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

John W. Cooney

March 5, 2024

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. The software was setup using Mac/linux OS but should work using Windows OS as well. 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.

VERSION 2 of software released on 6 March 2024!!! LARGE model performance enhancement, particularly with AACP detection. Added A LOT of new model input combinations as possibilities, including TROPDIFF, DIRTYIRDIFF, SNOWICE, and CIRRUS. Added the new and improved checkpoint files associated with those model runs. Optimal models and thresholds have been updated since the Version 1 release. Changed how we interpolate the GFS data onto the GOES grid. We now follow the methods outlined in Khlopenkov et al. (2021). New version changes how software decides between daytime and nighttime model runs. Previously used the maximum solar zenith angle within the domain and checked if that exceeded 85°, however, there was an issue of only nighttime models being used for CONUS domains due to how the satellite views the Earth and the data are gridded. Thus, now the software checks to see if more than 5% of the pixels in the domain have solar zenith angles that exceed 85°. If they do, then the software acts as if the domain is nighttime and if not the software acts as if it is daytime within the domain. Fixed major issue with looping over post-processing dates and the month or year changed during a real-time or archived model run. New version speeds up post-processing for OTs. Set variables to None after using them in order to clear cached memory. Catch and hide RunTime warnings that the User does not need to worry about.

Software Users prior to 6 March 2024 will need to pull the latest python files from GitHub. This should be very fast. Users MUST also download the new checkpoint files from 'https://science-

data.larc.nasa.gov/LaRC-SD-Publications/2023-05-05-001-JWC/data/ML_data.zip'. Once downloaded and unzipped, replace the old model_checkpoints subdirectory with the latest one that was just downloaded. This directory includes all of the improved model detection files as well as checkpoint files for the new input combinations.

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.

2.1 *Mac/Linux/Unix Users*

There are 11 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

11. Refer to Section 9.2 for setting up Google Cloud to download data files from there. This service is free and only requires users have a Gmail account. If you do not wish to download data from Google Cloud, see 5.2.2 for more information regarding where to obtain the necessary data.

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.

2.2 *Windows Users*

Note, one method Windows users could do is install a Linux Virtual environment and run software there. If that is not what you want to do, the software should work on Windows devices.

IMPORTANT NOTE: Windows users must have 64-bit installed!!! Tensorflow, a main component of machine learning, will not work otherwise.

I apologize in advance if this section is not written as well as others. I am not a Windows user and so I am not familiar with all of the nuances available to Windows operators. These are the 13 steps I found to work for installing the software on Windows. You may have a method that works more efficiently. If so, please email me so that I can better assist Windows users. 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 the Anaconda Powershell Prompt.

2. cd svrstormsig

3. git clone https://github.com/deeplycloudy/glmtools.git
4. cd glmtools
5. cp ../windows_environment.yml ./environment.yml
6. cp ../setup.py .
7. conda env create -f environment.yml
8. conda activate svrstormsig
9. pip install -e .
10. pip install tensorflow==2.7
11. conda install -c conda-forge pygrib
12. pip install opencv-python-headless
13. Refer to Section 9.2 for setting up Google Cloud to download data files from there. Follow steps 1-15 outlined. For step 16, I am unsure about .bash_profiles or .zshrc on Windows at this time, so instead in the Anaconda Powershell Prompt I found typing this works:

    $env:GOOGLE_APPLICATION_CREDENTIALS='key_path'

    where key_path is the local path and filename to the json key file. This service is free and only requires users have a gmail account. If you do not wish to download data from Google Cloud, see 5.2.2 for more information about where to obtain the necessary data.

2.3 *Software Path Setup*

This section displays what the directory and subdirectory structure of the software should look like after install. If the directory path structure is not setup like this, then you may run into errors when attempting to run the software.

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 3.3 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 *Model Input Data Acronyms*

IR refers to the clean infrared window (10.3μm) brightness temperatures.

VIS refers to the visible reflectance window (0.64μm).

GLM refers to the Geostationary Lightning Mapper flash extent density data.

WVIRDIFF (6.2μm - 10.3μm) refers to the difference between the water vapor window (6.2μm) and the clean IR window brightness temperatures.

