Thoughts:

One roadblock I stuck with initially was the dimension expansion you had in the original notebook, which was not that clear initially. (Maybe I was just dumb.)

I dropped different columns from the original dataset than you did, please see the correlation matrix shown below.

I used larger epoch as well. Unless there is something I am missing regarding why I should only use batch_size = 8, then, I will continue with larger epochs.

Changes will be highlighted down below.

---

1. Data Generation

In [1]:

```
import os
import pandas as pd
import numpy as np
import pickle
import ast

# Plotting libraries
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns
%matplotlib inline
```

In [2]:

```
# Universial data folder
# Inside, we have the CSV for each weather station, and the satellite imagery c
# shall be generated and stored inside a sub-folder
data_path = 'data_dir/'
csv_path = 'combined_dataset/'
```

In [3]:

```
# Get list of all CSV files
all_files = os.listdir(data_path + csv_path)

# Filter out the CSV files
csv_files = [file for file in all_files if file.endswith('.csv')]

# Now csv_files list contains all the names of csv files

# To get the full path of these csv files
csv_file_paths = [os.path.join(data_path, csv_path, file) for file in csv_files
```

In [4]:

```
# Inspection purpose
len(csv_file_paths)
```

Out[4]:

```
5
```

In [5]:
```
csv_file_paths
```

Out[5]:
```
['data_dir/combined_dataset/Take_2_2006Fall_2017Spring_GOES_meteo_combined_14815.csv',
 'data_dir/combined_dataset/Take_2_2006Fall_2017Spring_GOES_meteo_combined_14850.csv',
 'data_dir/combined_dataset/Take_2_2006Fall_2017Spring_GOES_meteo_combined_14819.csv',
 'data_dir/combined_dataset/Take_2_2006Fall_2017Spring_GOES_meteo_combined_04846.csv',
 'data_dir/combined_dataset/Take_2_2006Fall_2017Spring_GOES_meteo_combined_14845.csv']
```

`TO-DO`:

Change the index number for `csv_file_paths` to switch weather stations.

In [6]:
```
file_idx = 1
```

In [7]:
```
df_single_station = pd.read_csv(csv_file_paths[file_idx])

filename_curr = csv_file_paths[file_idx]
station_code = filename_curr[-9:-4]
```

In [8]:
```
# Inspection purpose
df_single_station.head(5)
```

Out[8]:

	Date_UTC	Time_UTC	Date_CST	Time_CST	File_name_for_1D_lake	File_nam
0	2006-10-01	00:00	2006-09-30	18:00	goes11.2006.10.01.0000.v01.nc-var1-t0.csv	T_goes11.2006.10.0
1	2006-10-01	01:00	2006-09-30	19:00	goes11.2006.10.01.0100.v01.nc-var1-t0.csv	T_goes11.2006.10.0
2	2006-10-01	02:00	2006-09-30	20:00	goes11.2006.10.01.0200.v01.nc-var1-t0.csv	T_goes11.2006.10.0
3	2006-10-01	03:00	2006-09-30	21:00	goes11.2006.10.01.0300.v01.nc-var1-t0.csv	T_goes11.2006.10.0
4	2006-10-01	04:00	2006-09-30	22:00	goes11.2006.10.01.0400.v01.nc-var1-t0.csv	T_goes11.2006.10.0

5 rows × 31 columns

Change column names for easier access.

In [9]:
```
# Check if 'Unnamed: 18' is in the DataFrame's columns
if 'Unnamed: 18' in df_single_station.columns:
    # Drop the column
    df_single_station = df_single_station.drop(columns=['Unnamed: 18'])
#     print('Dropped the empty column.')
else:
    print('Empty column does not exist.')

# Check if 'does_snow_24_120' is in the DataFrame's columns
if 'does_snow_24_120' in df_single_station.columns:
    # Drop the column
    df_single_station = df_single_station.drop(columns=['does_snow_24_120'])
#     print('Dropped the <does_snow_24_120> column.')
else:
    print('The <does_snow_24_120> column does not exist.')

# Check if 'precip_work_zone' is in the DataFrame's columns
if 'precip_work_zone' in df_single_station.columns:
    # Drop the column
    df_single_station = df_single_station.drop(columns=['precip_work_zone'])
#     print('Dropped the <precip_work_zone> column.')
else:
    print('The <precip_work_zone> column does not exist.')

# Check if 'is_snow_precip' is in the DataFrame's columns
if 'is_snow_precip' in df_single_station.columns:
    # Drop the column
    df_single_station = df_single_station.drop(columns=['is_snow_precip'])
#     print('Dropped the <is_snow_precip> column.')
else:
    print('The <is_snow_precip> column does not exist.')

# Check if 'is_precip' is in the DataFrame's columns
if 'is_precip' in df_single_station.columns:
    # Drop the column
    df_single_station = df_single_station.drop(columns=['is_precip'])
#     print('Dropped the <is_precip> column.')
else:
    print('The <is_precip> column does not exist.')

# Check if 'Wind Chill (F)' is in the DataFrame's columns
if 'Wind Chill (F)' in df_single_station.columns:
    # Drop the column
    df_single_station = df_single_station.drop(columns=['Wind Chill (F)'])
#     print('Dropped the <Wind Chill (F)> column.')
else:
    print('The <Wind Chill (F)> column does not exist.')

# Check if 'Heat Index (F)' is in the DataFrame's columns
if 'Heat Index (F)' in df_single_station.columns:
    # Drop the column
    df_single_station = df_single_station.drop(columns=['Heat Index (F)'])
#     print('Dropped the <Heat Index (F)> column.')
else:
    print('The <Heat Index (F)> column does not exist.')
```

In [10]:
```python
# Renaming
df_single_station.rename(columns={ "Temp (F)": "Temp_F", "RH (%)": "RH_pct",
                    "Dewpt (F)" : "Dewpt_F", "Wind Spd (mph)" : "Wind_Spd_mph",
                    "Wind Direction (deg)" : "Wind_Direction_deg", "Peak Wind Gu
                    "Low Cloud Ht (ft)" : "Low_Cloud_Ht_ft", "Med Cloud Ht (ft)'
                    "High Cloud Ht (ft)" : "High_Cloud_Ht_ft", "Visibility (mi)'
                    "Atm Press (hPa)" : "Atm_Press_hPa", "Sea Lev Press (hPa)" :
                    "Altimeter (hPa)" : "Altimeter_hPa", "Precip (in)" : "Precip
                    "Wind Chill (F)" : "Wind_Chill_F", "Heat Index (F)" : "Heat_
                    } , inplace = True)
```

In [11]:
```python
def missing_values(df):
    total_null = df.isna().sum()
    percent_null = total_null / df.count() # Total count of null values / Total
    missing_data = pd.concat([total_null, percent_null], axis = 1, keys = ['Tot
    return missing_data

missing_values_before = missing_values(df_single_station)
missing_values_before
```

Out[11]:

	Total Null	Percentage Null
Date_UTC	0	0.000000
Time_UTC	0	0.000000
Date_CST	0	0.000000
Time_CST	0	0.000000
File_name_for_1D_lake	0	0.000000
File_name_for_2D_lake	0	0.000000
Lake_data_1D	0	0.000000
data_usable	0	0.000000
cloud_count	0	0.000000
cloud_exist	0	0.000000
Temp_F	239	0.004991
RH_pct	239	0.004991
Dewpt_F	239	0.004991
Wind_Spd_mph	239	0.004991
Wind_Direction_deg	239	0.004991
Peak_Wind_Gust_mph	239	0.004991
Low_Cloud_Ht_ft	239	0.004991
Med_Cloud_Ht_ft	239	0.004991
High_Cloud_Ht_ft	239	0.004991
Visibility_mi	239	0.004991
Atm_Press_hPa	239	0.004991
Sea_Lev_Press_hPa	239	0.004991
Altimeter_hPa	239	0.004991
Precip_in	239	0.004991

In [12]:

```
# Replace any m, M values to nan (float type)
df_single_station['Temp_F'] = df_single_station['Temp_F'].replace(['m', 'M'], f

# Then, replace those nan values with the last numerical value in the column
df_single_station['Temp_F'] = df_single_station['Temp_F'].fillna(method='ffill'
```

In [13]:

```
# Replace any m, M values to nan (float type)
df_single_station['RH_pct'] = df_single_station['RH_pct'].replace(['m', 'M'], f

# Then, replace those nan values with the last numerical value in the column
df_single_station['RH_pct'] = df_single_station['RH_pct'].fillna(method='ffill'
```

In [14]:

```
# Replace any m, M values to nan (float type)
df_single_station['Dewpt_F'] = df_single_station['Dewpt_F'].replace(['m', 'M'],
```

```
# Then, replace those nan values with the last numerical value in the column
df_single_station['Dewpt_F'] = df_single_station['Dewpt_F'].fillna(method='ffil
```

In [15]:
```
# Replace any m, M values to nan (float type)
df_single_station['Wind_Spd_mph'] = df_single_station['Wind_Spd_mph'].replace([

# Then, replace those nan values with the last numerical value in the column
df_single_station['Wind_Spd_mph'] = df_single_station['Wind_Spd_mph'].fillna(me
```

In [16]:
```
# Replace any m, M values to nan (float type)
df_single_station['Wind_Direction_deg'] = df_single_station['Wind_Direction_deg

# Then, replace those nan values with the last numerical value in the column
df_single_station['Wind_Direction_deg'] = df_single_station['Wind_Direction_deg
```

> "Peak Wind Gust" refers to the highest instantaneous wind speed recorded during a specific period, typically over the course of a day. It represents the maximum force of wind experienced at a location and is usually caused by high-pressure systems or storms.

Therefore, we further replace any of the `NaN` values in the column `Peak_Wind_Gust_mph` with the value that is in the column `Wind_Spd_mph`.

