DAYLIGHT CALIBRATION METHOD FOR AGRICULTURAL REMOTE SENSING WITH A WIDE ANGLE "SKY CAMERA" AND DEEP LEARNING

by

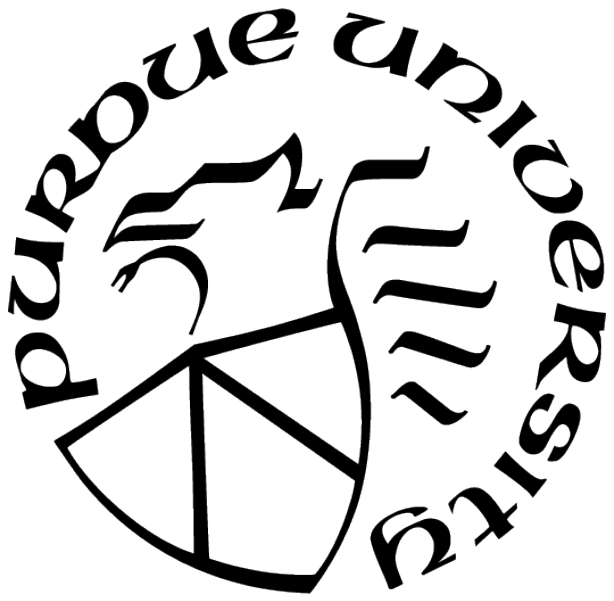
Thirawat Bureetes

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Agricultural and Biological Engineering



School of Agricultural and Biological Engineering

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Jian Jin, Chair

Agricultural and Biological Engineering

Dr. Ankita Raturi

Agricultural and Biological Engineering

Dr. Yang Yang

Institute for Plant Sciences

Approved by:

Dr. Nathan Mosier

To my father and mother. Please be proud of your son.

ACKNOWLEDGMENTS

Thank you to Professor Jian Jin for his endless support and advice on this research project. I have gained invaluable experience.

Thank you to the committee members: Professor Ankita Raturi and Professor Yang Yang, for constructive feedback that helps me to produce better work.

Thank you to the Kalufs for your prays and comments.

Thank you to members of the Plant Phenotyping Lab for help whenever I need it.

Thank you to my parents. You are the motivation that pushes me forward.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

UAV	unmanned aerial vehicle
DSLR	digital single-lens reflex camera
USB	universal serial bus
ISO	international organization for standard

ABSTRACT

An advancement of Unmanned Aerial Vehicle (UAV) technology accelerates airborne imaging in the agricultural sector. Various cameras, such as a multispectral camera or hyperspectral camera, are equipped with drones. It enables farmers to access several vegetation indices such as nitrogen or water stress. The image pixel value is a crucial variable of many indices calculations. However, the sunlight heavily influences image pixel values. The non-calibration data could lead to misinterpretation. The current daylight calibration method is using a white reference board made from highly reflective material. Including the white reference board provides sunlight intensity information. However, this method requires the white reference board's existence in every image. A spectrometer is a sensor that gives light spectrum intensity directly without the white reference board. This study develops a regression model to produce daylight intensity from spectrometer data. However, the result shows that the model outcome does not match with the white reference board. Although the spectrometer eliminates white reference board necessity, it cannot replace the white reference board method due to outcome incompatibility. A new daylight calibration method using sky information is introducing in this study. An RGB camera mounted with a wide-angle or fisheye lens pointing to the sky captures a sky's dynamics. A Convolutional Neural Network is trained using sky images. The model R-square is 0.997. The number of outliers, a prediction that mismatch its ground truth more than 10 percent, measures the model performance. From 921 samples, seven outliers existed or 0.8 percent. The proposed wide-angle camera solution can produce similar light intensity values as the physical white reference board method. This alternative method operation uses a single camera offering a higher practicable daylight calibration method for aerial remote sensing.

1. INTRODUCTION

1.1 Background Information

The United Nations [1] estimates that the world population increases by approximately 83 million people annually and will reach 9.8 billion people by 2050. The world food production capacity needs to improve to meet the fast-growing food demand due to the growing population. Historically, food production has been expanded by increasing agricultural land production area. A thousand years ago, the humanity used only 400 million hectares, or 4 percent of the land space, for agriculture purposes [2]. In the modern-day, agricultural land usage takes half of the global habitable land area, leaving another half for the forest, grassland, urban space, and freshwater combined. The land area is a limited resource. Thus increasing land usage for agricultural purposes and food production leads to decreased land area available for forests. Since 1990, 420 million hectares of forest have disappeared, according to the Food and Agriculture Organization of the United Nations [3]. The commercial agriculture industry is the primary source of deforestation. Therefore, expanding the farming area is not a sustainable solution for increasing global food production. Humanity needs to make the most yield of the existing agricultural land area. Adopting precision agriculture can improve productivity from existing farm land areas [4], [5]. For example, an intelligent irrigation schedule [6] raises a plant's growth to an optimal point. Accessing inside plant information such as nitrogen status [7] or water stress [8] also plays a crucial role in precision agriculture.

The emerging Unmanned Aircraft Vehicle (UAV) technology, with aerial remote sensing has potential for agricultural applications. UAV technology offers a high throughput capability to scan a massive land area in a fraction of time compared to human scanning or terrestrial robot. A drone equipped with various camera types such as a RGB , multispectral, or hyperspectral camera can capture images [9] from the sky. These images contain essential information that can calculate several vegetation indices, such as Normalized Difference Vegetation Index (NDVI). Although different types of cameras capture different information, all cameras share a common image acquisition mechanism. An array of photoreceptors inside the camera react to incoming photons. Each image pixel is a result of the light characteristics

measured by the sensor's array. The wavelength of light determines the color, while the light energy level designates brightness. The Sun is a primary light source in an outdoor environment. When the Sun's light hits objects, some light wavelengths are absorbed, and some are reflected. Objects in images are a result of reflected light that travels into the camera. The more sunlight impacting an object's surface results in more light reflected back. The same object in images can either look brighter or darker depending on the intensity of the sunlight.

Figure 1.1. Shadow impacts the pixel brightness [10]

Image 1.1 illustrates an impact due to non-homogeneous light condition. There are some areas under the cloud's shadow that are noticeable because of dimmer pixel values. Vegetation index calculations rely on the image's pixel values. Therefore, shadows or changes in light conditions affect vegetation index calculations as they alter pixel values. As a result, a daylight calibration method is needed to reduce the influence from brightness fluctuation.

1.2 A White Reference Calibration Method

A current standard daylight calibration method uses a white reference board made by a material that reflects nearly 100 percent of impacting light. Since the board surface can reflect most of the sunlight, a pixel of the white reference board in the image is equal to sunlight intensity. This pixel value is a reference value for the calibration process. Equation 1.1 shows the formula to calibrate the image using the white reference board method. x_{raw} represents a raw pixel value, x_{white} represents white reference board pixel value, and x_{cal} is a calibrated pixel value. Dividing each image's pixels by the reference value cancels the influence of fluctuated sunlight.

$$x_{cal} = \frac{x_{raw}}{x_{white}} \tag{1.1}$$

Although the white reference board method works well, it has some drawbacks. First, the white reference board reflectance varies according to the properties of the material utilized. Therefore, the pixel value over the white reference board changes depending on the white reference board material and effects the calibrated pixel value from equation 1.1. Second, applying the white reference board calibration method requires the board to exist in every image. Although drone imaging cuts down operation time dramatically, the light condition changes even faster the speed of the drones. Therefore, each image has to be calibrated with the light condition when the image is taken. Any images in which the white reference board is missing have no information to be appropriately calibrated. Practically, several white reference boards need to be evenly distributed over the targeted area before the imaging session. This operation is labor-intensive and time-consuming, which reduces the speed benefits of drone imaging. Additionally, the location of the white reference board in images is not static. Thus, it requires processing resources to locate the white reference board inside the image before initiating an image calibration. Alternative calibration methods that do not rely on a physical white reference board can streamline remote drone imaging operations.

A spectrometer is a sensor that measures light spectral intensity. The spectrometer has a fiber-optic tube guiding light into a processing box, which will break down incoming light and measure energy levels. If the fiber-optic tube is pointed upward to the sky, the input

light is the sunlight. Therefore, the resulting output from this spectrometer configuration is the spectral intensity of the sunlight. Consequently, the spectrometer can provide sunlight information that could be used to calibrate drone images. The spectrometer solves the white reference board method's drawback as the spectrometer does not require any physical white reference boards. There are several previous studies which attempted to calibrate aerial images utilizing spectrometers.

1.3 Literature Review

Zhang et al. (2019) [11] implemented a hyperspectral handheld device. That device calibrated its camera with a white reference board before capturing leaf images. Zhao et al. (2015) [12], Zhang at el. (2015) [13], and Li et al. (2017) [14] used a black and a white reference board to calibrate hyperspectral images. Even in the dark room where the light source was completely controllable, calibration helped improve output quality. All three studies share the same calibration formula as equation 1.2.

$$x_{cal} = \frac{x_{raw} - x_{black}}{x_{white} - x_{black}} \tag{1.2}$$

Where x_{raw} is raw pixel intensity, x_{black} is black reference pixel intensity, x_{white} is white reference intensity, and x_{cal} is calibrated pixel intensity. The formula shows that the calibrated value is a ratio between two reflectance values compared to black reference board. The numerator is an object light reflectance and the denominator is white reference board reflectance. Zhang et al. (2020) [15] discovered that since leaves are curved, the light impacting angle may vary. A flat white reference does not perform well with curved leaves. They developed a 3D white reference for calibrating hyperspectral images for curved leaves.

Wu et al. (2015) [16] demonstrated an effect of shadow on vegetation index interpretation. To solve the problem, a ground-based spectrometer was used to calibrate images before calculating vegetation indices. Burkart et al. (2017) [17] synchronized a ground-based spectrometer with an on-board drone spectrometer for white calibration. Singh et al. (2017) [18] used a ground-based spectrometer to capture the ambient light intensity. The information was linked to the white reference board's hyperspectral images. Bai et al. (2019) [19]

equipped a spectrometer on a spider camera. There were two fiber optic tubes feeding light into the spectrometer. One tube pointed downward to the plants while another one pointed upward to the sky.

1.4 Research Objective

This study aims to find alternative daylight calibration solutions that do not depend on white reference boards. The first candidate method is using a spectrometer. This study will utilize light spectral information provided by the spectrometer to calibrate images in the same manner as the white reference board. A regression model will be used to reveal a relationship between white reference board intensity and spectrometer's output. Therefore, the spectrometer can replace physical white reference boards.

This study also introduces an original solution: a sky camera. A human could get a sense of daylight brightness by considering objects on the sky such as clouds and the Sun. Thus, a neural network will be created to replicate that human ability. Using deep learning will train the neural network with sky images captured by a camera equipped with a wide-angle lens. As a result, the model can predict the daylight intensity directly from the information from the sky. Therefore, physical white reference boards are not necessary.

Both spectrometer and sky camera methods are evaluated based on their performance compared to the white reference board method. The goal is to find factors that effect the model's prediction accuracy, including the model's limitations.

2. METHODOLOGY OF THE STUDY

2.1 Equipment

A white reference board is the standard procedure for white referencing for drone imaging. Ground truth is a technical term that refers to a standard method or value that is used as a reference to evaluate other methods or values. Therefore, the white reference board is the ground truth in this study and will be used to benchmark with other approaches. The white reference board is made from a material that has a high light reflective property. With this property, most of the light that impacts the surface of the board will be reflected.

To collect white reference data, a drone with a digital camera flies over a whiteboard. Instead of flying a drone to capture images of white reference boards from the sky, an imaging station on the ground was set up in a way that simulated the camera on the flying drone. To achieve this setup, a Nikon D5300 DSLR camera was mounted on a tripod facing the ground, and the white reference board was placed beneath the camera.

Figure 2.1. A Nikon D5300 DSLR camera [20]

A ground station for a spectrometer was set up to imitate the spectrometer attached on-board with a drone. An Ocean Insight Flame spectrometer was attached to the same tripod that the Nikon DSLR camera was mounted on. The spectrometer has a fiber-optic tube that guides the light into a processing box. The processing box will measure each wavelength's energy level. In this study, the objective of the spectrometer is to measure the intensity of

the sunlight. Hence, the input fiber-optic tube was held with the tripod to point upward to the sky.

Figure 2.2. An Ocean Insight Flame spectrometer [21]

The third sensor was a Flir camera with a fish-eye lens that allows this camera to capture the sky at wide-angle. As this camera's purpose is to capture the sky's image, this camera will be called a sky-camera in this study. The mounting orientation was upward to the sky the perpendicular to the ground, like the spectrometer's fiber-optic tube. The figure 2.4 shows the overall tripod ground station setup.

Figure 2.3. A Flir camera with fish-eye lens

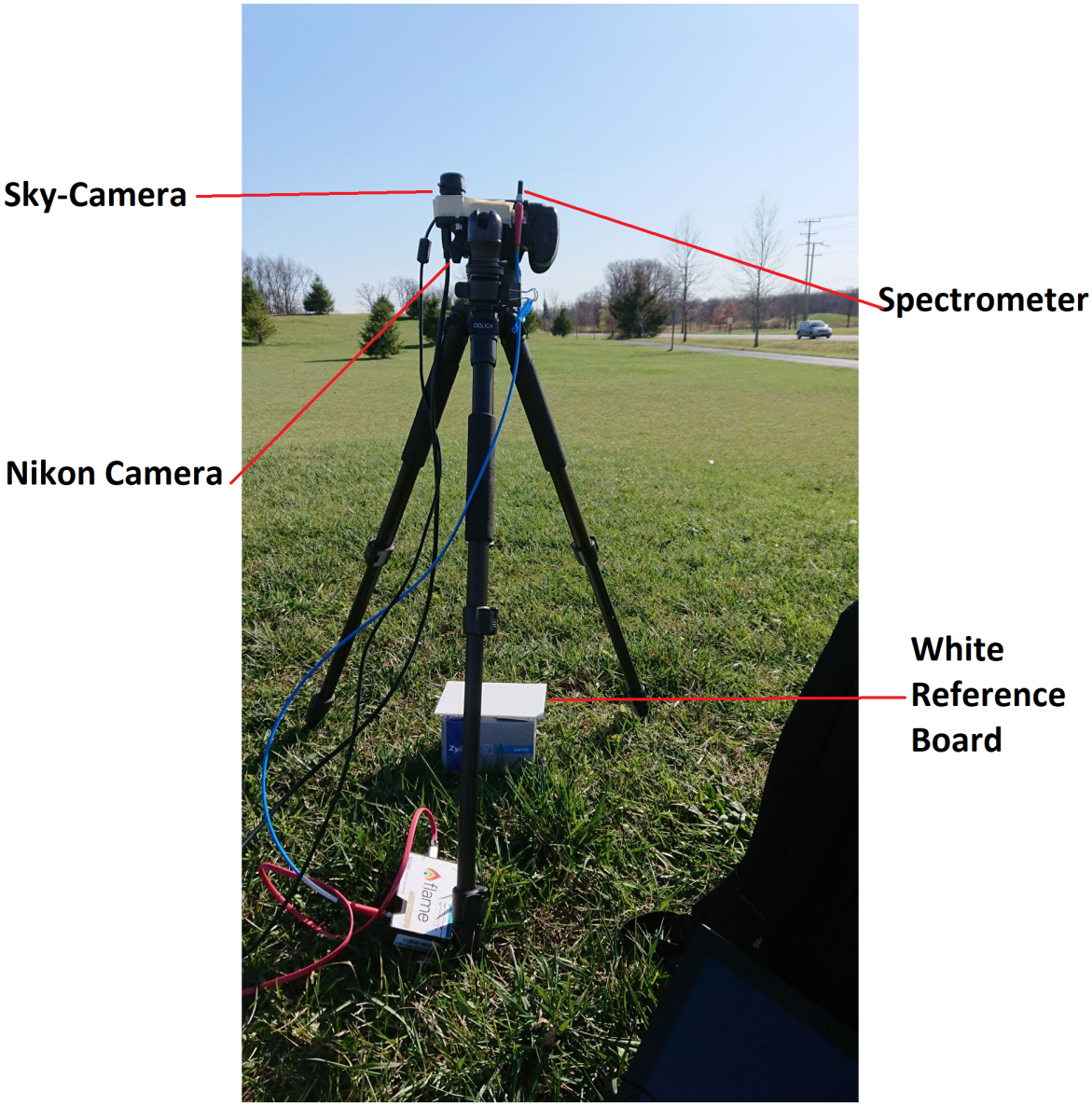


Figure 2.4. An imaging station setup

2.2 Data Collection

Three 3 sensors are supposed to capture information of the intensity of the Sun. As the Sun's position and sky's condition continuously change over time, the data collection process's key constraint is time synchronization among all sensors. Therefore, manually triggering each sensor individually does not comply with the constraint. An automatic data capturing script was written to overcome this issue. The script was written in Python3 and open-source modules that interface with each sensor. All sensors were connected to a "Raspberry Pi 4" controller via a USB port. The Raspberry Pi 4 operates an Ubuntu Server operating system version 20.04, including Python version 3.8 by default. To interface with the Nikon DSLR camera, spectrometer, and sky-camera, Python modules "libgphoto2" [22], "seabreeze" [23], and "pyspin" [24] were used respectively. Modules libgphoto2 and seabreeze are compatible with Python version 3.8. However, the module pyspin is compatible with Python version 3.7 or older. So, the Raspberry Pi 4, which has Python version 3.8, cannot run the module pyspin. Hence Python virtual environment of Python version 3.7 was set up to make the pyspin module functional. Another limitation of the pyspin module is its hardware architecture requirement. An "armhf" is a hardware architecture used in Raspberry Pi 3 and older versions. This hardware architecture does not meet the pyspin module's requirement. While Raspberry Pi 4 uses another hardware architecture named "arm64" which supports the pyspin module's functionality. In conclusion, interfacing with the sky-camera is more difficult than another 2 sensors in this study due to its specific requirements.

To ensure all data's consistency throughout the entire data collection process, each sensor's configuration parameters were fixed. Table 2.1 and table 2.2 show the configurations for the DSLR camera and sky-camera respectively. For the spectrometer, the sampling time was fixed to 15 microseconds.

By default, the sky-camera captures greyscale images. This setting and configurations in table 2.2 are restored to their original values every time the device is powered on. Consequently, the sky-camera has to reconfigure every time to capture color images. Unlike the

Table 2.1. DSLR camera parameter configurations

Parameter	Value	Unit
shutter speed	0.25	ms
aperture	5	-
ISO	200	-

Table 2.2. Sky-camera parameter configurations

Parameter	Value	Unit
shutter speed	0.05	ms
gain	0.07	-

sky-camera, the DSLR camera and spectrometer require configuration only once. Although the DSLR camera and spectrometer are power off, the configurations remain unchanged.

