3. `combine_ir_glm_vis` reads in the three data types and finds the subset of GLM data that matches the IR data. Function also converts the visible radiance to reflectance and then normalizes reflectance by the solar zenith angle (at every lat/lon pixel), IR radiance to brightness temperature, and calculates the latitude and longitude of the selected viewpoint. The latitude and longitude are then written to a netCDF file along with the data from the three modalities. IR data are written in terms of brightness temperature, VIS data are written in terms of reflectance normalized by the solar zenith angle, solar zenith angle in degrees, and GLM data are written in terms of flash extent density. The ncdump header information of an example file is provided in Section 13.2.2. The user cam choose whether or not they want to include GLM data or VIS data in the output netCDF file.

4. `glm_data_processing` contains utility functions that are used to grid/project the GOES data and convert it into desired units.

5. `img_from_three_modalities2` creates a composite PNG image from the combined netCDF file. Cartopy is used to map the IR/VIS data to a lat/lon grid. If VIS data are not available or the zenith angle in the middle of the image scene exceeds 85°, only the IR data are plotted. VIS data are plotted using greyscale while IR data are plotted using a rainbow color table. A colorblind version of the rainbow color table created using Google AI and named turbo_colormap is also available and is set as the function's default. Figures 1 and 2

provide an example on IR/VIS sandwich image and the color bar, respectively.

6.3 *Path Setup*

The default file path setup is listed below. The keywords in run_create_image_from_three_modalities will do most of the work and decide what directory to send the images based upon the keywords that are set.

1. GLM, IR, and VIS input files = '../../../data/goes-data/20200513/'. The files for each GOES file type is within subdirectories 'glm/', 'ir/', and 'vis/', respectively.

2. GLM outfile directory = '../../../data/goes-data/out_dir/'. Location of re-gridded GLM data file.

3. Combined IR, GLM, and VIS directory file directory = '../../../data/goes-data/combined_nc_dir/'.

4. Image output root directory = '../../../data/goes-data/aacp_results_imgs/' when the model is run even though the default output is '../../../data/goes-data/layered_img_dir/'. Specifying certain keywords will add qualifier subdirectories to output image file path. meso_sector is one such keyword. meso_sector is a LONG integer specifying the mesoscale domain sector to use to create maps (= 1 or 2). DEFAULT = 2 (sector 2).

6.4 *Removed Data*

IR BT $<$ 163 K or NaN are set to 500 K. This removes incomplete or invalid data and makes sure that those points will not be used by the model. Bad pixels will be sporadic, often on domain edges or in a bad scan line, and not possessing a spatial configuration that should look like an OT or AACP to a deep learning model. Often times such bad pixels/lines have anomalously cold temperatures. With that said, it would probably be best to have it register as a 0 weight in the scaled brightness temperature so that it is disregarded.

ONLY GLM flash extent density within $\pm$2.5 minutes of the GOES IR/VIS scan are used and written to the combined netCDF files.

Visible data are not mapped in output images if the solar zenith angle at any point of the domain exceeds 85°. In this case, ONLY IR data are mapped. This prevents retention of visible data with

artificially high reflectance values as a result of solar zenith angle transitioning from daytime to nighttime.

7 *Creating Numpy Arrays for Model Input*

7.1 *Overview*

Numpy arrays for IR, VIS, and GLM are used as model inputs into the the machine learning algorithm. Numpy arrays are created for a single time. All of the numpy arrays contain normalized values ranging between 0 and 1. The scripts necessary to create a numpy array has 2 parts: 1) Create model input variable numpy arrays and 2) Create corresponding csv file that gives how the numpy files are organized. The script used is create_vis_ir_numpy_arrays_from_netcdf_files2.py.

7.2 *Script Breakdown*

This first list shows the script breakdown for creating numpy mask files.

1. `run_write_severe_storm_post_processing` is the MAIN function that appends the output netCDF files. This function calculates brightness temperature differences and extracts object ID numbers.

2. `append_combined_ncdf_with_model_post_processing` is a function that is called by run_write_severe_storm_post_processing that opens the model output netCDF file and appends it with the post-processed data variables.

3. `download_gfs_analysis_files` is a function that uses the request python module to download GFS GRIB analysis files from NCAR.

4. `gfs_nearest_time` is a function that calculate the date and time of the GFS analysis or forecast file nearest to 'date'.

5. `gfs_interpolate_gridrad_tropT_to_goes_grid` is a function that interpolates tropopause temperatures from GFS onto a satellite data grid.

8 *Post-Processing Model Results*

8.1 *Overview*

One model runs are complete, the software now provides post-processed data into the output netCDF files. The post-processed products include object identification numbers, the brightness temperature difference between the anvil and the OT, as well as the GFS tropopause temperature.