In [17]:
```
# Replace any m, M values to nan (float type)
df_single_station['Peak_Wind_Gust_mph'] = df_single_station['Peak_Wind_Gust_mph

# Then, replace those nan values with the last numerical value in the column
df_single_station['Peak_Wind_Gust_mph'] = df_single_station['Peak_Wind_Gust_mph

df_single_station['Peak_Wind_Gust_mph'] = df_single_station['Peak_Wind_Gust_mph
```

In [18]:
```
# Replace any m, M values to nan (float type)
df_single_station['Low_Cloud_Ht_ft'] = df_single_station['Low_Cloud_Ht_ft'].rep

# Then, replace those nan values with the last numerical value in the column
df_single_station['Low_Cloud_Ht_ft'] = df_single_station['Low_Cloud_Ht_ft'].fil
```

In [19]:
```
# Replace any m, M values to nan (float type)
df_single_station['Med_Cloud_Ht_ft'] = df_single_station['Med_Cloud_Ht_ft'].rep

# Then, replace those nan values with the last numerical value in the column
df_single_station['Med_Cloud_Ht_ft'] = df_single_station['Med_Cloud_Ht_ft'].fil

df_single_station['Med_Cloud_Ht_ft'] = df_single_station['Med_Cloud_Ht_ft'].fil
```

In [20]:
```
# Replace any m, M values to nan (float type)
df_single_station['High_Cloud_Ht_ft'] = df_single_station['High_Cloud_Ht_ft'].r

# Then, replace those nan values with the last numerical value in the column
df_single_station['High_Cloud_Ht_ft'] = df_single_station['High_Cloud_Ht_ft'].f

df_single_station['High_Cloud_Ht_ft'] = df_single_station['High_Cloud_Ht_ft'].f
```

In [21]:
```
# Replace any m, M values to nan (float type)
df_single_station['Visibility_mi'] = df_single_station['Visibility_mi'].replace

# Then, replace those nan values with the last numerical value in the column
df_single_station['Visibility_mi'] = df_single_station['Visibility_mi'].fillna(
```

In [22]:
```
# Replace any m, M values to nan (float type)
df_single_station['Atm_Press_hPa'] = df_single_station['Atm_Press_hPa'].replace

# Then, replace those nan values with the last numerical value in the column
df_single_station['Atm_Press_hPa'] = df_single_station['Atm_Press_hPa'].fillna(
```

In [23]:
```
# Replace any m, M values to nan (float type)
df_single_station['Sea_Lev_Press_hPa'] = df_single_station['Sea_Lev_Press_hPa']

# Then, replace those nan values with the last numerical value in the column
df_single_station['Sea_Lev_Press_hPa'] = df_single_station['Sea_Lev_Press_hPa']
```

In [24]:
```
# Replace any m, M values to nan (float type)
df_single_station['Altimeter_hPa'] = df_single_station['Altimeter_hPa'].replace

# Then, replace those nan values with the last numerical value in the column
df_single_station['Altimeter_hPa'] = df_single_station['Altimeter_hPa'].fillna(
```

In [25]:
```
# Replace any m, M values to nan (float type)
df_single_station['Precip_in'] = df_single_station['Precip_in'].replace(['m', '

# Then, replace those nan values with the last numerical value in the column
df_single_station['Precip_in'].fillna(0.00, inplace = True)
```

After all the patch work, let's see how the situation is now with missing values.

In [26]:
```
missing_values_after = missing_values(df_single_station)
missing_values_after
```

Out[26]:

	Total Null	Percentage Null
Date_UTC	0	0.0
Time_UTC	0	0.0
Date_CST	0	0.0
Time_CST	0	0.0
File_name_for_1D_lake	0	0.0
File_name_for_2D_lake	0	0.0
Lake_data_1D	0	0.0
data_usable	0	0.0
cloud_count	0	0.0
cloud_exist	0	0.0
Temp_F	0	0.0
RH_pct	0	0.0
Dewpt_F	0	0.0
Wind_Spd_mph	0	0.0
Wind_Direction_deg	0	0.0
Peak_Wind_Gust_mph	0	0.0
Low_Cloud_Ht_ft	0	0.0
Med_Cloud_Ht_ft	0	0.0
High_Cloud_Ht_ft	0	0.0
Visibility_mi	0	0.0
Atm_Press_hPa	0	0.0
Sea_Lev_Press_hPa	0	0.0
Altimeter_hPa	0	0.0
Precip_in	0	0.0

In [27]:
```
df_daytime_only = df_single_station.loc[(df_single_station['Time_UTC'] >= '14:(
                    & (df_single_station['Time_UTC'] <= '21:00')]
df_daytime_only  = df_daytime_only .reset_index(drop=True)
# df_daytime_only.head(10)
```

In [28]:
```
# Summary
df_daytime_only.describe()
```

Out[28]:

	cloud_count	Temp_F	RH_pct	Dewpt_F	Wind_Spd_mph	Wind_Direct
count	16040.000000	16040.000000	16040.000000	16040.000000	16040.000000	16040
mean	3189.580860	35.412594	68.103491	25.379988	8.313529	183
std	782.601809	14.920630	15.099017	13.649343	4.870364	113
min	1.000000	-13.000000	10.000000	-20.000000	0.000000	0
25%	3192.500000	25.000000	58.000000	16.000000	5.000000	80
50%	3579.000000	34.000000	70.000000	25.000000	8.000000	210
75%	3599.000000	45.000000	79.000000	34.000000	11.000000	270
max	3599.000000	88.000000	100.000000	67.000000	32.000000	360

---

2. Cloud Image Generation

We will try to generate the images based on the 1-D lake data.

In [29]:
```
df_lat_lon = pd.read_csv('data_dir/lat_long_1D_labels_for_plotting.csv')
# df_lat_lon.head(5)
```

In [30]:
```
lat_lst = df_lat_lon['latitude'].to_list()
lon_lst = df_lat_lon['longitude'].to_list()
```

1-D Lake Imagery Data Conversion

In [31]:
```
def rectify(crap_string):
    return [0.0 if el == 'nan' else float(el) for el in crap_string.strip('][')
```

3. Feature Engineering for Snowfall Events

The fundamental criteria are the temperature to be below 32 F in the local area, and the precipitation larger than 0.01 inch.

In [32]:
```
df_daytime_only.loc[(df_daytime_only['Temp_F'] <= 32) & (df_daytime_only['Preci
df_daytime_only.loc[(df_daytime_only['Temp_F'] > 32) | (df_daytime_only['Precip
# df_daytime_only.head(5)
```

In [33]:
```
df_daytime_only = df_daytime_only.drop(['Date_UTC', 'Time_UTC', 'Date_CST', 'Ti
df_daytime_only = df_daytime_only.reset_index(drop=True)
# df_daytime_only.head()
```

In [34]:
```
df_daytime_only = df_daytime_only.drop(['data_usable', 'cloud_count', 'cloud_ex
df_daytime_only = df_daytime_only.reset_index(drop=True)
```

In [35]:
```
# Summary
df_daytime_only.describe()
```

Out[35]:

	Temp_F	RH_pct	Dewpt_F	Wind_Spd_mph	Wind_Direction_deg	Peak_
count	16040.000000	16040.000000	16040.000000	16040.000000	16040.000000	
mean	35.412594	68.103491	25.379988	8.313529	183.465087	
std	14.920630	15.099017	13.649343	4.870364	113.074909	
min	-13.000000	10.000000	-20.000000	0.000000	0.000000	
25%	25.000000	58.000000	16.000000	5.000000	80.000000	
50%	34.000000	70.000000	25.000000	8.000000	210.000000	
75%	45.000000	79.000000	34.000000	11.000000	270.000000	
max	88.000000	100.000000	67.000000	32.000000	360.000000	

In [36]:
```
df_daytime_only.LES_Snowfall.value_counts()
```

Out[36]:
```
0.0    15696
1.0      344
Name: LES_Snowfall, dtype: int64
```

I reckon it looks alright? We can then work on checking the correlations between the features.

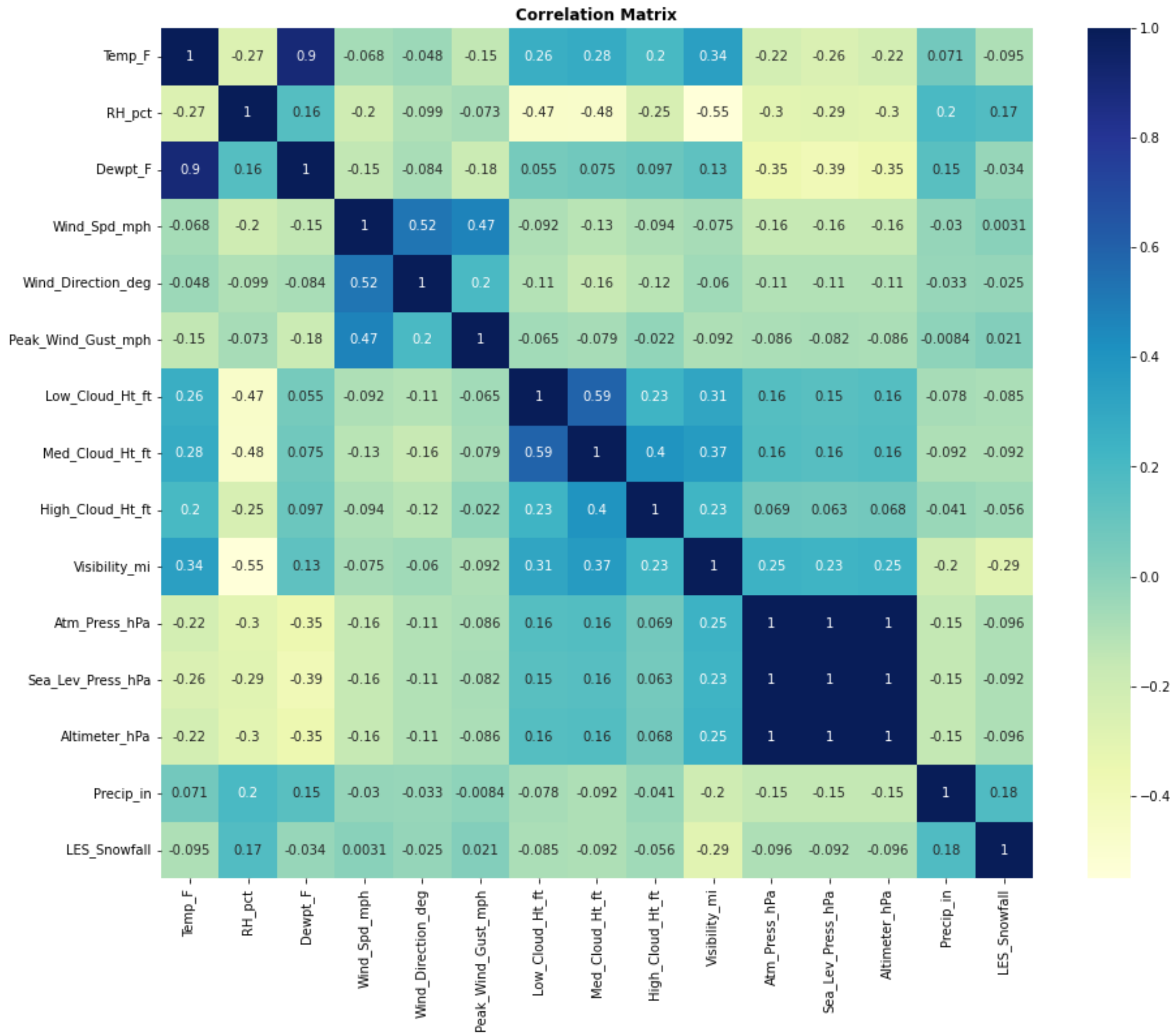
---

4. Correlations Between Features

In [37]:
```
# Correlation
correlation_matrix = df_daytime_only.corr(method = 'pearson')
plt.subplots(figsize=(15,12))

# Heatmap
sns.heatmap(correlation_matrix, annot = True, cmap = "YlGnBu")
plt.title("Correlation Matrix", size = 12, weight = 'bold')
```

```
/tmp/ipykernel_18656/113768846.py:2: FutureWarning: The default value of numer
ic_only in DataFrame.corr is deprecated. In a future version, it will default
to False. Select only valid columns or specify the value of numeric_only to si
lence this warning.
  correlation_matrix = df_daytime_only.corr(method = 'pearson')
```

Out[37]:
```
Text(0.5, 1.0, 'Correlation Matrix')
```



Observations from the above correlation plots:

- Few features are very heavily correated with each other (score >= 0.50)
  - **Temp_F** is highly correlated with **Dewpt_F**
  - **Wind_Spd_mph** is highly correlated with **Wind_Direction_deg**
  - **Atm_Press_hPa**, **Sea_Lev_Press_hPa**, and **Altimeter_hPa** are highly correlated to each other
- We also note some strong `negative correlation`, but all of them are greater than -0.5, hence we do not drop those features

We can drop the above columns since they imply to the same information, and keeping them as features will increase the model size.

