Hurricane Wind Speed Prediction Using Machine Learning and Deep Learning

Arsh Modak, Omkar Waghmare
https://github.com/arshmodak/Hurricane-Wind-Speed-Prediction-using-Deep-Learning-and-Machine-Learning

1. SUMMARY

1.1. Problem Statement and Overview

Tropical cyclones have become more destructive in the last decades due to increase in surface temperature as a result of global warming. Hurricanes/Tropical Cyclones are one of the costliest natural disasters globally because of the wide range of associated hazards. Hurricanes can cause upwards of 1000 deaths in a single event and are responsible for more than 100,000 deaths worldwide. During a tropical cyclone, humanitarian response efforts hinge on accurate risk approximation models that depend on wind speed measurements at different points in time throughout a storm's life cycle.

Direct measurements of the winds within a tropical cyclone are sparse, particularly, over open ocean. Thus, diagnosing the intensity of a tropical cyclone is initially performed using satellite measurements. According to the National Hurricane Center (NHC), an accurate assessment of intensity using satellite data remains a challenge.

For several decades, forecasters have relied on visual pattern recognition of complex cloud features in visible and infrared imagery. However, visual inspection is manual, subjective and often leads to inconsistent estimates. This is the reason why we want to design and develop a system using Deep Learning and Machine Learning which predicts the hurricane's speed using satellite images.

1.2. Data

The data was prepared by the NASA IMPACT team and Radiant Earth Foundation. It consists of single-band satellite images captured by GOES (Geostationary Operational Environmental Satellites) of 600 storms over two oceans (Atlantic Ocean and East Pacific).

There are a total of 114,634, 366x366 single-band images; 70,257 images in the train set and 44,377 in the test set. Each image is associated with the following metadata:

- *Image ID*: unique image identifier.
- *Storm ID*: unique storm identifier.
- *Ocean*: 1-Atlantic, 2-East Pacific.
- *Relative Time*: timestamp in milliseconds relative to the first image.
- *Wind Speed*: speed of the wind in knots.

1.3. Non-Technical Description of Proposed Methods

We will be creating a baseline CNN model and use state-of-the-art (SOTA) CNN architectures for our task. Additionally, we will perform feature extraction techniques to use them for our standard ML models.

We will perform hyperparameter tuning for all our models and evaluate them using metrics such as MSE, MAE and RMSE.

2. METHODS

2.1.Data Pre-processing

2.1.1. Metadata Extraction

Training and testing sets are comprised of two main folders, one for images along with a JSON file for its metadata and another file for its corresponding wind speed labels. We have extracted meta data regarding the images from multiple JSON files and created a dataset for future use.

2.1.2. Exploratory Data Analysis

We performed EDA on the metadata dataset that we created in the previous step. We also visualized the images of the storms at varying speeds. The results can be viewed in Section 3.

2.1.3. Image Processing

We converted our single-band images to three-band (RGB) images using two methods. In one method we manually added R, G, B to respective channels and then merged the image, whereas in the second method we directly merged the image without adding any color.

For creating the three-band images, we merge three distinct images at three distinct time steps that differ by a fixed interval. Based on research we found the best interval is 9 timesteps.

In Figure 1, we have described both these approaches, essentially both the images (bottom) are the same with just one major difference, bottom left image has colours added to each channel and the bottom right image has no colour added.