8.2 *Post-Processing Variables and Calculations*

8.2.1 *Object Identification Numbers*

There are 2 possible object types that the User can specify to the software, OT and AACP. In the configuration file, the User has the option to specify the likelihood threshold in which they would like the data to be post-processed around. The User can also not specify this or choose to use the optimal threshold found during testing (see Table 2 for the optimal thresholds for each model found during testing). The dataset is first cleaned by removing all pixels with output likelihood values less than 0.05. Next, the software uses scipy.ndimage.label to identify individual objects. Objects that contain a pixel that has a likelihood score that exceed the User specified likelihood threshold (or if not specified then the optimal threshold identified in our testing) move on to determine which pixels should be included as part of the object. For AACPs (OTs), the software retains all pixels that are at least 10% (50%) of the value of the maximum likelihood score in the object. Each pixel that satisfies this condition within the object is given the same ID number. Thus, the object Identification Number field shows all pixels that belong to an individual object region. The ID numbers apply uniquely to each satellite scan, i.e. ID number 1 in one scan will likely not be the same feature as ID number 1 in the next scan, and therefore cannot be used to track an object throughout its lifetime.

8.2.2 *IR OT-Anvil Brightness Temperature Difference*

IR OT-Anvil Brightness Temperature Difference (BTD) is calculated for OTs only. For each OT object ID, the software identifies the coldest BT in the object. Then, the software searches for anvil pixels within a 30x30 km domain surrounding the coldest pixel in the OT. A pixel is considered

to be anvil if it is not part of any OT object. The software the omits the coldest and warmest X% when calculating the mean anvil brightness temperature. The default for X is 20% which means that 20% of the warmest and coldest anvil pixel temperatures are removed in order to prevent the contribution of noise to the OT IR-anvil BTD calculation. This value can be specified by the User in the configuration file. The mean brightness temperature of the remaining anvil pixels is calculated and subtracted from the minimum BT in the OT in order to get IR OT-Anvil BTD.

8.2.3 *Tropopause Temperature*

Tropopause temperatures are calculated from GFS analysis files. GFS analysis files are downloaded from the NCAR data archive and will automatically download analysis times that encompass the date range that the model was run over, whether it was a real-time or archived model run. GFS tropopause temperatures are interpolated onto the satellite grid using scipy.interpn and then smoothed using a Gaussian filter. Tropopause temperatures are only included in the output netCDF files for OT object model runs.

8.3 *Script Breakdown*

This first list shows the script breakdown for appending the output netCDF files with post-processed data.

1. `create_vis_ir_numpy_arrays_from_netcdf_files2` is the MAIN function that creates IR, VIS, and GLM numpy files for a specific date and scan sector. The IR and GLM data are both interpolated onto the VIS data grid, if they were not already. This function uses the combined netCDF files (created from combine_ir_glm_vis) to normalize the IR, VIS, and GLM which is then output to numpy arrays. The numpy arrays are stored as with dimensions (1, number of latitude pixels, number of longitude pixels).

2. `fetch_convert_vis` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_files2 and normalizes the reflectance data to range from 0 to 1. This function returns the normalized reflectance array and whether the GOES scan is considered night or day (if middle of image zenith angle $> 85^{\circ}$ yields night). Middle of image solar zenith angle has been switched (effective February 3, 2022) such that if the zenith angle is $> 85^{\circ}$ anywhere in the image then

it is considered night. This was done in order to avoid poor detections in high solar zenith angle regions.

3. `fetch_convert_ir` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_files2 and normalizes the brightness temperatures by user specified minimum and maximum values. The default min and max values are 195 K and 225 K, respectively. Brightness temperature measurements closer to the min value are nearer to 1 when normalized. IMPORTANT NOTE!!! IR data values are set to -1 in regions that are not able to be scanned by the satellite. These regions come about due to the gridding process for CONUS and FULL disk scans. They are set to -1 in order to distinguish that there should never be any model detection at those points. When running the model, we can easily find those points and make sure the results likelihoods are set to 0 (not an OT or AACP).

4. `fetch_convert_glm` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_files2 and normalizes flash extent density data by user specified min and max values. The default min and max values are 0 and 20, respectively.

5. `merge_csv_files` is a function that merges ALL csv files created from the previous steps. Output is one big, combined csv file containing label csv files and IR/VIS/GLM files with dates and times.

9 *Google Cloud Platform (GCP)*

9.1 *Overview*

IMPORTANT NOTE: Google Cloud is not essential for this project. All of the software can be run without installing Google Cloud, however, users that want to run the model in real-time would be advised to install Google Cloud Services such that, data can be downloaded from the cloud.

Raw Level 1 GOES files are downloaded from Google public storage buckets and we often store files that are used as input into the model on our own storage buckets which can be accessed by Google's Virtual Machines (VMs). This project ran the model training, validation, and prediction on Google's VMs. This section will discuss the storage buckets in which the data are stored, the VMs, and code used to writing and reading from storage buckets. All of the scripts, made by John

Cooney, are capable of reading and writing the data locally as well as to and from the Google Cloud Storage Buckets. If the following subsections are not helpful, refer to the code for some examples on how to read and write from the GCP buckets.