TROPDIFF (10.3μm - GFS TropT) refers to the difference between the clean IR window brightness temperature and the GFS tropopause temperature.

SNOWICE (1.6μm) refers to the snow/ice channel.

CIRRUS (1.37μm) refers to the cirrus channel.

DIRTYIRDIFF (12.3μm - 10.3μm) refers to the difference between the dirty IR window (12.3μm) brightness temperatures and the clean IR window brightness temperatures.

3.3 *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 = 2022-08-23 23:00:00


#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 = 2022-08-24 00:02:00


#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/archived 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, goes-17, goes-18)
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.
#See extract_us_lat_lon_region in rdr_sat_utils_jwc.py for examples. User
#can also add their own regions there.
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 and best OT model at night is IR.
#Best AACP model for daytime is IR+DIRTYIRDIFF and best AACP model at night is IR+DIRTYIRDIFF.
optimal_params = y


#11)
#If optimal_params is n or no, what model inputs would you like to use? Keep in mind,
#visible data is not available during night time hours and thus, no model
#results files will be created at night if using VIS, snowice, or cirrus.
#Best OT model for daytime is IR+VIS
#Best OT model at night is IR.
#Best AACP model for daytime is IR+DIRTYIRDIFF
#Best AACP model at night is IR+DIRTYIRDIFF.
#(ex. IR, TROPDIFF, IR+VIS, IR+GLM, IR+WVIRDIFF, IR+VIS+GLM, IR+TROPDIFF,
#IR+CIRRUS, IR+SNOWICE, IR+DIRTYIRDIFF, VIS+TROPDIFF, GLM+TROPDIFF, IR+VIS+TROPDIFF,
#VIS+GLM+TROPDIFF, TROPDIFF+DIRTYIRDIFF, IR+VIS+DIRTYIRDIFF, VIS+TROPDIFF+DIRTYIRDIFF)
#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.
#Currently cannot plot due to licensing issues so this response does not matter.
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
```

3.4 *Optimal Models*

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

3.4.1 *OT Optimal Model From Testing*

During the day, the OT detection model that performed best during our tests was the IR+VIS MultiResUnet model. Table 2 shows the optimal threshold for this model is 0.25. 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.

3.4.2 *AACP Optimal Model From Testing*

During the day and night, the AACP detection model that performed best during our tests was the IR+DIRTYIRDIFF MultiResUnet model. Table 2 shows the optimal threshold for this model is 0.45. See Section 4.4 for more details regarding optimal model thresholds.

3.5 *Model Checkpoint Files*

The best model checkpoint files are included during the install process. These files were output during our model training 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.

3.6 *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 5.2 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 5.2 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.

3.7 *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. DEFAULT is to no longer write these files, starting with Version 2. To save on local memory, the model results will only be written to the netCDF files....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.

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 'Case Classifier' 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 was not input into the model at any time.

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

Date	Time Range (UTC)	Mesoscale Sector	Case Classifier
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

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. Most model input combinations only are available for MultiResUnet. Model inputs for OTs available include: IR only, TROPDIFF only, IR+VIS, IR+GLM, IR+WVIRDIFF, IR+VIS+GLM, IR+TROPDIFF, IR+CIRRUS, IR+SNOWICE, IR+DIRTYIRDIFF, VIS+TROPDIFF, GLM+TROPDIFF, IR+VIS+TROPDIFF, VIS+GLM+TROPDIFF, TROPDIFF+DIRTYIRDIFF, IR+VIS+DIRTYIRDIFF, and VIS+TROPDIFF+DIRTYIRDIFF. TROPDIFF stands for 10.3 micron IR BT - GFS Tropopause temperature, WVIRDIFF stands for 6.2 micron - 10.3 micron channel difference, and DIRTYIRDIFF stands for 12.3 micron - 10.3 micron channel difference. For each of these input combinations 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.