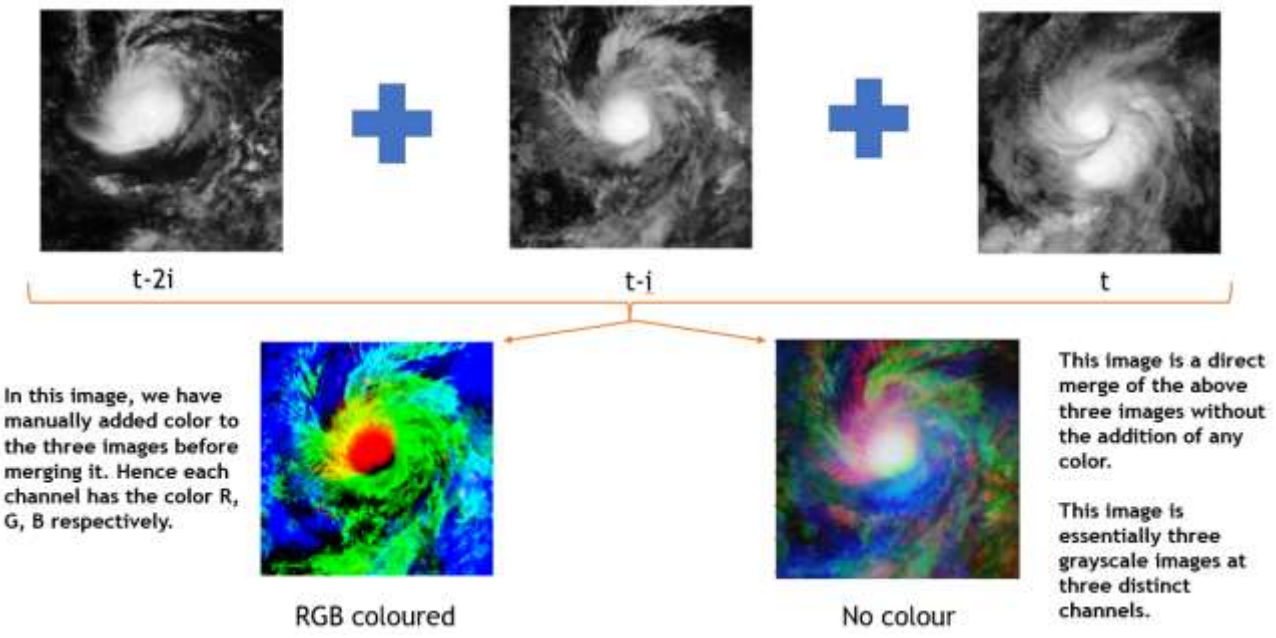


Figure 1: Image Processing results. Original images at different timesteps (top), manually added colors before merging images (bottom left), merging images without adding color (bottom right).

Additionally, for the baseline CNN, we resized out single-band images from 366x366 to 224x224. However, we also experimented with the original size.

2.2. Modelling

2.2.1. Model Creation and Training

For our baseline model we created a simple CNN architecture as described in Figure 2.1.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 224, 224]             416
         MaxPool2d-2         [-1, 16, 112, 112]               0
            Conv2d-3         [-1, 32, 112, 112]          12,832
         MaxPool2d-4           [-1, 32, 56, 56]               0
            Conv2d-5           [-1, 64, 56, 56]          51,264
         MaxPool2d-6           [-1, 64, 28, 28]               0
            Conv2d-7          [-1, 128, 28, 28]         204,928
         MaxPool2d-8          [-1, 128, 14, 14]               0
           Dropout-9                [-1, 25088]               0
           Linear-10                   [-1, 32]         802,848
          Dropout-11                   [-1, 32]               0
           Linear-12                    [-1, 1]              33
================================================================
Total params: 1,072,321
Trainable params: 1,072,321
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 14.55
Params size (MB): 4.09
Estimated Total Size (MB): 18.83
----------------------------------------------------------------
```

Figure 2.1: Description of the baseline CNN architecture

2.2.2. Hyperparameter Tuning, Evaluation and Comparison

We experimented with different learning rates, batch sizes, optimizers and image size for our baseline model. Since our response variable is continuous in nature, we evaluated our models using metrics such as MSE and RMSE. Figure 2.2 shows the model train and validation losses for various configurations. Figure 2.3 compares the model's performances on the test set with various configurations.

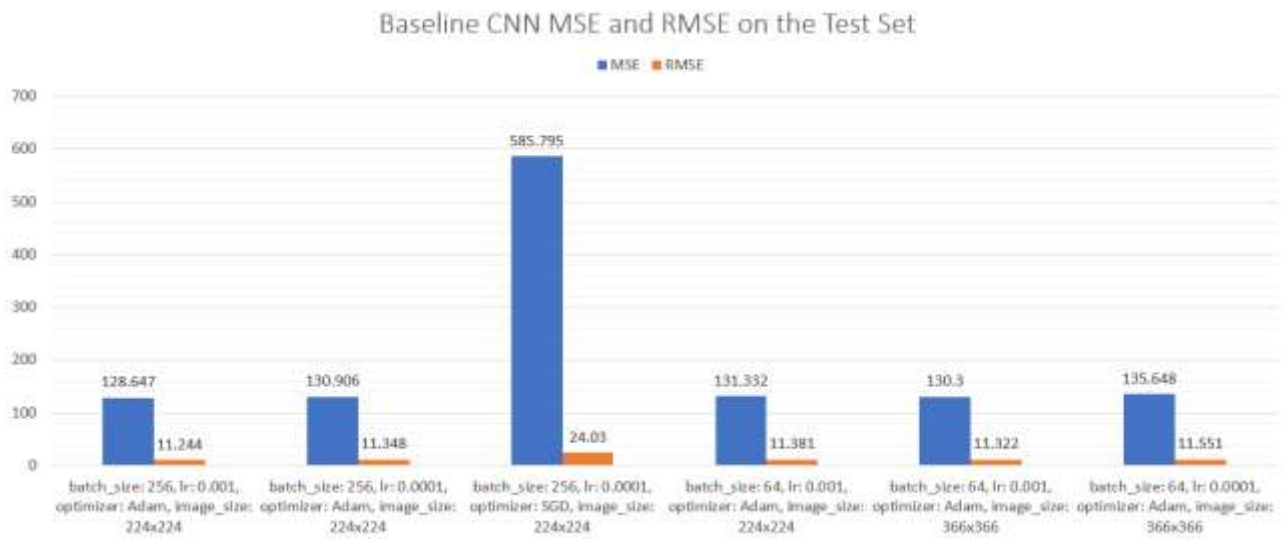


Figure 2.3: Baseline CNN Performance with various configurations on the Test Set

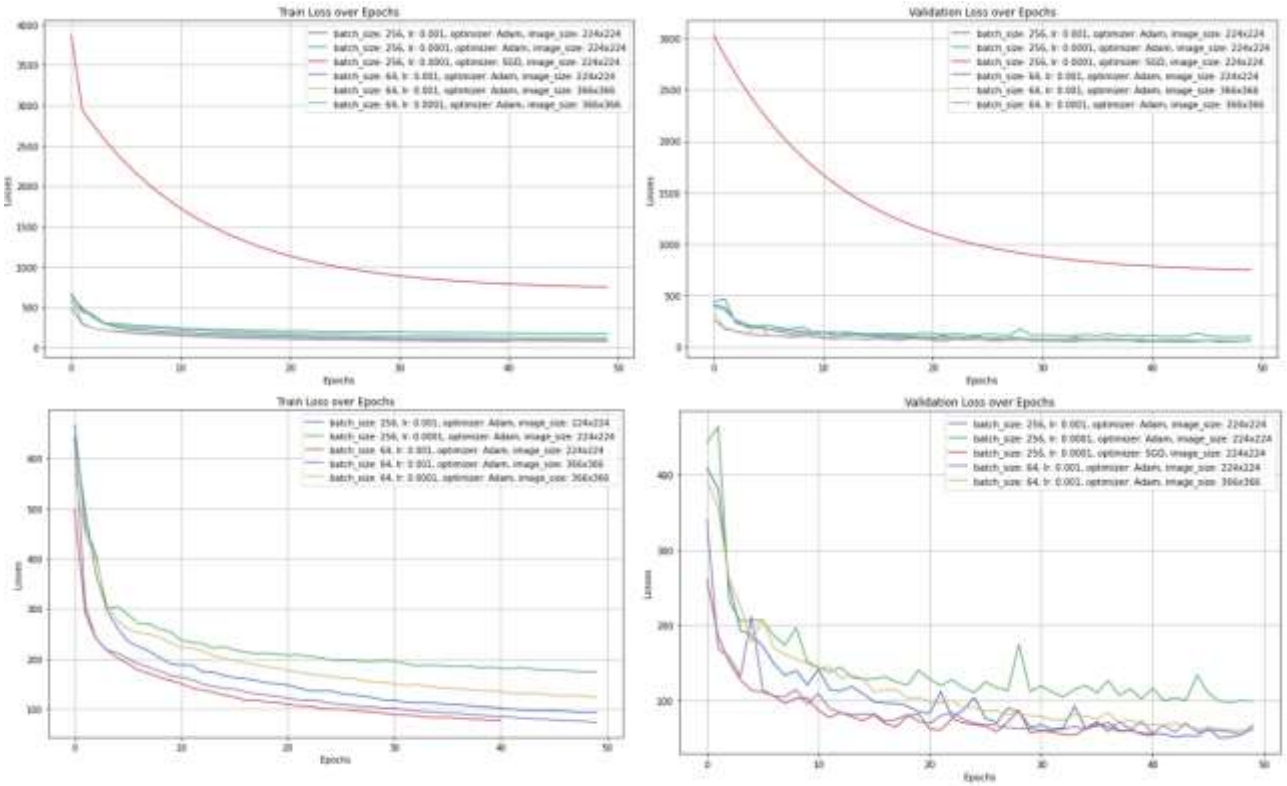


Figure 2.2: Top: Train Loss vs Epoch of all baseline CNN configurations (left), Validation Loss vs Epoch of all baseline CNN configurations (right). The red line is for the model with SGD, which performs the worst out of all and hence we discard it in the bottom visualization. Bottom: Train Loss vs Epoch of all baseline CNN configurations except for SGD (left), Validation Loss vs Epoch of all baseline CNN configurations except for SGD (right).

3. RESULTS

Figure 3 shows the outputs from the best performing hyper tuned baseline CNN model.

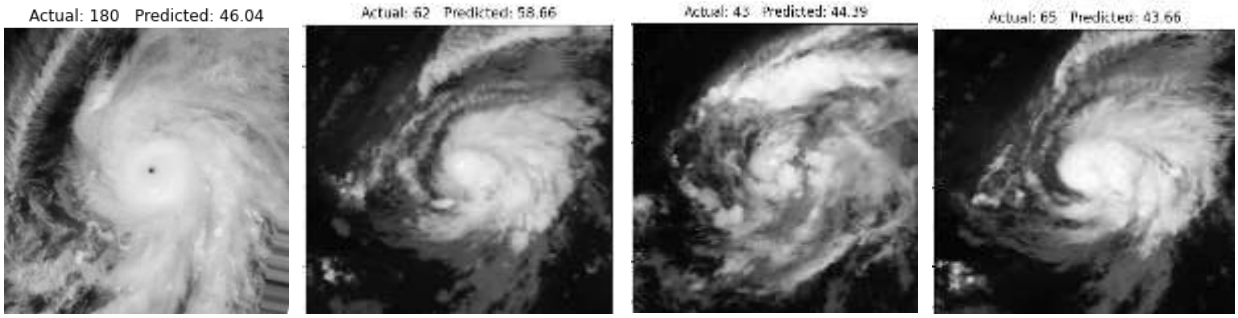


Figure 3: Sample outputs from the test set of the best performing baseline CNN

From the first image, we can clearly find out that the model does not work well for hurricanes with extremely high speed. The reason for this is extreme imbalance of data where high speed hurricane images are very low in number, some as few as just one image. For ranges within 25 to 70 knots, the model performs decently wherein it predicts almost 60% of the images correctly.

4. DISCUSSION

We need to balance the dataset in order to improve the models to work decently well for all wind speeds especially for extremely high speeds since they are the most devastating hurricanes. However, this being said, implementing this is easier said than done, mainly due to three reasons: first, the dataset comprises of images, second, using the same image multiple times may highly overfit our model, third, since this is a regression task, traditional data balancing/sampling techniques such as under/over sampling don't work. Additionally, if we choose to sample the images, we do not want to reduce the images for lower speeds since these hurricanes are the most frequent.

The current model can be used to predict hurricanes with speeds within the range on 25 to 70 knots with a decent error rate, however, the main purpose of this model serves as a baseline to compare the performance with bigger, better and more complex SOTA CNN architectures which we will see in the future phase of this project.

Instead of using the raw grayscale images, we could attempt to use our RGB converted images (discussed in Section 2.1.3) along with a more balanced dataset in order to improve the model's performance.

5. STATEMENT OF CONTRIBUTIONS

Both team members, Omkar and Arsh have contributed equally by doing each and every part of this project together.

6. REFERENCES

https://ieeexplore.ieee.org/document/9149719

https://mlhub.earth/10.34911/rdnt.xs53up

https://www.drivendata.org/competitions/72/predict-wind-speeds/

7. APPENDIX

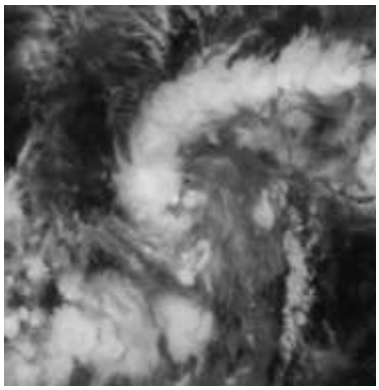
Function to create a data frame with links of the images at the specified time step interval.

```
def get_data_in_intervals(metadata, interval):
    """

    :param metadata: metadata dataframe
    :type metadata: pd.DataFrame
    :param interval: time interval
    :type interval: int
    :return: dataset with links of images at the specified interval
    :rtype: pd.DataFrame
    """

    data_dict = dict()
    for j, i in enumerate(range(0, metadata.shape[0])):
        i2 = i - interval
        i3 = i2 - interval
        if i3 > 0 and i2 > 0:
            lis = [metadata.image_path[i3], metadata.image_path[i2], metadata.image_path[i]]
            storm_id = metadata.storm_id[i3]
            wind_speed = metadata.wind_speed[i]

            if j not in data_dict.keys():
                data_dict[j] = {"image_list": lis, "wind_speed": wind_speed}
    df = pd.DataFrame.from_dict(data_dict, orient="index").reset_index().drop(columns="index")

    return df
```

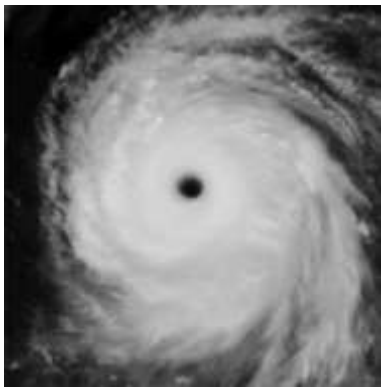
Function to create three-band images from single band images for pretrained models.

```python
def convert_to_threeband(img_list, add_color=False):
    """

    :param img_list: list of image paths
    :type img_list: list
    :param add_color: add color to each channel manually
    :type add_color: boolean
    :return: merged RGB image with or without color
    :rtype: image
    """
    def add_rgb_to_images(img):
        # create 1 pixel red image
        red = np.zeros((1, 1, 3), np.uint8)
        red[:] = (0, 0, 255)
        # resize lut to 256 values
        # # create 1 pixel blue image
        blue = np.zeros((1, 1, 3), np.uint8)
        blue[:] = (255, 0, 0)
        # resize lut to 256 values
        # # create 1 pixel green image
        green = np.zeros((1, 1, 3), np.uint8)
        green[:] = (0, 255, 0)
        # append the three images
        lut = np.concatenate((red, green, blue), axis=0)
        lut = cv2.resize(lut, (1, 256), interpolation=cv2.INTER_CUBIC)
        # apply lut
        result = cv2.LUT(img, lut)
        return result

    # Reading the three images
    img1 = cv2.imread(img_list[0], cv2.IMREAD_GRAYSCALE)  # t - 2*interval
    img2 = cv2.imread(img_list[1], cv2.IMREAD_GRAYSCALE)  # t - interval
    img3 = cv2.imread(img_list[2], cv2.IMREAD_GRAYSCALE)  # t

    # Merge the three images to convert to three equal channels
    img = cv2.merge((img3, img2, img1))

    # Add color if True
    if add_color:
        img = add_rgb_to_images(img)
    return img
```

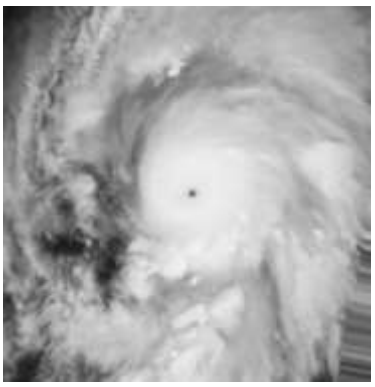
Images of the same storm at different wind speeds



Wind Speed: 30 knots

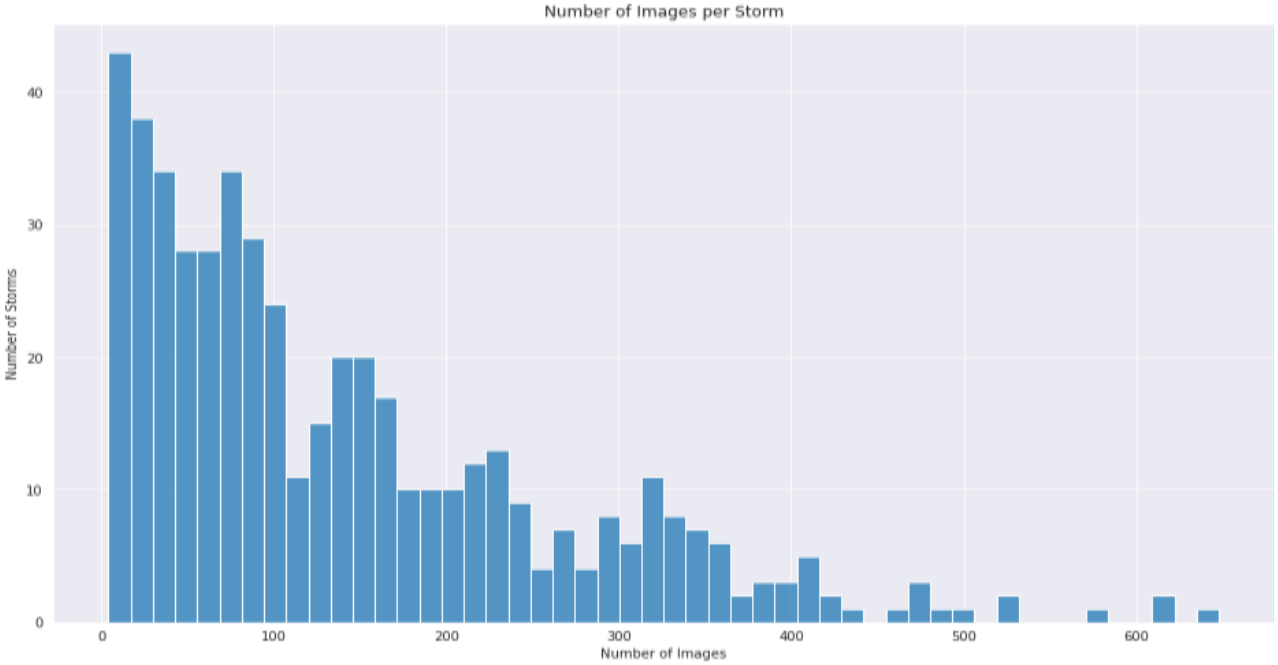


Wind Speed: 150 knots

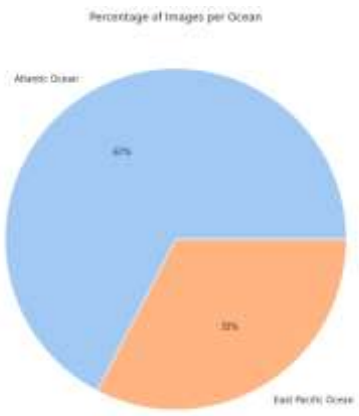


Wind Speed: 185 knots

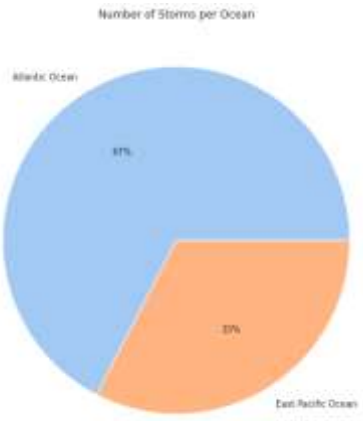
Number of Images for each Storm



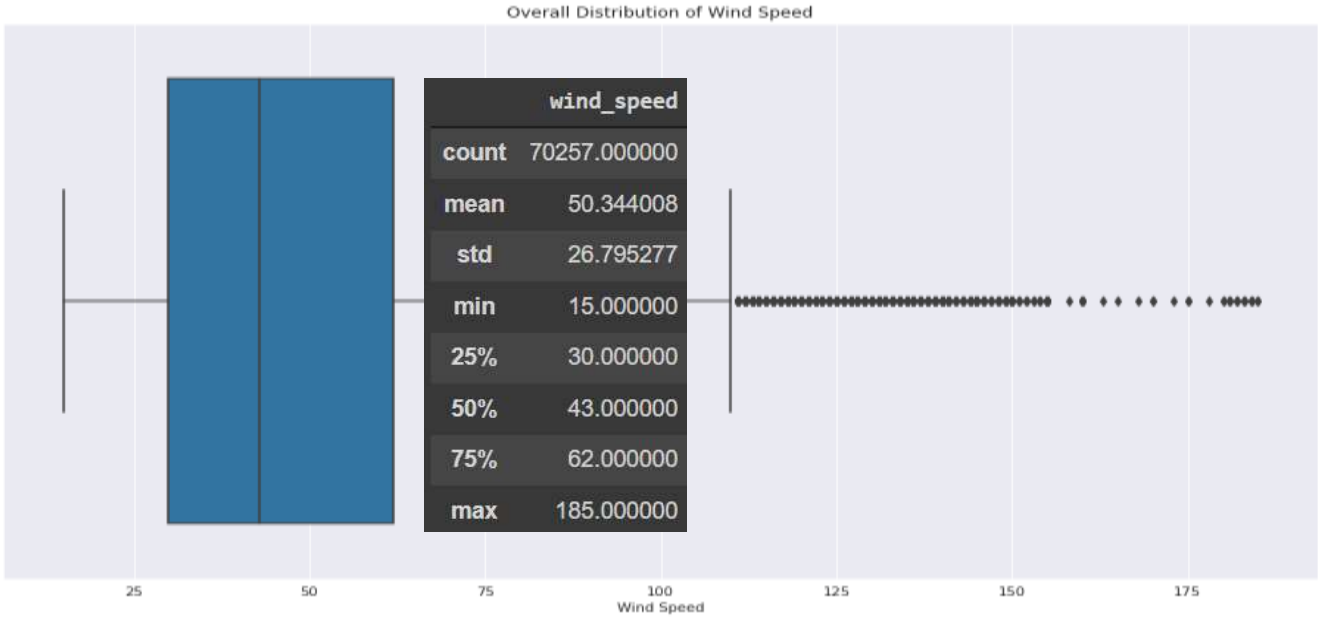
Percent of Images per Ocean



Percent of Storms per Ocean



Distribution of Wind Speeds



Wind Speed v/s Relative Time per Storm