9.2 *Installing Google Cloud*

To install Google Cloud, follow the directions on Google Cloud Install. Download the package for your operating system and then run the install script (from the root of the folder you extracted in the last step) as described on the website (ex. .'/google-cloud-sdk/install.sh'). Once complete, your .bash_profile or .zshrc whatever you are using should look similar to the following.

```
# The next line updates PATH for the Google Cloud SDK.
if [ -f '/Users/user-name/google-cloud-sdk/path.bash.inc' ]; then . '/Users/user-name/google-cloud-sdk/path.bash.inc'; fi

# The next line enables shell command completion for gcloud.
if [ -f '/Users/user-name/google-cloud-sdk/completion.bash.inc' ]; then . '/Users/user-name/google-cloud-sdk/completion.bash.inc'; fi
```

This should allow you to use gsutil commands. For downloading, you may need to run *gcloud iam service-accounts keys create /Users/user-name/google-cloud-sdk/google-service-account-file.json –iam-account=project-name@appspot.gserviceaccount.com* in your terminal in order to get necessary json file. Put this file in a location that you want and then export it in .zshrc or .bash_profile by adding a line similar to:

```
export GOOGLE_APPLICATION_CREDENTIALS=/Users/user-name/google-cloud-sdk/google-service-account-file.json
```

9.3 *Script Breakdown*

All of the code for reading and writing to and from the Google Cloud is located within the script *gcs_processing.py*. There are multiple functions that will list the files within a particular google storage bucket, read a file on the storage bucket, download a file from a storage bucket, and write a file to a storage bucket. The functions are capable of handling csv files, numpy files, image files, netCDF files, and model checkpoint files. Choose the function that best suits your needs.

NOTE, netCDF and model checkpoint files must be available locally in order to be read. Thus, we must download them from a Google Cloud storage bucket. Other file types can just be loaded into memory directly from the Google Cloud storage buckets so they do not need to be downloaded first.

10 *Software Updates*

10.1 *MAJOR REVISIONS*

10.1.1 *05 June 2023*

Added post-processing function that yields object ID numbers of OTs and AACPs as well as OT IR-anvil brightness temperature differences.

Fixed issue that occurred in full disk and CONUS scanned model runs. The issue occurred at the very edge of the domain where the satellite view is off in to space on the edge of Earth. This issue was fixed prior to release for OTs but AACPs needed to be fixed in a unique way.

10.1.2 *22 June 2023*

Added tropopause temperature to post-processing functionality for OT model runs. This software downloads GFS data from NCAR and then interpolates and smooths the tropopause temperature onto the satellite data grid before writing the output to the netCDF files.

10.2 *MINOR REVISIONS*

10.2.1 *13 June 2023*

Fixed issue where user wants to correct the model inputs or model type and not everything like the correct optimal model to be called followed with it.

Pass optimal_threshold for a given model to be written to output netCDF file rather than whatever is chosen by user for post-processing and plotting. The one chosen by the user is still passed into the subroutines for plotting and post-processing and is referred to as the likelihood_threshold in order to differentiate it from the optimal_thresh attribute in the raw model likelihood output variables. Users can identify objects in post-processing by specifying their own likelihood thresholds, rather than being forced to use the optimal thresholds found for a particular model.

Decrease size of files by improving compression by setting least_significant_digits keyword when creating the variables.

11 *Authors*

This software was produced through collaborative work between John W. Cooney, Kristopher M. Bedka, and Charles A. Liles at NASA Langley Research Center.

11.1 *Contact Information*

This software is intended for research and operational use. Users can contact the software team regarding its use, especially before publication or public presentation. This is the first official release of this software; these products that are still undergoing validation, testing, quality control, and debugging. Users are invited to address questions and provide feedback to the contacts below.

1. **John Cooney:** john.w.cooney@nasa.gov
2. **Kristopher Bedka:** kristopher.m.bedka@nasa.gov

12 *Disclaimer and Copyright Notices*

12.1 *Notices*

12.2 *Disclaimers*

RESULTING FROM USE OF THE SUBJECT SOFTWARE. FURTHER, GOVERNMENT AGENCY DISCLAIMS ALL WARRANTIES AND LIABILITIES REGARDING THIRD-PARTY SOFTWARE, IF PRESENT IN THE ORIGINAL SOFTWARE, AND DISTRIBUTES IT ”AS IS.”