During the day, the OT detection model that performed best during our tests was the IR+VIS 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 AACPs available include: IR only, TROPDIFF only, IR+VIS, IR+GLM, IR+WVIRDIFF, IR+VIS+GLM, IR+TROPDIFF, IR+CIRRUS, IR+SNOWICE, IR+DIRTYIRDIFF, VIS+TROPDIFF, GLM+TROPDIFF, IR+VIS+TROPDIFF, VIS+GLM+TROPDIFF, TROPDIFF+DIRTYIRDIFF, IR+VIS+DIRTYIRDIFF, and VIS+TROPDIFF+DIRTYIRDIFF. TROPDIFF stands for 10.3 micron IR BT - GFS Tropopause temperature, WVIRDIFF stands for 6.2 micron - 10.3 micron channel difference, and DIRTYIRDIFF stands for 12.3 micron - 10.3 micron channel difference.

During the day and night, the AACP detection model that performed best during our tests was

the IR+DIRTYIRDIFF 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 (10.3μm) range is 180 K to 230 K. VIS reflectance (0.64μm) range is 0 to 1. GLM flash extent density range is 0 to 20. WVIRDIFF (6.2μm - 10.3μm) range is -20 K to 10 K. TROPDIFF (10.3μm - GFS TropT) range is -15 K to 20 K SNOWICE (1.6μm) range is 0 to 1. CIRRUS (1.37μm) range is 0 to 1. DIRTYIRDIFF (12.3μm - 10.3μm) range is -1 to 2.

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. 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 at the VIS resolution of 0.5 km are shown in Table 2. IR resolution (2 km) optimal model thresholds are shown in Table 3.

5 *Model Input Data*

5.1 *Model Satellite Data Inputs*

Raw Level 1b GOES, GLM data, and GFS tropopause temperatures are used as the primary satellite inputs for training the deep learning models. The ABI is featured on the latest generation of

Table 2: Model types and optimal thresholds found from independent test cases for data on VIS 0.5 km resolution.

Severe Storm Signature	Model Type	Model Inputs	Optimal Model Threshold
OT	MultiResUnet	IR	0.40
OT	MultiResUnet	TROPDIFF	0.60
OT	MultiResUnet	IR+VIS	0.25
OT	MultiResUnet	IR+GLM	0.65
OT	MultiResUnet	IR+WVIRDIFF	0.30
OT	MultiResUnet	IR+SNOWICE	0.60
OT	MultiResUnet	IR+CIRRUS	0.50
OT	MultiResUnet	IR+DIRTYIRDIFF	0.45
OT	MultiResUnet	IR+TROPDIFF	0.25
OT	MultiResUnet	VIS+TROPDIFF	0.50
OT	MultiResUnet	TROPDIFF+GLM	0.55
OT	MultiResUnet	IR+VIS+GLM	0.40
OT	MultiResUnet	IR+VIS+TROPDIFF	0.15
OT	MultiResUnet	VIS+TROPDIFF+GLM	0.40
OT	MultiResUnet	IR+VIS+DIRTYIRDIFF	0.30
OT	MultiResUnet	VIS+TROPDIFF+DIRTYIRDIFF	0.45
OT	MultiResUnet	TROPDIFF+DIRTYIRDIFF	0.45
OT	Unet	IR	0.45
OT	Unet	IR+GLM	0.45
OT	Unet	IR+VIS	0.35
OT	Unet	IR+WVIRDIFF	0.45
OT	Unet	IR+VIS+GLM	0.45
OT	AttentionUnet	IR+VIS	0.65
OT	AttentionUnet	IR+VIS+GLM	0.20
AACP	MultiResUnet	IR	0.75
AACP	MultiResUnet	TROPDIFF	0.80
AACP	MultiResUnet	IR+VIS	0.50
AACP	MultiResUnet	IR+GLM	0.20
AACP	MultiResUnet	IR+WVIRDIFF	0.80
AACP	MultiResUnet	IR+SNOWICE	0.25
AACP	MultiResUnet	IR+CIRRUS	0.60
AACP	MultiResUnet	IR+DIRTYIRDIFF	0.50
AACP	MultiResUnet	IR+TROPDIFF	0.80
AACP	MultiResUnet	VIS+TROPDIFF	0.70
AACP	MultiResUnet	TROPDIFF+GLM	0.55
AACP	MultiResUnet	IR+VIS+GLM	0.35
AACP	MultiResUnet	IR+VIS+TROPDIFF	0.70
AACP	MultiResUnet	VIS+TROPDIFF+GLM	0.30
AACP	MultiResUnet	IR+VIS+DIRTYIRDIFF	0.30
AACP	MultiResUnet	VIS+TROPDIFF+DIRTYIRDIFF	0.25
AACP	MultiResUnet	TROPDIFF+DIRTYIRDIFF	0.75

Table 3: Model types and optimal thresholds found from independent test cases on IR 2.0 km resolution.

Severe Storm Signature	Model Type	Model Inputs	Optimal Model Threshold
OT	MultiResUnet	IR	0.20
OT	MultiResUnet	TROPDIFF	0.40
OT	MultiResUnet	IR+GLM	0.25
OT	MultiResUnet	IR+WVIRDIFF	0.55
OT	MultiResUnet	IR+DIRTYIRDIFF	0.25
OT	MultiResUnet	IR+TROPDIFF	0.40
OT	MultiResUnet	TROPDIFF+GLM	0.55
OT	MultiResUnet	TROPDIFF+DIRTYIRDIFF	0.45
AACP	MultiResUnet	IR	0.30
AACP	MultiResUnet	TROPDIFF	0.60
AACP	MultiResUnet	IR+GLM	0.30
AACP	MultiResUnet	IR+WVIRDIFF	0.55
AACP	MultiResUnet	IR+DIRTYIRDIFF	0.40
AACP	MultiResUnet	IR+TROPDIFF	0.20
AACP	MultiResUnet	TROPDIFF+GLM	0.50
AACP	MultiResUnet	TROPDIFF+DIRTYIRDIFF	0.15

NOAA/NASA geostationary satellites, including GOES-16, -17, and -18 (Schmit et al., 2017, 2018). The ABI samples 16 spectral bands at 0.5-2 km horizontal resolution every 1 min for mesoscale domains, 5 mins for CONUS, and 10 mins for full disk scans. Mesoscale domains are fixed areas, each $\sim$1000 km^2, chosen to provide detailed observations for areas of interest like severe storms. Raw Level 1b GOES-16 channel files used as model input includes the 0.64 μm (VIS band), 1.37 μm (CIRRUS band), 1.6 μm (SNOWICE band), 10.3 μm (IR band), and 12.3μm (dirty IR band). The IR band is interpolated to the VIS band resolution when both inputs are used together. Otherwise, the model software runs using the native IR data resolution.