But before doing this, let's work on **Atm_Press_hPa**, **Sea_Lev_Press_hPa**, and **Altimeter_hPa**, to see what is actually going on.

They are not identical to each other, but by nature, we know that they should be highly correlated. So, we are going to drop:

- **Dewpt_F**

- **Sea_Lev_Press_hPa** and **Altimeter_hPa**

We are being a little bit conservative here at the moment. The threshold for what constitutes "high" correlation can depend on the specific context and the dataset, but a common rule of thumb is to consider variables with a correlation coefficient above 0.8 or 0.9 to be highly correlated. However, there's no hard and fast rule, and the specific requirements of your project might necessitate a different threshold.

In [38]:
```
df_daytime_only = df_daytime_only.drop(['Dewpt_F', 'Sea_Lev_Press_hPa', 'Altime
df_daytime_only = df_daytime_only.reset_index(drop=True)

# Information about dataset shape
print('Total observations: ', df_daytime_only.shape[0])
print('Total number of features: ', df_daytime_only.shape[1])
# df_daytime_only.head()
```

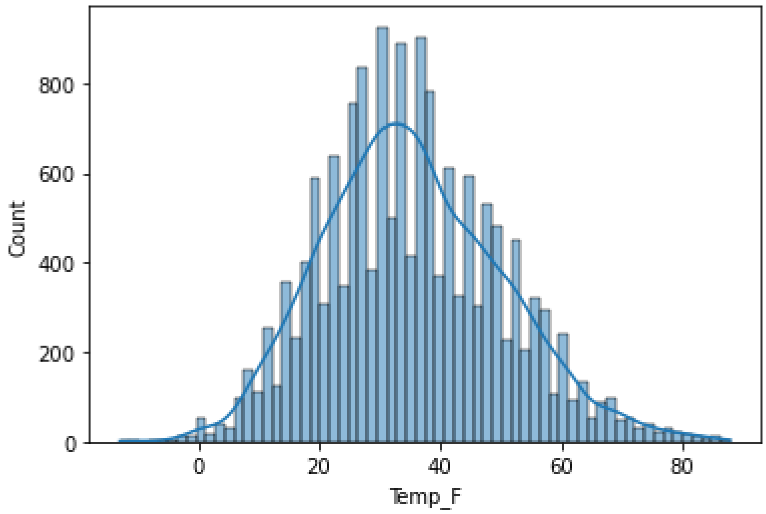
```
Total observations:  16040
Total number of features:  15
```

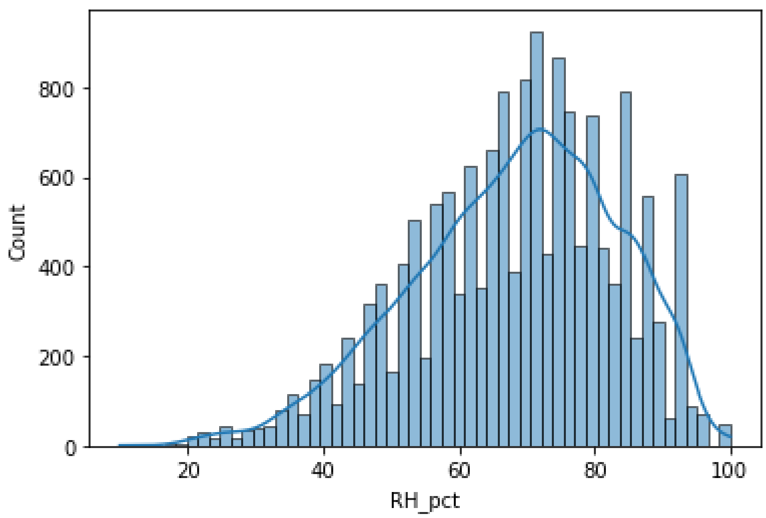
In [39]:
```
sns.pairplot(df_daytime_only)
```

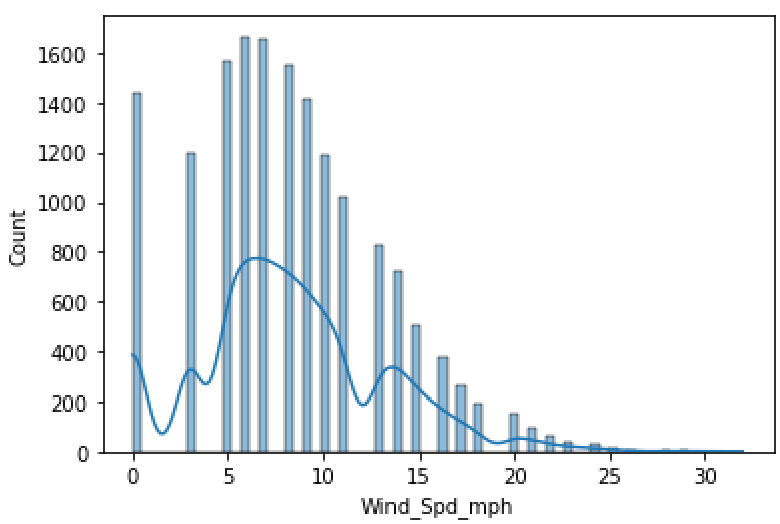
Out[39]:
```
<seaborn.axisgrid.PairGrid at 0x7f35234e9310>
```

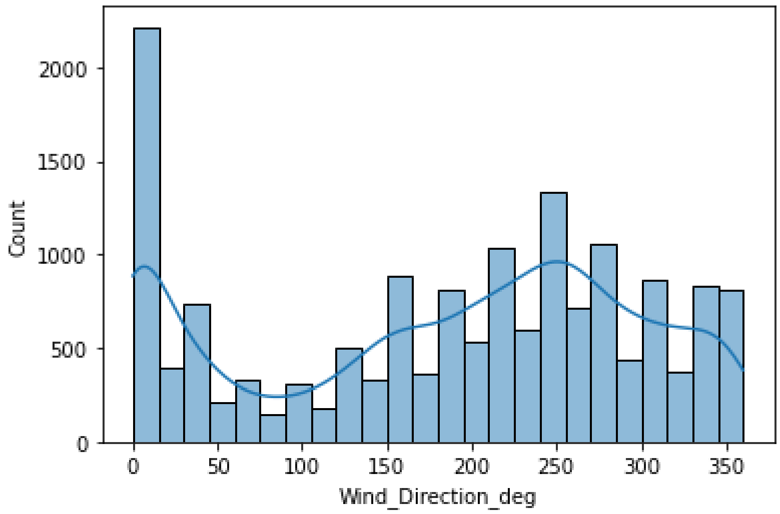
In [40]:

```
def distPlot(data):
    cols = data.columns[3:]
    for col in cols:
        sns.histplot(data[col], kde=True)
        plt.show()

distPlot(df_daytime_only)
```