The automated data collection script begins with checking the sensor's connection with the Raspberry Pi 4 controller and applying the sky-camera configuration. If all three sensors present in the system, the data capturing starts in the following order: sky-camera, spectrometer, and DSLR camera. The entire process takes approximately 0.3 seconds. Data from each sensor are stored in the Raspberry Pi 4 's SD card with a timestamp, adopting ISO8601 format, to indicate the time that data is captured. The data capturing process interval is set to 4 - 6 seconds depending on that particular date's weather. If the wind speed is high, causing rapid movement of the clouds, the interval time will be set to high 4 seconds to capture the sky's various conditions. On the other hand, if the sky condition rarely changes, the interval time is set to 6. The complete Python script and related modules are recorded in Appendix A.

The automated data capturing script will keep running over a period of several hours to collect images of the Sun's different positions. There is always a shadow as a result of the sunlight If there is a shadow lying down on the white reference board surface area, it potentially impacts the quality of ground truth values. Therefore, the white reference board was placed on a box to lift it from the ground. This prevented the white reference board

from grass's shadows. Still, shadows produced by the tripod's stand existed. Moreover, the Sun's position varies over time, causing changes in the shadow's angle. Consequently, the white reference board needed to be periodically moved away from the tripod's shadows.

To operate on the field without an electricity grid, a 5 Volt mobile power-bank was used to supply energy to the Raspberry Pi 4. The power was also transmitted to the sky-camera and spectrometer via USB cables. The DSLR camera was powered by its battery cell. The location where all data was taken was an open space across the Folk soccer field on McCormick road, northwest of Purdue West Lafayette campus. The GPS coordinates were latitude 40.4382° N and longitude 86.9370° W.

Figure 2.5. The data collection location

2.3 Data Processing

2.3.1 Ground-Truth

In this study, white reference board images taken by the DSLR Nikon D5300 camera are ground truth. Each image is 2,992 x 2,000 pixels in 8-bit RGB format. Figure 2.6 shows a sample of a ground truth image. The ground truth is an average pixel intensity of red, green, blue, and red-green-blue (RGB) combined over the white reference board area. Since the images are in 8-bit encoding, the possible theoretical intensity ranges from 0 to 255 ($2^8 - 1$).

Figure 2.6. A sample of a ground truth white reference image

Each image consists of both the white reference board and the background. White color in RGB format has high intensity values in all red (R), green (G), and blue (B) color spaces. In contrast, a background has high values in one or two color spaces. For example, grass has a high value in green color space compared to the other color spaces. Thus, a mask is created by differentiating each color space pair: red-green, red-blue, and green-blue. White reference board pixels have a lower color space difference than non-white reference board pixels. An Otsu algorithm [25] is used to find a threshold value for separating white reference board pixels from background pixels. As shown in figure 2.7, the mask can separate the white

reference board area from the background. A white pixel is classified as the white reference board, while a black pixel is classified as the background.

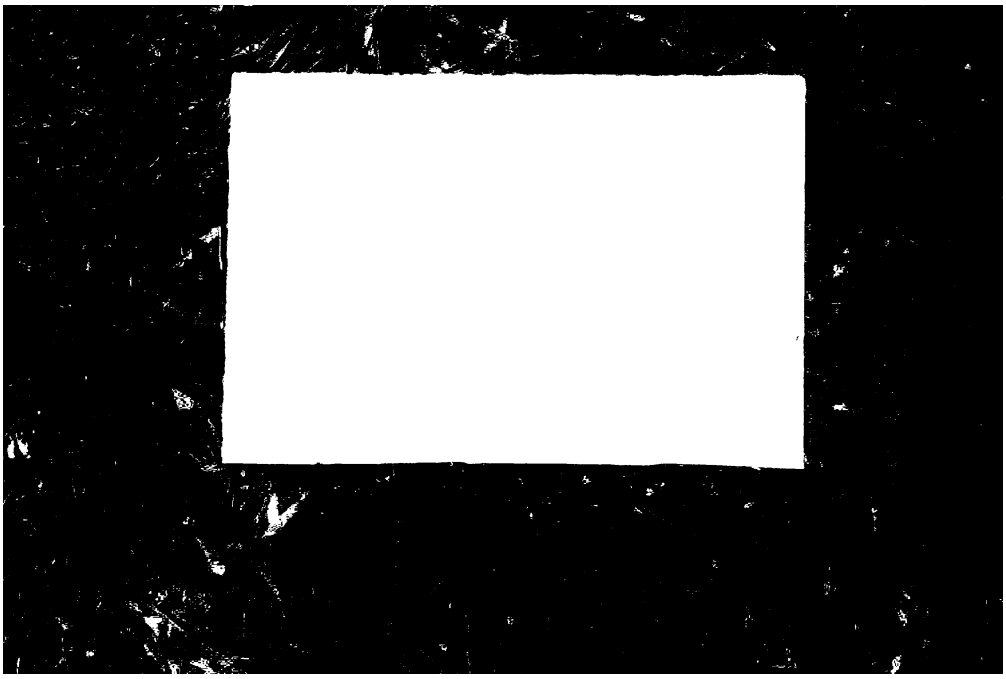


Figure 2.7. A sample mask to separate white reference board from background

However, some small areas that are not part of the white reference board are marked as white. Including these areas in a calculation will alter a true ground truth value. An algorithm is designed to filter out these undesired areas. The algorithm randomly creates 100 x 100 pixel square boxes over the image. Figure 2.8 shows 5 samples of random 100 x 100 pixel square boxes. The white reference board areas are recognized only if the complete area inside the 100 x 100 pixel box is white, drawing in green in the figure 2.8. If there are one or more black pixels inside a particular random box, drawing in red in figure 2.8, the algorithm will immediately reject that box. The algorithm finds 500 boxes over the white reference board surface area to complete ground truth extraction for each image. A minimum number of 500 boxes is used to ensure enough samples are distributed through the board surface area. The white reference ground truth extraction algorithm's source code is in Appendix B.

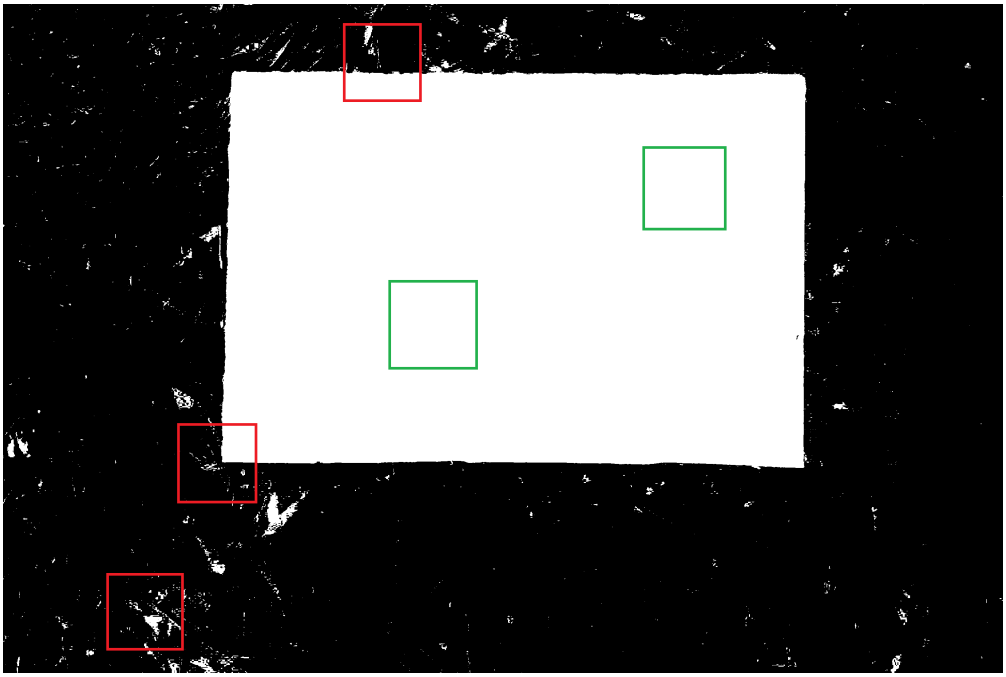
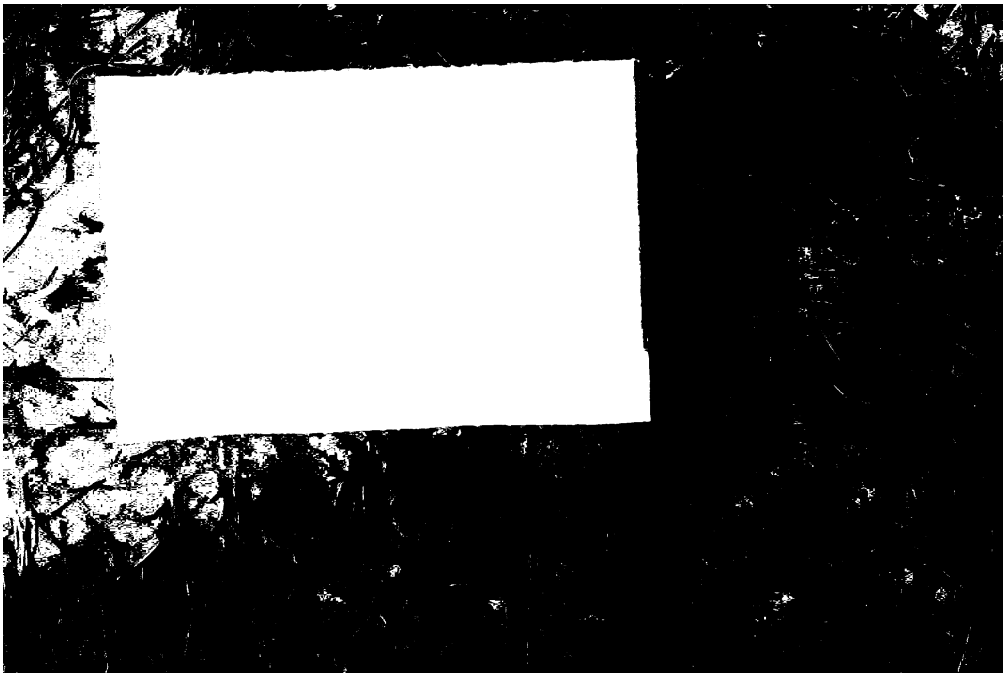


Figure 2.8. 5 random 100 x 100 pixel square boxes randomly created on mask

There is one drawback of this masking approach. It could not identify shadows on the white reference board. Pixel intensity values over the shadow areas tend to be lower than the rest area of the white reference board. If these areas are included in the ground truth calculation, the result value will be lower than the actual value. A white reference board in figure 2.9a has a shadow laying on its surface. The mask in figure 2.9b can separate the white reference board surface area from the background. Although parts of glass under shadow are marked as white, they would be excluded from the random boxes sampling process since they contain black pixels. However, the shadow areas are colored with white. Consequently, the random box sampling algorithm considers shadow areas as parts of the white reference board surface and includes these areas into the ground truth calculation. The random boxes algorithm cannot prevent this issue. Therefore, the images that have shadows over the white reference board area are excluded from the analysis to prevent potential inaccurate ground truth values. This exclusion process requires visual inspection of each image individually. Another exclusion method that can be applied computationally is considering a variance of 500 boxes' average intensity. The high variance indicates inconsistency of pixel intensity

among boxes casing from shadows. Thus, the images with high pixel intensity variance relative to pixel intensity average are excluded from the analysis.

(a) An original RGB image



(b) A white reference board mask

Figure 2.9. A sample of white reference images with shadows on surface area

2.3.2 Spectrometer

Raw data from the spectrometer is a series of 3,840 numbers. Each number represents the intensity of a particular wavelength. The Ocean Insight Flame spectrometer's measurement ranges from 344 nanometers to 1,065 nanometers. Thus, each raw data from the spectrometer is mapped to the corresponding wavelength. Figure 2.10 shows an example of spectrometer raw data. The x-axis represents wavelengths, while the y-axis represents wavelength's intensity levels.

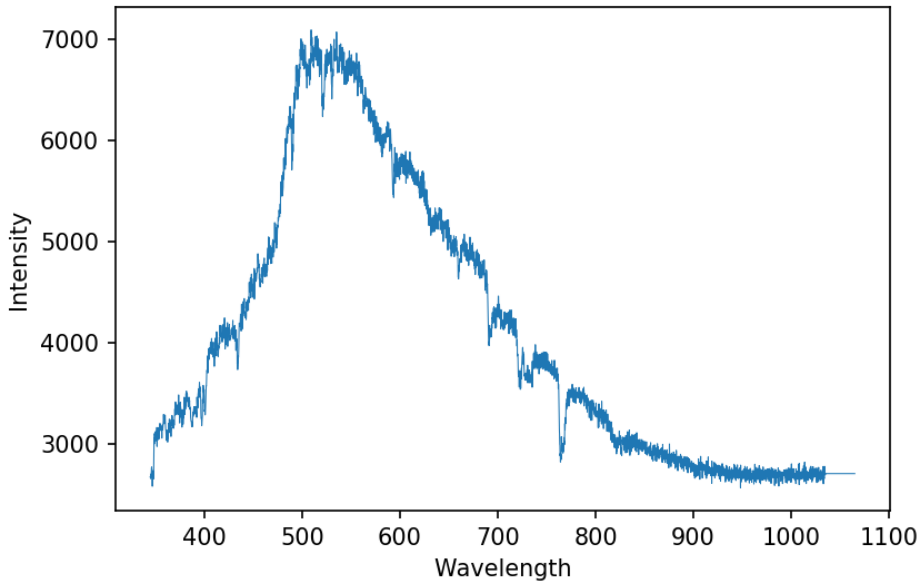


Figure 2.10. An example of spectrometer's raw data

Raw data from each sampling is saved as comma separate value (.csv) file format along with a timestamp that indicates sampling time. As each raw data is kept in individually, the entire spectrometer data is fragmented. A Python script is created to gather all raw data into a single file. The outcome file has 3,841 columns. The first 3,840 columns contain wavelength intensity values and the last column contains a timestamp. Wavelength columns can be easily extracted into a 3,840 x n matrix for regression analysis, where n is a number of samples.

2.3.3 Sky Image

Each sky image is 808 x 608 pixels in 8-bit RGB format. An example of sky images is shown in figure 2.11. As the data collecting process takes multiple days to complete, it is not easy to have a perfectly identical sky view. Additionally, the imaging station must be periodically moved to avoid tripod shadows laying down on the white reference board. Subsequently, the sky is not always the center of the image, which could cause a bias in sky analysis. Also, some portions of each image are ground areas due to the effect of a wide-angle camera. These portions are not necessary for this study as the main objective is to focus on the sky's dynamic.

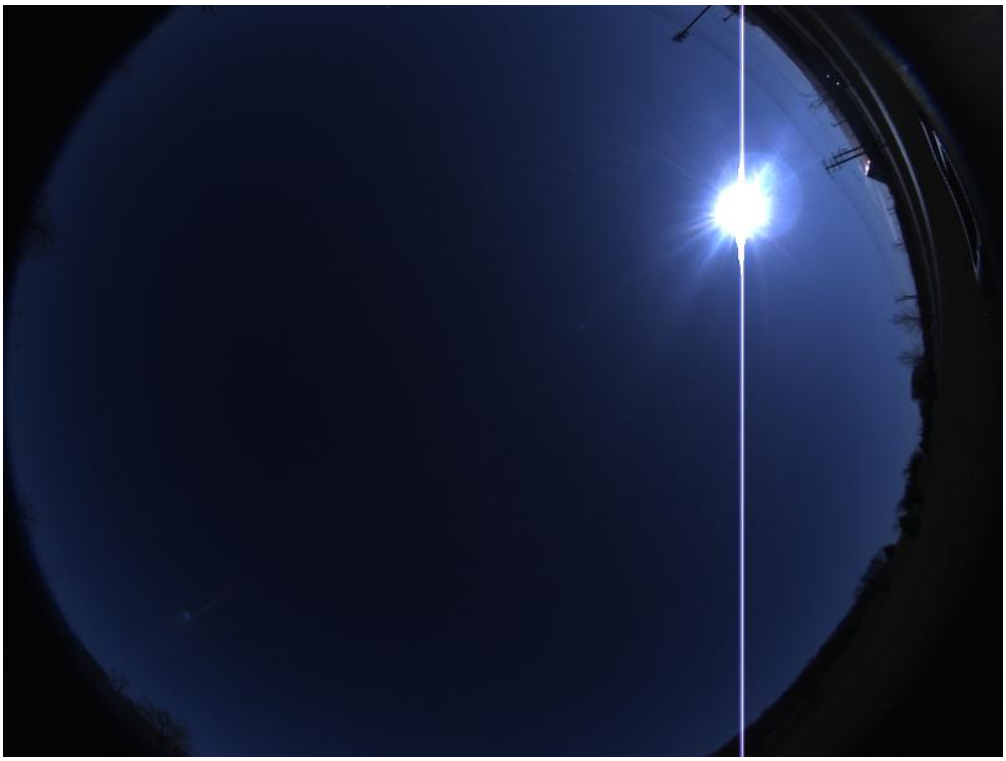
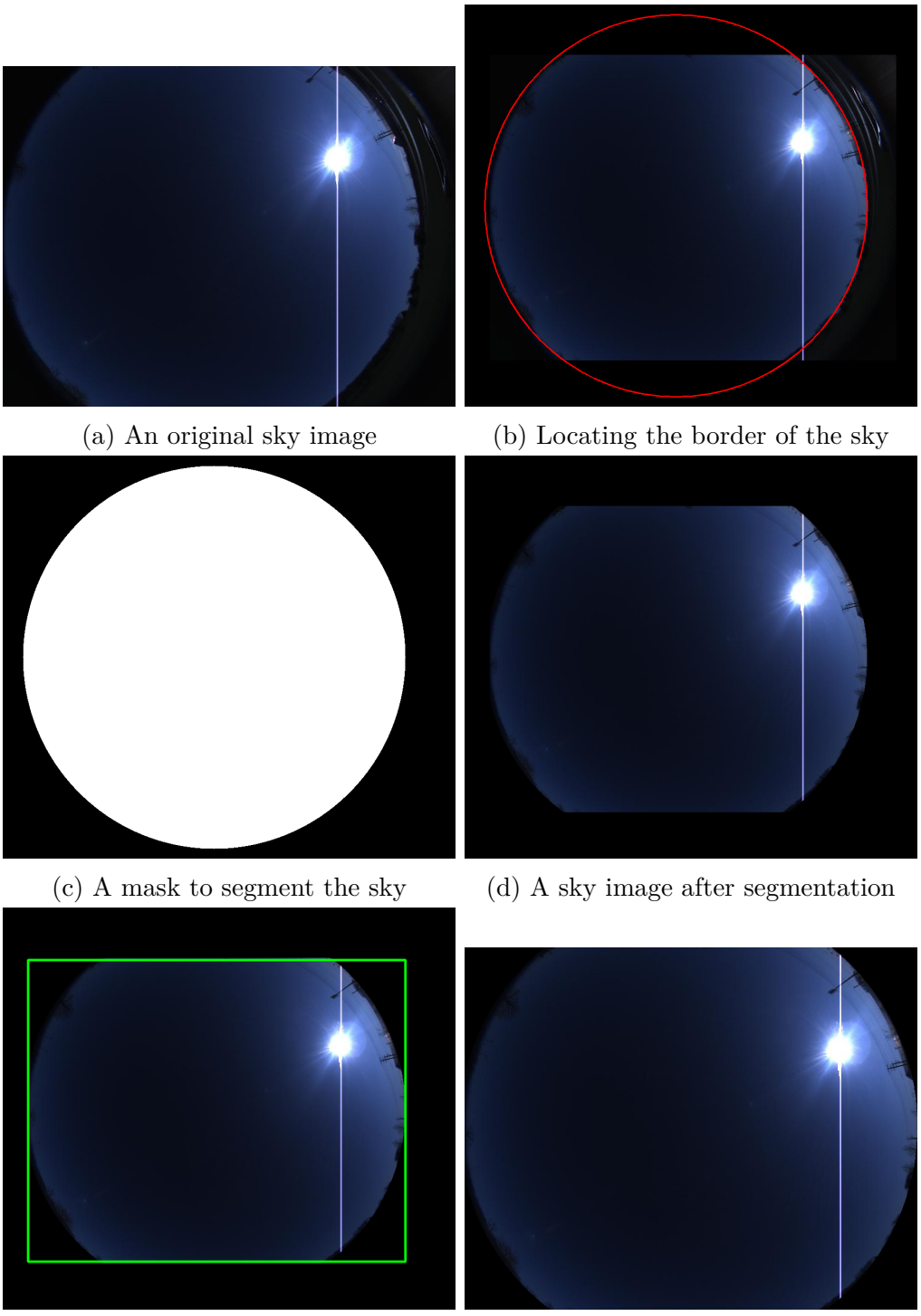