The flash extent density product measured by GLM can also be used as input into the deep learning model. Flash extent density indicates the number of flashes (both cloud-to-ground and cloud-to-cloud) that occur within a grid cell over a given time period. The horizontal resolution of the flash extent density data are 8 km with new GLM data files available every 20 seconds. GLM data are gridded using the glmtools Python library provided in Bruning et al. (2019). Only GLM files within $\pm$2.5 minutes of GOES IR/VIS start scan times are gridded and considered co-located. GLM data are also interpolated to the VIS data resolution of 0.5 km prior to being input into the model.

In addition to satellite imagery, the Global Forecast System (GFS) model tropopause temperature product was tested as input to the machine learning models. GFS was chosen due to its production of near real-time analysis files. GFS grib files are downloaded from the NCAR Research Data Archive. The GFS lapse-rate tropopause temperature product (TropT), available 6-hourly on a 0.25° x 0.25° longitude-latitude grid (National Centers for Environmental Prediction, National Weather Service, NOAA, U.S. Department of Commerce, 2015), is linearly interpolated in time and space from the GFS grid to the GOES ABI pixels. Occasionally, reanalysis tropopause temperature fields can show unrealistic spatial variability. At one location, the reanalysis may accurately capture the primary tropopause while nearby it may incorrectly label the secondary tropopause as the primary. This is most common near jet streams (Khlopenkov et al., 2021). To mitigate potential artifacts, tropopause temperature outliers are identified and replaced using spatial filtering methods described in Khlopenkov et al. (2021). For our purposes, we used a 10 grid box radius for the spatial filtering.

5.2 *Data Download Sources*

It is recommended that users download data from the Google Cloud Platform in order to get the full usage out of the software, but it is not required to run the software. Additional resources are included below. As a reminder, our software uses the L1b GOES data products.

5.2.1 *Google Cloud*

GOES and GLM data can be downloaded from Google Cloud (see Section 9 for more information about the Google Cloud Platform). All satellite imagery used in this study were obtained from the GCP (Google). Data are available in Google's public storage buckets within a minute of the satellite scan completion. Software functions that do the downloading are named run_download_goes_ir_vis_l1b_glm_l2_data.py and run_download_goes_ir_vis_l1b_glm_l2_data_parallel.py functions. Using Google Cloud, 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. See Section 9 for Google Cloud setup instructions.

GFS tropopause temperature data when used in real-time is also downloaded from Google Cloud. If tropopause temperatures in post-processing, then the data are downloaded from data.rda.ucar.

5.2.2 *Additional Resources*

The software is setup to be run if you have locally stored GOES files and do not want to download from the GCP. If you would prefer to download the data yourself to store and not use Google Cloud storage, please refer to page 14 of NOAA Website Documentation. I have provided this link because I think they do a better job of linking to sites and explaining the process than I would here. This page of the document refers to NOAA's CLASS data repository. As a reminder, our software uses the L1b GOES data products.

A similar but easier website to navigate might be NCEI's Archive Information Request System (AIRS). No account is needed for this site (AIRS Website). Description of downloading GOES data from AIRS is also on page 14 of the pdf link sent above. Both NOAA's class system and AIRS will allow you to download GOES data using FTP.