12.3 *Waiver and Indemnity*

RECIPIENT AGREES TO WAIVE ANY AND ALL CLAIMS AGAINST THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND SUBCONTRACTORS, AS WELL AS ANY PRIOR RECIPIENT. IF RECIPIENT’S USE OF THE SUBJECT SOFTWARE RESULTS IN ANY LIABILITIES, DEMANDS, DAMAGES, EXPENSES OR LOSSES ARISING FROM SUCH USE, INCLUDING ANY DAMAGES FROM PRODUCTS BASED ON, OR RESULTING FROM, RECIPIENT’S USE OF THE SUBJECT SOFTWARE, RECIPIENT SHALL INDEMNIFY AND HOLD HARMLESS THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND SUBCONTRACTORS, AS WELL AS ANY PRIOR RECIPIENT, TO THE EXTENT PERMITTED BY LAW. RECIPIENT’S SOLE REMEDY FOR ANY SUCH MATTER SHALL BE THE IMMEDIATE, UNILATERAL TERMINATION OF THIS AGREEMENT.

13 *Appendix*

13.1 *Anaconda Installation*

This following anaconda installation steps are primarily useful for Mac users. Please see Windows Anaconda Install for better steps on how to install anaconda using Windows. Note that the steps refer to the anaconda packages, NOT Anaconda-Navigator.app.

1. Go to: Anaconda Install Website
2. Click download. (This will take you to bottom of web page)
3. Install the 64-Bit Graphical Installer for your system
4. Locate your download and double click it
5. NOTE: when you install Anaconda, it modifies your .bash_profile (this can be important for later)

6. Agree to License Agreement and install anaconda

7. Once complete, close the installer and move it to the trash

8. Activate conda by going to the command line within terminal source <path toconda>/bin/activate. The <path toconda> can change. The default location for Anaconda is:

   ```
   Linux = /home/<your_username>/Anaconda3
   Windows = C:\Users\<your_username>\Anaconda3
   Mac = /Users/<your_username>/Anaconda3
   ```

   For Mac, if anaconda was installed at the system level, the anaconda package may be located in your /opt directory. If you cannot find the package and, specifically, the anaconda3 directory + bin subdirectory folder then go into your finder and search for anaconda. This should give you the location.