Figure 2.11. An example of sky images

To have a consistent information format, the sky should be the center of each sky image. Furthermore, each image should not contain or contain as little as possible of non-sky portions. Figure 2.12 shows the process of removing non-sky parts and centering the sky. The process starts by locating the border of the sky. A red circle in figure 2.12b represents the

border of original image (figure 2.12a). After locating the sky's border by the circle, its geometry can be obtained to produce a mask shown in figure 2.12c. The white color represents the area of the red circle, which is the area of the sky, while the black area is the non-sky part, which will be removed. After applying sky segmentation, the resulting image will no longer have any areas that are not the sky. Figure 2.12d shows the result after segmentation process. The next step is to find the center of the sky to define the final image's boundary. The original image's height is 608 pixels. During segmentation, the image has not been cropped vertically. For this reason, the maximum height of the segmented image remains 608 pixels. However, the width of images could vary. To ensure that all images are in the same size, the image's width to height ratio is set to 5:4. As a result, the final image size is 760 x 608 pixels. A green box in figure 2.12e shows the boundary of the image where the center of the sky is the center of the image. In the final step, the image is cropped according to the green box. The figure 2.12f shows the final outcome. A source code that used this process is in Appendix C.



(a) An original sky image

(b) Locating the border of the sky

(c) A mask to segment the sky

(d) A sky image after segmentation

(e) Locating the center of the sky

(f) The final outcome

Figure 2.12. The process of removing non-sky part and centering the sky

2.4 Deep Learning Architecture For Sky Image

The goal of analyzing a sky image is to predict the sunlight intensity when the sky image is taken. Same as the spectrometer, the white reference board is used as the ground truth. Deep learning neural network is used to convert a sky image, as input, into a single value corresponding to ground truth sunlight intensity. The neural network contains 3 convolutional layers, 3 pooling layers, and 3 fully connected layers. A pooling layer follows each convolutional layer. This pattern repeats 3 times then followed by 3 consecutive fully connected layers. Figure 2.13 shows the architecture of this neural network.

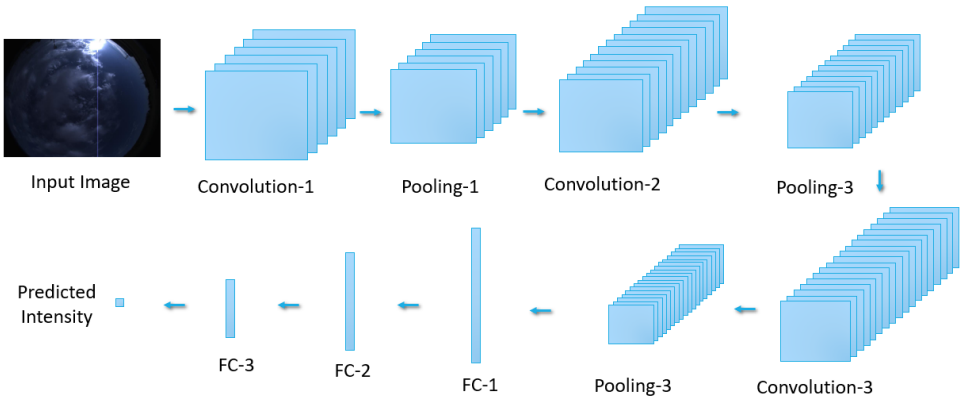


Figure 2.13. An architecture of the deep learning neural network. FC stands for Fully Connected layer

An optimizer is an algorithm that manages the weights used to calculate outcome values. This deep learning neural network's optimizer is a stochastic gradient descent named "Adam" [26] An optimizer's learning rate is set to 0.005. A loss function is a function that calculates loss or difference between ground truth and predicted value. Since this is regression analysis, Mean Squared Error (MSE) is used. PyTorch [27] is an open-source machine learning framework base on Python. An implementation of the deep learning neural network in this study is done using PyTorch.

Before sending the data into the neural network, each sky image has to go through 3 pre-processing steps: normalization, standardization, and resizing. Each pre-processing step helps to improve the performance of the neural network. The sky images are in RGB format, which means each pixel has three values representing the intensity of red, blue, and green

color, respectively. With 8-bits encoding, there are 2^8 or 256 possible values ranging from 0 to 255. In the normalization process, the minimum of the original value is replaced by 0, while the maximum of the original value is replaced by 1. Any values between the minimum and the maximum are calculated by the equation 2.1 where x, min, max, and x_{norm} are original value, minimum value, maximum value, and normalized value that is ranging from 0 to 1.

$$x_{norm} = \frac{x - min}{max - min} \tag{2.1}$$

The second step is standardization. In this step, normalized values are standardized by the mean and standard deviation. The equation 2.2 describes the process where x_{norm}, μ, σ, and x_{std} are normalized value, mean, standard deviation, and standardized value respectively. In this study, both mean and standard deviation are set to 0.5 for all the data set images. As a result, the standardized values range from -1 to 1.

$$x_{std} = \frac{x_{norm} - \mu}{\sigma} \tag{2.2}$$

The last step is resizing the images. The sky images' original size is 808 x 608 pixels, so the ratio of width to height is 1.329. The width of resized images is set to 64 pixels. To keep the original ratio, the height of resized images is 48.158 pixels. As the number of width or height of images has to be an integer, the height of resized images is rounded to 48 pixels. This downsizing procedure is applied using bilinear interpolation. Each pre-processing step is applied independently to each red, green, and blue layer of the images. Therefore, the outcome of pre-processing steps is a 3 channel array of 64 x 48 pixels. Each value ranges from -1 to 1. Then each image is fed into the deep learning neural network as following steps.

1. Convolutional layer 1. This layer applies 5 x 5 convolution operation to 3 channel arrays of 64 x 48 pixels input. The output is 16 channel arrays of 60 x 44 pixels.

2. Pooling layer 1. This layer is 2 x 2 max pooling layer. It creates a 2 x 2 pixel scope then searches for the maximum values inside the scope. The width and height of input

are half while the number of the channel remains unchanged. As a result, the output from this layer is 16 channel arrays of 30 x 22 pixels.

3. Convolutional layer 2 applies 3 x 3 convolution operation to the output from the previous step. The result is 64 channel arrays of 28 x 20 pixels.

4. Pooling layer 2 applies a similar operation as pooling layer 1 to the output from convolution layer 2. The output from this layer is 64 channel arrays of 14 x 10 pixels.

5. Convolutional layer 3 applies 3 x 3 convolution operation similar to convolutional layer 2. The output from this layer is 256 channel arrays of 12 x 8 pixels.

6. Pooling layer 3 repeats the same procedures as in pooling layer 1 and pooling layer 2. The outcome is 256 channel arrays of 6 x 4 pixels.

7. At this point, there are 256 x 6 x 4 or 6,144 pixels in total. All pixels are rearranged into a vector of 6,144 features before feeding to fully connected layer 1. This layer maps a 6,144-feature vector into a 1,024-feature vector.

8. Fully connected layer 2 continues mapping features from 1,024 to 32 features.

9. Fully connected layer 3 predicts the final values from 32 features. This value is a predicted intensity corresponding to a particular sky image.

This neural network structure can predict the intensity of red, green, blue, or average RGB. Same sky images can be used to train neural networks regardless of the color of the output. Still, the ground truth used in the training process has to match with the desired output. For example, to predict average RGB intensity, ground truth data has to be average RGB intensity. Although neural network architecture is identical, different ground yields different neural network instances. Hence, there are 4 discrete neural network instances for each color.

The neural network architecture described earlier is designed to fit with the original sky images before the segmentation process mentioned in section 2.3.3. Unlike the original size of the sky images, the post-segmentation size is 760 x 608 pixels. This causes size an

incompatible issue. Therefore, the neural network architecture has to be revised according to the size of post-segmentation sky images. Similar to original sky image, 3 pre-processing steps apply to each segmented sky image. Normalization and standardization are perfectly identical but resizing step is modified. To keep the width to height ratio 760 to 608 or 5 to 4 consistent, the images are downsized to 80 x 64 pixels using bilinear interpolation. The structure of the 9 layers remains unchanged. However, the size of the result from each convolution and pooling layer is different. Each layer's differences between neural network structure for original sky images and segmented sky images are summarized in table 2.3. Three numbers in each row from initial to polling-3 layer represent width x height x channel, respectively. For example, original sky images' initial sizes are 808 pixels wide, 608 pixels high, and 3 channels. In the fully connected layer 1 to 3, all pixels are rearranged into a 1-dimension vector. The first number is the vector's size before mapping, while the following number is the vector's size after mapping.

Table 2.3. A result from each layer comparison between neural network structure for original sky images and segmented sky images

Operation or Layer	**Original sky images**	**Segmented sky images**
Initial	808 x 608 x 3	760 x 608 x 3
Resizing	64 x 48 x 3	80 x 64 x 3
Convolution-1	60 x 44 x 16	76 x 60 x 16
Polling-1	30 x 22 x 16	38 x 30 x 16
Convolution-2	28 x 20 x 64	36 x 28 x 64
Polling-2	14 x 10 x 64	18 x 14 x 64
Convolution-3	12 x 8 x 256	16 x 12 x 256
Polling-3	6 x 4 x 256	8 x 6 x 256
Fully connected-1	6,144 to 1,024	12,288 to 1,024
Fully connected-2	1,024 to 32	1,024 to 32
Fully connected-3	32 to 1	32 to 1

The sky images are randomly divided into 2 sets: a training set and a validation set. The ratio between the training set and the validation set is 7 to 3. The training set is used to train the neural network. Then validation set is used to evaluate the prediction performance of the trained neural network. A source code that is used to implement the deep learning neural network is in appendix E.

3. ANALYSIS AND INTERPRETATION OF DATA

3.1 Ground-Truth Analysis

There are 4,978 samples captured over five days: November 16^{th}, November 18^{th}, November 20^{th}, December 10^{th}, and December 11^{th}, 2020. Diverging sampling time helps to collect various sky conditions. Figure 3.1 shows a ground-truth histogram of all samples. The x-axis indicates ground truth values: daylight intensities, while the y-axis represents the number of samples in that particular truth value. Each histogram bin covers five ground truth values, e.g., 0 - 4, 5 - 9, 10-15. A theoretical ground-truth value ranges from 0 to 255. However, practically, extreme values are not captured. Low ground-truth values (e.g., 5 or 10) occur when the sky is dark. The dark sky is not normal working conditions in the farming industry, especially in the field. Very high ground-truth values (e.g., 200 or more) do not appear in the histogram in 3.1 due to the DSLR camera's setting. Increasing the camera's sensitivity or exposure time can expand the range of ground-truth values beyond 200. However, it will be likely that pixel intensity values are saturated at the maximum value (255). Having a marginal gap with the current configuration ensures that pixel intensity values will not be saturated. Overall, the span of ground-truth values ranges from 14 to 201. This range covers 73 percent of all possible values.

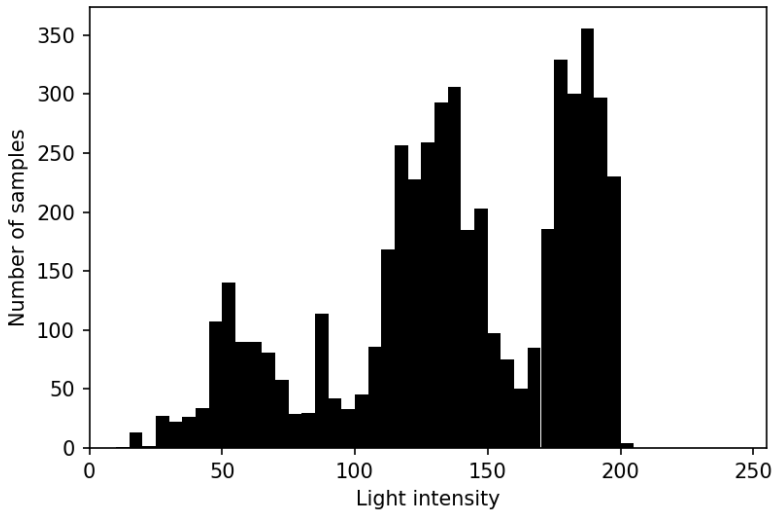


Figure 3.1. A ground-truth histogram. Theoretical value ranges from 0 to 255.

The process in section 2.3.1 excludes some samples from the data set due to value inconsistency caused by shadows. After the inspection, 3,067 samples, or 61.6 percent, remain in the data set. Figure 3.2 visualizes a remaining samples distribution compare to the original data set. The black color and blue color represent the remaining samples and original samples, respectively. The remaining data set spans almost identically to the original data set. However, the filtering process impacts some ranges dramatically. The high ground-truth values are mostly filtered out. Also, the process removes ground-truth values less than 27 altogether. In summary, the remaining data set ranges from 27 to 201, which equals 68 percent of the theoretically possible values.

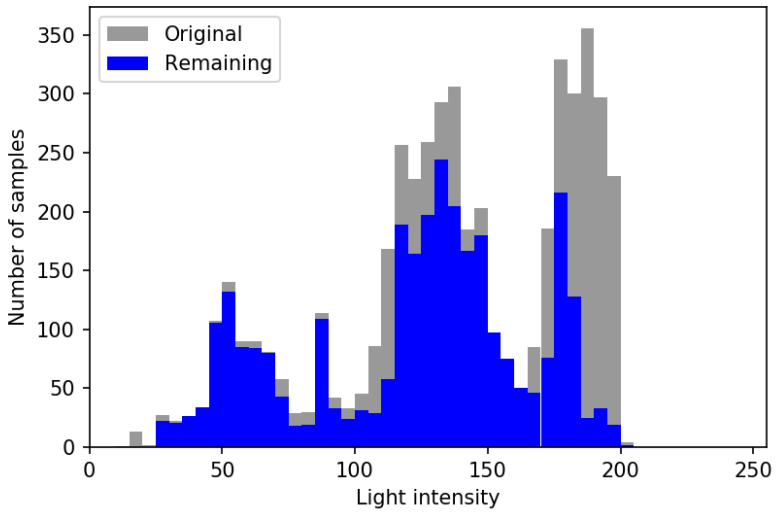
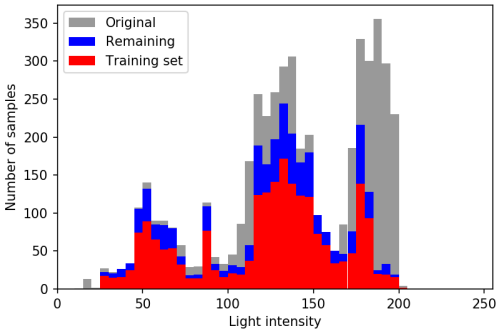
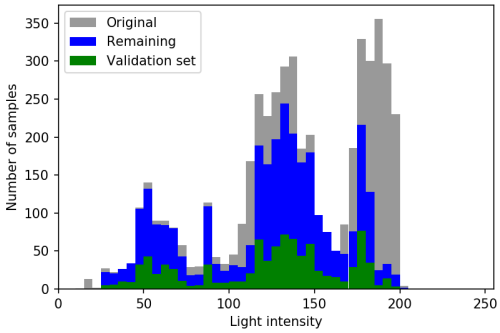


Figure 3.2. A histogram of ground-truth values after filtering process in section 2.3.1.

Each remaining sample is randomly assigned into two groups: a training data set and a validation data set. The ratio between these two groups is 7 to 3. Thus, there are 2,146 and 921 samples in the training data set and validation data set. Figure 3.3a and figure 3.3b shows the training data set distribution and the validation data set distribution respectively. Plot plots show that samples in the training set and validation set cover values as much as possible. It is crucial to have various situations in training set to train the model. Also, distributed validation set helps to evaluate the model performance more precisely.



(a) A distribution of training set.



(b) A distribution of validation set.

Figure 3.3. A distribution of training set and validation set.

3.2 Spectrometer Regression With An Area Feature

Each spectrometer value is the intensity of the daylight in a particular band. Therefore, a summation of each spectrometer data equals overall daylight intensity. Thus, an area under a spectrometer raw data plot represents an accumulated daylight intensity at the sampling moment. Although both ground-truth values and spectrometer data indicate daylight intensity, they are on different scales. Therefore, there must be a coefficient that links these two quantities.

$$y_n = \alpha x_n + c \tag{3.1}$$

A parameter x_n in equation 3.1 designates areas under raw spectrometer data of sample n. A parameter y_n represents corresponding ground-truth values. A coefficient α and constant c are the linear relationships between areas under raw spectrometer data and ground-truth values. Ordinary Least Square (OLS) Linear regression analysis is a tool to find a coefficient α and constant c. The first step is to rearrange areas under raw spectrometer data of sample and ground-truth values into a vector of n elements.