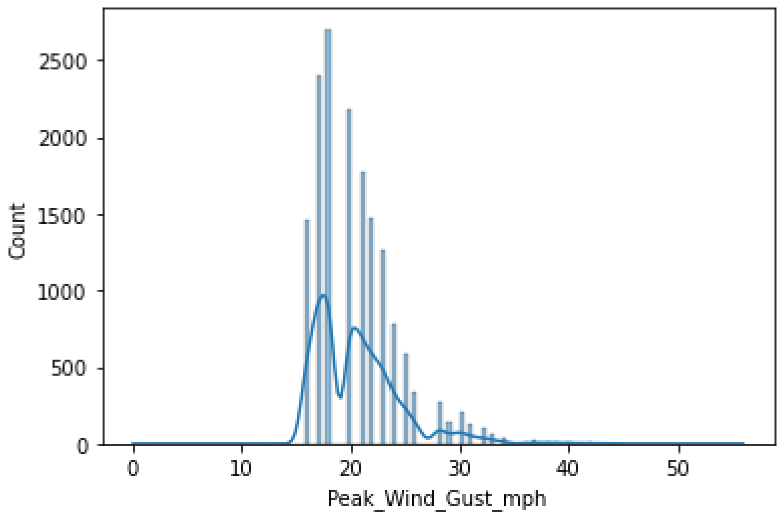

Count
Temp_F


Count
RH_pct

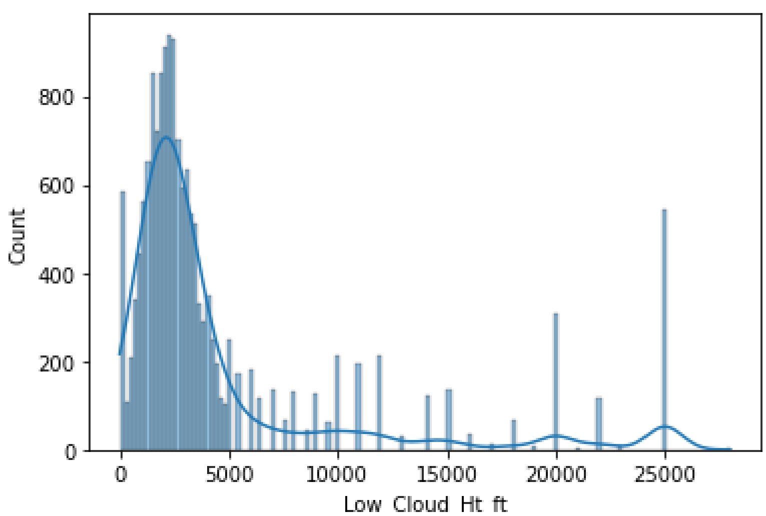

Count
Wind_Spd_mph



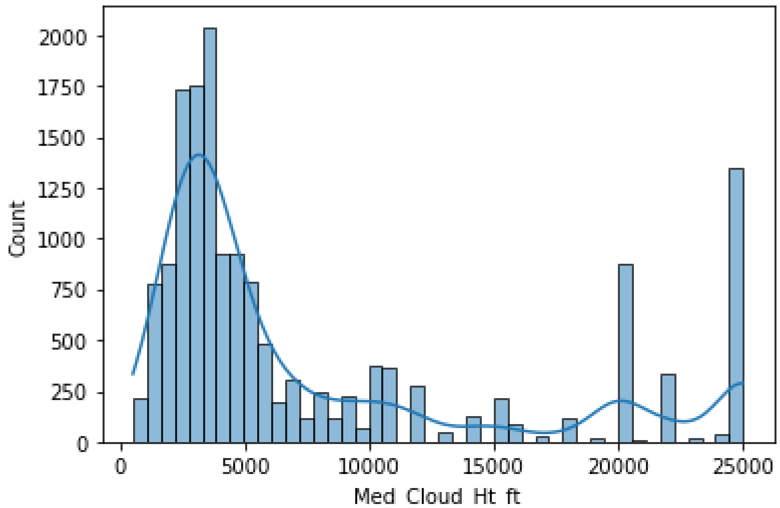
0
500
1000
1500
2000
Count
0
50
100
150
200
250
300
350
Wind_Direction_deg

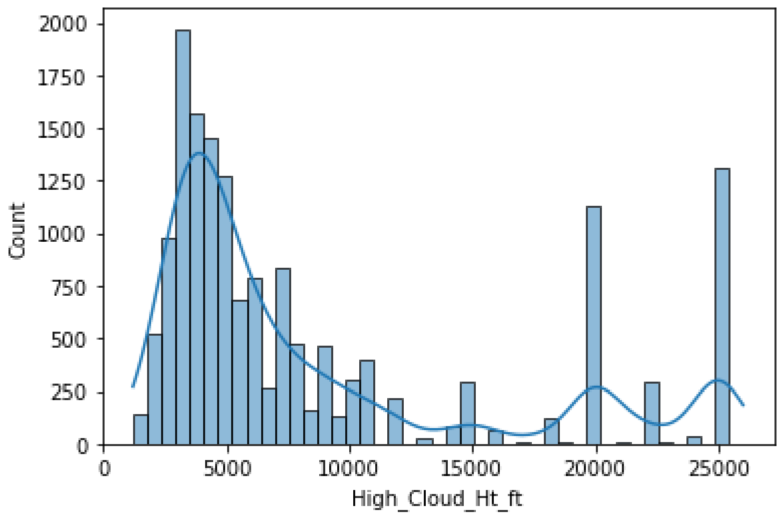


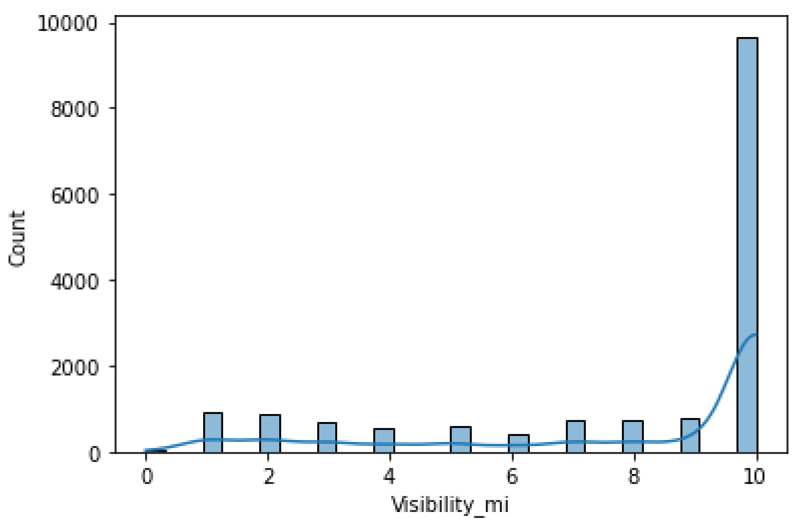
0
500
1000
1500
2000
2500
Count
0
10
20
30
40
50
Peak_Wind_Gust_mph

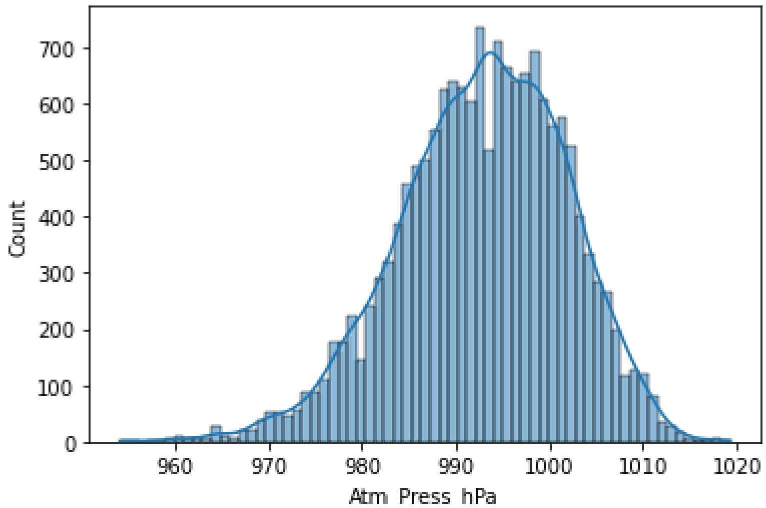


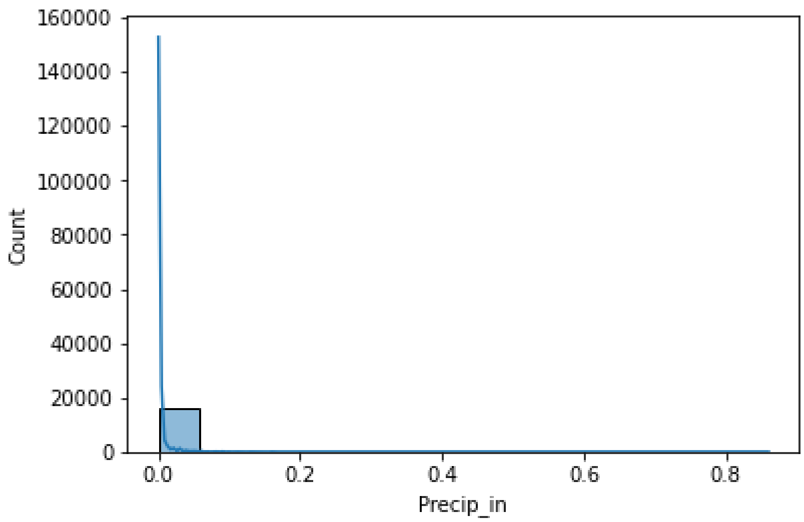
0
200
400
600
800
Count
0
5000
10000
15000
20000
25000
Low_Cloud_Ht_ft

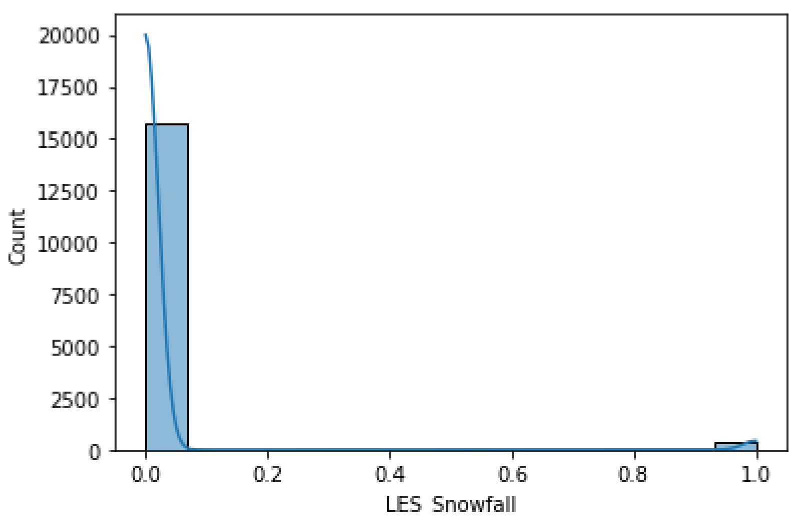

Count
Med_Cloud_Ht_ft


Count
High_Cloud_Ht_ft


Count
Visibility_mi





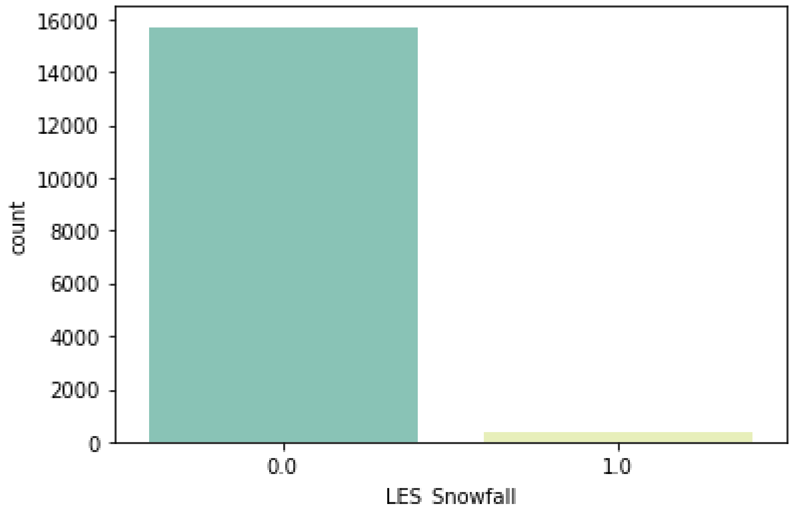


In [41]:
```
df_daytime_only['LES_Snowfall'].value_counts()
```

Out[41]:
```
0.0    15696
1.0      344
Name: LES_Snowfall, dtype: int64
```

In [42]:
```
sns.countplot(x = df_daytime_only['LES_Snowfall'], palette=["#7fcdbb", "#edf8b1
```

Out[42]:
```
<Axes: xlabel='LES_Snowfall', ylabel='count'>
```



---

5. Feature Engineering: Precipitation

Adding a New Column For Precipitation

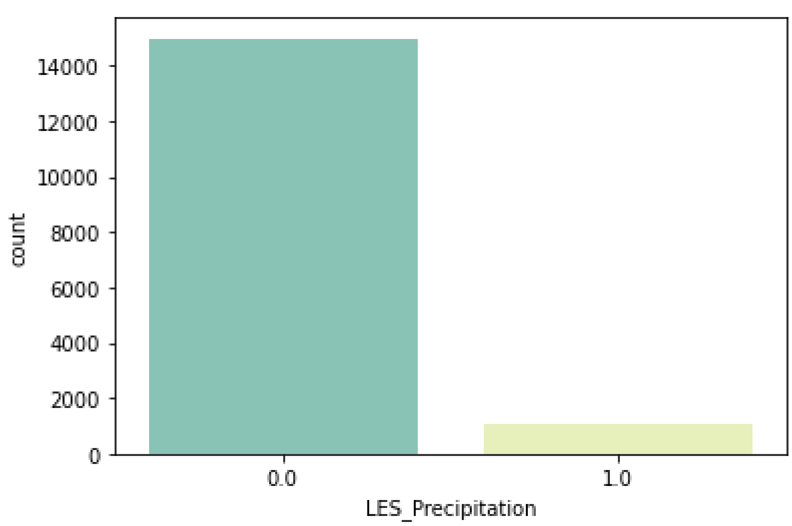
There is no fancy masking being applied yet. We will do that in another experiment.