9. In terminal enter *conda init* (this may or may not yield a successful output message).

13.2 *netCDF File Contents*

13.2.1 *GLM Gridder netCDF File*

Example GLM gridder file is shown below.

```
netcdf 20191201800333_gridded_data {
dimensions:
x = 2500 ;
y = 1500 ;
dim_0 = 1 ;
variables:
short x(x) ;
x:_FillValue = -999s ;
x:axis = "X" ;
x:long_name = "GOES fixed grid projection x-coordinate" ;
x:standard_name = "projection_x_coordinate" ;
x:units = "rad" ;
x:add_offset = -0.151844 ;
x:scale_factor = 5.6e-05 ;
short y(y) ;
y:_FillValue = -999s ;
y:axis = "Y" ;
y:long_name = "GOES fixed grid projection y-coordinate" ;
y:standard_name = "projection_y_coordinate" ;
y:units = "rad" ;
y:add_offset = 0.151844 ;
y:scale_factor = -5.6e-05 ;
int goes_imager_projection ;
```

```
goes_imager_projection:long_name = "GOES-R ABI fixed grid projection" ;
goes_imager_projection:grid_mapping_name = "geostationary" ;
goes_imager_projection:perspective_point_height = 35786023. ;
goes_imager_projection:semi_major_axis = 6378137. ;
goes_imager_projection:semi_minor_axis = 6356752.31414 ;
goes_imager_projection:inverse_flattening = 298.2572221 ;
goes_imager_projection:latitude_of_projection_origin = 0. ;
goes_imager_projection:longitude_of_projection_origin = -75. ;
goes_imager_projection:sweep_angle_axis = "x" ;
byte DQF(y, x) ;
DQF:_FillValue = -1b ;
DQF:grid_mapping = "goes_imager_projection" ;
DQF:number_of_qf_values = 6 ;
DQF:units = "1" ;
DQF:standard_name = "status_flag" ;
DQF:long_name = "GLM data quality flags" ;
DQF:flag_values = 0, 1 ;
DQF:flag_meanings = "valid, invalid" ;
DQF:_Unsigned = "true" ;
double nominal_satellite_subpoint_lat ;
nominal_satellite_subpoint_lat:_FillValue = -999. ;
nominal_satellite_subpoint_lat:long_name = "nominal satellite subpoint latitude (platform latitude)" ;
nominal_satellite_subpoint_lat:standard_name = "latitude" ;
nominal_satellite_subpoint_lat:units = "degrees_north" ;
double nominal_satellite_subpoint_lon(dim_0) ;
nominal_satellite_subpoint_lon:_FillValue = -999. ;
nominal_satellite_subpoint_lon:long_name = "nominal satellite subpoint longitude (platform longitude)" ;
nominal_satellite_subpoint_lon:standard_name = "longitude" ;
nominal_satellite_subpoint_lon:units = "degrees_east" ;
short flash_extent_density(y, x) ;
flash_extent_density:_FillValue = 0s ;
flash_extent_density:standard_name = "flash_extent_density" ;
flash_extent_density:long_name = "Flash extent density" ;
flash_extent_density:units = "Count per nominal    3136 microradian^2 pixel per 1.0 min" ;
flash_extent_density:grid_mapping = "goes_imager_projection" ;
flash_extent_density:add_offset = 0. ;
flash_extent_density:scale_factor = 0.0625 ;
flash_extent_density:_Unsigned = "true" ;
short flash_centroid_density(y, x) ;
flash_centroid_density:_FillValue = 0s ;
flash_centroid_density:standard_name = "flash_centroid_density" ;
flash_centroid_density:long_name = "Flash centroid density" ;
flash_centroid_density:units = "Count per nominal    3136 microradian^2 pixel per 1.0 min" ;
flash_centroid_density:grid_mapping = "goes_imager_projection" ;
flash_centroid_density:add_offset = 0. ;
flash_centroid_density:scale_factor = 1. ;
flash_centroid_density:_Unsigned = "true" ;
short average_flash_area(y, x) ;
average_flash_area:_FillValue = 0s ;
average_flash_area:standard_name = "average_flash_area" ;
average_flash_area:long_name = "Average flash area" ;
average_flash_area:units = "km^2 per flash" ;
average_flash_area:grid_mapping = "goes_imager_projection" ;
average_flash_area:add_offset = 0. ;
average_flash_area:scale_factor = 10. ;
average_flash_area:_Unsigned = "true" ;
short total_energy(y, x) ;
total_energy:_FillValue = 0s ;
total_energy:standard_name = "total_energy" ;