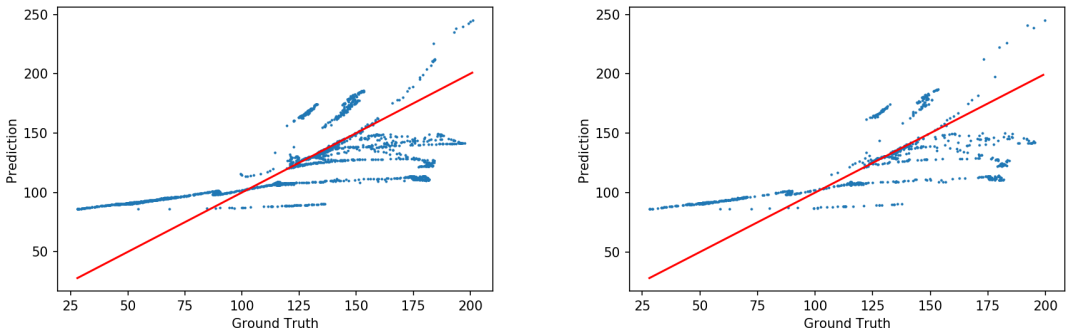
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_{n-1} \\ x_n \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_{n-1} \\ y_n \end{bmatrix}$$

Then use the equation 3.2 to find an optimized α.

$$\alpha = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \tag{3.2}$$

An optimized α is a result of applying samples in the training set. An area under raw spectrometer data x is an explanatory parameter. Thus using α and x yields a predicted daylight intensity $\hat{y}$. A predicted intensity value supposes to be close to its ground-truth value. A plot 3.4 illustrates a prediction result from linear regression model. Ground-truth intensities are on the x-axis, and predicted intensities are on the y-axis. A diagonal red line indicates locations where Ground-truth intensities and predicted intensities are perfectly equal. Plot 3.4a shows prediction result from training set. And plot 3.4b shows prediction result from validation set. Both graphs present an almost identical distribution. Most dots locate far from the red line, indicating that predicted values are different from their ground-truth values. A model is created based on a linear relationship assumption. However, the result does not show a linear relationship between two quantities in the big picture. Yet, there are some linearity existed. Each dot from graph 3.4b is color according to its sampling date. The result is shown in figure 3.5.

Linear trends exist in purple (November 16^{th}),blue (November 18^{th}), and yellow (November 20^{th}) points. Green points do not have any linear trends. The model completely fails to handle the samples from December 11^{th}, 2020. Black points show two lines. However, both lines are not connected the each other. Although the model does not produce accurate predictions from samples in the validation set, evidence shows that predicted values linearly



(a) Predictions from training set.

(b) Predictions from validation set.

Figure 3.4. Prediction from training set and validation set from linear regression.

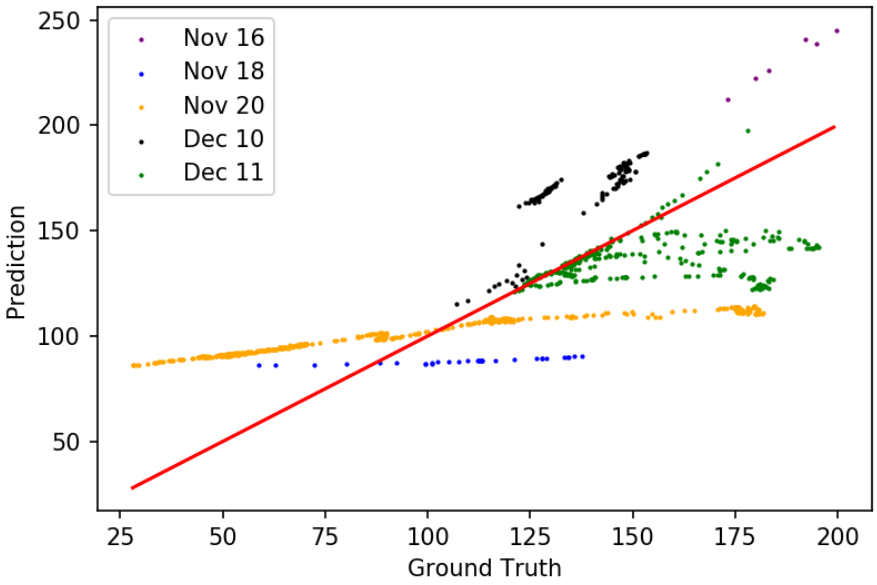
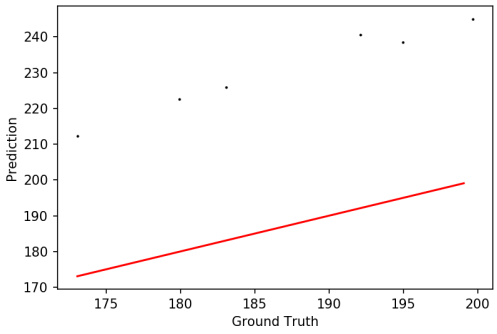
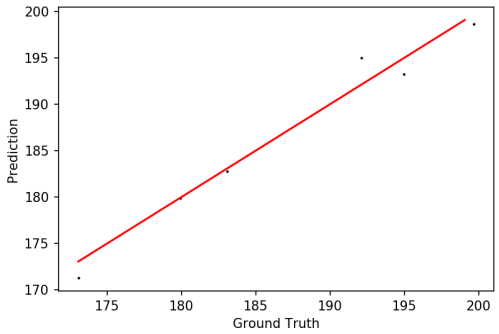


Figure 3.5. The prediction result from the validation data set. Using different color to identity the sampling data.

relate to ground-truth values on some particular days. The further analysis focuses on each sampling day individually. Figure 3.6a shows the cropped area around the purple points for better visualization. The color is changed to black, making better contrast from the background and red line (the correct line).



(a) Original prediction results.



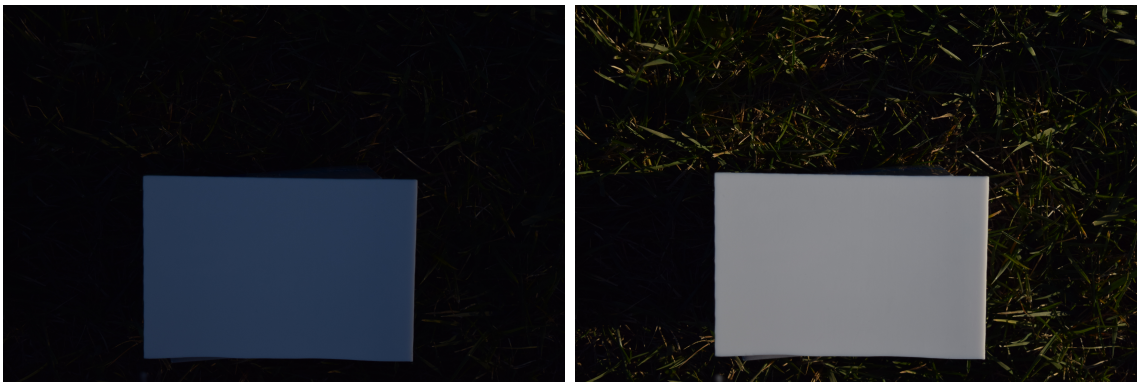
(b) Prediction results after retrain the model.

Figure 3.6. Prediction result from November 16^{th}.

All points in figure 3.6a located away from the red line, indicating that the prediction is not correct. However, they aligned in a precise linear pattern. The linear pattern proves that a linear relationship between an accumulated spectrum intensity and a ground truth intensity exists. A dedicated regression model is trained exclusively by points from November 16^{th}. A result is shown in figure 3.6b. Prediction values fit well with their corresponding ground-truth values. The result demonstrates that the regression method cloud delivers a good product in the scope of a single day.

Samples from November 18^{th}, blue points in figure 3.5, also shows a linear trend. However, the line looks almost like a horizontal line. In other words, predicted values stay constant regardless of ground truth values. The predicted values range from 86 to 90, while ground truth values range from 59 to 138. The correct line should have a slope equals to 1. Nevertheless, the actual slope for these points is only 0.05. The minimum ground-truth value is the same sample that produces the minimum predicted value. Likewise, the maximum ground-truth value is the same sample that produces the maximum predicted value. Figure 3.7a shows the ground truth image when its value equals to 59 (day's minimum). And Figure 3.7bhows the ground truth image when its value equals 138 (day's maximum). A difference between the minimum ground-truth value and maximum ground-truth value is reasonable considering the white reference board's brightness. In contrast, the predicted values barely

change between two points of time. Although there is a linear relationship between ground truth values and predicted values, the slope or rate of change does not reflect the reality.



(a) Minimum ground truth image. (b) Maximum ground truth image.

Figure 3.7. Ground truth images when the values are minimum and maximum on November 18^{th}.

Yellow points in figure 3.5, represent samples from November 20^{th}, 2020, shows linearity. Figure 3.26b is a plot that contains samples exclusively from November 20^{th}, 2020 to provide better visualization. There are six noticeable groups in the plot differentiated by six colors. Points in red and purple do not show any trends or patterns. Like samples from December 10^{th} and December 11^{th}, a regression model cannot deliver accurate predictions. On the other hand, green, yellow, blue, and grey points have a linear trend. Although a linear trend indicating a relationship between ground truth values predicted values exists, the rate of change or slope is not correct. Although all samples are from the same day, each group's slope is not equal. Therefore, it is impossible to have a linear regression model predicting intensity from all samples correctly. Building several models to handle each condition is not a practical approach. Lastly, the yellow group has an inverse slope which is counter-intuitive. A predicted intensity decrease while a ground truth intensity increases. All evidence reveals an impracticability of the regression model from spectrometer data.

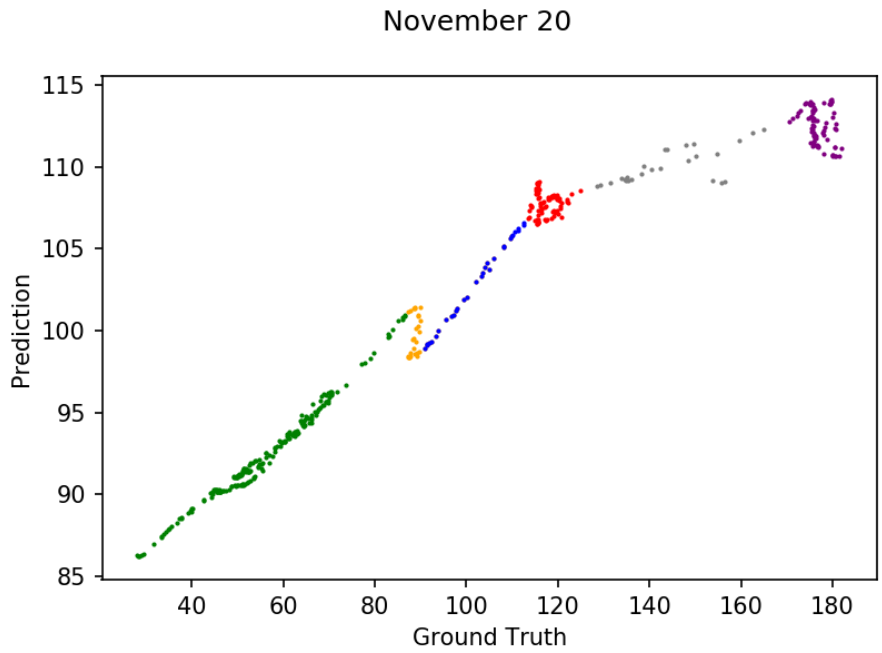


Figure 3.8. Prediction results exclusively to the samples from November 20^{th}, 2020.

3.3 Spectrometer Regression With Full Features

Each raw spectrometer contains 3,840 band's intensity information. In section 3.2, a regression model with areas under spectrometer plot does not perform well. There is only one feature used to build a linear relationship between spectrometer data and ground-truth intensities. However, there are 3,840 features from each spectrometer sample. The following equation defines a new linear relationship utilizing all features provided by the spectrometer.

$$\begin{aligned}\hat{y}_n &= \alpha_0 + x_{n,1}\alpha_1 + x_{2,1}\alpha_2 + ... + x_{n,3840}\alpha_{3840} \\ &= \alpha_0 + \sum_{m=1}^{3840} x_{n,m}\alpha_m\end{aligned}$$

An Ordinary Least Square (OLS) linear regression finds an optimized set of coefficients (α). All features need to arrange into a matrix. $x_{n,m}$ represents m^{th} wavelength intensity from n^{th} sampling. y^n represents ground-truth value that corresponding to n^{th} sample. α_m

represents a coefficient of m^{th} wavelength. Equation 3.4 shows the solution from the ordinary least square method.

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,3840} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,3840} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ x_{n-1,1} & x_{n-1,2} & x_{n-1,3} & \dots & x_{n-1,3840} \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,3840} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ . \\ . \\ . \\ \alpha_{3839} \\ \alpha_{3840} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_{n-1} \\ y_n \end{bmatrix} \tag{3.3}$$

$$\alpha = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.4}$$

The same training set as section 3.2 is used to find an optimized set of coefficients. Figure 3.9 shows the prediction result. All dots are behind the red line indicating that predictions are perfectly correct.

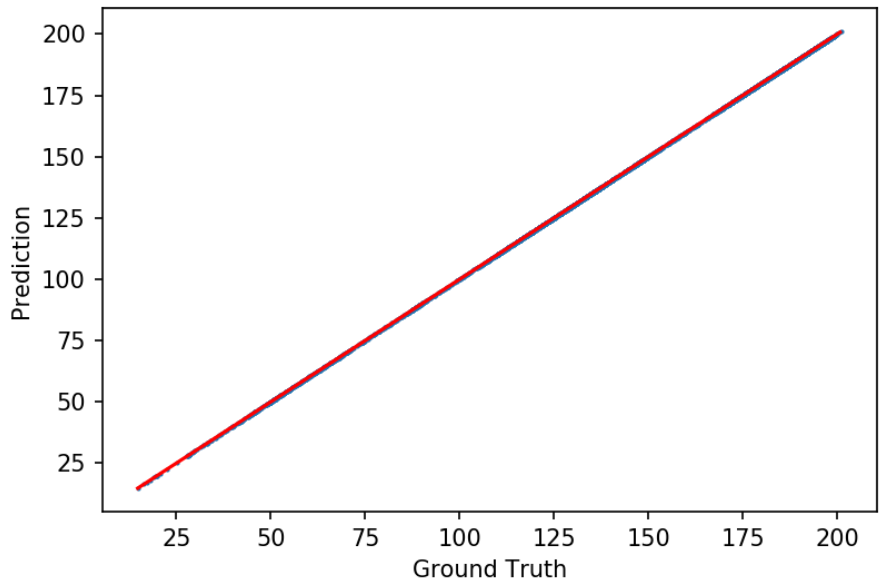


Figure 3.9. Prediction results from training data set.

This phenomenon occurs because there are 3,840 coefficients while there are 2,146 samples. The number of variables is more than observations. Therefore this is an overdetermined system. The solution can exist in overdetermined systems but potentially unstable. In this particular case, the solution exists. Predicted values match ground-truth values perfectly. The validation set has not been seen by the linear regression model before. Therefore, the validation set is a fair evaluator to measure the model's prediction ability. Figure 3.10 shows prediction results from the validtaion set. The predictions distribute widely and away from the red line. The model fails accurately predict unseen data in the validation.

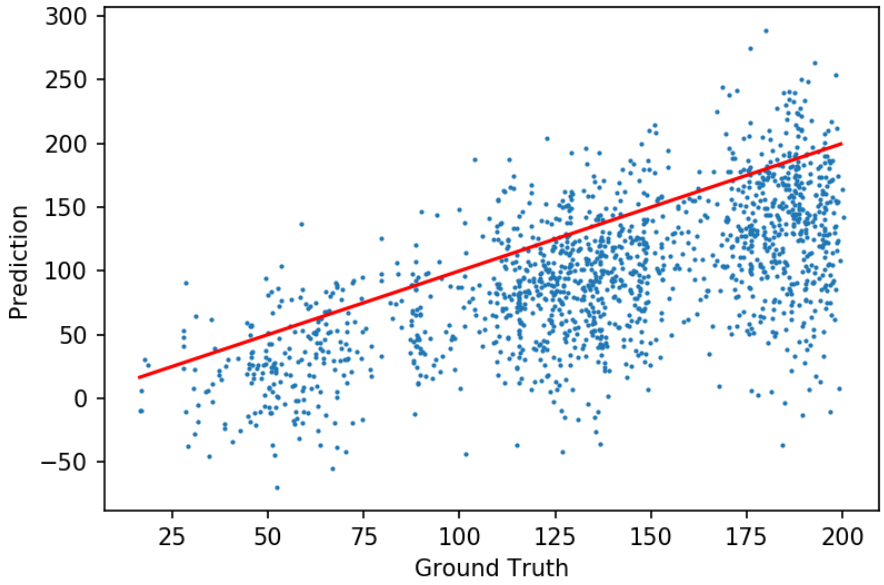


Figure 3.10. Prediction results from validation data set.

Figure 3.11 shows the regression coefficients. Intuitively, each wavelength's intensity should contribute to accumulating the overall intensity spectrum. Coefficients spread out in both positive and negative regions. As a result, any coefficient is most likely to have a pair with equal value but opposite sign to cancel each other out.

By nature, intensities of nearby wavelengths correlate to each other. For example, a light whose wavelength is 500 nanometers is a green light. An intensity of 500 nanometers of light is related to 490 nanometers and 510 nanometers of light. Spectrometer raw data

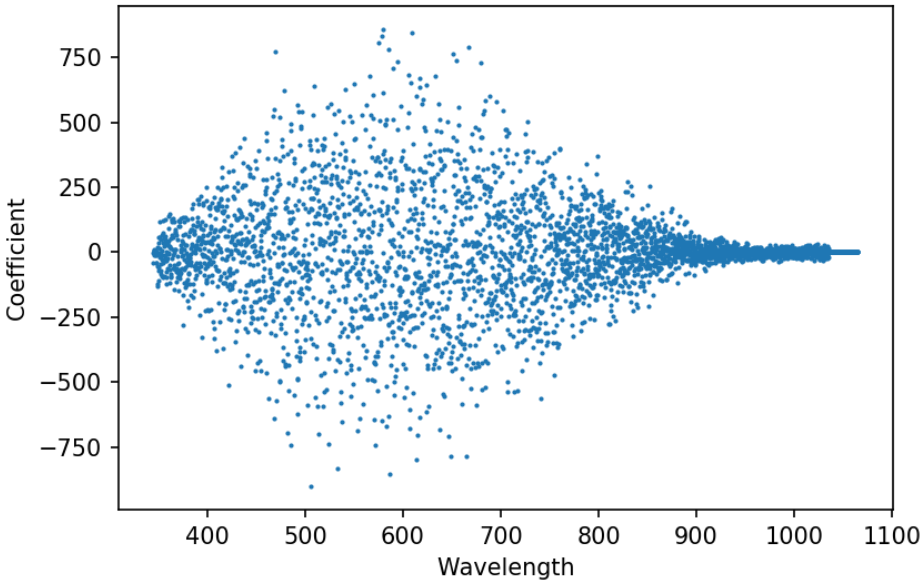


Figure 3.11. Optimized coefficient values from Ordinary Least Square linear regression.

has 3,840 intensity values that cover wavelengths from 344 nanometers to 1,065 nanometers. Thus, any two adjacent values are 0.188 nanometers apart from each other. Therefore, the raw spectrometer data potentially have a high correlation issue. Figure 3.12 examines a correlation between each value for the entire raw spectrometer data set. Instead of using all 3,840 variables, a graph uses every 20 variable (wavelength 1^{th}, 20^{th}, 40^{th}, ..., 3840^{th}) to calculate correlation. As a result, there are 192 variables in a correlation matrix in figure 3.12. Each axis shows wavelength. The brighter color, e.g., yellow, indicates a high correlation, while the darker color, e.g., purple, represents a low correlation. Light from 345 nanometers to 930 nanometers, or 80 percent of the spectrometer range, is highly correlated. Another 20 percent or from 930 nanometers to 1,065 nanometers also has a high correlation issue.