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 $\pm$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 can 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.

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/'.

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, snowice, and cirrus data are not available if the solar zenith angle in 5% or more of the domain exceeds 85°. In this case, ONLY IR and WV data channels are used. This prevents retention of 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, TROPDIFF, DIRTYIRDIFF, CIRRUS, SNOWICE, WVIRDIFF, 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_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*

Once 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. read database and health and monitoring data technique

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. The GFS lapse-rate tropopause temperature product (TropT), available 6-hourly on a 0.25° x 0.25° longitude-latitude grid (National Centers for Environmental Prediction, National Weather Service, NOAA, U.S. Department of Commerce, 2015), is linearly interpolated in time and space from the GFS grid to the GOES ABI pixels. Occasionally, reanalysis tropopause temperature fields can show unrealistic spatial variability. At one location, the reanalysis may accurately capture the primary tropopause while nearby it may incorrectly label the secondary tropopause as the primary. This is most common near jet streams (Khlopenkov et al., 2021). To mitigate potential artifacts, tropopause temperature outliers are identified and replaced using spatial filtering methods described in Khlopenkov et al. (2021). For our purposes, we used a 10 grid box radius for the spatial filtering. Tropopause temperatures are only included in the output netCDF files for OT object model runs or for runs that used TROPDIFF as model input.

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 $> 5\%$ of pixels in domain $> 85^{\circ}$ yields 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_trop` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_files2 and normalizes the IR BT - TropT difference data to range from 0 to 1. The default min and max values are -15 and 20, respectively.

5. `fetch_convert_snowice` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_files2

and normalizes the snow/ice reflectance data to range from 0 to 1. The default min and max values are 0 and 1, respectively.

6. `fetch_convert_cirrus` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_files2 and normalizes the cirrus reflectance data to range from 0 to 1. The default min and max values are 0 and 1, respectively.

7. `fetch_convert_dirtyirdiff` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_f and normalizes the dirtyIR - IR difference data to range from 0 to 1. The default min and max values are -1 and 2, respectively.

8. `fetch_convert_irdiff` is a function that is called by create_vis_ir_numpy_arrays_from_netcdf_files2 and normalizes the WV-IR difference data to range from 0 to 1. The default min and max values are -20 and 10, respectively.

9. `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.

10. `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 1: 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.

IMPORTANT NOTE 2: In order to use Google Cloud services, the user must have a gmail account. Downloading data from the Google Cloud public storage repository is free with a gmail account.

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*

The following steps are used to install Google Cloud on your device. I have included as many details and separated the tasks to be small so that they are as easy to follow as possible. Hopefully, Google does not change how this process.

Step 1: To install Google Cloud, follow the directions on Google Cloud Install. Download the package for your operating system.

Step 2: Follow instructions on the page (steps 2a, 2b, and 2c). Even if it says Optional, I recommend doing those steps. The instructions include unzipping the downloaded file, running 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'), and allowing the install to update your PATH in your .bash_profile or .zshrc whatever you are using. Your .bash_profile or .zshrc should look similar to the following after completing these steps:

```
# 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
```

Step 3: Initialize the gcloud CLI by running 'gcloud init'. Running this will give the user prompts they need to answer.

First, the user will be prompted for a configuration name. This configuration name can be anything that the user wants, as long as it is not already being used by another user project. In testing, I chose tmptestproj2-7363823364 because it was very unlikely to have been used by someone else previously. You can choose something that corresponds more closely to your project though.

Second, the user will be prompted for an account to be associated with this configuration (user must have a Gmail account in order for this to work).You can login with new account when prompted.

This will redirect the user to their default web browser.

Click on your gmail account or enter it in and login to the account.

Click 'Allow' at bottom of page to authenticate the Google Cloud CLI.

User will next be prompted to create a project ID.

You can again name this anything you would like as long as it is not being used by another Google Cloud user.

Example: In testing, I chose a random assortment (e.g. tmptestproj-673527384)

Step 4: Once signed in, click select a project (next to 'GoogleCloud text' in left hand corner).

Step 5: Select your project name.

Step 6: Select the 3 horizontal lines to the left of 'GoogleCloud' text.

Step 7: Scroll down to 'IAM & Admin'.

Step 8: Within 'IAM & Admin' menu, Click on 'Service Accounts'.

Step 9: Once at Service Accounts page, Click '+ CREATE SERVICE ACCOUNT' near top of page.

Step 10: Type in a Service Account name. Again, this can be anything you want. Once you have typed in the Service Account name, Click Done. You have now created a service account.

Step 11: Click the 3 vertical dots next to the service account that you just enabled.

Step 12: Scroll down and Click Manage Keys.

Step 13: Click 'ADD KEY' on webpage.

Step 14: Click 'Create new key' (Create a JSON key type). This will download the json file to your local computer.

Step 15: Move the json file that was just downloaded into your google-cloud-sdk directory.

Step 16: Within your .zshrc, .basrc, or whatever it is that you use, type:

export GOOGLE_APPLICATION_CREDENTIALS=path-to-json-file

(e.g. in my file, 'export GOOGLE_APPLICATION_CREDENTIALS=/Users/jack/google-cloud-sdk/tmptestproj-673527384-a2baeabf78a0.json').

Step 17: Open a new Terminal window. All new Terminal windows should now be able to use

Google Cloud which will be accessed by the software code.

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 *Google Cloud Problems Users May Encounter*

9.3.1 *Local Web Proxy/Firewall Error*

When running the 'glcoud init' command (to initialize the glcoud CLI utilities), some users have reported running into an error message that looks like the following:

```
ERROR: Reachability Check failed.
    httplib2 cannot reach https://accounts.google.com:
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificate chain (_ssl.c:1129)

    httplib2 cannot reach https://cloudresourcemanager.googleapis.com/v1beta1/projects:
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificate chain (_ssl.c:1129)

    httplib2 cannot reach https://www.googleapis.com/auth/cloud-platform:
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificate chain (_ssl.c:1129)

    httplib2 cannot reach https://dl.google.com/dl/cloudsdk/channels/rapid/components-2.json:
[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate in certificate chain (_ssl.c:1129)

Network connection problems may be due to proxy or firewall settings.
```

The key step to fixing this issue was doing: gcloud config set core/custom_ca_certs_file $SSL_CERT_FILE. Your system should then populate this environment variable with a specific tls-ca-bundle.pem file on it, allowing the gcloud products to all work.

If the above fix does not work for you, I found some other resources that might be helpful and included them here. One a possible solution to the problem is setting: gcloud config set auth/disable_ssl_validation True. This would disable the ssl validation. Some other links that may be useful are provided here: Stack Overflow Link and Google Cloud Documentation Link. Your resident IT expert may be best to discuss the proxy issue with though as certain user's settings may have different permissions and allowances than others.

9.4 *Script Breakdown*

All of the code for reading and writing to and from the Google Cloud is located within the script *gcs_processing.py*. All of the software code calls the functions in this script. 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.

10.1.3 *23 August 2023*

Added Windows .yml file to the git repo. This should allow Windows users an easier time to setup and install the software. Minor changes to path in run_tf_1_channel_plume_updraft_day_predict.py function as well that occurred when re split was being used and the pattern being searched for included path separators.

10.1.4 *6 March 2024 VERSION 2 Release*

VERSION 2 of software!!! LARGE model performance enhancement, particularly with AACP detection. Added A LOT of new model input combinations as possibilities, including TROPDIFF, DIRTYIRDIFF, SNOWICE, and CIRRUS. Added the new and improved checkpoint files associated with those model runs. Optimal models and thresholds have been updated since the Version 1 release. Changed how we interpolate the GFS data onto the GOES grid. We now follow the methods outlined in Khlopenkov et al. (2021). New version changes how software decides between daytime and nighttime model runs. Previously used the maximum solar zenith angle within the domain and checked if that exceeded 85°, however, there was an issue of only nighttime models being used for CONUS domains due to how the satellite views the Earth and the data are gridded. Thus, now the software checks to see if more than 5% of the pixels in the domain have solar zenith angles that exceed 85°. If they do, then the software acts as if the domain is nighttime and if not the software acts as if it is daytime within the domain. Fixed major issue with looping over post-processing dates and the month or year changed during a real-time or archived model run. New version speeds up post-processing for OTs. Set variables to None after using them in order to clear cached memory. Catch and hide RunTime warnings that the User does not need to worry about.

Software Users prior to 6 March 2024 will need to pull the latest python files from GitHub. This should be very fast. Users MUST also download the new checkpoint files from 'https://science-data.larc.nasa.gov/LaRC-SD-Publications/2023-05-05-001-JWC/data/ML_data.zip'. Once downloaded and unzipped, replace the old model_checkpoints subdirectory with the latest one that was just downloaded. This directory includes all of the improved model detection files as well as checkpoint files for the new input combinations.

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.

10.2.2 *21 August 2023*

In post-processing, some users experienced a 'Runtime Warning:invalid value encountered in cast' when initializing the ot_id array. This was likely due to differences in OS's. Numpy is aware of the issue but the line of code has been changed to ot_id = np.full_like(res, 0).astype('uint16'). This should remove the issue for those users.

ReadME documentation was updated for Google Cloud and includes additional ways to download GOES L1b data for ingestion into the software.

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*

No Warranty: THE SUBJECT SOFTWARE IS PROVIDED ”AS IS” WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THE SUBJECT SOFTWARE WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE SUBJECT SOFTWARE WILL BE ERROR FREE, OR ANY WARRANTY THAT DOCUMENTATION, IF PROVIDED, WILL CONFORM TO THE SUBJECT SOFTWARE. THIS AGREEMENT DOES NOT, IN ANY MANNER, CONSTITUTE AN ENDORSEMENT BY GOVERNMENT AGENCY OR ANY PRIOR RECIPIENT OF ANY RESULTS, RESULTING DESIGNS, HARDWARE, SOFTWARE PRODUCTS OR ANY OTHER APPLICATIONS 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 IM-

MEDIATE, 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
```


Bruning, E. C., et al., 2019: Meteorological imagery for the geostationary lightning mapper. *Journal of Geophysical Research: Atmospheres*, **124 (24)**, 14 285–14 309.

Google, ????: gcp-public-data-goes-16. https://console.cloud.google.com/marketplace/product/noaa-public/goes.

Schmit, T. J., P. Griffith, M. M. Gunshor, J. M. Daniels, S. J. Goodman, and W. J. Lebair, 2017: A closer look at the abi on the goes-r series. *Bulletin of the American Meteorological Society*, **98**, 681–698.

Schmit, T. J., S. S. Lindstrom, J. J. Gerth, and M. M. Gunshor, 2018: Applications of the 16 spectral bands on the advanced baseline imager (abi). *J. Operational Meteor.*, **6 (4)**, 33–46.

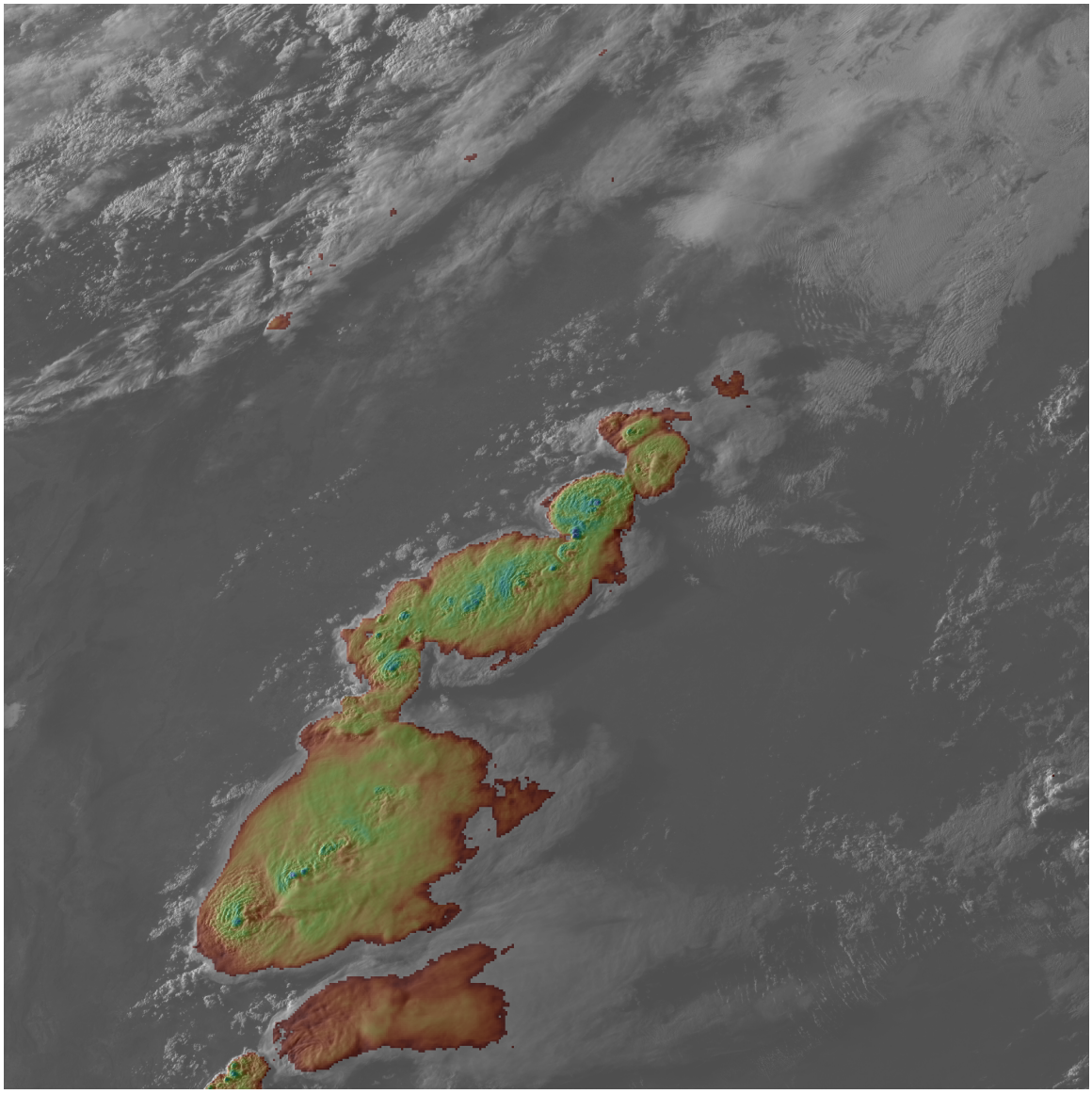

Figure 1: Example IR/VIS image for combined netCDF file: OR_ABI_L1b_M2_COMBINED_s20201350016495_e20201350016564_c20201350017016.

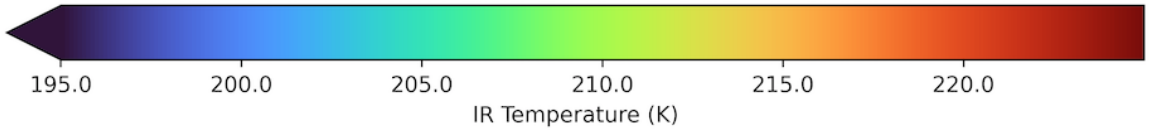

Figure 2: IR color bar.