In [43]:
```
import os
os.environ["TF_GPU_ALLOCATOR"]="cuda_malloc_async"
```

In [44]:
```
df_daytime_only.loc[df_daytime_only['Precip_in'] > 0, 'LES_Precipitation'] = 1
df_daytime_only.loc[df_daytime_only['Precip_in'] <= 0, 'LES_Precipitation'] = (
# df_daytime_only
```

In [45]:
```
sns.countplot(x = df_daytime_only['LES_Precipitation'], palette=["#7fcdbb", "#e
```

Out[45]:
```
<Axes: xlabel='LES_Precipitation', ylabel='count'>
```



In [46]:
```
import tensorflow as tf
from tensorflow import keras
```

```
from tensorflow.keras import layers

import io
import imageio
from IPython.display import Image, display
from ipywidgets import widgets, Layout, HBox
```

In [47]:
```
from tqdm import tqdm
import cv2

images = []
for idx in tqdm(range(df_daytime_only.shape[0])):
    # im shape -> (64, 64)
    im = cv2.imread('data_dir/lake-michigan-images-64/' + str(idx) + '.png')
    # Storing 1 channel, since the images are grayscale, and cropping
    images.append(im[8:-8,8:-8,0])
    # images shape -> (35, 64, 64)
```

```
100%|██████████| 16040/16040 [00:00<00:00, 18463.93it/s]
```

10. Predicting rain from past imagery *and* meteo

We're going to start with one daytime's worth of cloud imagery, and one datyime plus one nighttime worth's of meteo data.

We're going to use a ConvLSTM2D for imagery, and an LSTM for meteo.

Instead of predicting cloud frames, which we know is challenging based on our past experiments, we're going to attempt to predict daily precipitation.

So now I need to go back to my original dataset, which includes nighttime meteo data:

Data prep for cloud imagery and meteo datasets

Meteo training and validation

We remove some highly correlated features, and redundant ones.

Specifically, we will remove 'Date_UTC', 'Time_UTC', 'Date_CST', 'Time_CST', 'File_name_for_1D_lake', 'File_name_for_2D_lake', 'Lake_data_1D', 'Lake_data_2D', 'Dewpt_F', 'Peak_Wind_Gust_mph', and maybe 'Altimeter_hPa' because highly correlated with 'Atm_Press_hPa'.

Our network will consists of two networks, a ConvLSTM2D network for Cloud imagery, and an LSTM network for meteo data.

Each observation will consists of sequences: A sequence of 8 daytime hours for the imagery network, and a sequence of 24 hours for the meteo network.

First, we are going to attempt to determine if it rains at all the next day, from information from the previous day (imagery *and* meteo).

We are going to say that it rains on any day if it rains for at least one hour and more than 10% of...

If successful, we can attempt to push the boundary and predict longer into the future.

This is how we can going to create our LSTM tensor for meteo data:

In [48]:
```
meteo_les = df_single_station.drop(
    [ 'Date_UTC', 'Time_UTC', 'Date_CST', 'Time_CST', 'File_name_for_1D_lake',
     'Lake_data_1D', 'Dewpt_F', 'Sea_Lev_Press_hPa', 'Altimeter_hPa', 'data_usa
       'cloud_exist' ], axis=1)
```

In [49]:
```
meteo_les.head()
```

Out[49]:

	Temp_F	RH_pct	Wind_Spd_mph	Wind_Direction_deg	Peak_Wind_Gust_mph	Low_Cloud_Ht
0	51.0	92.0	0.0	0.0	0.0	370
1	48.0	96.0	0.0	0.0	0.0	370
2	49.0	92.0	3.0	220.0	3.0	370
3	48.0	100.0	0.0	0.0	0.0	250
4	50.0	92.0	3.0	180.0	3.0	700

In [50]:
```
len(meteo_les)
```

Out[50]:
```
48121
```

In [51]:
```
# x3 = tf.keras.preprocessing.timeseries_dataset_from_array(meteo_les, None, 24
#                                                             batch_size=50000)
```

In [52]:
```
# sequence_length: Length of the output sequences (in number of timesteps).
# sequence_stride: Period between successive output sequences. For stride s, ou
x3 = tf.keras.preprocessing.timeseries_dataset_from_array(meteo_les, None, sequ
                                                           batch_size=50000)
```

```
2023-08-19 18:35:58.211685: I tensorflow/core/platform/cpu_feature_guard.cc:19
3] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical operati
ons:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate co
mpiler flags.
2023-08-19 18:35:58.685102: I tensorflow/core/common_runtime/gpu/gpu_process_s
tate.cc:222] Using CUDA malloc Async allocator for GPU: 0
2023-08-19 18:35:58.685307: I tensorflow/core/common_runtime/gpu/gpu_device.c
c:1532] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 46524
MB memory:  -> device: 0, name: NVIDIA A40, pci bus id: 0000:22:00.0, compute
capability: 8.6
```

In [53]:
```
for batch in x3:
    print(batch.shape)
```

```
    print('--')
```

```
(2004, 48, 11)
--
```

So we have 2005 observations of 24 hours of meteo data consisting of 11 features.

But first we need to split our dataset into training and validation.

We agreed to use the first 30,000 rows of `filtered_les` as training data. Since 8 observations of that dataset correspond to 24 observations of the `meteo_les` dataset, we have 3 times more meteo observations than imagery, So we will use:

In [54]:
```
# meteo_train_batched = tf.keras.preprocessing.timeseries_dataset_from_array(me
#                                                                    sampling_r
```

In [55]:
```
meteo_train_batched = tf.keras.preprocessing.timeseries_dataset_from_array(mete
                                                                     sampling_rat
```

In [56]:
```
## meteo_train = None
for batch in meteo_train_batched:
    meteo_train = batch
    print(meteo_train.shape)
    print('--')
```

```
(1686, 48, 11)
--
```

Since we started our validation dataset for imagery at index 13,050 and we had 15,959 instances of imagery, let's agree to use the last 2,500 instances of imagery as our validation dataset (skipping some intermediate `nan` instances). That corresponds to $2500 \times 3 = 7500$ rows of meteo data.

In [57]:
```
# meteo_val_batched = tf.keras.preprocessing.timeseries_dataset_from_array(mete
#                                                                    sampling_r
```

In [58]:
```
meteo_val_batched = tf.keras.preprocessing.timeseries_dataset_from_array(meteo_
                                                                     sampling_rat
```

In [59]:
```
meteo_val = None
for batch in meteo_val_batched:
    meteo_val = batch
    print(meteo_val.shape)
    print('--')
```

```
(311, 48, 11)
--
```

So we have about 3 times more traiing data than test data.

Cloud imagery training and validation datasets

We can probably use `les_filtered` to gather our imagery data, just liked we did previously. But now our training dataset will consist of 8 hours of imagery and the label will be rain or not *the next day*.

Let's create our imagery training data:

In [60]:
```
# cloud_train_batched = tf.keras.preprocessing.timeseries_dataset_from_array(in
#                                                                             sa
```

In [61]:
```
cloud_train_batched = tf.keras.preprocessing.timeseries_dataset_from_array(imag
                                                                           samp
```

In [62]:
```
cloud_train = None
for batch in cloud_train_batched:
    cloud_train = batch
    cloud_train = np.expand_dims(cloud_train, axis=-1)
    print(cloud_train.shape)
    cloud_train = cloud_train / 255
    print('--')
```

```
(1686, 16, 48, 48, 1)
--
```

And test data:

In [63]:
```
# cloud_val_batched = tf.keras.preprocessing.timeseries_dataset_from_array(imag
#                                                                          samp
```

In [64]:
```
cloud_val_batched = tf.keras.preprocessing.timeseries_dataset_from_array(images
                                                                         sampli
```

In [65]:
```
cloud_val = None
for batch in cloud_val_batched:
    cloud_val = batch
    cloud_val = np.expand_dims(cloud_val, axis=-1)
    print(cloud_val.shape)
    cloud_val = cloud_val / 255
    print('--')
```

```
(311, 16, 48, 48, 1)
--
```

Final rain classification label

Finally, let's create our label:

This is how much precipitation in 24 hours:

In [66]:
```
rain_train = []
for batch in meteo_train:
    batch =batch
    batch = np.expand_dims(batch, axis=0)
    for i in range(batch.shape[0]):
        rain_train.append(sum(batch[i,:,-1]))
#                rain_train.append(sum(batch[i,:,-1].numpy()))


len(rain_train)
```

Out[66]: 1686

Let's train for *serious* rain, more than 0.10 precipitation per day (is that enough?):

In [67]:
```
rain_train_b = [1 if 0.10 <= r else 0 for r in rain_train]
```

In [68]:
```
rain_train_c = np.array(rain_train_b)
rain_train_c.shape
```

Out[68]: (1686,)

In [69]:
```
rain_val = []
for batch in meteo_val:
    batch = np.expand_dims(batch, axis=0)
    for i in range(batch.shape[0]):
        rain_val.append(sum(batch[i,:,-1]))


print(batch.shape[0])
len(rain_val)
```

```
1
```

Out[69]: 311

In [70]:
```
rain_val_b = [1 if 0.10 <= r else 0 for r in rain_val]
```

In [71]:
```
rain_val_c = np.array(rain_val_b)
rain_val_c.shape
```

Out[71]: (311,)

Network

Imagery Network

In [72]:
```
cloud_train.shape, rain_train_c.shape, cloud_val.shape, rain_val_c.shape
```

Out[72]: ((1686, 16, 48, 48, 1), (1686,), (311, 16, 48, 48, 1), (311,))

First, let's learn how shapes get transformed through convolution.