Ridge regression can handle a multicollinearity data set. It is a modified Ordinary Least Square (OLS) linear regression using L2 regularization on estimated coefficients. Equation 3.4 describes the solution of Ordinary Least Square linear regression. A solution for Ridge

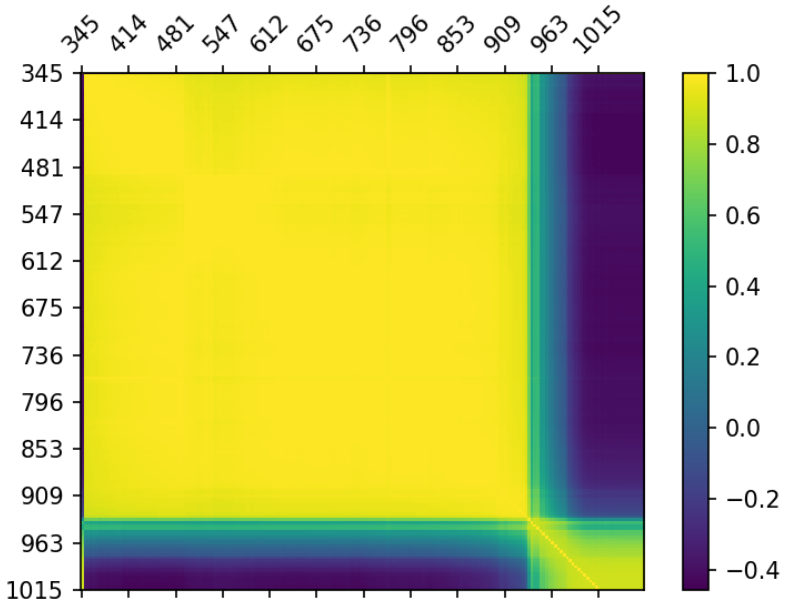


Figure 3.12. A spectrometer data correlation matrix.

regression adds $k\mathbf{I}$, where k is a regularization strength and $\mathbf{I}$ is an identity matrix. Equation 3.5 describe the solution of Ridge regression.

$$\hat{\mathbf{W}}_R = (\mathbf{X}^T\mathbf{X} + k\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.5}$$

This study fixes a regularization strength for Ridge regression to 0.1. The same training set is used to compute an optimized coefficient set. Figure 3.13 shows the optimized coefficient values from Ridge linear regression. Comparing to optimized coefficient values from Ordinary Least square linear regression, figure 3.11, the new optimized coefficient values are less distributed. Figure 3.14a and figure 3.14b shows predictions from training set and predictions from validation set respectively.

The prediction from the training data set using the coefficient set from Ridge regression is not perfect, unlike the result from the training set using the coefficient from the Ordinary Least Square method (figure 3.9). The prediction shows that Ridge regression's coefficients become more stable. The prediction from the validation set performs significantly better compared to the Ordinary Least Square model. The validation set prediction result has R-

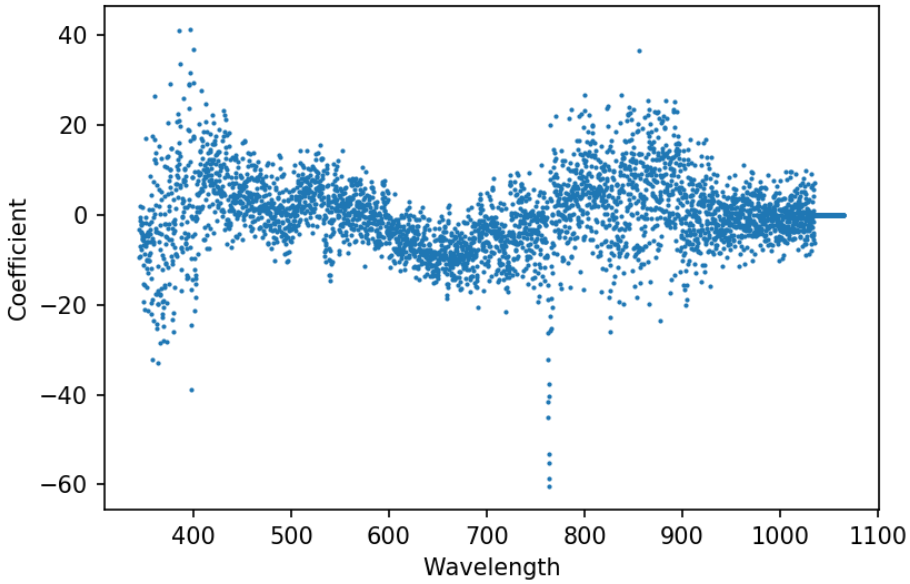
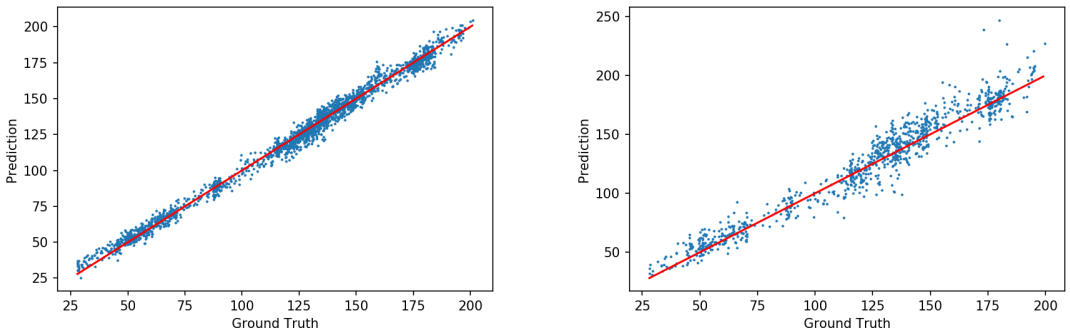


Figure 3.13. Optimized coefficient values from Ridge linear regression.

squared equals to 0.937. R-square is a numeric indicator that measures a regression model's prediction performance. The maximum possible R-square is 1, which means all predictions perfectly fit the actual values. In general, a higher R-square value shows a better model's accuracy. Overall, the Ridge regression model demonstrates an excellent ability to produce accurate daylight intensity values.

The regression model generates a predicted intensity by considering each raw spectrometer band intensity. Intuitively, the higher the spectrometer values mean, the brighter the sunlight. Figure 3.15 shows raw data plots from two random samples in the validation set. The values in the top left corner are each line's predicted intensity. The regression model predicts 167.2 and 181.6 for samples in the orange line and blue line, respectively. The predicted values show contradiction to the raw spectrometer data plots. The orange line's predicted value is less than the blue line's predicted value. The plot shows that the orange line has higher raw spectrum intensities than the blue line. The contradiction is a result of the overfitting issue. Although the model performs well with the validation set, the validation set is sampled from the same population pool as the training set. The overfitting issue



(a) Predictions from training set.

(b) Predictions from validation set.

Figure 3.14. Prediction from training set and validation set using the coefficient set from Ridge regression

impacts the model's ability to accurately predict data outside the training set, e.g., the data from other sampling days. As a result, this regression model is not a functional model to be used in practical scenarios.

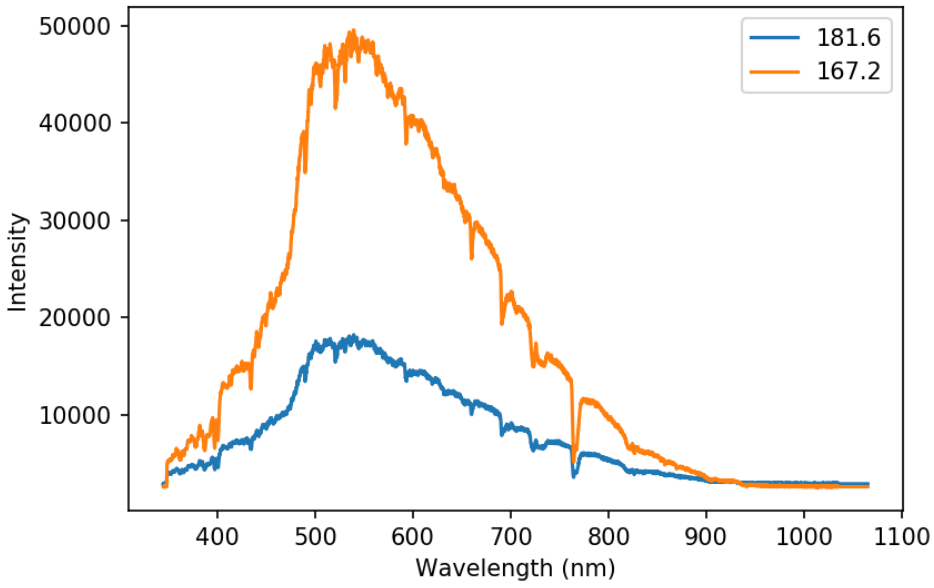


Figure 3.15. Two random samples from validation data set. The numbers on the top left corner are their prediction values.

3.4 Deep Learning From Original Sky Images

Each sky image links with its corresponding ground-truth value using a timestamp to find the connection. Each sky image is assigned into two groups: a training set and a validation according to the group, its ground truth belongs to in section 3.1. The training data set is used to train the convolutional neural network described in section 2.4 A neural network analyzes each image and provides a predicted value. The difference between a predicted value and a ground-truth value is "an error" or "a loss". The training process continues until the loss is minimized, which means the neural network predictions are close to the ground-truth values. Figure 3.16 shows prediction results from the training set. The x-axis is ground-truth values, and the y-axis is neural network predictions. Each blue dot represents one prediction. A red diagonal dash line from the lower-left corner to the upper-right corner is the correct line which means a prediction value equals its corresponding ground truth. The closer of the blue dots to the correct line means the more accurate the predictions.

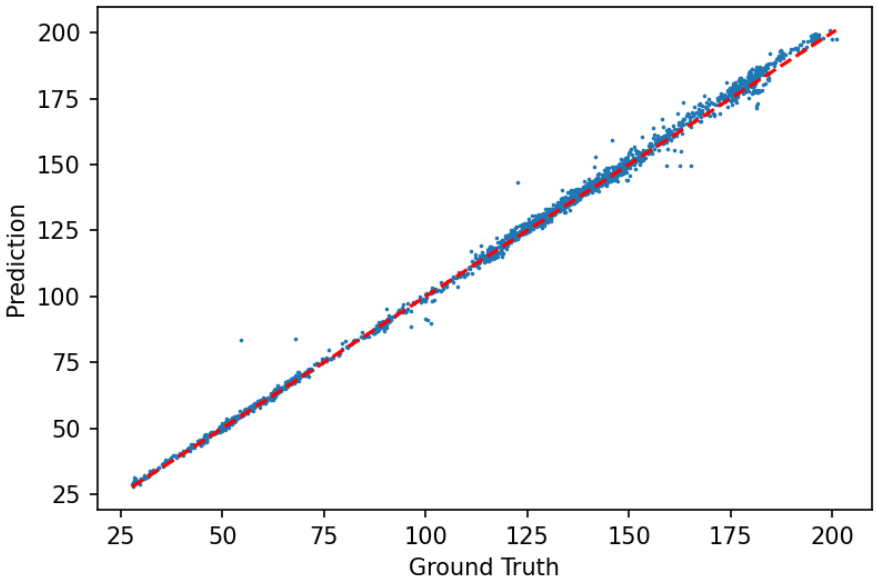


Figure 3.16. Prediction results from training set.

The neural network has analyzed images in training set during the training process, including their ground truth values. The neural network adjusts its predictions from the loss

function's feedback. In contrast, the validation data set is unseen from the neural network. The neural network does not know the ground-truth values of samples from the validation set. Therefore, the validation set's prediction result is better to evaluate a deep learning neural network performance than the training set's prediction result. Figure 3.17 shows predictions result from the validation set.

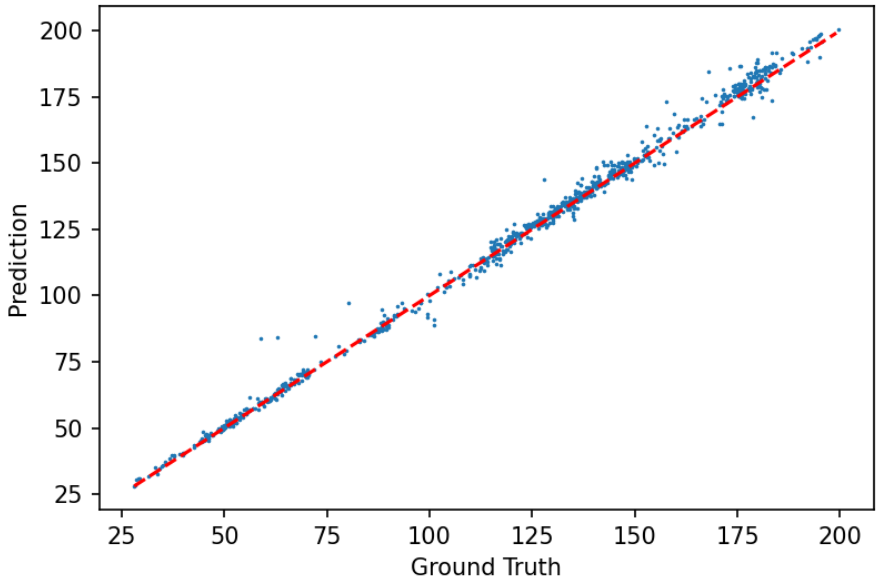


Figure 3.17. Prediction results from validation set.

An R-square of validation data set prediction is 0.997. This number is close to 1, telling that predictions are highly equivalent to their ground truth values. Figure 3.18 shows a histrogram distribution of prediction's errors. A positive value means a prediction is more than its ground truth value. While a negative value means a prediction is less than its ground truth value. The blue color is a histogram that illustrates an actual error distribution. The x-axis represents error values, and the y-axis indicates error probabilities. Most of the errors locate nearby 0 (a red dash line), which supports a claim that the predictions are highly similar to their ground-truth values. Error's mean and standard deviation are 1.08 and 2.82, respectively. The orange line in figure 3.18 shows a Gaussian distribution of the errors according to mean and standard deviation.

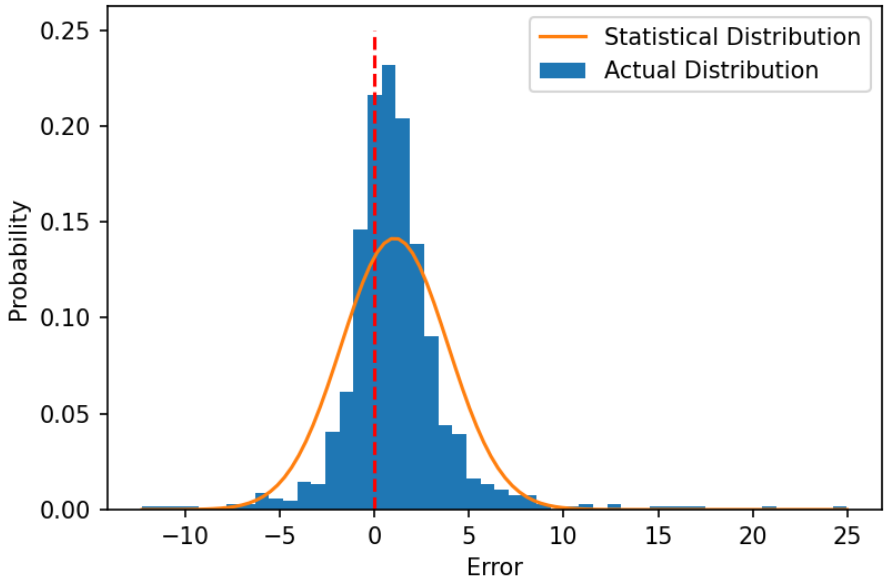


Figure 3.18. A validation set prediction error distribution.

Although most of the predictions locate close to 0, some predictions perform poorly. Equation 3.6, where determines an error percentage relative to its ground-truth value. A prediction error, ground truth value, and error percentage are designated as err, gt, and $err_{percent}$ respectively. Any predictions that miss their ground truth values more than 10 percent (-10 $\leq$ error percentage $\leq$ 10) are classified as "outliers".

$$err_{percent} = \frac{err \times 100}{gt} \tag{3.6}$$

There are seven outliers out of 921 predictions or 0.8 percent. From seven outliers occurring in this model, there are two groups according to their timestamp. One outlier is a result of a sample at 2:37:49 p.m. on December 10th, 2020. The rest six outliers are from samples during four minutes period on November 18th, 2020.