total_energy:long_name = "Total radiant energy" ;
total_energy:units = "nJ" ;
```

```
total_energy:grid_mapping = "goes_imager_projection" ;
total_energy:add_offset = 0. ;
total_energy:scale_factor = 1.52597e-06 ;
total_energy:_Unsigned = "true" ;
short group_extent_density(y, x) ;
group_extent_density:_FillValue = 0s ;
group_extent_density:standard_name = "group_extent_density" ;
group_extent_density:long_name = "Group extent density" ;
group_extent_density:units = "Count per nominal    3136 microradian^2 pixel per 1.0 min" ;
group_extent_density:grid_mapping = "goes_imager_projection" ;
group_extent_density:add_offset = 0. ;
group_extent_density:scale_factor = 0.25 ;
group_extent_density:_Unsigned = "true" ;
short group_centroid_density(y, x) ;
group_centroid_density:_FillValue = 0s ;
group_centroid_density:standard_name = "group_centroid_density" ;
group_centroid_density:long_name = "Group centroid density" ;
group_centroid_density:units = "Count per nominal    3136 microradian^2 pixel per 1.0 min" ;
group_centroid_density:grid_mapping = "goes_imager_projection" ;
group_centroid_density:add_offset = 0. ;
group_centroid_density:scale_factor = 1. ;
group_centroid_density:_Unsigned = "true" ;
short average_group_area(y, x) ;
average_group_area:_FillValue = 0s ;
average_group_area:standard_name = "average_group_area" ;
average_group_area:long_name = "Average group area" ;
average_group_area:units = "km^2 per group" ;
average_group_area:grid_mapping = "goes_imager_projection" ;
average_group_area:add_offset = 0. ;
average_group_area:scale_factor = 1. ;
average_group_area:_Unsigned = "true" ;
short minimum_flash_area(y, x) ;
minimum_flash_area:_FillValue = 0s ;
minimum_flash_area:standard_name = "minimum_flash_area" ;
minimum_flash_area:long_name = "Minimum flash area" ;
minimum_flash_area:units = "km^2" ;
minimum_flash_area:grid_mapping = "goes_imager_projection" ;
minimum_flash_area:add_offset = 0. ;
minimum_flash_area:scale_factor = 10. ;
minimum_flash_area:_Unsigned = "true" ;

// global attributes:
:cdm_data_type = "Image" ;
:Conventions = "CF-1.7" ;
:id = "93cb84a3-31ef-4823-89f5-c09d88fc89e8" ;
:institution = "DOC/NOAA/NESDIS > U.S. Department of Commerce, National Oceanic and Atmospheric Administration, National Environmental Satellite, Data,
:instrument_type = "GOES R Series Geostationary Lightning Mapper" ;
:iso_series_metadata_id = "f5816f53-fd6d-11e3-a3ac-0800200c9a66" ;
:keywords = "ATMOSPHERE > ATMOSPHERIC ELECTRICITY > LIGHTNING, ATMOSPHERE > ATMOSPHERIC PHENOMENA > LIGHTNING" ;
:keywords_vocabulary = "NASA Global Change Master Directory (GCMD) Earth Science Keywords, Version 7.0.0.0.0" ;
:license = "Unclassified data.  Access is restricted to approved users only." ;
:Metadata_Conventions = "Unidata Dataset Discovery v1.0" ;
:naming_authority = "gov.nesdis.noaa" ;
:processing_level = "National Aeronautics and Space Administration (NASA) L2" ;
:project = "GOES" ;
:standard_name_vocabulary = "CF Standard Name Table (v25, 05 July 2013)" ;
:summary = "The Lightning Detection Gridded product generates fields starting from the GLM Lightning Detection Events, Groups, Flashes product.  It cor
:title = "GLM L2 Lightning Detection Gridded Product" ;
:dataset_name = "OR_GLM-L2-GLMC-M3_G16_s20191201801400_e20191201802400_c20203501353540.nc" ;
:date_created = "2020-12-15T13:53:54.935958Z" ;
:instrument_ID = "FM1" ;
```

```
:orbital_slot = "GOES-East" ;
:platform_ID = "G16" ;
:production_data_source = "Postprocessed" ;
:production_environment = "DE" ;
:production_site = "TTU" ;
:scene_id = "CONUS" ;
:spatial_resolution = "2km at nadir" ;
:time_coverage_end = "2019-04-30T18:02:40Z" ;
:time_coverage_start = "2019-04-30T18:01:40Z" ;
:timeline_id = "ABI Mode 3" ;
}
```

Size of file depends on how many GLM files are used to create this gridded file. A single gridded file is created for all of the raw GLM data files in specified directory. Creating the gridded GLM files also takes a good chunk of time to do so it might be beneficial for the user to create these files prior to running main script. As long as the files are named using a similar convention to that in the main program, there should not be any issue reading it in.

13.2.2 *Combine IR, VIS, and GLM Data netCDF File*

Example IR, VIS, and GLM combined netCDF file is shown below.

```
netcdf OR_ABI_L1b_M2_COMBINED_s20231692357560_e20231692358033_c20231692358050 {
dimensions:
Y = 2000 ;
X = 2000 ;
time = 1 ;
variables:
float longitude(Y, X) ;
longitude:least_significant_digit = 3LL ;
longitude:long_name = "longitude -180 to 180 degrees east" ;
longitude:standard_name = "longitude" ;
longitude:units = "degrees_east" ;
float latitude(Y, X) ;
latitude:least_significant_digit = 3LL ;
latitude:long_name = "latitude -90 to 90 degrees north" ;
latitude:standard_name = "latitude" ;
latitude:units = "degrees_north" ;
float time(time) ;
time:long_name = "J2000 epoch mid-point between the start and end image scan in seconds" ;
time:standard_name = "time" ;
time:units = "seconds since 2000-01-01 12:00:00" ;
int visible_reflectance(time, Y, X) ;
visible_reflectance:_FillValue = -2147483648 ;
visible_reflectance:long_name = "Visible Reflectance Normalized by Solar Zenith Angle" ;
visible_reflectance:standard_name = "Visible_Reflectance" ;
visible_reflectance:units = "reflectance_normalized_by_solar_zenith_angle" ;
visible_reflectance:coordinates = "longitude latitude time" ;
visible_reflectance:add_offset = 1.009126f ;
visible_reflectance:scale_factor = 4.614115e-10f ;
int solar_zenith_angle(time, Y, X) ;
solar_zenith_angle:_FillValue = -2147483648 ;
solar_zenith_angle:least_significant_digit = 2LL ;
solar_zenith_angle:long_name = "Solar Zenith Angle" ;
```

```
solar_zenith_angle:standard_name = "solar_zenith_angle" ;
solar_zenith_angle:units = "degrees" ;
solar_zenith_angle:add_offset = 76.50061f ;
solar_zenith_angle:scale_factor = 3.591184e-09f ;
int imager_projection ;
imager_projection:long_name = "GOES-R ABI fixed grid projection" ;
imager_projection:satellite_name = "G16" ;
imager_projection:grid_mapping_name = "geostationary" ;
imager_projection:perspective_point_height = 35786023. ;
imager_projection:semi_major_axis = 6378137. ;
imager_projection:semi_minor_axis = 6356752.31414 ;
imager_projection:inverse_flattening = 298.2572221 ;
imager_projection:latitude_of_projection_origin = 0. ;
imager_projection:longitude_of_projection_origin = -75. ;
imager_projection:sweep_angle_axis = "x" ;
imager_projection:bounds = "-1923606.125,-922098.4375,2665593.5,3667101.25" ;
imager_projection:bounds_units = "m" ;
int ir_brightness_temperature(time, Y, X) ;
ir_brightness_temperature:_FillValue = -2147483648 ;
ir_brightness_temperature:long_name = "Infrared Brightness Temperature Image resampled onto VIS data grid" ;
ir_brightness_temperature:standard_name = "IR_brightness_temperature" ;
ir_brightness_temperature:units = "kelvin" ;
ir_brightness_temperature:coordinates = "longitude latitude time" ;
ir_brightness_temperature:add_offset = 254.1493f ;
ir_brightness_temperature:scale_factor = 2.284545e-08f ;
int glm_flash_extent_density(time, Y, X) ;
glm_flash_extent_density:_FillValue = -2147483648 ;
glm_flash_extent_density:least_significant_digit = 4LL ;
glm_flash_extent_density:long_name = "Flash extent density within +/- 2.5 min of time variable resampled onto VIS data grid and then smoothed using Gau
glm_flash_extent_density:standard_name = "Flash_extent_density" ;
glm_flash_extent_density:units = "Count per nominal 3136 microradian^2 pixel per 1.0 min" ;
glm_flash_extent_density:coordinates = "longitude latitude time" ;
glm_flash_extent_density:add_offset = 4. ;
glm_flash_extent_density:scale_factor = 1.86264515009832e-09 ;
float ir_vis_glm_ot(time, Y, X) ;
ir_vis_glm_ot:standard_name = "IR+VIS+GLM_OT_Model_Results" ;
ir_vis_glm_ot:least_significant_digit = 3LL ;
ir_vis_glm_ot:optimal_thresh = 0.4 ;
ir_vis_glm_ot:missing_value = 0.f ;
ir_vis_glm_ot:units = "dimensionless" ;
ir_vis_glm_ot:coordinates = "longitude latitude time" ;
ir_vis_glm_ot:valid_range = 0.f, 1.f ;
ir_vis_glm_ot:checkpoint_file = "/Users/jwcooney/python/code/aacp/data/model_checkpoints/ir_vis_glm/updraft_day_model/2022-02-18/multiresunet/chosen_ir
ir_vis_glm_ot:model_type = "Multiresunet" ;
ir_vis_glm_ot:long_name = "IR+VIS+GLM OT Multiresunet Machine Learning Detection Likelihood" ;
ushort ir_vis_glm_ot_id_number(time, Y, X) ;
ir_vis_glm_ot_id_number:description = "The object Identification Number field shows all pixels that belong to an individual object region.  The ID numb
ir_vis_glm_ot_id_number:standard_name = "IR+VIS+GLM OT ID Number" ;
ir_vis_glm_ot_id_number:missing_value = 0US ;
ir_vis_glm_ot_id_number:units = "dimensionless" ;
ir_vis_glm_ot_id_number:likelihood_threshold = 0.4 ;
ir_vis_glm_ot_id_number:coordinates = "longitude latitude time" ;
ir_vis_glm_ot_id_number:model_type = "Multiresunet" ;
ir_vis_glm_ot_id_number:long_name = "IR+VIS+GLM OT Identification Number" ;
float ir_vis_glm_ot_anvilmean_brightness_temperature_difference(time, Y, X) ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:description = "Minimum brightness temperature within an OT Minus Anvil Brightness Temperature
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:standard_name = "IR+VIS+GLM OT - Anvil BTD" ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:least_significant_digit = 2LL ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:missing_value = NaNf ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:units = "K" ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:likelihood_threshold = 0.4 ;
```

```
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:coordinates = "longitude latitude time" ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:valid_range = -50.f, 0.f ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:model_type = "Multiresunet" ;
ir_vis_glm_ot_anvilmean_brightness_temperature_difference:long_name = "IR+VIS+GLM Overshooting Top Minus Anvil Brightness Temperature Difference" ;
float tropopause_temperature(time, Y, X) ;
tropopause_temperature:standard_name = "GFS Tropopause" ;
tropopause_temperature:least_significant_digit = 2LL ;
tropopause_temperature:missing_value = NaNf ;
tropopause_temperature:units = "K" ;
tropopause_temperature:coordinates = "longitude latitude time" ;
tropopause_temperature:valid_range = 160.f, 310.f ;
tropopause_temperature:long_name = "Temperature of the Tropopause Retrieved from GFS Interpolated and Smoothed onto Satellite Grid" ;

// global attributes:
:Conventions = "CF-1.8" ;
string :description = "This file combines unscaled IR data, Visible data, and GLM data into one NetCDF file. The VIS data is on its original grid but i
:geospatial_lat_min = "25.43004" ;
:geospatial_lat_max = "37.462128" ;
:geospatial_lon_min = "-98.60013" ;
:geospatial_lon_max = "-84.41396" ;
:x_inds = "" ;
:y_inds = "" ;
:spatial_resolution = "0.5km at nadir" ;
}
```

These files are each up to $\sim$ 45 MB. A single file is created for each IR/VIS file. The file above scales the data to save space as well as resamples the GLM and IR data to the VIS data grid. The GLM data are smoothed using a Gaussian smoother after resampling the GLM data but prior to writing the netCDF file.