`TO-DO`: Need to figure out why it uses 21-long

Assume our input consists of a 21-long sequence of 48×48 images. One way to process the sequences is to add the sequence dimension as a channel and use the traditional `Conv2D` api.

Conv2D(`filters` , `kernel_size` , `strides=(1, 1)` , ...)

Note that the shape-shifting operators are `filters` and `strides` .

In [73]:
```
# from keras.layers import Dropout, GlobalAveragePooling2D, MaxPooling2D

# input_cnn = layers.Input(shape=(48,48,21))
# print("layers.Input(shape=(48,48,21))", input_cnn.shape)
# x = layers.Conv2D(3, (3, 3), (2,2), padding='same', activation='selu')(input_
# print("layers.Conv2D(3, (3, 3), (2,2)", x.shape)
# x = MaxPooling2D(pool_size=(2,2))(x)
# print("MaxPooling2D(pool_size=(2,2))", x.shape)
# x = layers.Conv2D(6, (3, 3), (2,2), padding='same', activation='selu')(x)
# print("layers.Conv2D(6, (3, 3), (2,2)", x.shape)
# x = MaxPooling2D(pool_size=(2,2))(x)
# print("MaxPooling2D(pool_size=(2,2)", x.shape)
# x = GlobalAveragePooling2D()(x)
# print("GlobalAveragePooling2D", x.shape)
# x.shape
```

In [74]:
```
from keras.layers import Dropout, GlobalAveragePooling2D, MaxPooling2D

input_cnn = layers.Input(shape=(48,48,45))
print("layers.Input(shape=(48,48,21))", input_cnn.shape)
x = layers.Conv2D(3, (3, 3), (2,2), padding='same', activation='selu')(input_cn
print("layers.Conv2D(3, (3, 3), (2,2)", x.shape)
x = MaxPooling2D(pool_size=(2,2))(x)
print("MaxPooling2D(pool_size=(2,2))", x.shape)
x = layers.Conv2D(6, (3, 3), (2,2), padding='same', activation='selu')(x)
print("layers.Conv2D(6, (3, 3), (2,2)", x.shape)
x = MaxPooling2D(pool_size=(2,2))(x)
print("MaxPooling2D(pool_size=(2,2)", x.shape)
x = GlobalAveragePooling2D()(x)
print("GlobalAveragePooling2D", x.shape)
x.shape
```

```
layers.Input(shape=(48,48,21)) (None, 48, 48, 45)
layers.Conv2D(3, (3, 3), (2,2) (None, 24, 24, 3)
MaxPooling2D(pool_size=(2,2)) (None, 12, 12, 3)
layers.Conv2D(6, (3, 3), (2,2) (None, 6, 6, 6)
MaxPooling2D(pool_size=(2,2) (None, 3, 3, 6)
GlobalAveragePooling2D (None, 6)
```

Out[74]:
```
TensorShape([None, 6])
```

Instead, if we want to use the more orthodox ConvLSTM2D(`filters` , `kernel_size` , `strides=(1, 1)` , ...) on 8-long sequences of images (with one gray channel):

In [75]:
```
cloud_train.shape, rain_train_c.shape, cloud_val.shape, rain_val_c.shape
```

Out[75]:
```
((1686, 16, 48, 48, 1), (1686,), (311, 16, 48, 48, 1), (311,))
```

In [76]:
```
cloud_train.shape[2:]
```

Out[76]:
```
(48, 48, 1)
```

We can stack 3 `ConvLSTM2D` layers with batch normalization, followed by a `Conv3D` layer for the spatiotemporal outputs.

Conv3D api is: `layers.Conv3D( filters, kernel_size, strides=(1, 1, 1), ...`.

Here are some examples:

In [77]:
```
# # The inputs are 28x28x28 volumes with a single channel, and the batch size i
# input_shape =(4, 28, 28, 28, 1)
# x = tf.random.normal(input_shape)
# print(x.shape)
# y = tf.keras.layers.Conv3D(2, 3, activation='relu', padding="same", input_sha
# print(y.shape)
```

In [78]:
```
# # With extended batch shape [4, 7], e.g. a batch of 4 videos of 3D frames, wi
# input_shape = (4, 7, 28, 28, 28, 1)
# x = tf.random.normal(input_shape)
# print(x.shape)
# y = tf.keras.layers.Conv3D(2, 3, activation='relu', padding="same", input_sha
# print(y.shape)
```

Ok, here's our stack of 3 `ConvLSTM2D` layers with batch normalization, followed by a `Conv3D` layer for the spatiotemporal outputs.

Since the padding is `same` and `stride` defaults to `(1,1)`, our images remain the same size through the stack.

In [79]:
```
# Construct the input layer with no definite frame size (None below could be re
inp = layers.Input(shape=(None, *cloud_train.shape[2:]))
print("layers.Input(shape=", inp.shape)

x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(5, 5),
    padding="same",
    return_sequences=True,
    activation="relu",
)(inp)
print("ConvLSTM2D filters=64, kernel_size=(5, 5)", x.shape)
x = layers.BatchNormalization()(x)
print("BatchNormalization", x.shape)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(3, 3),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
print("ConvLSTM2D filters=64, kernel_size=(3, 3)", x.shape)
x = layers.BatchNormalization()(x)
print("BatchNormalization", x.shape)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(1, 1),
```

```
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
print("ConvLSTM2D filters=64, kernel_size=(1, 1)", x.shape)
x = layers.Conv3D(
    filters=1, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
)(x)
print("Conv3D kernel_size=(3, 3, 3)", x.shape)
```

```
layers.Input(shape= (None, None, 48, 48, 1)
ConvLSTM2D filters=64, kernel_size=(5, 5) (None, None, 48, 48, 64)
BatchNormalization (None, None, 48, 48, 64)
ConvLSTM2D filters=64, kernel_size=(3, 3) (None, None, 48, 48, 64)
BatchNormalization (None, None, 48, 48, 64)
ConvLSTM2D filters=64, kernel_size=(1, 1) (None, None, 48, 48, 64)
Conv3D kernel_size=(3, 3, 3) (None, None, 48, 48, 1)
```

So this network is appropriate when the input is `z` number of observations, each one a sequence of `t` grayscale images of size 48×48, i.e. `(None=z, None=t, 48, 48, 1)`, *and the label is a sequence of exactly the same size*: `(None=z, None=t, 48, 48, 1)`.

If we want to predict rain or not, which is just a binary label, then we need a network that reduces from `(None=z, None=t, 48, 48, 1)` to `{None, q)`, where q will be the vector to be concatenated with the final meteo vector and then passed through a Dense layer for the final binary rain yes/no!

So it makes sense to *slowly* reduce the size of the image, *and* also to flatten the size of the sequence. We can slowly reduce the size of our images with strides larger than `(1,1)` in our convolutions, and we can flatten our sequence to a single dimension with `return_sequences=False` in our last convolution layer. For example, this way:

In [80]:
```
# Construct the input layer with no definite frame size (None below could be re
inp = layers.Input(shape=(None, *cloud_train.shape[2:]))
print("layers.Input(shape=", inp.shape)

# x = layers.ConvLSTM2D(
#     filters=64,
#     kernel_size=(5, 5),
#     strides=(2, 2),
#     padding="same",
#     return_sequences=True,
#     activation="relu",
# )(inp)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(7, 7),
    strides=(1, 1),
    padding="same",
    return_sequences=True,
    activation="relu",
)(inp)
x = layers.Dropout(0.2)(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(5, 5),
```

```
    strides=(1, 1),
    padding="same",
    return_sequences=True,
    activation="relu",
# )(inp)
)(x)
print("ConvLSTM2D filters=64, kernel_size=(5, 5), return_sequences=True", x.sha
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.2)(x)
print("BatchNormalization", x.shape)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(5, 5),
    strides=(2, 2),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(3, 3),
    strides=(2, 2),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.Dropout(0.2)(x)
x = layers.ConvLSTM2D(
    filters=32,
    kernel_size=(3, 3),
    strides=(1, 1),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
print("ConvLSTM2D filters=64, kernel_size=(3, 3), return_sequences=True", x.sha
x = layers.BatchNormalization()(x)
print("BatchNormalization", x.shape)
x = layers.ConvLSTM2D(
    filters=32,
    kernel_size=(1, 1),
    strides=(2, 2),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
print("ConvLSTM2D filters=64, kernel_size=(1, 1), return_sequences=True", x.sha
x = layers.Conv3D(
    filters=16, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
)(x)

# Note that MaxPooling2D takes in a 4D input and downsamples the input along it
# taking the maximum value over an input window (of size defined by pool_size)
# is shifted by strides along each dimension. The first dim is observattions, t
# the last dim is channels. So it downsamples only the two intermediate dimensi
# x = MaxPooling2D(pool_size=(2,2))(x[:, :, :, :, 0])
# .. it will work but it will assume that the number of channels is 6 and leave
# want to downsample the sequence dimension!

# Better to use an additional convolution layer with return_sequences=False
```

```
print("Conv3D kernel_size=(3, 3, 3)", x.shape)
x = layers.ConvLSTM2D(
    filters=16,
    kernel_size=(1, 1),
    strides=(2, 2),
    padding="same",
    return_sequences=False,
    activation="relu",
)(x)
x = layers.Dropout(0.2)(x)
print("ConvLSTM2D filters=1, kernel_size=(1, 1), return_sequences=False", x.sha
x = layers.BatchNormalization()(x)
print("BatchNormalization", x.shape)

#x = layers.Dense(1)(x)
#print("Dense", x.shape)
x = GlobalAveragePooling2D()(x)
print("GlobalAveragePooling2D", x.shape)
```

```
layers.Input(shape= (None, None, 48, 48, 1)
ConvLSTM2D filters=64, kernel_size=(5, 5), return_sequences=True (None, None,
48, 48, 64)
BatchNormalization (None, None, 48, 48, 64)
ConvLSTM2D filters=64, kernel_size=(3, 3), return_sequences=True (None, None,
12, 12, 32)
BatchNormalization (None, None, 12, 12, 32)
ConvLSTM2D filters=64, kernel_size=(1, 1), return_sequences=True (None, None,
6, 6, 32)
Conv3D kernel_size=(3, 3, 3) (None, None, 6, 6, 16)
ConvLSTM2D filters=1, kernel_size=(1, 1), return_sequences=False (None, 3, 3,
16)
BatchNormalization (None, 3, 3, 16)
GlobalAveragePooling2D (None, 16)
```