The first situation occurred at 2:37:49 p.m. on December 10th, 2020. The sky at that time was partially cloudy. From the sky image in figure 3.20a, the model predicts an intensity value is 143.9. The ground truth value (3.20d) is 128.0. The prediction is 15.9, or 12.5 percent, lower than the ground truth. Considering two following sky images at 2:38:04

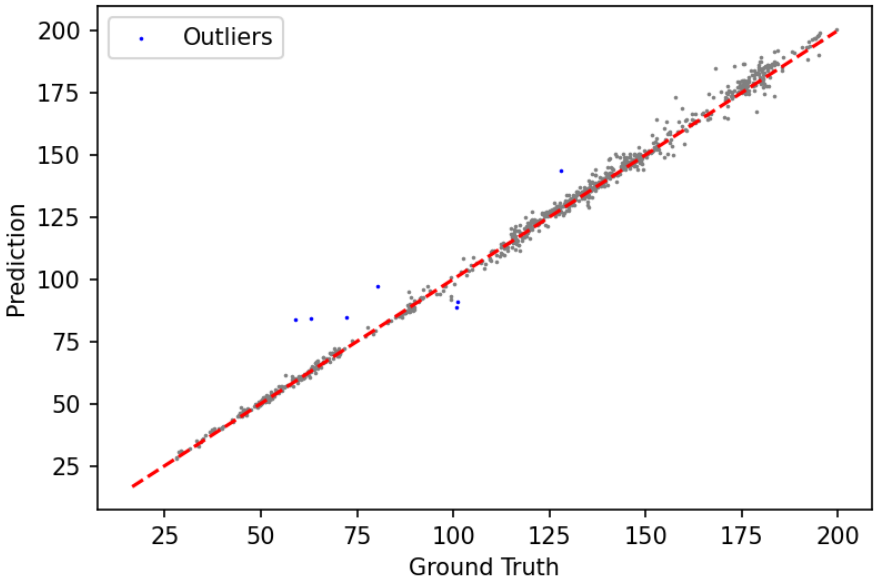
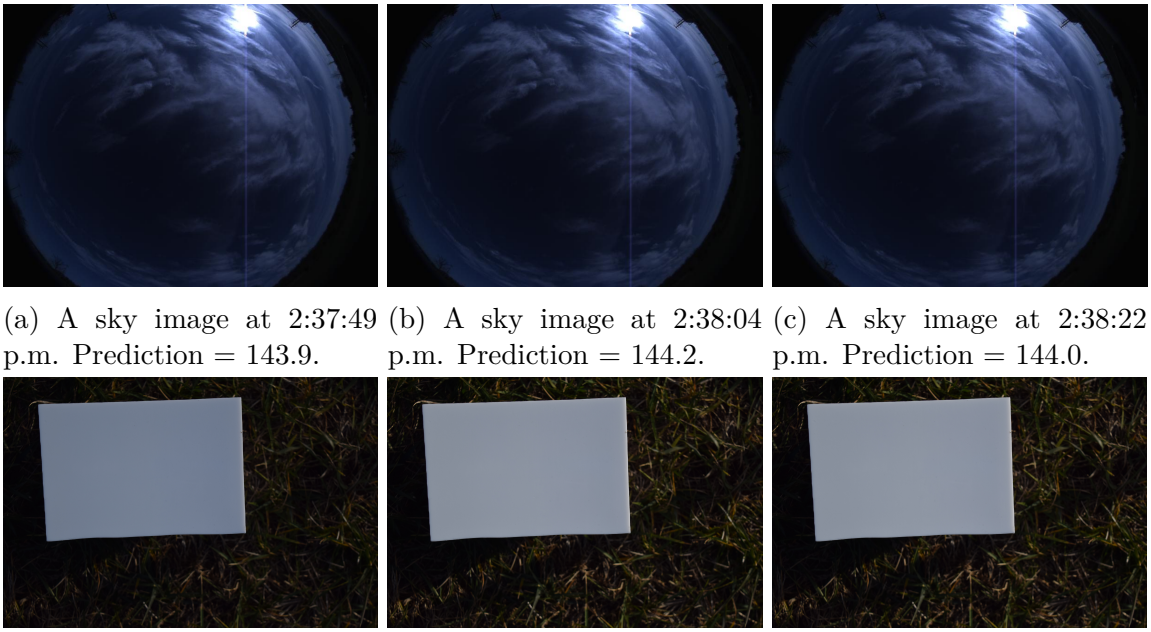


Figure 3.19. Prediction results from filtered validation data set.

p.m. (3.20b) and 2:38:22 p.m.(3.20c), the Sun's location and the sky condition are visually identical for all three images. Additionally, the model predictions are consistent, ranging from 143.9 to 144.2 or 144.05 $\pm$ 0.15. Meanwhile, ground truth images at at 2:38:04 p.m. (3.20e) and 2:38:22 p.m.(3.20f) are brighter than the ground truth image (3.20d). Ground truth values increase, reflecting changes of brightness. The model makes correct predictions for the latter two situations. However, a change on ground truth image from 2:37:49 p.m. to 2:38:04 p.m. does not appear on the sky. Since the model relies on the sky condition, therefore, it cannot produce a correct prediction in this situation.

Another situation occurred on November 18th, 2020. Figure 3.21 illustrates ground truth values and prediction of 26 samples from 3:30 p.m. to 3:50 p.m on November 18th, 2020. The blue line represents ground truth values, and the orange line represents prediction values. A red region indicates the time when outliers occurred. From 3:42 p.m., the ground truth values start dropping until 3:46 p.m. before rebounding rapidly. The predictions follow the same trend as ground truth values. However, it cannot keep up with ground truth changing rates. As a result, from 3:44:43 p.m. to 3:47:50 p.m., there are six consecutive outliers.



(a) A sky image at 2:37:49 p.m. Prediction = 143.9.
(b) A sky image at 2:38:04 p.m. Prediction = 144.2.
(c) A sky image at 2:38:22 p.m. Prediction = 144.0.
(d) A ground truth at 2:37:49 p.m. Value = 127.9.
(e) A ground truth at 2:38:04 p.m. Value = 147.8.
(f) A ground truth at 2:38:22 p.m. Value = 149.3.

Figure 3.20. 3 continuous sky images corresponding ground truth values images on December 10th, 2020.

Figure 3.22 presents 4 samples from from 3:40 p.m. to 3:50. Sky images on the top row are time-synchronized with ground truth images in the bottom row. From left to right, the sampling times are 3:42:16 p.m. (figure 3.22a and 3.22e), 3:44:14 p.m. (figure 3.22b and 3.22f), 3:46:14 p.m. (figure 3.22c and 3.22g), and 3:48:36 p.m. (figure 3.22d and 3.22h), respectively. There are no clouds in any sample sky images during this period. Moreover, the Sun locates at the top edge of the figure 3.22a. Eventually, the Sun disappears from the latter sky images. Consequently, the model has little information about objects in the sky. Thus, predictions are not accurate in this condition. Figure 3.22f, 3.22g, and 3.22d explain the phenomenon in the red region in figure 3.21. Figure 3.22g is , sampling at 3:46:14 p.m., the valley the ground truth line. The image is comparatively darker than the other two, which are captured before and after this image. It indicates that ground truth values quickly drop until 3:46:14, then rise back.

A deep learning model can also predict each Red, Green, and Blue bands individually using the same neural network architecture. Instead of training the model with RGB average

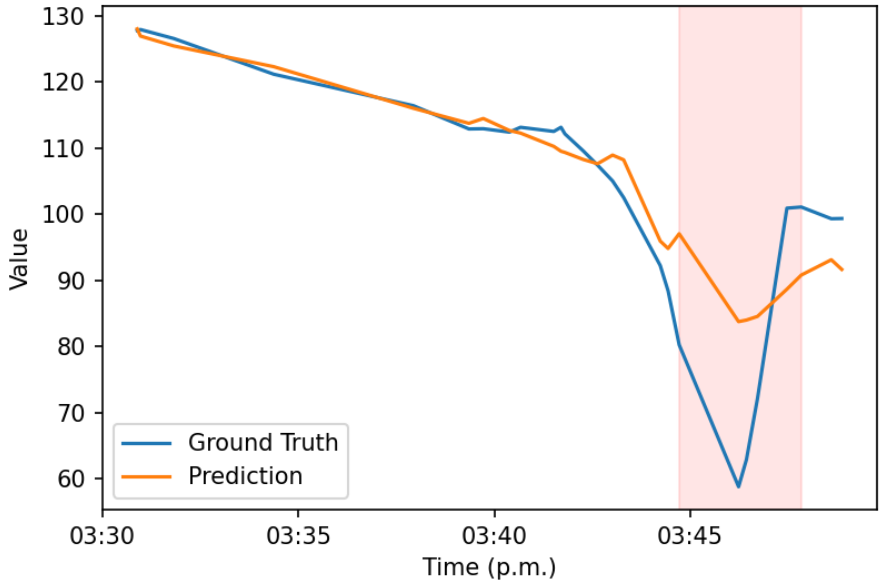


Figure 3.21. Prediction results from 3:30 p.m. to 3:50 p.m on November 18th, 2020.

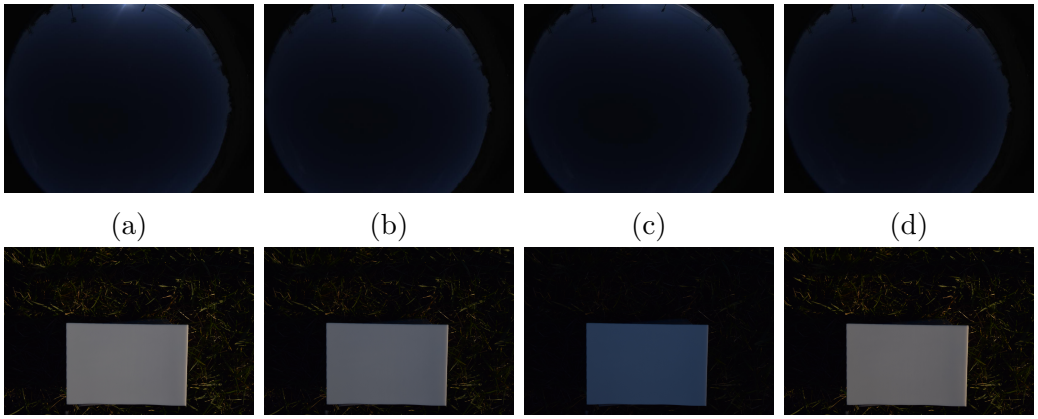
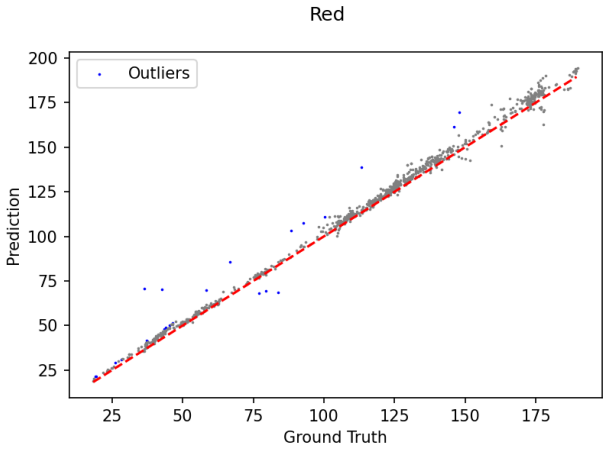
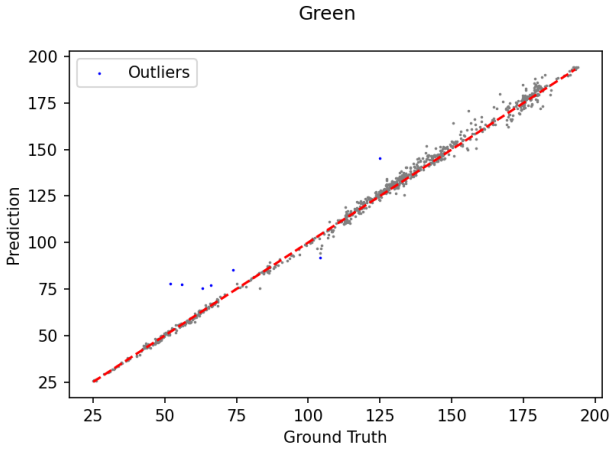


Figure 3.22. 4 samples sky images corresponding ground truth images on November 18th, 2020, from 3:40 p.m. to 3:50.

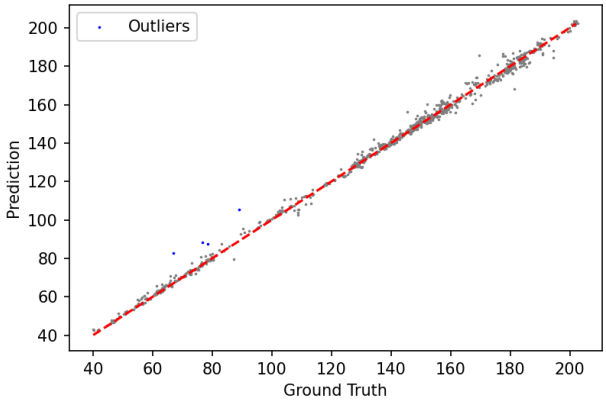
ground-truth values, each Red, Green, and Blue pixel intensities are used separately. For example, training the model with red pixel ground-truth values yields a model that is capable of prediction daylight intensity in red color band. Therefore, there are 3 models trained to predict with Red, Green, and Blue. The prediction results and their outliers for each model are shown in figure 3.23.



(a) Red



(b) Green



(c) Blue

Figure 3.23. Prediction result and outliers from Red, Green, and Blue bands.

All four outliers are from November 18^{th} simultaneously as outliers from the average RGB model exist. There are seven outliers in the green model's prediction. Similar to the blue model, all six from seven outliers are from November 18^{th} at the same time as the average RGB model's outliers. Moreover, the last outlier is from December 10^{th} is identical to the average RGB model's December 10^{th} outliers. The red model, on the other hand, performs significantly worse than the other two models. There are 23 outliers from 921 predictions or 2.5 percent Seven outliers match with the average RGB model's outliers. The remaining 16 outliers are from November 20^{th}, December 10^{th}, and December 11^{th}. However, the causes of these outliers fall into the same reasons as previously outliers. On November 20^{th} outliers occurs from samples whose timestamp are later than 4:10 p.m. Therefore, the Sun no longer appears in the sky camera's frame. The same reason explains outliers from December 10^{th} when outliers are from three samples whose timestamp is at 3:46 p.m. And the data collection stopped after that minute. One outlier from December 11^{th} has a timestamp at 12:25:09 p.m. The Sun is observable. The exact phenomenon that happened on December 10^{th} at 2:37:49 is the cause of the error. The sky did not change at that particular moment. However, there was a brightness drop that reduces the pixel intensity of the white reference board. Therefore, the predicted value is higher than the ground-truth value.

3.5 Deep Learning From Segmented Sky Images

Section 2.3.3 shows procedures to remove a non-sky part from the sky images and centralize the sky to the center of each image. Segmented sky images are divided into a segmented training data set and a segmented validation data set. Each image is assigned according to the process in section 3.1. Hence, the timestamp of each group's image is identical to previous models. A new neural network is trained by the segmented train data set. The segmented training data set's prediction result is in appendix E. Figure 3.24 shows prediction results from segmented validation data set. Each prediction is compared with its ground truth to calculate an error rate. Outliers are predictions whose error rate is more than 10 percent, the same criteria as previous analysis. There are nine predictions identified as outliers showing in blue color in figure 3.24. The outliers can be separated into two groups by their times-

tamp. The first group consists of one sample from December 12th, 2020. Another group has eight outliers from November 18th, 2020. The single outlier on December 10th, 2020, has a timestamp at 2:37:49 p.m., identical to the outlier found in section 3.4. The prediction from the current model is 142.9, while its ground truth value is 127.9. The new prediction is a bit closer to its ground truth value compared to the prediction (143.9) from the model in section 3.4. However, the difference is not significant. As 3.20 showing observable brightness changing on ground truth images while the sky is static, neither model can deliver correct prediction in this particular situation.

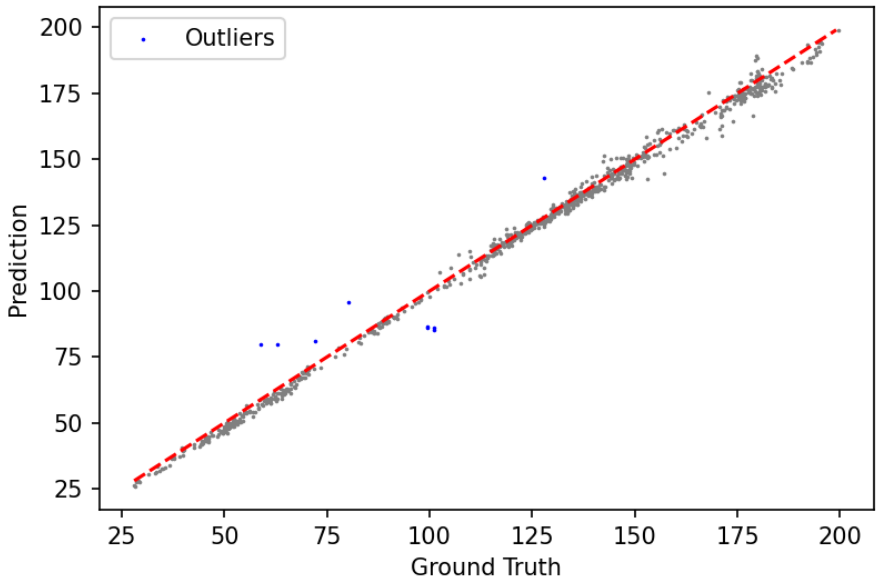


Figure 3.24. Prediction results from segmented validation data set.

Other eight outliers occurred on November 18th, 2020. The first outlier has a timestamp at 3:44:43 p.m., which is the same time as the model in section 3.4 first outlier on November 18th, 2020. The remaining seven outliers happened continuously after the first outlier until 3:48:52 p.m., which is the last sample of the day. Figure 3.25 shows ground truth values (blue line), predictions from original sky images (orange line), and predictions from segmented sky images (green line). All outliers from the original sky image model exist (orange region) in the segmented sky image model (green area) due to a lack of object in the sky as demonstrating

in figure 3.22. Although the last two samples are not outliers in the original sky image model, their error rates are 6.2 and 7.8 percent, respectively. These error rates are less than the outlier's threshold at 10 percent, still significantly higher than the average error rate (1.08 percent). Prediction error rates from both models tend to increase during the 5 minutes before the first outlier occurred.

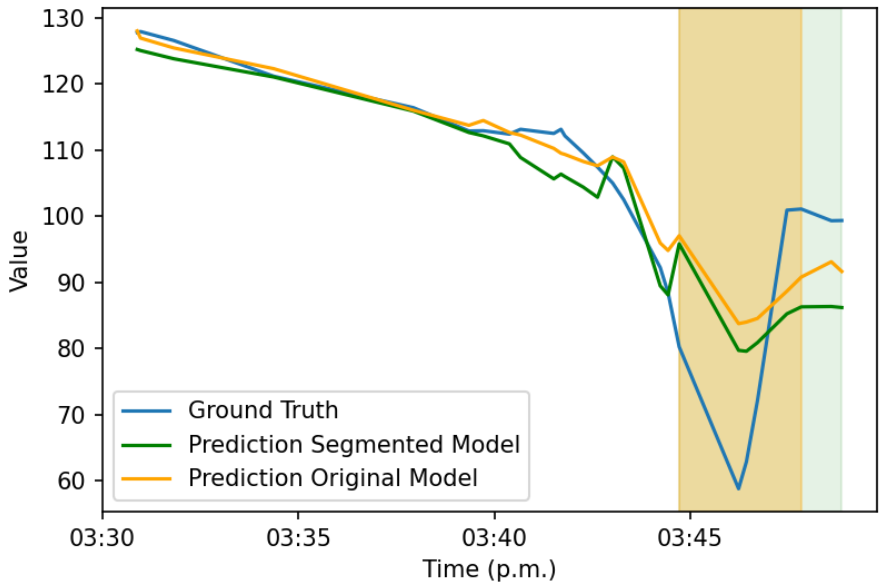
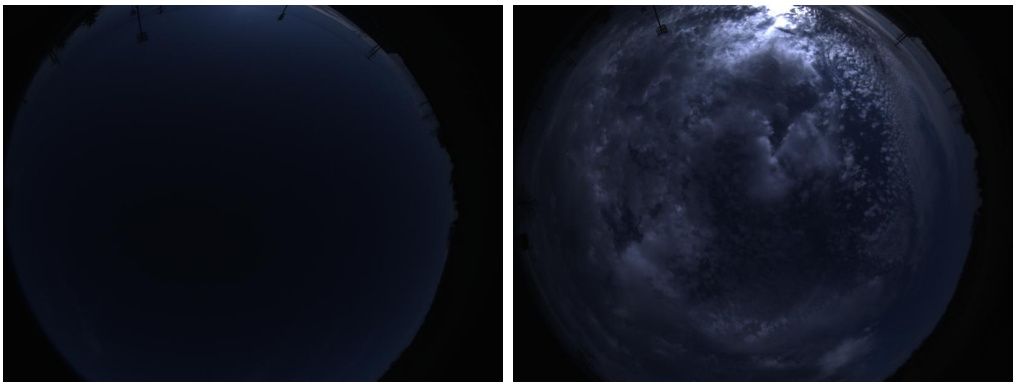


Figure 3.25. Prediction results from 3:30 p.m. to 3:50 p.m on November 18th, 2020 from two models.