13.3 *GOES Storage Bucket Download Examples*

A few examples of how the GOES-16 IR, VIS, and GLM data can be downloaded. For these examples, the data are downloaded to another Google Cloud Storage Bucket named, 'goes-data'. The data can also be downloaded to local directories as well.

```
gsutil -m cp -r gs://gcp-public-data-goes-16/GLM-L2-LCFA/2019/125/1[8-9]/**** gs://goes-data/20190505-06/glm

gsutil -m cp -r gs://gcp-public-data-goes-16/ABI-L1b-RadM/2019/128/0[0-3]/*C13_G16_* gs://goes-data/20190507-08/ir

gsutil -m cp -r gs://gcp-public-data-goes-16/ABI-L1b-RadM/2019/128/0[0-3]/*C02_G16_* gs://goes-data/20190507-08/vis
```

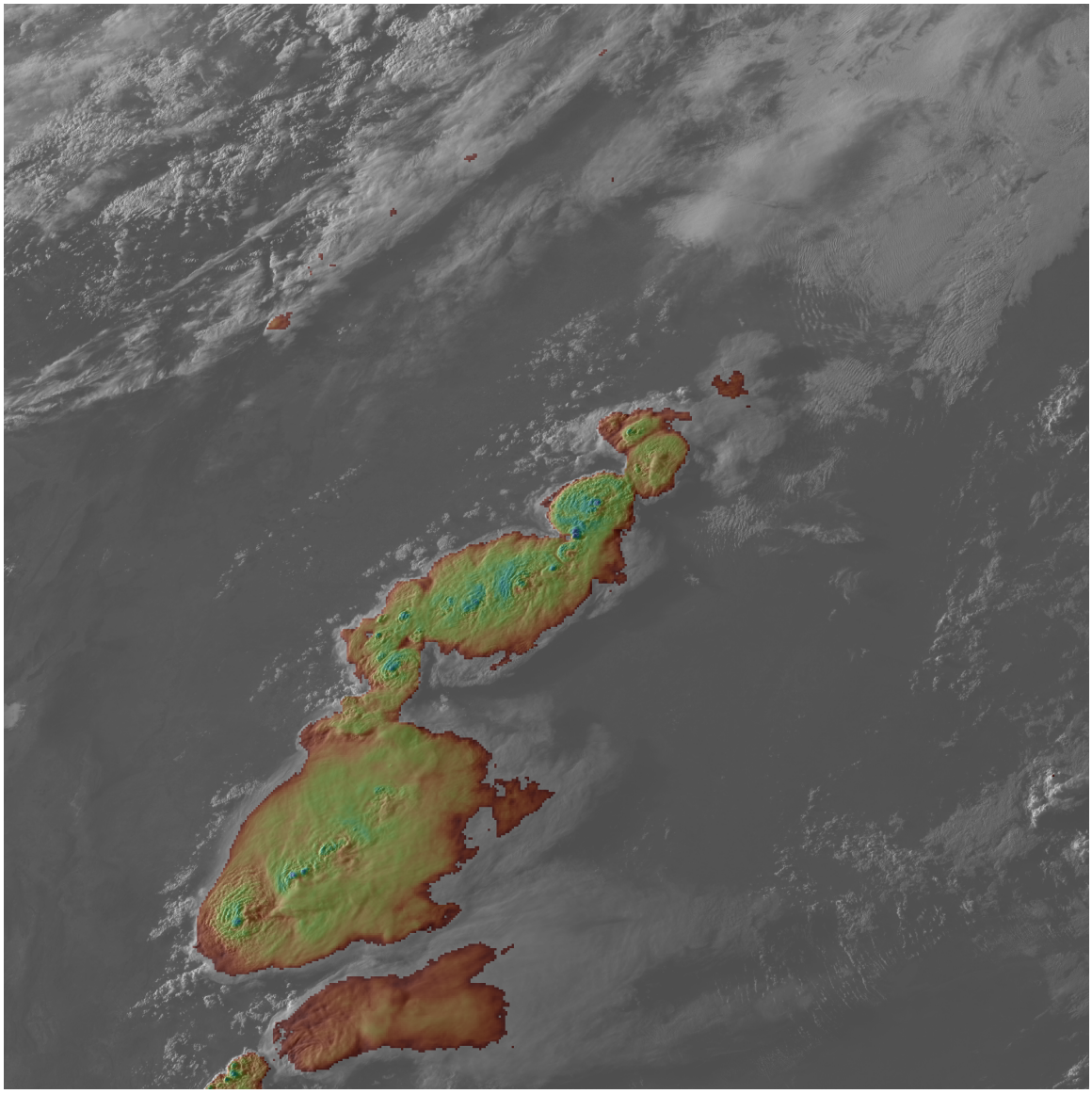

Figure 1: Example IR/VIS image for combined netCDF file: OR_ABI_L1b_M2_COMBINED_s20201350016495_e20201350016564_c20201350017016.

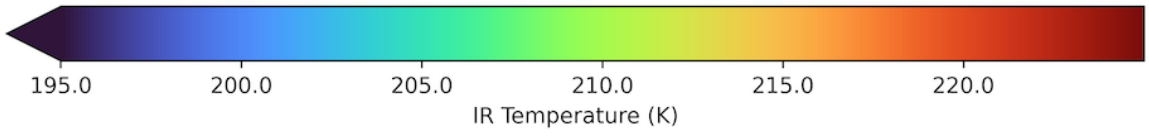

Figure 2: IR color bar.