Meteo network

In [81]:
```
meteo_train.shape, rain_train_c.shape, meteo_val.shape, rain_val_c.shape
```

Out[81]:
```
(TensorShape([1686, 48, 11]), (1686,), TensorShape([311, 48, 11]), (311,))
```

In [82]:
```
meteo_train.shape[1:]
```

Out[82]:
```
TensorShape([48, 11])
```

In [83]:
```
# RNN = layers.LSTM
# hidden_size = 8
# data_shape = (24, 11)
# data = layers.Input(shape= data_shape)
# meteo_inp = layers.Input(shape=(None, *meteo_train.shape[1:]))
# print("layers.Input(shape=", meteo_inp.shape)
# lstm1 = RNN(hidden_size, input_shape=(24, data_shape[1]), return_sequences= T
# lstm2 = RNN(hidden_size, input_shape=(24, hidden_size), return_sequences= Fal
# lstm2.shape
```

In [84]:
```
RNN = layers.LSTM
hidden_size = 16
data_shape = (48, 11)
data = layers.Input(shape= data_shape)
```

```
meteo_inp = layers.Input(shape=(None, *meteo_train.shape[1:]))
print("layers.Input(shape=", meteo_inp.shape)

from tensorflow.keras.layers import Bidirectional
lstm1 = Bidirectional(RNN(data_shape[1], input_shape=(48, data_shape[1]), retur
lstm2 = Bidirectional(RNN(hidden_size, input_shape=(48, data_shape[1]), return_
lstm3 = Bidirectional(RNN(hidden_size, input_shape=(48, hidden_size), return_se
lstm4 = Bidirectional(RNN(hidden_size, input_shape=(48, hidden_size), return_se
lstm5 = Bidirectional(RNN(hidden_size, input_shape=(48, hidden_size), return_se


# lstm1 = RNN(hidden_size, input_shape=(24, data_shape[1]), return_sequences= T
# lstm2 = RNN(hidden_size, input_shape=(24, hidden_size), return_sequences= Tru
# lstm3 = RNN(hidden_size, input_shape=(24, hidden_size), return_sequences= Tru
# lstm4 = RNN(hidden_size, input_shape=(24, hidden_size), return_sequences= Fal
# lstm2 = RNN(hidden_size, input_shape=(24, hidden_size), return_sequences= Fal
# lstm2.shape
```

```
layers.Input(shape= (None, None, 48, 11)
```

Imagery + meteo

Our final classification into rain or no rain, based on a balanced amount of information from both imagery and meteo:

In [85]:

```
# Flatten the output of CNN
#flattened = layers.Flatten()(conv6)

# Connect the CNN output and RNN output to a dense layer with 1 neuron for fina
final = layers.Concatenate(axis=1)([lstm5, x])
print("layers.Concatenate(axis=1)([lstm5, x])", final.shape)
out = layers.Dense(1, activation='sigmoid')(final)
print("layers.Dense(1)", out.shape)
```

```
layers.Concatenate(axis=1)([lstm5, x]) (None, 48)
layers.Dense(1) (None, 1)
```

In [86]:

```
# Using both, images and numerical data as input
#inp = layers.Input(shape=(None, *cloud_train.shape[2:]))
#data = layers.Input(shape= (24, 11))
model = keras.models.Model([inp, data], out)
#model = keras.models.Model(inp, x)

# Build model
model.compile(loss=keras.losses.binary_crossentropy, optimizer=keras.optimizers
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, None, 48, 4  0           []
                                8, 1)]

 conv_lstm2d_3 (ConvLSTM2D)     (None, None, 48, 48  815616      ['input_3[0]
[0]']
                                , 64)

 dropout (Dropout)              (None, None, 48, 48  0           ['conv_lstm2d
_3[0][0]']
                                , 64)

 conv_lstm2d_4 (ConvLSTM2D)     (None, None, 48, 48  819456      ['dropout[0]
[0]']
                                , 64)

 batch_normalization_2 (BatchNo  (None, None, 48, 48  256        ['conv_lstm2d
_4[0][0]']
 rmalization)                   , 64)

 dropout_1 (Dropout)            (None, None, 48, 48  0           ['batch_norma
lization_2[0][0]']
                                , 64)

 conv_lstm2d_5 (ConvLSTM2D)     (None, None, 24, 24  819456      ['dropout_1
[0][0]']
                                , 64)

 conv_lstm2d_6 (ConvLSTM2D)     (None, None, 12, 12  295168      ['conv_lstm2d
_5[0][0]']
                                , 64)

 dropout_2 (Dropout)            (None, None, 12, 12  0           ['conv_lstm2d
_6[0][0]']
                                , 64)

 conv_lstm2d_7 (ConvLSTM2D)     (None, None, 12, 12  110720      ['dropout_2
[0][0]']
                                , 32)

 batch_normalization_3 (BatchNo  (None, None, 12, 12  128        ['conv_lstm2d
_7[0][0]']
 rmalization)                   , 32)

 input_4 (InputLayer)           [(None, 48, 11)]     0           []

 conv_lstm2d_8 (ConvLSTM2D)     (None, None, 6, 6,   8320        ['batch_norma
lization_3[0][0]']
                                32)

 bidirectional (Bidirectional)  (None, 48, 22)       2024        ['input_4[0]
[0]']

 conv3d_1 (Conv3D)              (None, None, 6, 6,   13840       ['conv_lstm2d
_8[0][0]']
```

```
                                16)

 bidirectional_1 (Bidirectional  (None, 48, 32)      4992        ['bidirection
al[0][0]']
 )

 conv_lstm2d_9 (ConvLSTM2D)     (None, 3, 3, 16)     2112        ['conv3d_1[0]
[0]']

 bidirectional_2 (Bidirectional  (None, 48, 32)      6272        ['bidirection
al_1[0][0]']
 )

 dropout_3 (Dropout)            (None, 3, 3, 16)     0           ['conv_lstm2d
_9[0][0]']

 bidirectional_3 (Bidirectional  (None, 48, 32)      6272        ['bidirection
al_2[0][0]']
 )

 batch_normalization_4 (BatchNo  (None, 3, 3, 16)    64          ['dropout_3
[0][0]']
 rmalization)

 bidirectional_4 (Bidirectional  (None, 32)          6272        ['bidirection
al_3[0][0]']
 )

 global_average_pooling2d_1 (Gl  (None, 16)          0           ['batch_norma
lization_4[0][0]']
 obalAveragePooling2D)

 concatenate (Concatenate)      (None, 48)           0           ['bidirection
al_4[0][0]',
                                                                  'global_aver
age_pooling2d_1[0][0
                                                                 ]']

 dense (Dense)                  (None, 1)            49          ['concatenate
[0][0]']

==============================================================================
====================
Total params: 2,911,017
Trainable params: 2,910,793
Non-trainable params: 224
______________________________________________________________________________
____________________
```

Training

In [87]:

```
# # Define some callbacks to improve training
# early_stopping = keras.callbacks.EarlyStopping(monitor="val_loss", patience=1
# reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor="val_loss", patience=1(

# # Define modifiable training hyperparameters
# epochs = 100
# batch_size = 16
```

```
# from datetime import datetime
# now = datetime.now()
# current_time = now.strftime("%H:%M:%S")
# print("Started training at", current_time)

# # Fit the model to the training data
# history = model.fit(
#     [cloud_train, meteo_train],
#     rain_train_c,
#     batch_size=batch_size,
#     epochs=epochs,
#     validation_data=([cloud_val, meteo_val], rain_val_c),
#     callbacks=[early_stopping, reduce_lr],
# )

# now = datetime.now()
# current_time = now.strftime("%H:%M:%S")
# print("Finished training at", current_time)
```