As the time pass in the afternoon, the Sun's angle is getting low. At a particular time, depending on the season, the Sun may be out of the sky camera's frame. The samples in this study were collected during early winter. Therefore the Sun disappears from the sky earlier than in other seasons. As a result, the deep learning model that uses sky images to predict a light intensity loses its ability to produce accurate predictions when the Sun no longer presents in the sky. Especially on the clear sky day, e.g., on November 18th, 2020, no other objects exist. However, November 20th, 2020, was a cloudy day, and some samples were collected on this day. The Sun is not directly observable in the images taken on November 20th after 3:30 p.m. Nevertheless, there are several cloudy remaining in the sky. Also, the sunlight projects its light onto the clouds. Figure 3.26a shows the sky image

taken on November 18th, 2020, at 3:46:14 p.m., when the prediction error rate is the highest from both the original sky image model and segmented sky image model. Figure 3.26b shows the sky image taken on November 20th, 2020 at 3:46:13 p.m., about the same time to figure 3.26a. However, the prediction figure 3.26b has very low error rate. Figure 3.27 shows an error rate comparison of sky images from November 18th and November 20th during 3:30 p.m. and during 3:50 p.m. Predictions on November 20th, a cloudy sky day, do not show any signs of high error rate, while predictions on November 18th, a clear sky day, have high error rates and outliers after 3:40 p.m. Both days are two days apart from each other, so the Sun's angle does not change significantly. The most significant difference which influences the model's prediction accuracy is clouds in the sky. This is evidence showing that the Sun location and the sky condition contribute to the deep learning model's performance.



(a) A sky image taken on November 18th, 2020 at 3:46:14 p.m.

(b) A sky image taken on November 20th, 2020 at 3:46:13 p.m.

Figure 3.26. A distribution of filtered training data set and filtered validation data set compare to filtered data set population.

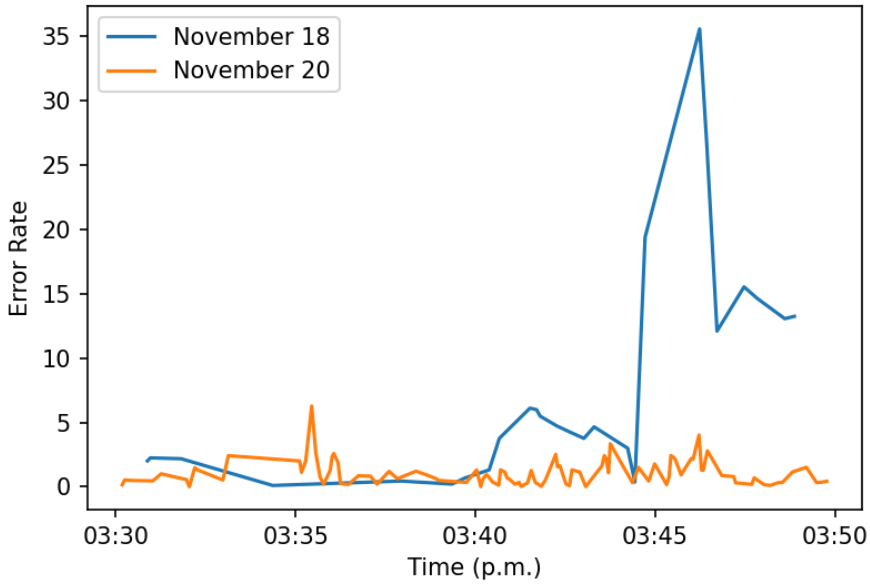


Figure 3.27. Error rate comparison of samples from 3:30 p.m. to 3:50 p.m on November 18th, 2020 and November 20th, 2020.

4. CONCLUSIONS AND FUTURE WORKS

4.1 Conclusions

An accumulated spectrum intensity is used as a feature of Ordinary Least Square linear regression model to predict light intensity. The result shows poor prediction results. However, there are some linear trends embedded in the result. The data are divided into groups according to their sampling date. Sample from November 16^{th} displace from the correctly if using the overall regression to predict. On the other hand, using only data from that particular day to train a separated linear regression yield an excellent prediction result. Data on November 20^{th} shows several linear lines in a single day. Each line has a different slope and does not connect seamlessly to other lines. Although a linear relationship between predicted values ground truth values exists, it is not universal. It is correct only during a certain period of time. The model fails to produce accurate prediction results in most cases. In conclusion, the spectrometer data with a linear regression model cannot be a practical alternative to the white reference calibration method.

Expanding utilization of full 3,840 spectral features provided by the spectrometer data with a linear regression model shows an excellent result. However, there are several issues behind the scene. The first issue is multicollinearity among 3,840 spectrum intensity. Ordinary Least Square regression fits perfectly with the training data set. However, it completely fails to handle the validation data set. The new model using Ridge regression solves multicollinearity issues. The new model creates good intensity predictions from both the prediction data set and validation data set. Nevertheless, some prediction values are counter-intuitive. Each spectrometer value represents spectrum intensity. So, the higher raw spectrometer data relates to the higher light intensity. The Ridge regression model occasionally predicts lower light intensity while the raw spectrometer data is comparatively high. This phenomenon is a result of the overfitting issue. Overfitting model is unlikely able to corp with unseen data. Consequently, the regression model could not be practical as it does not predict light intensity reasonably.

A Convolutional Neural Network (CNN) is built to analyze sky images. Overall, the CNN prediction accuracy is excellent. Most of the predictions locate near the correct line.

From 921 samples, seven outliers, or 0.8 percent of all samples, have more than 10 percent error. One outlier is from December 10^{th}. The rest outliers are from November 18^{th}. Outliers from the second group occur continuously due to lacking sky information. Since sampling time was in late November, the Sun's angle is already down at 3:30 p.m. Moreover, the sky was clear. As a result, the sky images from that period are the plain sky. A separated CNN is trained using segmented sky images which non-sky part is removed. An outcome from the new model is similar to the previous model. Both models have an outlier on December 10^{th} at the same points of time. Moreover, both share a group of outliers on November 18^{th}. Other than these two situations, the CNN model performs well.

By nature of the deep learning model, the more training data, the better is the model. The only issue found is when there is no object in the sky: neither the Sun nor clouds. However, during the growing season, the Sun will remain in the sky longer than the sampling time for this study. Therefore the possibility of inaccurate predictions is low. The wide-angle camera with a deep learning model exhibits a good result in this study and also potentially be better with more training data. It is a suitable substituting sunlight calibration method to the white reference board.

4.2 Future Works

All data in this study were taken from last November to early December. These months are generally outside of production season. The Sun's position varies depending on months. Therefore, the model must be trained by the data inside the production period. The current data set is from 5 sampling data from 10 am. to 4 pm. The data sampling must be expanded in both times, from morning to afternoon, and more number of days. A validation data set is currently randomly selected from the same population pool to the training data set. The model does not learn from the validation set directly. Therefore, the validation set can be used to evaluate the model performance. However, a testing data set, an independent data set separated from the training data set and validation data set, is a more appropriate option to evaluate the actual model performance.

REFERENCES

[1] UN, *World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100*, 2017. [Online]. Available: https://www.un.org/development/desa/en/news/population/world-population-prospects-2017.html.

[2] H. Ritchie and M. Roser, "Land use," *Our World in Data*, 2013, https://ourworldindata.org/land-use.

[3] FAO and UNEP, *The State of the World's Forests 2020*. FAO and UNEP, 2020, ISBN: 978-92-5-132419-6. DOI: 10.4060/ca8642en.

[4] F. K. Van Evert, D. Gaitán-Cremaschi, S. Fountas, and C. Kempenaar, "Can precision agriculture increase the profitability and sustainability of the production of potatoes and olives?" *Sustainability*, vol. 9, no. 10, 2017, ISSN: 2071-1050. DOI: 10.3390/su9101863. [Online]. Available: https://www.mdpi.com/2071-1050/9/10/1863.

[5] G. Gyarmati and T. Mizik, "The present and future of the precision agriculture," in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, 2020, pp. 593–596. DOI: 10.1109/SoSE50414.2020.9130481.

[6] R. M. Math and N. V. Dharwadkar, "An intelligent irrigation scheduling and monitoring system for precision agriculture application," *International Journal of Agricultural and Environmental Information Systems (IJAEIS)*, vol. 11, no. 4, pp. 1–24, 2020, ISSN: 1947-3192. DOI: 10.4018/IJAEIS.2020100101. [Online]. Available: https://www.igi-global.com/article/an-intelligent-irrigation-scheduling-and-monitoring-system-for-precision-agriculture-application/262595.

[7] A. Chlingaryan, S. Sukkarieh, and B. Whelan, "Machine learning approaches for crop yield prediction and nitrogen status estimation in precision agriculture: A review," *Computers and Electronics in Agriculture*, vol. 151, pp. 61–69, 2018, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2018.05.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917314710.

[8] G. Ezenne, L. Jupp, S. Mantel, and J. Tanner, "Current and potential capabilities of uas for crop water productivity in precision agriculture," *Agricultural Water Management*, vol. 218, pp. 158–164, 2019, ISSN: 0378-3774. DOI: https://doi.org/10.1016/j.agwat.2019.03.034. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378377418318298.

[9] A. M. de Oca, L. Arreola, A. Flores, J. Sanchez, and G. Flores, "Low-cost multispectral imaging system for crop monitoring," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 443–451. DOI: 10.1109/ICUAS.2018.8453426.

[10] LoVerdeIndustrial, *What factors may help increase the land value per square meter?* Jun. 24, 2019. [Online]. Available: http://verdeindustrial.com/2019/06/10/what-factors-may-help-increase-the-land-value-per-square-meter.

[11] L. Zhang, L. Wang, J. Wang, Z. Song, T. U. Rehman, T. Bureetes, D. Ma, Z. Chen, S. Neeno, and J. Jin, "Leaf scanner: A portable and low-cost multispectral corn leaf scanning device for precise phenotyping," *Computers and Electronics in Agriculture*, vol. 167, p. 105 069, 2019, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2019.105069. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919306477.

[12] Y. R. Zhao, K. Q. Yu, and Y. He, "Hyperspectral imaging coupled with random frog and calibration models for assessment of total soluble solids in mulberries"," *Journal of Analytical Methods in Chemistry*, vol. 2015, Sep. 14, 2015. DOI: 10.1155/2015/343782.

[13] B. Zhang, S. Fan, J. Li, W. Huang, C. Zhao, M. Qian, and L. Zheng, "Detection of early rottenness on apples by using hyperspectral imaging combined with spectral analysis and image processing," *Food Analytical Methods*, vol. 8, no. 8, pp. 2075–2086, Sep. 1, 2015, ISSN: 1936-976X. DOI: 10.1007/s12161-015-0097-7. [Online]. Available: https://doi.org/10.1007/s12161-015-0097-7.

[14] X. Li, R. Li, M. Wang, Y. Liu, B. Zhang, and J. Zhou, "Hyperspectral imaging and their applications in the nondestructive quality assessment of fruits and vegetables," *Hyperspectral Imaging In Agriculture, Food And Environment*, vol. 3, Dec. 20, 2017. DOI: 10.5772/intechopen.72250.

[15] L. Zhang, J. Jin, L. Wang, P. Huang, and D. Ma, "A 3d white referencing method for soybean leaves based on fusion of hyperspectral images and 3d point clouds," *Precision Agriculture*, vol. 21, no. 6, pp. 1173–1186, Dec. 1, 2020, ISSN: 1573-1618. DOI: 10.1007/s11119-020-09713-7. [Online]. Available: https://doi.org/10.1007/s11119-020-09713-7.

[16] T. Wu, L. Zhang, and C. Huang, "An analysis of shadow effects on spectral vegetation indices using a ground-based imaging spectrometer," in *2015 7th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2015, pp. 1–4. DOI: 10.1109/WHISPERS.2015.8075503.

[17] A. Burkart, S. Cogliati, A. Schickling, and U. Rascher, "A novel uav-based ultra-light weight spectrometer for field spectroscopy," *IEEE Sensors Journal*, vol. 14, no. 1, pp. 62–67, 2014. DOI: 10.1109/JSEN.2013.2279720.

[18] K. D. Singh and C. Nansen, "Advanced calibration to improve robustness of drone-acquired hyperspectral remote sensing data," in *2017 6th International Conference on Agro-Geoinformatics*, 2017, pp. 1–6. DOI: 10.1109/Agro-Geoinformatics.2017.8047061.

[19] G. Bai, Y. Ge, D. Scoby, B. Leavitt, V. Stoerger, N. Kirchgessner, S. Irmak, G. Graef, J. Schnable, and T. Awada, "NU-spidercam: A large-scale, cable-driven, integrated sensing and robotic system for advanced phenotyping, remote sensing, and agronomic research," *Computers and Electronics in Agriculture*, vol. 160, May 1, 2019, ISSN: 0168-1699. DOI: 10.1016/j.compag.2019.03.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169918314170.

[20] A. Johnson and B. Brittion, *Nikon d5300 review*, Feb. 12, 2014. [Online]. Available: https://www.dpreview.com/reviews/nikon-d5300.

[21] *Flame vis-nir spectrometers*. [Online]. Available: https://www.oceaninsight.com/products/spectrometers/general-purpose-spectrometer/flame-series/flame-vis-nir/.

[22] J. Easterbrook, *Gphoto2*, version 2.2.3, Oct. 12, 2020. [Online]. Available: https://github.com/gphoto/gphoto2.

[23] A. Poehlmann, *Seabreeze*, version 1.3.0, Aug. 23, 2020. [Online]. Available: https://github.com/ap--/python-seabreeze.

[24] *Pyspin*, version 1.1.1, Apr. 15, 2017. [Online]. Available: https://github.com/lord63/py-spin.

[25] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. DOI: 10.1109/TSMC.1979.4310076.

[26] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[27] A. Paszke, S. Gross, S. Chintala, and G. Chanan, *Pytorch*, version 1.8.0, Mar. 4, 2021. [Online]. Available: https://github.com/pytorch/pytorch.

A. DATA CAPTURE SOURCE CODE

```
import PySpin
import datetime
import gphoto2 as gp
import seabreeze
seabreeze.use('pyseabreeze')
import seabreeze.spectrometers as sb
import schedule
import time
import pandas as pd

class SystemParams:

    FILEPATH = './data/'

    def __init__(self):
        '''
            Initial all sensors.
        '''
        self.interval = 3 # seconds
        self.target_capture = 5
        self.integration_time = 15 # microseconds
        self.num_capture = 0
        self.cam_sys = PySpin.System.GetInstance()
        self.cam_list = self.cam_sys.GetCameras()
        self.num_cameras = self.cam_list.GetSize()
        for i, self.cam in enumerate(self.cam_list):
            self.init_cam()
```

```
        print(f'{datetime.datetime.now().isoformat()}' \
              f': Finish fisheye initiation')
        self.init_spectrometer()
        print(f'{datetime.datetime.now().isoformat()}' \
              f': Finish spectrometer initiation')
        self.init_dslr()
        print(f'{datetime.datetime.now().isoformat()}' \
              f': Finish DSLR initiation')
        print()

    def fisheye_capture(self, timestamp):
        '''
            Capture sky image. The name of captured
            is designated with timestamp.
        '''
        try:
            try:
                image_result = self.cam.GetNextImage(1000)

                width = image_result.GetWidth()
                height = image_result.GetHeight()
                image_converted = image_result.Convert(
                    PySpin.PixelFormat_RGB8,
                    PySpin.HQ_LINEAR)

                filename = f'{self.FILEPATH}' \
                           f'fisheye/Fisheye-{timestamp}.jpg'
                image_converted.Save(filename)
```

```
                image_result.Release()

            except PySpin.SpinnakerException as ex:
                print('Error: %s' % ex)



        except PySpin.SpinnakerException as ex:
            print('Error: %s' % ex)

        self.num_capture = self.num_capture + 1
        print(f'{datetime.datetime.now().isoformat()}' \
              f': Finish fisheye capturing')

    def init_cam(self):
        # Inital sky camera
        self.cam.Init()
        self.nodemap_tldevice = self.cam.GetTLDeviceNodeMap()
        self.nodemap = self.cam.GetNodeMap()
        self.cam.AcquisitionMode.SetValue(
            PySpin.AcquisitionMode_Continuous)
        self.cam.BeginAcquisition()

    def deinit_cam(self):
        # Clear sky camera at the end of the process
        self.cam.EndAcquisition()
        self.cam.DeInit()
        del self.cam
        self.cam_list.Clear()
```

```
        self.cam_sys.ReleaseInstance()

    def init_spectrometer(self):
        # Inital spectrometer
        self.specs = sb.list_devices()
        self.spec = sb.Spectrometer(self.specs[0])
        self.spec.integration_time_micros(self.integration_time*1000)
        self.wavelengths = self.spec.wavelengths()

    def deinit_spectrometers(self):
        # Clear spectrometer before end the process
        sb.Spectrometer.close(self.spec)

    def get_intensity(self, timestamp):
        # Get data from spectrometer. Name the data with timestamp
        df = pd.DataFrame(columns=self.wavelengths)
        data = self.spec.intensities()
        adding_row = pd.Series(data, index=self.wavelengths)
        df = df.append(adding_row, ignore_index=True)
        filename = f'{self.FILEPATH}spectrometer/spec-{timestamp}.csv'
        df.to_csv(filename, sep=',', index=False)
        print(f'{datetime.datetime.now().isoformat()} ' \
              f': Finish spectrometer capturing')

    def init_dslr(self):
        # Initial dslr camera
        self.camera = gp.Camera()
        self.camera.init()
```

```
    def deinit_dslr(self):
        # Clear dslr camera
        self.camera.exit()

    def capture_dslr(self, timestamp):
        '''
            Trigger the dslr shutter.
            A image is transfer to raspberry pi SD card.
        '''
        filename = f'{SystemParams.FILEPATH}dslr/DSLR-{timestamp}.jpg'
        print(f'{datetime.datetime.now().isoformat()}: DSLR captured')
        file_path = self.camera.capture(gp.GP_CAPTURE_IMAGE)
        camera_file = self.camera.file_get(
            file_path.folder, file_path.name, gp.GP_FILE_TYPE_NORMAL)
        camera_file.save(filename)
        print(f'{datetime.datetime.now().isoformat()}' \
              f': Finish DSLR capturing')

def job(system):
    # Define task for scheduler
    print(f'{datetime.datetime.now().isoformat()}' \
          f': Start capturing number {system.num_capture+1}')
    timestamp = datetime.datetime.now().replace(microsecond=0).isoformat()
    system.fisheye_capture(timestamp)
    system.get_intensity(timestamp)
    system.capture_dslr(timestamp)
    print(f'{datetime.datetime.now().isoformat()}' \
          f': Finish capturing number {system.num_capture}\n')
```

```
if __name__ == '__main__':
    print(f'{datetime.datetime.now().isoformat()}: Start the program')
    system = SystemParams()
    schedule.every(system.interval).seconds.do(job, system)
    while system.num_capture < system.target_capture:
        schedule.run_pending()
        time.sleep(1)

    system.deinit_cam()
    system.deinit_spectrometers()
    system.deinit_dslr()
    print(f'{datetime.datetime.now().isoformat()}: Complete process')
```

B. GROUND TRUTH EXTRACTION SOURCE CODE

```
import numpy as np
import pandas as pd
import cv2 as cv
import matplotlib.pyplot as plt
from pathlib import Path
import re
from datetime import datetime
from tqdm import tqdm

class Image():
    '''
        Class for storing images.
    '''

    def __init__(self, path):
        self.path = path

    def load(self):
        filename = f'{self.path.parent}/{self.path.name}'
        self.image = cv.imread(filename)
        self.image_gray = cv.cvtColor(self.image, cv.COLOR_BGR2GRAY)

    def show(self):
        plt.imshow(cv.cvtColor(self.image, cv.COLOR_BGR2RGB))

    def clear(self):
        self.image = None
```

```
        self.image_gray = None


class ImageDSLR(Image):
    '''
        This class specifically designed to handle ground-truth images.
    '''
    def __init__(self, path):
        self.path = path
        self.image = None
        self.mask = None
        self.intensities = None



    def create_mask(self):
        if self.mask is not None:
            return self.mask

        color_diff = np.diff(self.image, axis=2)
        color_diff = np.sum(color_diff, axis=2).astype(np.uint8)
        _, mask = cv.threshold(color_diff, 0, color_diff.max(),
                               cv.THRESH_BINARY+cv.THRESH_OTSU)
        self.mask = (mask / (2**8-1)).astype(np.uint8)

        return self.mask

    def random_line(self, min, max, size):
        x1 = np.random.randint(min, max)
        x2 = x1 + size
        while x2 > max:
```

```
        x1 = np.random.randint(min, max)
        x2 = x1 + size

    return x1, x2

def random_box(self, h_min, h_max, w_min, w_max, size):
    h1, h2 = self.random_line(h_min, h_max, size)
    w1, w2 = self.random_line(w_min, w_max, size)

    return h1, h2, w1, w2

def sample_white_box(self, size):
    if self.mask is None:
        self.create_mask()

    h, w, _ = self.image.shape
    h1, h2, w1, w2, = self.random_box(0, h, 0, w, size)
    sample_mask = self.mask[h1:h2, w1:w2]
    while sample_mask.sum() < size**2:
        h1, h2, w1, w2, = self.random_box(0, h, 0, w, size)
        sample_mask = self.mask[h1:h2, w1:w2]
    box = self.image[h1:h2, w1:w2, :]
    intensity = np.mean(box)
    bgr = np.mean(np.mean(box, axis=0), axis=0)

    return intensity, bgr

def find_intensity(self, size, number, re_cal=False):
    if self.intensities is not None and not re_cal:
```

```
            return self.intensities

        if self.image is None:
            self.load()

        intensities = []
        bgr_array = []
        for i in range(number):
            intensity, bgr = self.sample_white_box(size)
            intensities.append(intensity)
            bgr_array.append(bgr)

        self.intensities = np.array(intensities)
        self.bgr_array = np.array(bgr_array)
        self.clear()

        return self.intensities, self.bgr_array

    def save_mark(self, path):
        filename = str(path)
        cv.imwrite(filename, self.mask*255)


# Indicate folders that files are kept in
root = Path('Z:') / 'jin' / 'temp' / 'Tam' / 'data'
path = root / 'dslr'
download = Path('W:') / 'My Documents'/ 'Downloads'

columns = ['timestamp', 'mean', 'var', 'filename',]
```

```
dslr_imgs = []
for x in tqdm(path.iterdir()):
  dslr_imgs.append(ImageDSLR(x))

for img in tqdm(dslr_imgs):
  name = img.path.name
  timestamp_text = re.findall(r'DSLR-(.*)(\..*)', name)[0][0]
  timestamp = datetime.strptime(timestamp_text, '%Y-%m-%dT%H_%M_%S')
  row = df.loc[df['timestamp'] == timestamp]
  r, _ = row.shape
  if r == 0:
    intensities, _ = img.find_intensity(100, 500)
    img.save_mark(download / 'sample' / f'{timestamp_text}.png')
    df = df.append({
      'filename': name,
      'timestamp': timestamp,
      'mean': intensities.mean(),
      'var': intensities.var()
    }, ignore_index=True)
```

C. SKY IMAGE SEGMENTATION SOURCE CODE

```
import numpy as np
import pandas as pd
import cv2 as cv
import matplotlib.pyplot as plt
from pathlib import Path
from tqdm import tqdm
import PIL
from datetime import datetime, date

class Image():
    '''
        Class for storing images.
    '''

    def __init__(self, path):
        self.path = path

    def load(self):
        filename = f'{self.path.parent}/{self.path.name}'
        self.image = cv.imread(filename)
        self.image_gray = cv.cvtColor(self.image, cv.COLOR_BGR2GRAY)

    def show(self):
        plt.imshow(cv.cvtColor(self.image, cv.COLOR_BGR2RGB))

    def clear(self):
        self.image = None
```

```
        self.image_gray = None


root = Path('Z:') / 'jin' / 'temp' / 'Tam'
path = root / 'data'
fisheye_path = path / 'fisheye'
download = Path('W:') / 'My Documents'/ 'Downloads'

filename = path / 'intensites.csv'
intensity = pd.read_csv(filename)
intensity['timestamp'] =  pd.to_datetime(intensity['timestamp'],
                                         format='%Y-%m-%dT%H_%M_%S')

intensity['ratio'] = intensity['mean'] / intensity['var']
filtered_df = intensity.loc[intensity.ratio > 10]


images = filtered_df.loc[filtered_df.timestamp.dt.date ==
                         date(year=2020, month=11, day=16)]

for _, row in tqdm(images.iterrows()):
    timestamp = row.timestamp
    image_name =  fisheye_path /
                  f'Fisheye-{datetime.strftime(
                                timestamp, "%Y-%m-%dT%H_%M_%S")}.jpg'
    img = Image(image_name)
    img.load()

    h_start = 100
```

```
w_start = 50
canvas = np.zeros((800, 900, 3), dtype=np.uint8)
canvas[h_start:h_start+608, w_start:w_start+808, :] = img.image

mask = np.zeros((800, 900, 3), dtype=np.uint8)
center_coordinates = (420, 400)
radius = 380
color = (255, 255, 255)
thickness = -1
mask = cv.circle(mask, center_coordinates, radius, color, thickness)

masked_canvas = cv.bitwise_and(canvas, mask)
width = 50
hieght = 104
final_img = masked_canvas[hieght:hieght+600, width:width+750, :]

filename = download / 'cleaned_skyimage'/ image_name.name
cv.imwrite(str(filename), final_img);

img.clear()
```

D. REGRESSION ANALYSIS SOURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
import re
from datetime import datetime
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from tqdm import tqdm
import random

root = Path('Z:') / 'jin' / 'temp' / 'Tam'
path = root / 'data'

a = val_df.loc[random.randint(0, val_df.count()[0]-1)]



filename = path / 'spec.csv'
spec = pd.read_csv(filename)
spec['timestamp'] =  pd.to_datetime(spec['timestamp'],
                                    format='%Y-%m-%dT%H_%M_%S')

wavelength = spec.columns.to_numpy()[:-1].astype(np.float)

filename = path / 'intensites.csv'
```

```
intensity = pd.read_csv(filename)
intensity['timestamp'] =  pd.to_datetime(intensity['timestamp'],
                                         format='%Y-%m-%dT%H_%M_%S')

filename = path / 'low_train.csv'
train_df = pd.read_csv(filename)
train_df['timestamp'] =  pd.to_datetime(train_df['timestamp'],
                                        format='%Y-%m-%dT%H_%M_%S')



train_x = []
train_y = []
train_timestamps = []
for index, row in tqdm(train_df.iterrows()):
    timestamp = row['timestamp']
    row_search = spec.loc[spec['timestamp'] == timestamp]
    r, _ = row_search.shape
    if r == 1:
        train_timestamps.append(timestamp)
        train_y.append(row['mean'])
        train_x.append(row_search.iloc[0].to_numpy()[:-1])

train_y = np.array(train_y, dtype=np.float)
train_x = np.array(train_x, dtype=np.float)
scaler = MinMaxScaler()
train_x = scaler.fit_transform(train_x)

reg = Ridge(alpha=0.1).fit(train_x, train_y)
```

```
fig, ax = plt.subplots(dpi=150)
ax.scatter(wavelength, reg.coef_, s=1)
ax.set_xlabel('Wavelength')
ax.set_ylabel('Coefficient');
filename = download / 'gr-spec-weight-low-ridge.png'
plt.savefig(filename)

x = train_y
y = reg.predict(train_x)
ref_line = np.arange(x.min(), x.max())
fig, ax = plt.subplots(dpi=150)
ax.plot(ref_line, ref_line, c='red')
ax.scatter(x, y, s=1)
fig.suptitle('Training Set')
ax.set_xlabel('Ground Truth')
ax.set_ylabel('Prediction');
print(f'R-squared = {r2_score(x, y):.3f}')
print(f'MSE = {mean_squared_error(x, y):.3f}')
filename = download / 'gr-spec-training-low-ridge.png'
fig.savefig(filename)

filename = path / 'low_val.csv'
val_df = pd.read_csv(filename)
val_df['timestamp'] =  pd.to_datetime(val_df['timestamp'],
                                      format='%Y-%m-%dT%H_%M_%S')



val_x = []
val_y = []
```

```
val_timestamps = []
for index, row in tqdm(val_df.iterrows()):
    timestamp = row['timestamp']
    row_search = spec.loc[spec['timestamp'] == timestamp]
    r, _ = row_search.shape
    if r == 1:
        val_timestamps.append(timestamp)
        val_y.append(row['mean'])
        val_x.append(row_search.iloc[0].to_numpy()[:-1])

val_y = np.array(val_y, dtype=np.float)
val_x = np.array(val_x, dtype=np.float)
scaler = MinMaxScaler()
val_x = scaler.fit_transform(val_x)

x = val_y
y = reg.predict(val_x)
ref_line = np.arange(x.min(), x.max())
fig, ax = plt.subplots(dpi=150)
ax.plot(ref_line, ref_line, c='red')
ax.scatter(x, y, s=1)
fig.suptitle('Validation Set')
ax.set_xlabel('Ground Truth')
ax.set_ylabel('Prediction');
print(f'R-squared = {r2_score(x, y):.3f}')
print(f'MSE = {mean_squared_error(x, y):.3f}')
filename = download / 'gr-spec-val-low-ridge.png'
fig.savefig(filename)
```

E. DEPP LEARNING SOURCE CODE

```
import torch
from torchvision import transforms, utils
from torch.utils.data import Dataset, DataLoader, random_split
from torch.autograd import Variable
import torch.nn.functional as F
import torch.nn as nn
import numpy as np
import pandas as pd
import cv2 as cv
import matplotlib.pyplot as plt
from pathlib import Path
import re
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from tqdm.notebook import tqdm
import PIL

class Image():
    '''
        Class for storing images.
    '''

    def __init__(self, path):
      self.path = path

    def load(self):
```

```
        filename = f'{self.path.parent}/{self.path.name}'
        self.image = cv.imread(filename)
        self.image_gray = cv.cvtColor(self.image, cv.COLOR_BGR2GRAY)

    def show(self):
        plt.imshow(cv.cvtColor(self.image, cv.COLOR_BGR2RGB))

    def clear(self):
        self.image = None
        self.image_gray = None


if torch.cuda.is_available():
    dev = 'cuda:0'
else:
    dev = 'cpu'

    path = Path('.') / 'drive' / 'My Drive' / 'Data' / 'fisheye'

fisheye_imgs = []
for x in path.iterdir():
    fisheye_imgs.append(Image(x))

filename = f'{path.parent}/intensites.csv'
df = pd.read_csv(filename)
df['timestamp'] =  pd.to_datetime(df['timestamp'],
                                  format='%Y-%m-%dT%H_%M_%S')
```

```
filename = f'{path.parent}/low_train.csv'
train_df = pd.read_csv(filename)
train_df['timestamp'] =  pd.to_datetime(train_df['timestamp'],
                                        format='%Y-%m-%dT%H_%M_%S')

train_features = []
train_targets = []
for image in tqdm(fisheye_imgs):
  name = image.path.name
  timestamp_text = re.findall(r'Fisheye-(.*)(\..*)', name)[0][0]
  timestamp = datetime.strptime(timestamp_text, '%Y-%m-%dT%H_%M_%S')
  row = train_df.loc[train_df['timestamp'] == timestamp]
  r, _ = row.shape
  if r == 1:
    intensity = row['mean'].item()
    variance = row['var'].item()
    # if intensity / variance > 10:
    train_features.append(image.path)
    train_targets.append(intensity)
train_targets = np.array(train_targets, dtype=np.float32)
train_targets = torch.from_numpy(train_targets)

filename = f'{path.parent}/low_val.csv'
val_df = pd.read_csv(filename)
val_df['timestamp'] =  pd.to_datetime(val_df['timestamp'],
                                      format='%Y-%m-%dT%H_%M_%S')


val_features = []
```

```
val_targets = []
for image in tqdm(fisheye_imgs):
  name = image.path.name
  timestamp_text = re.findall(r'Fisheye-(.*)(\..*)', name)[0][0]
  timestamp = datetime.strptime(timestamp_text, '%Y-%m-%dT%H_%M_%S')
  row = val_df.loc[val_df['timestamp'] == timestamp]
  r, _ = row.shape
  if r == 1:
    intensity = row['mean'].item()
    variance = row['var'].item()
    # if intensity / variance > 10:
    val_features.append(image.path)
    val_targets.append(intensity)
val_targets = np.array(val_targets, dtype=np.float32)
val_targets = torch.from_numpy(val_targets)

class SkyImageDataset(Dataset):
  def __init__(self, features, targets, transformations):
    self.features = features
    self.targets = targets
    self.transformations = transformations

  def __len__(self):
    return len(self.targets)

  def __getitem__(self, index):
    intensity = self.targets[index]
    path = self.features[index]
    image = PIL.Image.open(path)
```

```
        image = self.transformations(image)
        return image, intensity

transformations = transforms.Compose([
    transforms.Resize(size=(48, 64)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5],
                         std=[0.5, 0.5, 0.5])
])

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 5)
        self.conv2 = nn.Conv2d(16, 64, 3)
        self.conv3 = nn.Conv2d(64, 256, 3)
        self.fc1 = nn.Linear(256 * 4 * 6, 1024)
        self.fc2 = nn.Linear(1024, 32)
        self.fc3 = nn.Linear(32, 1)
        self.pool = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 256 * 4 * 6)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
```

```
        return x

train_dataset = SkyImageDataset(train_features,
                                train_targets,
                                transformations)
val_dataset = SkyImageDataset(val_features,
                              val_targets,
                              transformations)
train_data_loader = DataLoader(train_dataset, batch_size=10, shuffle=True)
val_data_loader = DataLoader(val_dataset, batch_size=10, shuffle=True)

net = Net()
net.to(dev)
optimizer = torch.optim.Adam(net.parameters(), lr=0.005)
loss_func = torch.nn.MSELoss()
val_losses = []
train_losses = []
epochs = 250
for t in tqdm(range(epochs)):
  acc_loss = []
  for images, intensities in train_data_loader:
    images = images.to(dev)
    intensities = intensities.to(dev)
    prediction = net(images)
    optimizer.zero_grad()
    loss = loss_func(prediction, intensities.float().unsqueeze(1))
    loss.backward()
    optimizer.step()
    acc_loss.append(loss.item())
```

```
    train_losses.append(np.mean(acc_loss))

    acc_loss = []
    for images, intensities in val_data_loader:
      images = images.to(dev)
      prediction = net(images)
      loss_mse = mean_squared_error(intensities.cpu().numpy(),
                                    prediction.cpu().data.numpy())
      acc_loss.append(loss_mse)
    val_losses.append(np.mean(acc_loss))



fig, ax = plt.subplots(dpi=150)
ax.plot(np.log10(train_losses), label='Training')
ax.plot(np.log10(val_losses), label='Validation')
plt.legend(loc='best');
ax.set_xlabel('iteration')
ax.set_ylabel('MSE');
filename = path.parent / 'training_loss_filtered.png'
plt.savefig(filename)

predictions = []
truths = []
fig, ax = plt.subplots(dpi=150)
for images, intensities in tqdm(train_data_loader):
  images = images.to(dev)
  prediction = net(images)
  predictions = predictions + \
                prediction.data.cpu().numpy().ravel().tolist()
```

```
    truths = truths + intensities.cpu().numpy().tolist()

predictions = np.array(predictions)
truths = np.array(truths)
ref_line = np.arange(truths.min(), truths.max())
ax.plot(ref_line, ref_line, 'r--')
ax.scatter(truths, predictions, s=0.5)
fig.suptitle('Training Set')
ax.set_xlabel('Ground Truth')
ax.set_ylabel('Prediction');
print(f'R-squared = {r2_score(predictions, truths):.3f}')
print(f'MSE = {mean_squared_error(predictions, truths):.3f}')
filename = path.parent / 'training_filtered.png'
plt.savefig(filename)

predictions = []
truths = []
fig, ax = plt.subplots(dpi=150)
for images, intensities in tqdm(val_data_loader):
    images = images.to(dev)
    prediction = net(images)
    predictions = predictions + \
                  prediction.cpu().data.numpy().ravel().tolist()
    truths = truths + intensities.cpu().numpy().tolist()

predictions = np.array(predictions)
truths = np.array(truths)
ref_line = np.arange(truths.min(), truths.max())
ax.plot(ref_line, ref_line, 'r--')
```

```
ax.scatter(truths, predictions, s=0.5)
fig.suptitle('Validation Set')
ax.set_xlabel('Ground Truth')
ax.set_ylabel('Prediction');
print(f'R-squared = {r2_score(predictions, truths):.3f}')
print(f'MSE = {mean_squared_error(predictions, truths):.3f}')
filename = path.parent / 'val_filtered.png'
plt.savefig(filename)

filename = f'{path.parent}/model_filtered_new_final.pth'
torch.save(net.state_dict(), filename)
```