In [88]:

```
cloud_train.shape, meteo_train.shape
```

Out[88]:

```
((1686, 16, 48, 48, 1), TensorShape([1686, 48, 11]))
```

In [89]:

```
# Define some callbacks to improve training
early_stopping = keras.callbacks.EarlyStopping(monitor="val_loss", patience=15)
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor="val_loss", patience=10)

# Define modifiable training hyperparameters
epochs = 100
batch_size = 32

from datetime import datetime
now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Started training at", current_time)

# Fit the model to the training data
history = model.fit(
    [cloud_train, meteo_train],
    rain_train_c,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=([cloud_val, meteo_val], rain_val_c),
    callbacks=[early_stopping, reduce_lr],
)

now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Finished training at", current_time)
```

```
Started training at 18:36:06
Epoch 1/100
2023-08-19 18:36:27.766771: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384]
Loaded cuDNN version 8101
2023-08-19 18:36:29.633725: I tensorflow/stream_executor/cuda/cuda_blas.cc:178
6] TensorFloat-32 will be used for the matrix multiplication. This will only b
e logged once.
```

```
53/53 [==============================] - 82s 1s/step - loss: 0.5652 - accurac
y: 0.7461 - val_loss: 0.5728 - val_accuracy: 0.7010 - lr: 0.0010
Epoch 2/100
53/53 [==============================] - 57s 1s/step - loss: 0.5285 - accurac
y: 0.7533 - val_loss: 0.5464 - val_accuracy: 0.7395 - lr: 0.0010
Epoch 3/100
53/53 [==============================] - 57s 1s/step - loss: 0.5179 - accurac
y: 0.7539 - val_loss: 0.5209 - val_accuracy: 0.7524 - lr: 0.0010
Epoch 4/100
53/53 [==============================] - 57s 1s/step - loss: 0.4987 - accurac
y: 0.7539 - val_loss: 0.5247 - val_accuracy: 0.7203 - lr: 0.0010
Epoch 5/100
53/53 [==============================] - 58s 1s/step - loss: 0.5016 - accurac
y: 0.7533 - val_loss: 0.5128 - val_accuracy: 0.7299 - lr: 0.0010
Epoch 6/100
53/53 [==============================] - 58s 1s/step - loss: 0.4846 - accurac
y: 0.7669 - val_loss: 0.5484 - val_accuracy: 0.7524 - lr: 0.0010
Epoch 7/100
53/53 [==============================] - 58s 1s/step - loss: 0.4810 - accurac
y: 0.7663 - val_loss: 0.4860 - val_accuracy: 0.7621 - lr: 0.0010
Epoch 8/100
53/53 [==============================] - 57s 1s/step - loss: 0.4757 - accurac
y: 0.7770 - val_loss: 0.5781 - val_accuracy: 0.7267 - lr: 0.0010
Epoch 9/100
53/53 [==============================] - 57s 1s/step - loss: 0.4716 - accurac
y: 0.7800 - val_loss: 0.5329 - val_accuracy: 0.7460 - lr: 0.0010
Epoch 10/100
53/53 [==============================] - 57s 1s/step - loss: 0.4810 - accurac
y: 0.7800 - val_loss: 0.4856 - val_accuracy: 0.7781 - lr: 0.0010
Epoch 11/100
53/53 [==============================] - 57s 1s/step - loss: 0.4682 - accurac
y: 0.7794 - val_loss: 0.5165 - val_accuracy: 0.7395 - lr: 0.0010
Epoch 12/100
53/53 [==============================] - 58s 1s/step - loss: 0.4731 - accurac
y: 0.7800 - val_loss: 0.4852 - val_accuracy: 0.7846 - lr: 0.0010
Epoch 13/100
53/53 [==============================] - 57s 1s/step - loss: 0.4657 - accurac
y: 0.7859 - val_loss: 0.4963 - val_accuracy: 0.7781 - lr: 0.0010
Epoch 14/100
53/53 [==============================] - 57s 1s/step - loss: 0.4582 - accurac
y: 0.7859 - val_loss: 0.4790 - val_accuracy: 0.7685 - lr: 0.0010
Epoch 15/100
53/53 [==============================] - 57s 1s/step - loss: 0.4564 - accurac
y: 0.7865 - val_loss: 0.5149 - val_accuracy: 0.7749 - lr: 0.0010
Epoch 16/100
53/53 [==============================] - 58s 1s/step - loss: 0.4525 - accurac
y: 0.7912 - val_loss: 0.5079 - val_accuracy: 0.7974 - lr: 0.0010
Epoch 17/100
53/53 [==============================] - 57s 1s/step - loss: 0.4400 - accurac
y: 0.8019 - val_loss: 0.4997 - val_accuracy: 0.7653 - lr: 0.0010
Epoch 18/100
53/53 [==============================] - 58s 1s/step - loss: 0.4594 - accurac
y: 0.7924 - val_loss: 0.5437 - val_accuracy: 0.7814 - lr: 0.0010
Epoch 19/100
53/53 [==============================] - 57s 1s/step - loss: 0.4678 - accurac
y: 0.7835 - val_loss: 0.5009 - val_accuracy: 0.7781 - lr: 0.0010
Epoch 20/100
53/53 [==============================] - 58s 1s/step - loss: 0.4467 - accurac
y: 0.8001 - val_loss: 0.4933 - val_accuracy: 0.7814 - lr: 0.0010
Epoch 21/100
```

```
53/53 [==============================] - 58s 1s/step - loss: 0.4478 - accurac
y: 0.7900 - val_loss: 0.5026 - val_accuracy: 0.7814 - lr: 0.0010
Epoch 22/100
53/53 [==============================] - 57s 1s/step - loss: 0.4457 - accurac
y: 0.7942 - val_loss: 0.4834 - val_accuracy: 0.7878 - lr: 0.0010
Epoch 23/100
53/53 [==============================] - 57s 1s/step - loss: 0.4542 - accurac
y: 0.7894 - val_loss: 0.4997 - val_accuracy: 0.7556 - lr: 0.0010
Epoch 24/100
53/53 [==============================] - 58s 1s/step - loss: 0.4414 - accurac
y: 0.7936 - val_loss: 0.5152 - val_accuracy: 0.7685 - lr: 0.0010
Epoch 25/100
53/53 [==============================] - 57s 1s/step - loss: 0.4224 - accurac
y: 0.8037 - val_loss: 0.5040 - val_accuracy: 0.7621 - lr: 1.0000e-04
Epoch 26/100
53/53 [==============================] - 57s 1s/step - loss: 0.4149 - accurac
y: 0.8102 - val_loss: 0.5014 - val_accuracy: 0.7653 - lr: 1.0000e-04
Epoch 27/100
53/53 [==============================] - 57s 1s/step - loss: 0.4099 - accurac
y: 0.8144 - val_loss: 0.4982 - val_accuracy: 0.7685 - lr: 1.0000e-04
Epoch 28/100
53/53 [==============================] - 58s 1s/step - loss: 0.4093 - accurac
y: 0.8132 - val_loss: 0.5006 - val_accuracy: 0.7781 - lr: 1.0000e-04
Epoch 29/100
53/53 [==============================] - 58s 1s/step - loss: 0.4032 - accurac
y: 0.8144 - val_loss: 0.5034 - val_accuracy: 0.7717 - lr: 1.0000e-04
Finished training at 19:04:16
```

That looks pretty good :-) It looks like I can keep on training, too! The first couple epoch took too long since I was running out of memory (forgot to close other notebooks), so the memory overflew into the RAM off from the GPU.

Let's look at accuracy:

In [90]:
```
cloud_val.shape, tf.convert_to_tensor(cloud_val).shape, meteo_val.shape
```

Out[90]:
```
((311, 16, 48, 48, 1),
 TensorShape([311, 16, 48, 48, 1]),
 TensorShape([311, 48, 11]))
```

In [91]:
```
# Select a random example from the cloud imagery validation dataset
example_index = np.random.choice(range(len(cloud_val)), size=1)[0]
print("Picked index", example_index,"from validation dataset.")
example_clouds = tf.convert_to_tensor(cloud_val[example_index]) # all 8 frames

# Select the same example from the meteo validation dataset
example_meteo = meteo_val[example_index]

# input
#np.expand_dims([example_clouds, example_meteo], axis=0)
# [example_clouds, example_meteo]
```

```
Picked index 191 from validation dataset.
```

In [92]:
```
# pred_input_combo = np.expand_dims([example_clouds, example_meteo], axis = 0)
```

`TO-DO`: See if can fix it later down the line.

In [93]:
```
# pred_input_combo = np.array(pred_input_combo, dtype=object)
```

In [94]:
```
# tf.convert_to_tensor(pred_input_combo, dtype=tf.float32)
```

In [95]:
```
# model.predict(pred_input_combo)
```

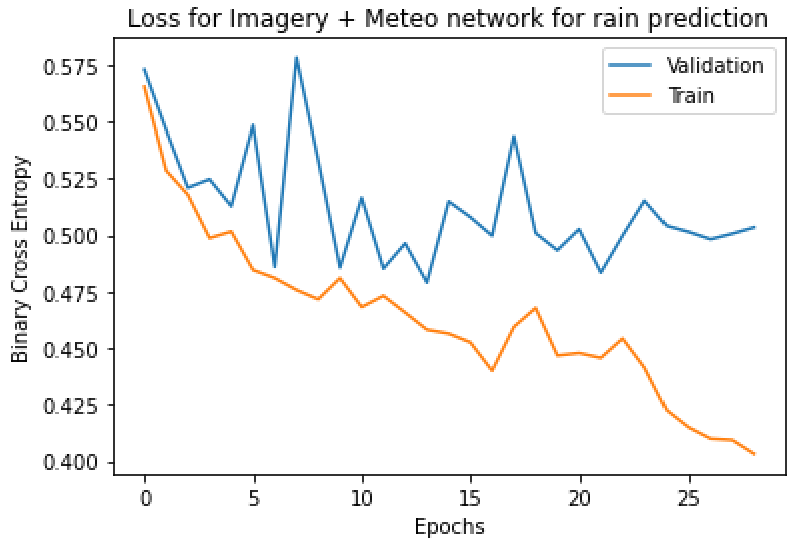
In [ ]:
```

```

In [96]:
```
pred = model([cloud_val, meteo_val])

# Convert to array
pred = np.array(pred)

# Assigning class based on prediction
pred[pred >= 0.5] = 1
pred[pred < 0.5] = 0
#pred[pred != 1] = 0

# Class-wise accuracy
classwise1 = ((np.array(pred)[:,0] == np.array(rain_val_c))*(rain_val_c==1)).su
classwise0 = ((np.array(pred)[:,0] == np.array(rain_val_c))*(rain_val_c==0)).su
```

In [97]:
```
print(f'Total Accuracy: \t {((np.array(pred)[:,0] == np.array(rain_val_c)).sum(
print('-'*30)
print('--Class wise Accuracy of Test--')
print('-'*30)
print(f'Class 0: \t {classwise0*100:.3f}')
print(f'Class 1: \t {classwise1*100:.3f}')
```

```
Total Accuracy:          77.170
------------------------------
--Class wise Accuracy of Test--
------------------------------
Class 0:         88.532
Class 1:         50.538
```

In [98]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(history.history['val_loss'], label='Validation')
plt.plot(history.history['loss'], label='Train')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Binary Cross Entropy')
plt.title('Loss for Imagery + Meteo network for rain prediction')
plt.savefig('data_dir/Losses-imagery-and-meteo-rain-prediction-48-'+station_coc
```



In [ ]:

```

```

In [99]:

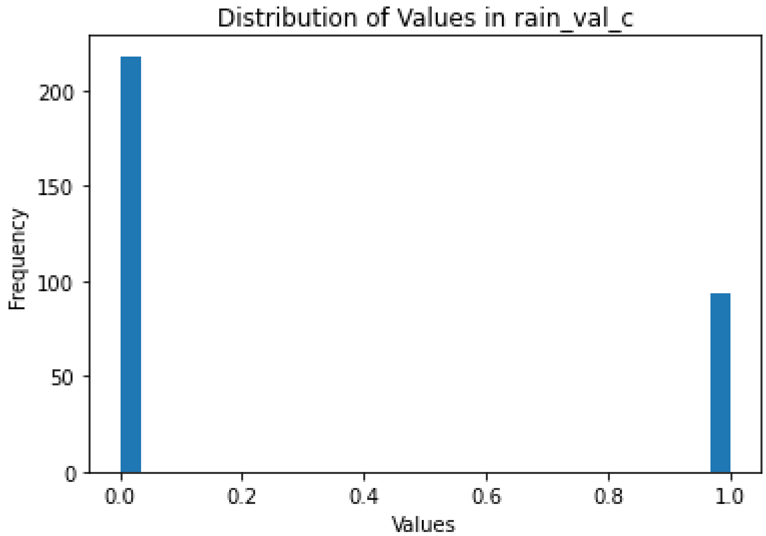
```
rain_val_c
```

Out[99]:

```
array([1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1])
```

In [100…

```
import matplotlib.pyplot as plt

# Assuming rain_val_c is your array
plt.hist(rain_val_c, bins=30)  # Change bins to get a different granularity
plt.title('Distribution of Values in rain_val_c')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```



In [101…
```
rain_val_series = pd.Series(rain_val_c)
value_counts = rain_val_series.value_counts()
value_counts
```

Out[101]:
```
0    218
1     93
dtype: int64
```

In [ ]:

In [102…
```
rain_train_c
```

Out[102]:
```
array([0, 0, 1, ..., 1, 0, 0])
```

In [103…
```
rain_train_series = pd.Series(rain_train_c)
value_counts = rain_train_series.value_counts()
value_counts
```

Out[103]:
```
0    1271
1     415
dtype: int64
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: