Applying the Histogram of Oriented Gradients Algorithm for Detecting Grass Lay Direction

Ben Sunshine

2024-05-10

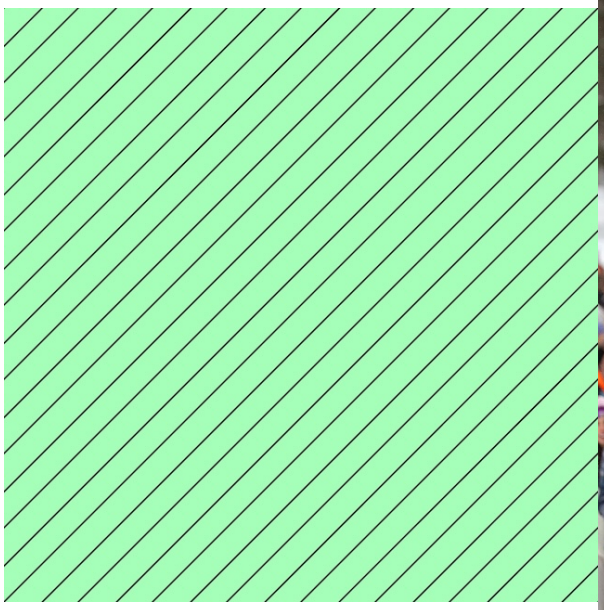
Table of contents

1 Abstract

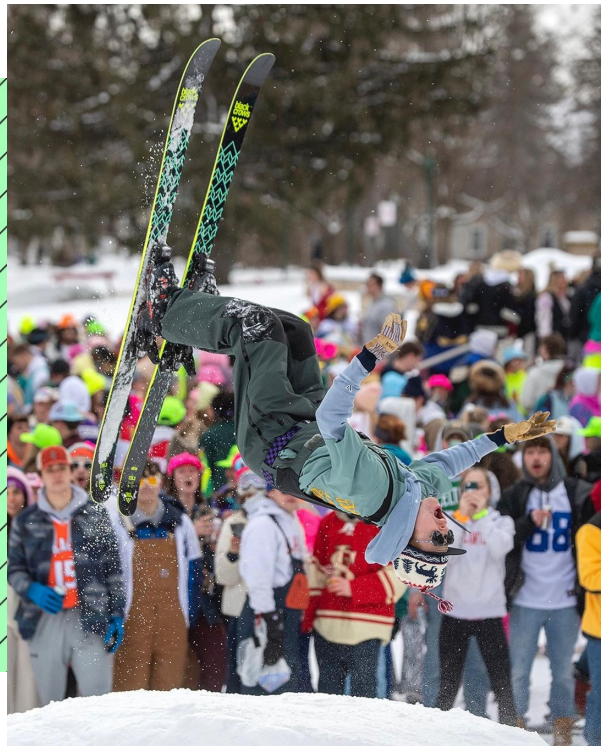
Subsistence-oriented indigenous communities across Alaska rely heavily on Traditional Ecological Knowledge (TEK), a holistic understanding of their environment acquired through generations of observation and cultural transmission. Among the Anishinaabek tradition, sweetgrass symbolizes wisdom and knowledge, passed down from elders to younger generations. Indigenous hunters and gatherers have long observed the alignment of grass and plants after the growing season as indicative of prevailing wind directions. Predominant wind direction serves a crucial role to subsistence practitioners when hunting, fishing, settling, and keeping track of changing weather. Due to the remote and harsh conditions, traditional weather stations are absent to measure shifts in historically predominant wind directions. On islands like St. Lawrence Island in Savoonga, AK, natives have observed a shift from historically predominant northerly wind patterns to southerly and easterly and dominated winds. In a previous study Dr. Jon Rosales (Environmental Studies) and his team collected images of grass lay from St. Lawrence University's Living Laboratory and manually attempted to measure grass lay angles and relate them with wind data. This research project seeks to reinforce Traditional Ecological Knowledge (TEK) with Scientific Ecological Knowledge (SEK) to develop our understanding of Alaskan indigenous wisdom and its relation with modern scientific findings. We investigated the Histogram of Oriented Gradients (HOG) algorithm to automate the measurement of grass lay angles. We applied the algorithm to various images sampled from the internet and the Living Laboratory to test its viability.

2 Data

To evaluate the algorithm's performance, we collected images with varying levels of complexity. Beginning with basic geometric shapes and diagonal lines sourced from Google Images, we established a foundational data set for initial testing. We incorporated aerial cityscapes including San Francisco, Salt Lake City, and Detroit from Mapbox, aiming to assess the algorithm's ability in identifying urban grid structures such as streets and highways. Additionally, we included images of grass sourced from both Google Images and Dr. Jon Rosales (Environmental Studies) at St. Lawrence University's Living Lab. The images sourced from Google Images were primarily The Living Lab images included aerial and close up shots, with special attention given to aerial images which featured a northern indicator line. To ensure consistent orientation, each aerial image was manually rotated to align north facing direction upwards before analysis, with the northern indicator subsequently removed to avoid introducing artificial lines in the image.



(a) Diagnol Lines



(b) Skiing Backflip(St. Lawrence University)

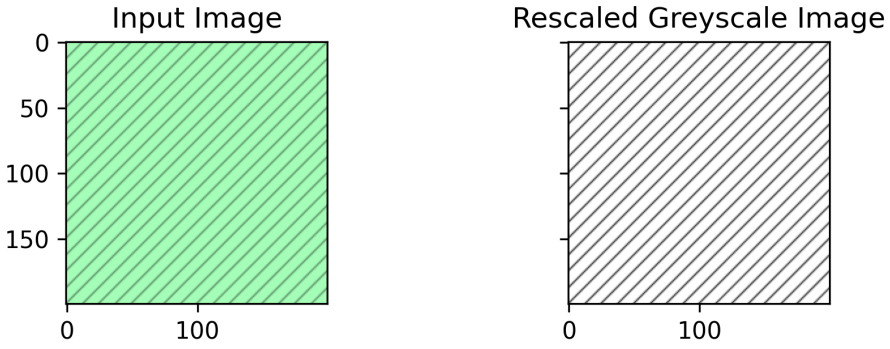
(c) Downtown San Francisco, CA

(d) Aerial Grass Image(St. Lawrence University Living Laboratory)

Figure 2.1: Sample of Images for Evaluation

3 Methods

The HOG algorithm, introduced by Navneet Dalal and Bill Triggs in 2005, is a popular technique for object detection in images. The algorithm can identify gradient magnitudes and angles at each pixel in an image. The preliminary steps involved using the 'skimage' library from Python to preprocess the images of interest. This included loading, resizing, and converting the images to grayscale. Images were rescaled to standardize their resolutions and preserve their aspect ratios to prevent distortion that could affect the accuracy of angle identification. Converting the images to grayscale was necessary because it allowed for focusing on a single channel to represent pixel intensity, rather than three channels (red, green, and blue).



(a) Colored Image with Diagonal Lines

Figure 3.1: Rescaling and Converting Image to Greyscale

The HOG features were then computed for the resized images, which involved calculating the gradient magnitudes and angles at each pixel. The gradient magnitude at each pixel is comprised of the gradients in the 'x' and 'y' directions. The gradient in the x-direction is computed by subtracting the pixel value to the left of pixel of interest is subtracted from the pixel value to its right. Similarly, the gradient in the y-direction is calculated by pixel value below the pixel of interest is subtracted from the pixel value above the pixel of interest.

$G_x = I(r, c+1)I(r, c-1)$

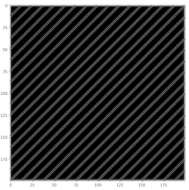
$G_y = I(r+1, c)I(r-1, c)$

Now to calculate the gradient magnitude at the pixel of interest, the Pythagorean Theorem can be utilized where the gradient magnitude is equal to the square root of the x-gradient squared plus the y-gradient squared. The angle at a given pixel can be calculated by taking the inverse tangent of its y-gradient divided by its x-gradient. It is important to note all angles produced by this algorithm are between zero and one hundred eighty degrees. This occurs, because the inverse tangent function used for calculating a given pixel's angle cannot

distinguish between all four quadrants.

$$Magnitude(\mu) = \sqrt{G_x^2 + G_y^2}$$

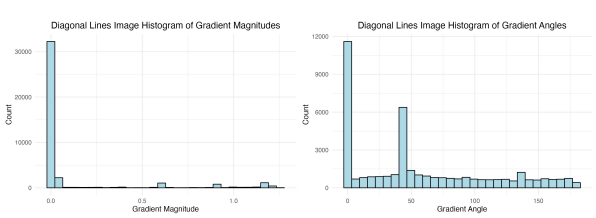
$$Angle(\Theta) = tan^1(\frac{G_y}{G_x})$$



(a) Gradient Magnitudes of Diagonal Lines Image

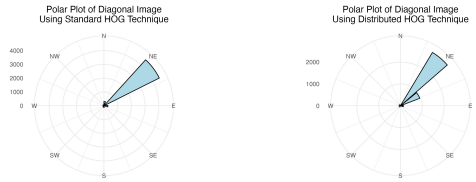
Figure 3.2: Plotting Gradient Magnitudes as Image

Next, histograms are constructed to visualize the distribution of gradient magnitudes and angles. Two different techniques for creating gradient angle histograms were implemented. The first histogram was created by counting the number of angles that fell into their respective bins. The second scheme factors in a pixel's gradient magnitude and its allocation to its bordering bins. Here, the weight assigned to each bin is calculated by the angle's deviation from the center of its central bin. This approach allows for a more representative histogram which splits angles between bins and takes their magnitudes into account. Lastly, these histograms are converted to polar histograms so the primary angles can be visualized and compared to their original images.



(a) Histograms of Gradient Magnitudes

(a) Histograms of Gradient Angles



(a) Polar Plot Using Standard Histogram Technique

(a) Polar Plot Using Distributed Histogram Technique

Figure 3.6: Plotting Histograms of Gradients

Part I

Results

4 Aerial Cityscapes

4.1 Load R Packages

```
library(reticulate)
library(tidyverse)
library(mapsapi)
library(mapboxapi)
library(magick)
Sys.which("python")
```

python
"/Users/bensunshine/.virtualenvs/r-reticulate/bin/python"

4.2 Download Aerial City Images

```
key <- Sys.getenv("mapbox_key")
```

```
map <- static_mapbox(
  access_token = key,
  style_url = "mapbox://styles/mapbox/satellite-v9",
  width = 300,
  height = 200,
  image = T, latitude = 37.792004, longitude = -122.428079, zoom = 12
)

magick::image_write(map, "images/san_francisco_scale_zoom_12.png")
```

```
points_of_interest <- tibble::tibble(
  longitude = c(-112.065945, -111.853948,
                -111.852956, -112.023371),

  latitude = c(40.794275, 40.791516,
               40.502308, 40.502308)
```

```r
  )

prepped_pois <- prep_overlay_markers(
  data = points_of_interest,
  marker_type = "pin-l",
  label = 1:4,
  color = "#fff",
)

map <- static_mapbox(
  access_token = key,
  style_url = "mapbox://styles/mapbox/satellite-v9",
  width = 800,
  height = 1200,
  image = T,
  latitude = 40.7,
  longitude = -111.876183, zoom = 12
)

magick::image_write(map, "images/salt_lake_city_zoom_12.png")
```

```r
map <- static_mapbox(
  access_token = key,
  style_url = "mapbox://styles/mapbox/satellite-v9",
  width = 1200,
  height = 800,
  image = T,
  latitude = 42.336322,
  longitude = -83.048705, zoom = 12
)

magick::image_write(map, "images/detroit_zoom_12.png")
```

4.3 Load Python Libraries

```python
import matplotlib.pyplot as plt
import pandas as pd

# jupyter only inline output command
```

(a) San Francisco Cityscape

(a) Salt Lake City Cityscape

(a) Detroit Cityscape

Figure 4.3: Aerial Cityscape Images

```
#%matplotlib inline

from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
from skimage import data, exposure


import matplotlib.pyplot as plt
from skimage import io
from skimage import color
from skimage.transform import resize
import math
from skimage.feature import hog
import numpy as np
```

4.4 Collect HOG Features for Aerial Cityscapes

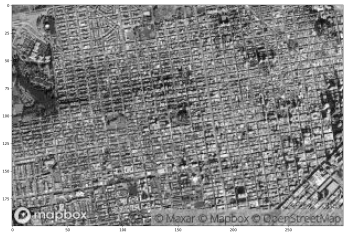
```
# List for storing images
img_list = []

# SF aerial
img_list.append(color.rgb2gray(io.imread("images/san_francisco_scale_zoom_12.png")))

# Salt Lake City Aerial
img_list.append(color.rgb2gray(io.imread("images/salt_lake_city_zoom_12.png")))

# Detroit Aerial
img_list.append(color.rgb2gray(io.imread("images/detroit_zoom_12.png")))


#img = color.rgb2gray(io.imread("images/grass_image2.jpg"))

# img = color.rgb2gray(io.imread("images/b_test_image_copy.jpg"))
#img = color.rgb2gray(io.imread("images/long_grass_sample.jpeg"))
#img = color.rgb2gray(io.imread("images/diagnol_lines.jpg"))

#img = color.rgb2gray(io.imread("images/san_francisco_scale_zoom_12.png"))

#img = color.rgb2gray(io.imread("images/diagnol_lines_flipped.jpg"))
#img = color.rgb2gray(io.imread("images/long_grass_sample_cropped.jpg"))

# aerial rotated image
#img = color.rgb2gray(io.imread("images/living_lab_aerial/aerial_grass_living_lab_rotated.jpg

# zoomed internet photo
#img = color.rgb2gray(io.imread("images/dead_grass_zoom.jpeg"))


# zoomed in 11
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_11.jpg"))

# zoomed in 12
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_12.jpg"))

# zoomed in 16
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_16_side.jpg"))
```

```
# real one
#img = color.rgb2gray(io.imread("images/living_labs_real_grass_image.jpg"))

# List to store magnitudes for each image
mag_list = []

# List to store angles for each image
theta_list = []


for x in range(len(img_list)):
    # Get image of interest
    img = img_list[x]

    rescaled_file_path = f"images/plots/aerial_cities/{x}.jpg"

    # Determine aspect Ratio
    aspect_ratio = img.shape[0] / img.shape[1]
    print("Aspect Ratio:", aspect_ratio)

    # Hard-Code height to 200 pixels
    height = 200

    # Calculate witdth to maintain same aspect ratio
    width = int(height / aspect_ratio)
    print("Resized Width:", width)

    # Resize the image
    resized_img = resize(img, (height, width))

    # Replace the original image with the resized image
    img_list[x] = resized_img

    # if (x == 1):
    #   plot_width = 8
    #   plot_height = 15
    # else:
    #   plot_width = 15
    #   plot_height = 9
    #
    # plt.figure(figsize=(plot_width, plot_height))
```

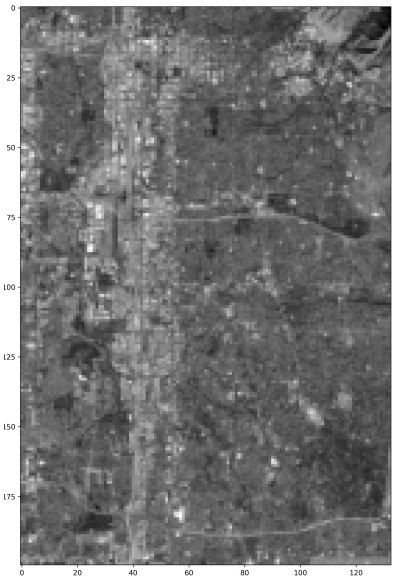
```
# plt.imshow(resized_img, cmap="gray")
# plt.axis("on")
# plt.tight_layout()
# plt.savefig(rescaled_file_path, dpi=300)
# plt.show()


# list for storing all magnitudes for image[x]
mag = []

# list for storing all angles for image[x]
theta = []

for i in range(height):
    magnitudeArray = []
    angleArray = []

    for j in range(width):
        if j - 1 < 0 or j + 1 >= width:
            if j - 1 < 0:
                Gx = resized_img[i][j + 1] - 0
            elif j + 1 >= width:
                Gx = 0 - resized_img[i][j - 1]
        else:
            Gx = resized_img[i][j + 1] - resized_img[i][j - 1]

        if i - 1 < 0 or i + 1 >= height:
            if i - 1 < 0:
                Gy = 0 - resized_img[i + 1][j]
            elif i + 1 >= height:
                Gy = resized_img[i - 1][j] - 0
        else:
            Gy = resized_img[i + 1][j] - resized_img[i - 1][j]

        magnitude = math.sqrt(pow(Gx, 2) + pow(Gy, 2))
        magnitudeArray.append(round(magnitude, 9))

        if Gx == 0:
            angle = math.degrees(0.0)
        else:
            angle = math.degrees(math.atan(Gy / Gx))
            if angle < 0:
```

```
                    angle += 180

            angleArray.append(round(angle, 9))

        mag.append(magnitudeArray)
        theta.append(angleArray)

    # add list of magnitudes to list[x]
    mag_list.append(mag)

    # add list of angles to angle list[x]
    theta_list.append(theta)
```

```
Aspect Ratio: 0.6666666666666666
Resized Width: 300
Aspect Ratio: 1.5
Resized Width: 133
Aspect Ratio: 0.6666666666666666
Resized Width: 300
```



(a) San Francisco, CA



(a) Salt Lake City, UT

(a) Detroit, MI

Figure 4.6: Aerial Cityscape Images Rescaled and Converted to Greyscale

4.5 Build Data Frames for Each Aerial Cityscape

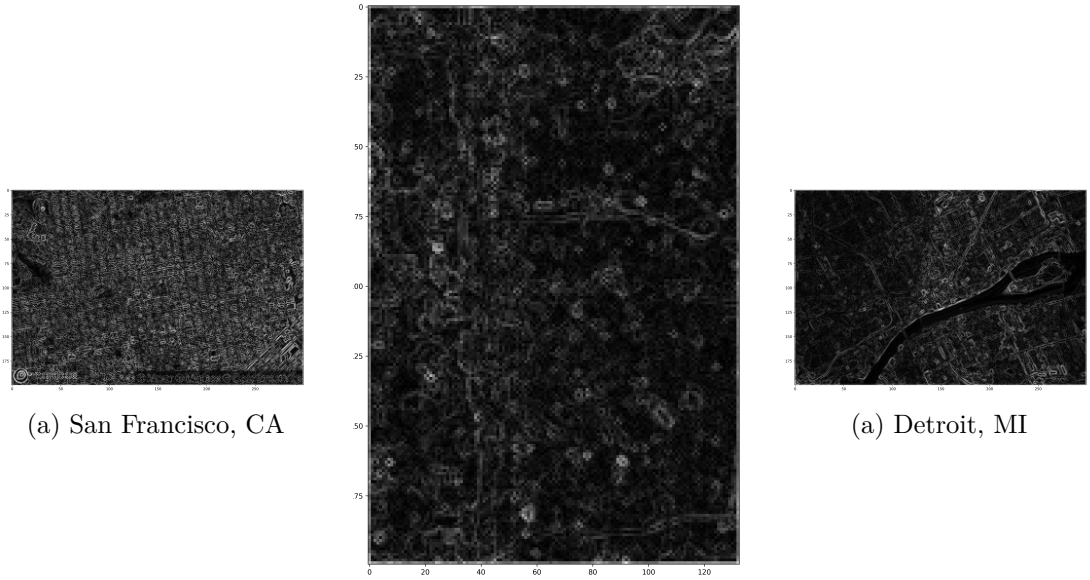
```
mag_sf = np.array(mag_list[0])
theta_sf = np.array(theta_list[0])


mag_salt_lake = np.array(mag_list[1])
theta_salt_lake = np.array(theta_list[1])


mag_detroit = np.array(mag_list[2])
theta_detroit = np.array(theta_list[2])
```

4.6 Plot Magnitudes as Image for each Aerial Cityscape

```
# plt.figure(figsize=(15, 8))
# #plt.title('San Francisco, CA Gradient Magnitudes')
# plt.imshow(mag_list[0], cmap="gray")
# plt.axis("on")
# #plt.show()
# plt.tight_layout()
# plt.savefig("images/plots/aerial_cities/sf_mag.png", dpi=300)
```

```
# plt.figure(figsize=(8, 15))
# #plt.title('Salt Lake City, UT Gradient Magnitudes')
# plt.imshow(mag_list[1], cmap="gray")
# plt.axis("on")
# #plt.show()
# plt.tight_layout()
# plt.savefig("images/plots/aerial_cities/salt_lake_mag.png", dpi=300)
```

```
# plt.figure(figsize=(15, 8))
# #plt.title('Detroit, MI Gradient Magnitudes')
# plt.imshow(mag_list[2], cmap="gray")
# plt.axis("on")
# #plt.show()
# plt.tight_layout()
# plt.savefig("images/plots/aerial_cities/detroit_mag.png", dpi=300)
```



(a) San Francisco, CA

(a) Salt Lake City, UT

(a) Detroit, MI

Figure 4.9: Aerial Cityscape Magnitudes

4.7 Create Data Frame for Each Image

```
# San Francisco DF
sf_hog_df <- data.frame(mag = as.vector(py$mag_sf),
                        theta = as.vector((py$theta_sf))) %>%
  mutate(radian = theta*(pi/180))
```

```
# Salt Lake City DF
salt_lake_hog_df <- data.frame(mag = as.vector(py$mag_salt_lake),
                        theta = as.vector((py$theta_salt_lake))) %>%
  mutate(radian = theta*(pi/180))
```

```
# Detroit DF
detroit_hog_df <- data.frame(mag = as.vector(py$mag_detroit),
                        theta = as.vector((py$theta_detroit))) %>%
  mutate(radian = theta*(pi/180))
```

```
# List of all Data frames
standard_df_list = list(sf_hog_df,
```

```
                          salt_lake_hog_df,
                          detroit_hog_df)
```

4.8 Create Histograms of Gradient Magnitudes and Angles for Aerial Cityscapes

```
sf_histogram_mag_plot <-
  ggplot(standard_df_list[[1]],
         aes(x = mag)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Magnitude",
       y = "Count",
       title = "San Francisco Cityscape Image Histogram of Gradient Magnitudes"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#sf_histogram_mag_plot

sf_mag_filter <- 0.4

ggsave("images/plots/aerial_cities/sf_histogram_mag_plot.jpg", sf_histogram_mag_plot, width =
```

```
sf_histogram_theta_plot <-
  ggplot(standard_df_list[[1]],
         aes(x = theta)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Angle",
       y = "Count",
       title = "San Francisco Cityscape Image Histogram of Gradient Angles"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#sf_histogram_theta_plot

ggsave("images/plots/aerial_cities/sf_histogram_theta_plot.jpg", sf_histogram_theta_plot, wi
```

```r
salt_lake_histogram_mag_plot <-
  ggplot(standard_df_list[[2]],
         aes(x = mag)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Magnitude",
       y = "Count",
       title = "Salt Lake City Image Histogram of Gradient Magnitudes"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#salt_lake_histogram_mag_plot

salt_lake_mag_filter <- 0.12

ggsave("images/plots/aerial_cities/salt_lake_histogram_mag_plot.jpg", salt_lake_histogram_mag
```

```r
salt_lake_histogram_theta_plot <-
  ggplot(standard_df_list[[2]],
         aes(x = theta)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Angle",
       y = "Count",
       title = "Salt Lake City Image Histogram of Gradient Angles"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#salt_lake_histogram_theta_plot

ggsave("images/plots/aerial_cities/salt_lake_histogram_theta_plot.jpg", salt_lake_histogram_t
```

```r
detroit_histogram_mag_plot <-
  ggplot(standard_df_list[[3]],
         aes(x = mag)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Magnitude",
       y = "Count",
       title = "Detroit Image Histogram of Gradient Magnitudes"
```

```
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#detroit_histogram_mag_plot

detroit_mag_filter <- 0.15

ggsave("images/plots/aerial_cities/detroit_histogram_mag_plot.jpg", detroit_histogram_mag_plo
```

```
detroit_histogram_theta_plot <-
  ggplot(standard_df_list[[3]],
         aes(x = theta)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Angle",
       y = "Count",
       title = "Detroit, MI Image Histogram of Gradient Angles"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#detroit_histogram_theta_plot

ggsave("images/plots/aerial_cities/detroit_histogram_theta_plot.jpg", detroit_histogram_theta
```

4.9 Build New Distributed Histogram Data Frames

```
# function to calculate the contributions to neighboring bins
calculate_bin_contributions <- function(angle, magnitude, num_bins) {
  bin_width <- 180 / num_bins
  contributions <- numeric(num_bins)

  # get the central bin
  central_bin <- floor(angle / bin_width) %% num_bins
  next_bin <- (central_bin + 1) %% num_bins

  # get contributions to neighboring bins
  weight <- (1 - abs((angle %% bin_width) / bin_width)) * magnitude
```

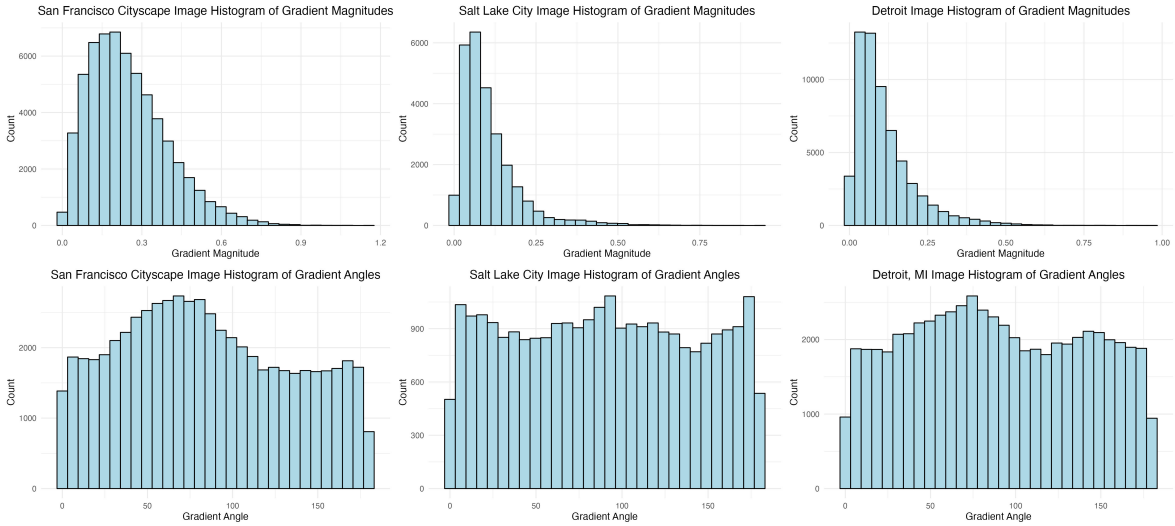


Figure 4.10: Aerial Cityscape Magnitudes and Angles

```
  contributions[central_bin + 1] <- weight
  contributions[next_bin + 1] <- magnitude - weight

  return(list(contributions[1],
         contributions[2],
         contributions[3],
         contributions[4],
         contributions[5],
         contributions[6],
         contributions[7],
         contributions[8],
         contributions[9])
         )
}
```

```
filtered_aerial_standard_df_list <-list(sf_hog_df %>%
                                  filter(mag >= sf_mag_filter),
                                salt_lake_hog_df %>%
                                  filter(mag >= salt_lake_mag_filter),
                                detroit_hog_df %>%
                                  filter(mag >= detroit_mag_filter))
```

```
aerial_contribution_df_list <- list()
```

```r
# Define the number of bins
num_bins <- 9

for (i in 1:length(filtered_aerial_standard_df_list)){

  aerial_contribution_hog_df <-
    filtered_aerial_standard_df_list[[i]] %>%
    rowwise() %>%
    mutate(`0` = calculate_bin_contributions(theta, mag, 9)[[1]],
           `20` = calculate_bin_contributions(theta, mag, 9)[[2]],
           `40` = calculate_bin_contributions(theta, mag, 9)[[3]],
           `60` = calculate_bin_contributions(theta, mag, 9)[[4]],
           `80` = calculate_bin_contributions(theta, mag, 9)[[5]],
           `100` = calculate_bin_contributions(theta, mag, 9)[[6]],
           `120` = calculate_bin_contributions(theta, mag, 9)[[7]],
           `140` = calculate_bin_contributions(theta, mag, 9)[[8]],
           `160` = calculate_bin_contributions(theta, mag, 9)[[9]],
           )

  aerial_split_histo_df <-
    aerial_contribution_hog_df %>%
    pivot_longer(names_to = "bin",
                 values_to = "contribution",
                 cols = 4:ncol(aerial_contribution_hog_df)) %>%
    mutate(bin = as.numeric(bin)) %>%
    group_by(bin) %>%
    summarise(contribution_sum = sum(contribution))


  aerial_contribution_df_list[[i]] <- aerial_split_histo_df

}
```

4.10 Generate Polar Plots for Standard Historgrams

```r
sf_plot <-
  ggplot(filtered_aerial_standard_df_list[[1]],
         aes(x = theta)) +
  geom_histogram(colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
```

```
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of San Francisco, CA Image\nUsing Standard HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/aerial_cities/sf_standard_polar_plot.jpg", sf_plot, width = 6, height =
```

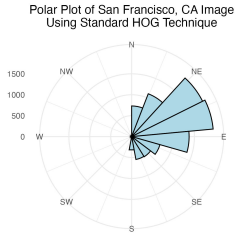
```
salt_lake_plot <-
  ggplot(filtered_aerial_standard_df_list[[2]],
         aes(x = theta)) +
  geom_histogram(colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Salt Lake City, UT Image\nUsing Standard HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/aerial_cities/salt_lake_standard_polar_plot.jpg", salt_lake_plot, width
```

```
detroit_plot <-
  ggplot(filtered_aerial_standard_df_list[[3]],
         aes(x = theta)) +
  geom_histogram(colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Detroit, MI Image\nUsing Standard HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/aerial_cities/detroit_standard_polar_plot.jpg", detroit_plot, width = 6
```

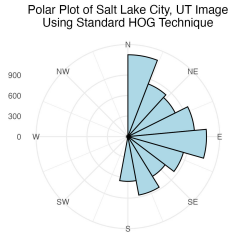
```
all_standard_city_plots <- ggpubr::ggarrange(sf_plot,
                                             salt_lake_plot,
                                             detroit_plot)

ggsave("images/plots/aerial_cities/all_standard_polar_plots.jpg",
       all_standard_city_plots,
       width = 7,
       height = 7)
```

4.11 Generate Polar Plots for Distributed Historgrams

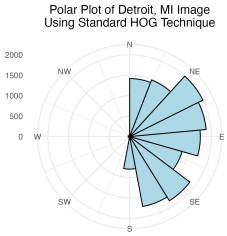
```
sf_split_plot <-
  ggplot(aerial_contribution_df_list[[1]],
         aes(x = bin, y = contribution_sum)) +
  geom_histogram(stat = "identity",
                 colour = "black",
                 fill = "lightblue",
```



(a) San Francisco, CA



(a) Salt Lake City, UT



(a) Detroit, MI

Figure 4.13: Aerial Cityscape Standard Polar Plots

```
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of San Francisco, CA Image\nUsing Distributed HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/aerial_cities/sf_contribution_polar_plot.jpg", sf_split_plot, width = 6
```

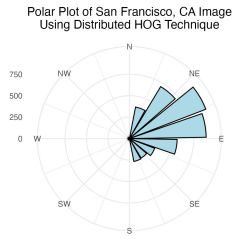
```
salt_lake_split_plot <-
  ggplot(aerial_contribution_df_list[[2]],
         aes(x = bin, y = contribution_sum)) +
  geom_histogram(stat = "identity",
                 colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
```

```
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Salt Lake City, UT Image\nUsing Distributed HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/aerial_cities/salt_lake_contribution_polar_plot.jpg", salt_lake_split_pl
```

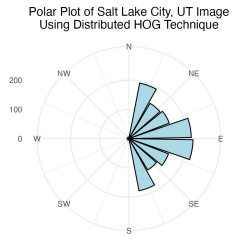
```
detroit_split_plot <-
  ggplot(aerial_contribution_df_list[[3]],
         aes(x = bin, y = contribution_sum)) +
  geom_histogram(stat = "identity",
                 colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Detroit, MI Image\nUsing Distributed HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/aerial_cities/detroit_contribution_polar_plot.jpg", detroit_split_plot,
```

```
all_aerial_contribution_plots <- ggpubr::ggarrange(sf_split_plot,
                                                   salt_lake_split_plot,
                                                   detroit_split_plot)

ggsave("images/plots/aerial_cities/all_aerial_contribution_plots.jpg",
```

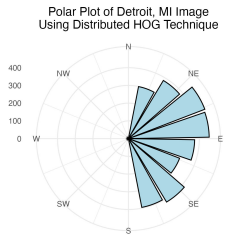
```
    all_aerial_contribution_plots,
    width = 7,
    height = 7)
```



(a) San Francisco, CA



(a) Salt Lake City, UT



(a) Detroit, MI

Figure 4.16: Aerial Cityscape Distributed Method Polar Plots

5 Grass Images

```
library(reticulate)
library(tidyverse)
library(mapsapi)
library(mapboxapi)
library(magick)
Sys.which("python")
```

```
                                                    python
"/Users/bensunshine/.virtualenvs/r-reticulate/bin/python"
```

```
import matplotlib.pyplot as plt
import pandas as pd

# jupyter only inline output command
#%matplotlib inline

from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
from skimage import data, exposure


import matplotlib.pyplot as plt
from skimage import io
from skimage import color
from skimage.transform import resize
import math
from skimage.feature import hog
import numpy as np
```

5.1 Collect HOG Features for Grass Images

```
# List for storing images
img_list = []

# Internet Grass Image
img_list.append(color.rgb2gray(io.imread("images/grass_image2.jpg")))

# Living Labs Rotated Aerial Grass
img_list.append(color.rgb2gray(io.imread("images/living_lab_aerial/aerial_grass_living_lab_r

# Living Labs Grass Close-up
img_list.append(color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_12.jpg")))


#img = color.rgb2gray(io.imread("images/grass_image2.jpg"))

# img = color.rgb2gray(io.imread("images/b_test_image_copy.jpg"))
#img = color.rgb2gray(io.imread("images/long_grass_sample.jpeg"))
#img = color.rgb2gray(io.imread("images/diagnol_lines.jpg"))

#img = color.rgb2gray(io.imread("images/san_francisco_scale_zoom_12.png"))

#img = color.rgb2gray(io.imread("images/diagnol_lines_flipped.jpg"))
#img = color.rgb2gray(io.imread("images/long_grass_sample_cropped.jpg"))

# aerial rotated image
#img = color.rgb2gray(io.imread("images/living_lab_aerial/aerial_grass_living_lab_rotated.jpg

# zoomed internet photo
#img = color.rgb2gray(io.imread("images/dead_grass_zoom.jpeg"))


# zoomed in 11
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_11.jpg"))

# zoomed in 12
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_12.jpg"))

# zoomed in 16
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_16_side.jpg"))
```

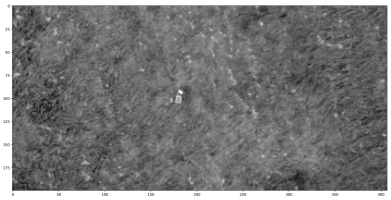
```
# real one
#img = color.rgb2gray(io.imread("images/living_labs_real_grass_image.jpg"))

# List to store magnitudes for each image
mag_list = []

# List to store angles for each image
theta_list = []


for x in range(len(img_list)):
    # Get image of interest
    img = img_list[x]

    rescaled_file_path = f"images/plots/grass/{x}.jpg"

    # Determine aspect Ratio
    aspect_ratio = img.shape[0] / img.shape[1]
    print("Aspect Ratio:", aspect_ratio)

    # Hard-Code height to 200 pixels
    height = 200

    # Calculate witdth to maintain same aspect ratio
    width = int(height / aspect_ratio)
    print("Resized Width:", width)

    # Resize the image
    resized_img = resize(img, (height, width))

    # Replace the original image with the resized image
    img_list[x] = resized_img

    # plt.figure(figsize=(plot_width, plot_height))
    # plt.imshow(resized_img, cmap="gray")
    # plt.axis("on")
    # plt.tight_layout()
    # plt.savefig(rescaled_file_path, dpi=300)
    # plt.show()


    # list for storing all magnitudes for image[x]
```

```
    mag = []

    # list for storing all angles for image[x]
    theta = []

    for i in range(height):
        magnitudeArray = []
        angleArray = []

        for j in range(width):
            if j - 1 < 0 or j + 1 >= width:
                if j - 1 < 0:
                    Gx = resized_img[i][j + 1] - 0
                elif j + 1 >= width:
                    Gx = 0 - resized_img[i][j - 1]
            else:
                Gx = resized_img[i][j + 1] - resized_img[i][j - 1]

            if i - 1 < 0 or i + 1 >= height:
                if i - 1 < 0:
                    Gy = 0 - resized_img[i + 1][j]
                elif i + 1 >= height:
                    Gy = resized_img[i - 1][j] - 0
            else:
                Gy = resized_img[i + 1][j] - resized_img[i - 1][j]

            magnitude = math.sqrt(pow(Gx, 2) + pow(Gy, 2))
            magnitudeArray.append(round(magnitude, 9))

            if Gx == 0:
                angle = math.degrees(0.0)
            else:
                angle = math.degrees(math.atan(Gy / Gx))
                if angle < 0:
                    angle += 180

            angleArray.append(round(angle, 9))

        mag.append(magnitudeArray)
        theta.append(angleArray)

    # add list of magnitudes to list[x]
```

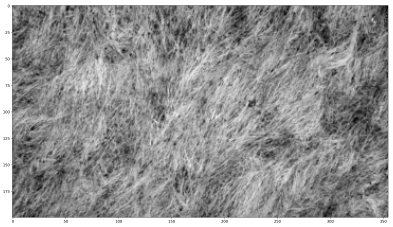
```
    mag_list.append(mag)

    # add list of angles to angle list[x]
    theta_list.append(theta)
```

```
Aspect Ratio: 0.662751677852349
Resized Width: 301
Aspect Ratio: 0.4904214559386973
Resized Width: 407
Aspect Ratio: 0.5625
Resized Width: 355
```

(a) Internet Grass Image



(a) Living Labs Aerial Image



(a) Living Labs Close-Up Image

Figure 5.3: Grass Images Rescaled and Converted to Greyscale

5.2 Build Data Frames for Each Grass Image

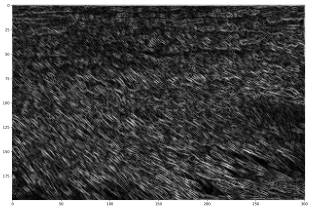
```
mag_internet_grass = np.array(mag_list[0])
theta_internet_grass = np.array(theta_list[0])


mag_aerial_living_lab = np.array(mag_list[1])
theta_aerial_living_lab = np.array(theta_list[1])


mag_close_up_living_lab = np.array(mag_list[2])
theta_close_up_living_lab = np.array(theta_list[2])
```

5.3 Plot Magnitudes as Image for each Grass Images

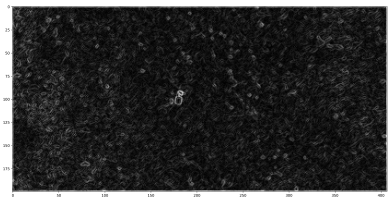
```
# plt.figure(figsize=(15, 8))
# #plt.title('San Francisco, CA Gradient Magnitudes')
# plt.imshow(mag_list[0], cmap="gray")
# plt.axis("on")
# #plt.show()
# plt.tight_layout()
# plt.savefig("images/plots/grass/internet_grass_mag.png", dpi=300)
```

```
# plt.figure(figsize=(15, 8))
# #plt.title('Salt Lake City, UT Gradient Magnitudes')
# plt.imshow(mag_list[1], cmap="gray")
# plt.axis("on")
# #plt.show()
# plt.tight_layout()
# plt.savefig("images/plots/grass/aerial_living_lab_grass_mag.png", dpi=300)
```

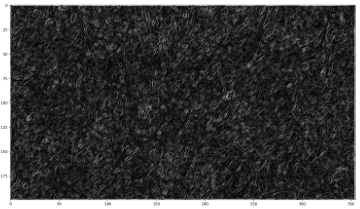
```
# plt.figure(figsize=(15, 8))
# #plt.title('Detroit, MI Gradient Magnitudes')
# plt.imshow(mag_list[2], cmap="gray")
# plt.axis("on")
# #plt.show()
# plt.tight_layout()
# plt.savefig("images/plots/grass/close_up_living_lab_grass_mag.png", dpi=300)
```



(a) Internet Grass



(a) Aerial Living Labs



(a) Close-Up Living Labs

Figure 5.6: Grass Image Magnitudes

5.4 Create Data Frame for Each Image

```
# Diagonal DF
internet_grass_hog_df <- data.frame(mag = as.vector(py$mag_internet_grass),
                                    theta = as.vector((py$theta_internet_grass))) %>%
  mutate(radian = theta*(pi/180))

# San Francisco DF
aerial_living_lab_hog_df <- data.frame(mag = as.vector(py$mag_aerial_living_lab),
                                    theta = as.vector((py$theta_aerial_living_lab))) %>%
  mutate(radian = theta*(pi/180))

# Internet Grass DF
close_up_living_lab_hog_df <- data.frame(mag = as.vector(py$mag_close_up_living_lab),
                                    theta = as.vector((py$theta_close_up_living_lab))) %>%
  mutate(radian = theta*(pi/180))


# List of all Data frames
grass_standard_df_list = list(internet_grass_hog_df,
                        aerial_living_lab_hog_df,
                        close_up_living_lab_hog_df)
```

5.5 Create Histograms of Gradient Magnitudes and Angles for Grass Images

```
internet_grass_histogram_mag_plot <-
  ggplot(grass_standard_df_list[[1]],
         aes(x = mag)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Magnitude",
       y = "Count",
       title = "Internet Grass Image Histogram of Gradient Magnitudes"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#internet_grass_histogram_mag_plot
```

```r
internet_grass_mag_filter <- 0.3

ggsave("images/plots/grass/internet_grass_histogram_mag_plot.jpg", internet_grass_histogram_m



internet_grass_histogram_theta_plot <-
  ggplot(grass_standard_df_list[[1]],
         aes(x = theta)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Angle",
       y = "Count",
       title = "Internet Grass Image Histogram of Gradient Angles"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#internet_grass_histogram_theta_plot

ggsave("images/plots/grass/internet_grass_histogram_theta_plot.jpg", internet_grass_histogra
```

```r
aerial_living_lab_histogram_mag_plot <-
  ggplot(grass_standard_df_list[[2]],
         aes(x = mag)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Magnitude",
       y = "Count",
       title = "Aerial Living Labs Image Histogram of Gradient Magnitudes"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#aerial_living_lab_histogram_mag_plot

aerial_living_lab_mag_filter <- 0.08

ggsave("images/plots/grass/aerial_living_lab_histogram_mag_plot.jpg",
       aerial_living_lab_histogram_mag_plot, width = 6, height = 4, dpi = 300)
```

```
aerial_living_lab_histogram_theta_plot <-
  ggplot(grass_standard_df_list[[2]],
         aes(x = theta)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Angle",
       y = "Count",
       title = "Aerial Living Labs Image Histogram of Gradient Angles"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#aerial_living_lab_histogram_theta_plot

ggsave("images/plots/grass/aerial_living_lab_histogram_theta_plot.jpg",
       aerial_living_lab_histogram_theta_plot, width = 6, height = 4, dpi = 300)
```

```
close_up_living_lab_histogram_mag_plot <-
  ggplot(grass_standard_df_list[[3]],
         aes(x = mag)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Magnitude",
       y = "Count",
       title = "Close-Up Living Labs Histogram of Gradient Magnitudes"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#close_up_living_lab_histogram_mag_plot

close_up_living_lab_mag_filter <- 0.12

ggsave("images/plots/grass/close_up_living_lab_histogram_mag_plot.jpg", close_up_living_lab_h



close_up_living_lab_histogram_theta_plot <-
  ggplot(grass_standard_df_list[[3]],
         aes(x = theta)) +
  geom_histogram(colour = "black", fill = "lightblue") +
```

```
  scale_x_continuous() +
  labs(x = "Gradient Angle",
       y = "Count",
       title = "Close-Up Living Labs Image Histogram of Gradient Angles"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#close_up_living_lab_histogram_theta_plot

ggsave("images/plots/grass/close_up_living_lab_histogram_theta_plot.jpg", close_up_living_lab
```

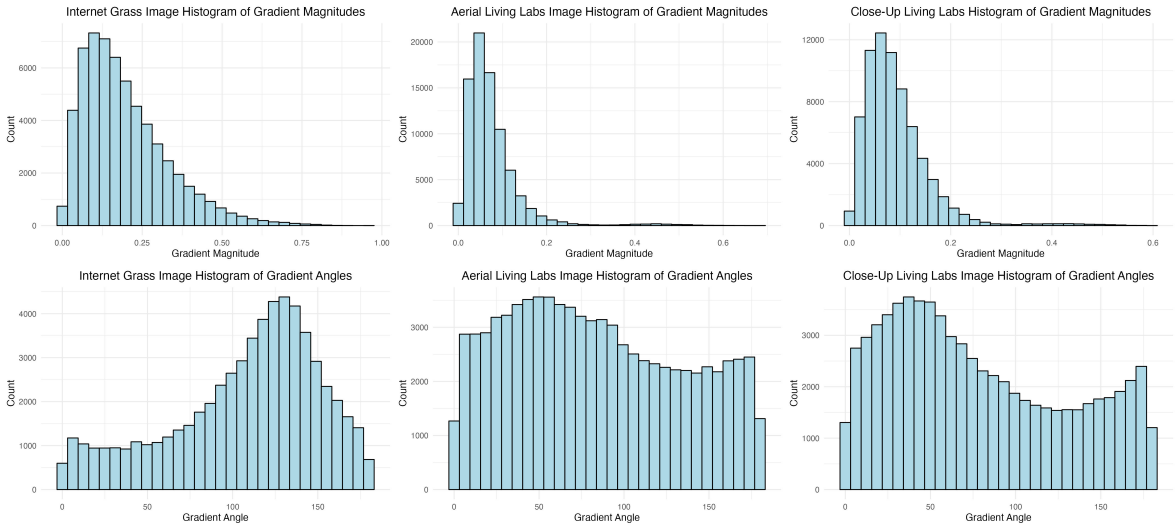


Figure 5.7: Grass Image Magnitudes and Angles

5.6 Build New Distributed Histogram Data Frames for Grass Images

```
# Define the number of bins
num_bins <- 9

filtered_grass_standard_df_list <-list(internet_grass_hog_df %>%
                                         filter(mag >= internet_grass_mag_filter),
                                       aerial_living_lab_hog_df %>%
                                         filter(mag >= aerial_living_lab_mag_filter),
                                       close_up_living_lab_hog_df %>%
```

```
                                        filter(mag >= close_up_living_lab_mag_filter))
grass_contribution_df_list <- list()


calculate_bin_contributions <- function(angle, magnitude, num_bins) {
  bin_width <- 180 / num_bins
  contributions <- numeric(num_bins)

  # get the central bin
  central_bin <- floor(angle / bin_width) %% num_bins
  next_bin <- (central_bin + 1) %% num_bins

  # get contributions to neighboring bins
  weight <- (1 - abs((angle %% bin_width) / bin_width)) * magnitude

  contributions[central_bin + 1] <- weight
  contributions[next_bin + 1] <- magnitude - weight

  return(list(contributions[1],
         contributions[2],
         contributions[3],
         contributions[4],
         contributions[5],
         contributions[6],
         contributions[7],
         contributions[8],
         contributions[9])
         )
}

for (i in 1:length(filtered_grass_standard_df_list)){

  grass_contribution_hog_df <-
    filtered_grass_standard_df_list[[i]] %>%
    rowwise() %>%
    mutate(`0` = calculate_bin_contributions(theta, mag, 9)[[1]],
           `20` = calculate_bin_contributions(theta, mag, 9)[[2]],
           `40` = calculate_bin_contributions(theta, mag, 9)[[3]],
           `60` = calculate_bin_contributions(theta, mag, 9)[[4]],
           `80` = calculate_bin_contributions(theta, mag, 9)[[5]],
           `100` = calculate_bin_contributions(theta, mag, 9)[[6]],
           `120` = calculate_bin_contributions(theta, mag, 9)[[7]],
```

```
            `140` = calculate_bin_contributions(theta, mag, 9)[[8]],
            `160` = calculate_bin_contributions(theta, mag, 9)[[9]],
            )

  grass_split_histo_df <-
    grass_contribution_hog_df %>%
    pivot_longer(names_to = "bin",
                 values_to = "contribution",
                 cols = 4:ncol(grass_contribution_hog_df)) %>%
    mutate(bin = as.numeric(bin)) %>%
    group_by(bin) %>%
    summarise(contribution_sum = sum(contribution))


  grass_contribution_df_list[[i]] <- grass_split_histo_df

}
```

5.7 Generate Grass Image Polar Plots for Standard Historgrams

```
internet_grass_plot <-
  ggplot(filtered_grass_standard_df_list[[1]],
         aes(x = theta)) +
  geom_histogram(colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Internet Grass Image\nUsing Standard HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))
```

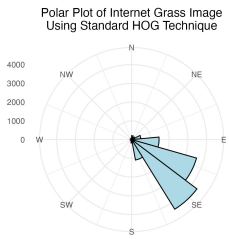
```
ggsave("images/plots/grass/internet_grass_standard_polar_plot.jpg", internet_grass_plot, wid
```

```
aerial_living_lab_plot <-
  ggplot(filtered_grass_standard_df_list[[2]],
         aes(x = theta)) +
  geom_histogram(colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Aerial Living Labs Image\nUsing Standard HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/grass/aerial_living_lab_standard_polar_plot.jpg", aerial_living_lab_plo
```

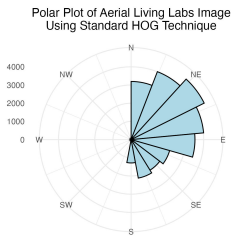
```
close_up_living_lab_plot <-
  ggplot(filtered_grass_standard_df_list[[3]],
         aes(x = theta)) +
  geom_histogram(colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Close-Up Living Lab Image\nUsing Standard HOG Technique") +
```

```
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/grass/close_up_living_lab_standard_polar_plot.jpg", close_up_living_lab
```

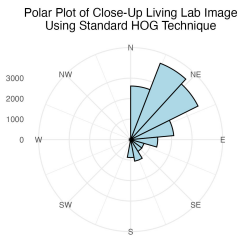
```
all_standard_grass_plots <- ggpubr::ggarrange(internet_grass_plot,
                                              aerial_living_lab_plot,
                                              close_up_living_lab_plot)

ggsave("images/plots/grass/all_grass_standard_polar_plots.jpg",
       all_standard_grass_plots,
       width = 7,
       height = 7)
```



(a) Internet Grass Image



(a) Aerial Living Labs Image



(a) Close-Up Living Labs Image

Figure 5.10: Aerial Cityscape Standard Polar Plots

5.8 Generate Grass Image Polar Plots for Distributed Historgrams

```
internet_grass_split_plot <-
  ggplot(grass_contribution_df_list[[1]],
         aes(x = bin, y = contribution_sum)) +
  geom_histogram(stat = "identity",
                 colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
```

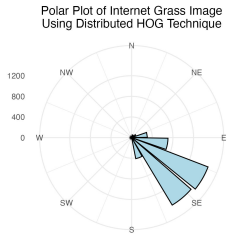
```
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Internet Grass Image\nUsing Distributed HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/grass/internet_grass_contribution_polar_plot.jpg", internet_grass_split_
```

```
aerial_living_lab_split_plot <-
  ggplot(grass_contribution_df_list[[2]],
         aes(x = bin, y = contribution_sum)) +
  geom_histogram(stat = "identity",
                 colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Aerial Living Lab Image\nUsing Distributed HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/grass/aerial_living_lab_contribution_polar_plot.jpg", aerial_living_lab_
```

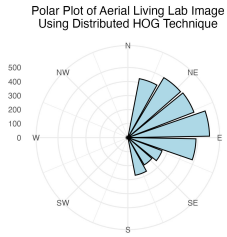
```
close_up_living_lab_split_plot <-
  ggplot(grass_contribution_df_list[[3]],
         aes(x = bin, y = contribution_sum)) +
```

```
  geom_histogram(stat = "identity",
                 colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Close-Up Living Lab Image\nUsing Distributed HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/grass/close_up_living_lab_contribution_polar_plot.jpg", close_up_living_
```

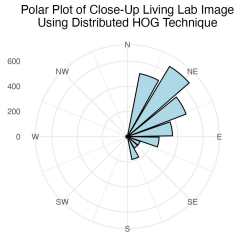
```
all_grass_contribution_plots <- ggpubr::ggarrange(internet_grass_split_plot,
                                                  aerial_living_lab_split_plot,
                                                  close_up_living_lab_split_plot)

ggsave("images/plots/grass/all_grass_contribution_plots.jpg",
       all_grass_contribution_plots,
       width = 7,
       height = 7)
```



(a) Internet Grass Image



(a) Aerial Living Labs Image



(a) Close-Up Living Labs Grass Image

Figure 5.13: Distributed Method Polar Plots for Grass Images

6 Backflip Image

```
library(reticulate)
library(tidyverse)
library(mapsapi)
library(mapboxapi)
library(magick)
Sys.which("python")
```

```
                                                    python
"/Users/bensunshine/.virtualenvs/r-reticulate/bin/python"
```

```
import matplotlib.pyplot as plt
import pandas as pd

# jupyter only inline output command
#%matplotlib inline

from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
from skimage import data, exposure


import matplotlib.pyplot as plt
from skimage import io
from skimage import color
from skimage.transform import resize
import math
from skimage.feature import hog
import numpy as np
```

6.1 Collect HOG Features for Backflip Image

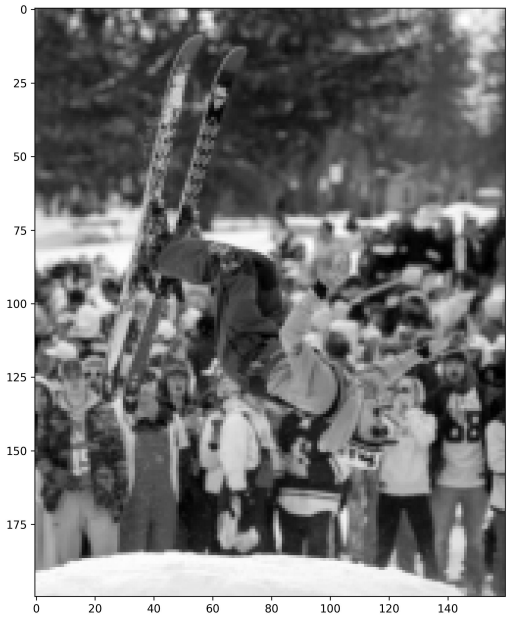
```
# List for storing images
img_list = []

# SF aerial
img_list.append(color.rgb2gray(io.imread("images/TitusFlip.jpg")))


#img = color.rgb2gray(io.imread("images/grass_image2.jpg"))

# img = color.rgb2gray(io.imread("images/b_test_image_copy.jpg"))
#img = color.rgb2gray(io.imread("images/long_grass_sample.jpeg"))
#img = color.rgb2gray(io.imread("images/diagnol_lines.jpg"))

#img = color.rgb2gray(io.imread("images/san_francisco_scale_zoom_12.png"))

#img = color.rgb2gray(io.imread("images/diagnol_lines_flipped.jpg"))
#img = color.rgb2gray(io.imread("images/long_grass_sample_cropped.jpg"))

# aerial rotated image
#img = color.rgb2gray(io.imread("images/living_lab_aerial/aerial_grass_living_lab_rotated.jpg

# zoomed internet photo
#img = color.rgb2gray(io.imread("images/dead_grass_zoom.jpeg"))


# zoomed in 11
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_11.jpg"))

# zoomed in 12
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_12.jpg"))

# zoomed in 16
#img = color.rgb2gray(io.imread("images/living_lab_aerial/LL_zoomed_in_16_side.jpg"))



# real one
#img = color.rgb2gray(io.imread("images/living_labs_real_grass_image.jpg"))

# List to store magnitudes for each image
mag_list = []
```

```
# List to store angles for each image
theta_list = []


for x in range(len(img_list)):
    # Get image of interest
    img = img_list[x]

    rescaled_file_path = f"images/plots/backflip/{x}.jpg"

    # Determine aspect Ratio
    aspect_ratio = img.shape[0] / img.shape[1]
    print("Aspect Ratio:", aspect_ratio)

    # Hard-Code height to 200 pixels
    height = 200

    # Calculate witdth to maintain same aspect ratio
    width = int(height / aspect_ratio)
    print("Resized Width:", width)

    # Resize the image
    resized_img = resize(img, (height, width))

    # Replace the original image with the resized image
    img_list[x] = resized_img

    # if (x == 1):
    #   plot_width = 8
    #   plot_height = 15
    # else:
    #   plot_width = 15
    #   plot_height = 8

    # plt.figure(figsize=(15, 8))
    # plt.imshow(resized_img, cmap="gray")
    # plt.axis("on")
    # plt.tight_layout()
    # plt.savefig(rescaled_file_path, dpi=300)
    # plt.show()

    # list for storing all magnitudes for image[x]
```

```
    mag = []

    # list for storing all angles for image[x]
    theta = []

    for i in range(height):
        magnitudeArray = []
        angleArray = []

        for j in range(width):
            if j - 1 < 0 or j + 1 >= width:
                if j - 1 < 0:
                    Gx = resized_img[i][j + 1] - 0
                elif j + 1 >= width:
                    Gx = 0 - resized_img[i][j - 1]
            else:
                Gx = resized_img[i][j + 1] - resized_img[i][j - 1]

            if i - 1 < 0 or i + 1 >= height:
                if i - 1 < 0:
                    Gy = 0 - resized_img[i + 1][j]
                elif i + 1 >= height:
                    Gy = resized_img[i - 1][j] - 0
            else:
                Gy = resized_img[i + 1][j] - resized_img[i - 1][j]

            magnitude = math.sqrt(pow(Gx, 2) + pow(Gy, 2))
            magnitudeArray.append(round(magnitude, 9))

            if Gx == 0:
                angle = math.degrees(0.0)
            else:
                angle = math.degrees(math.atan(Gy / Gx))
                if angle < 0:
                    angle += 180

            angleArray.append(round(angle, 9))

        mag.append(magnitudeArray)
        theta.append(angleArray)

    # add list of magnitudes to list[x]
```

```
    mag_list.append(mag)

    # add list of angles to angle list[x]
    theta_list.append(theta)
```

```
Aspect Ratio: 1.25
Resized Width: 160
```



(a) Skiing Backflip

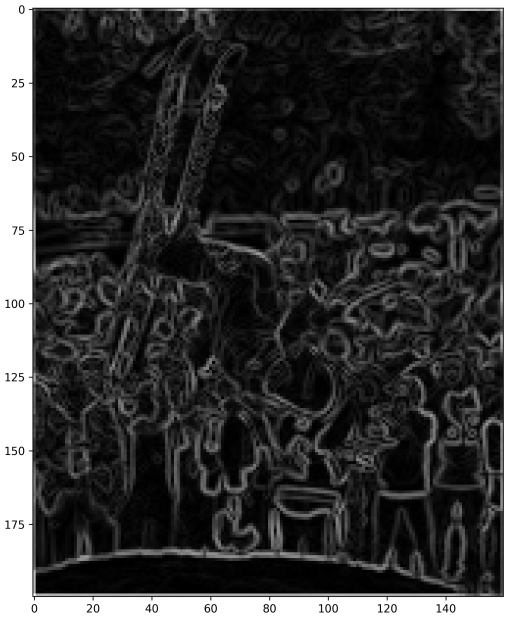
Figure 6.1: Skiing Backflip Image Rescaled and Converted to Greyscale

6.2 Build Data Frames for Backflip Image

```
mag_flip = np.array(mag_list[0])
theta_flip = np.array(theta_list[0])
```

6.3 Plot Magnitudes as Image for Diagonal Lines

```
# plt.figure(figsize=(15, 8))
# #plt.title('San Francisco, CA Gradient Magnitudes')
# plt.imshow(mag_list[0], cmap="gray")
# plt.axis("on")
# #plt.show()
# plt.tight_layout()
# plt.savefig("images/plots/backflip/backflip_mag.png", dpi=300)
```



(a) Skiing Backflip Image

Figure 6.2: Skiing Backflip Cityscape Magnitudes as Image

6.4 Create Data Frame for Backflip Image

```
# Flip DF
backflip_hog_df <- data.frame(mag = as.vector(py$mag_flip),
                              theta = as.vector((py$theta_flip))) %>%
  mutate(radian = theta*(pi/180))
```

```
# List of all Data frames
flip_standard_df_list = list(backflip_hog_df)
```

6.5 Create Histograms of Gradient Magnitudes and Angles for Backflip Image

```
flip_histogram_mag_plot <-
  ggplot(flip_standard_df_list[[1]],
         aes(x = mag)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Magnitude",
       y = "Count",
       title = "Skiing Backflip Image Histogram of Gradient Magnitudes"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#flip_histogram_mag_plot

flip_mag_filter <- 0.2

ggsave("images/plots/backflip/backflip_histogram_mag_plot.jpg", flip_histogram_mag_plot, wid



flip_histogram_theta_plot <-
  ggplot(flip_standard_df_list[[1]],
         aes(x = theta)) +
  geom_histogram(colour = "black", fill = "lightblue") +
  scale_x_continuous() +
  labs(x = "Gradient Angle",
       y = "Count",
       title = "Skiing Backflip Image Histogram of Gradient Angles"
       ) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

#flip_histogram_theta_plot
```

```
ggsave("images/plots/backflip/backflip_histogram_theta_plot.jpg", flip_histogram_theta_plot,
```

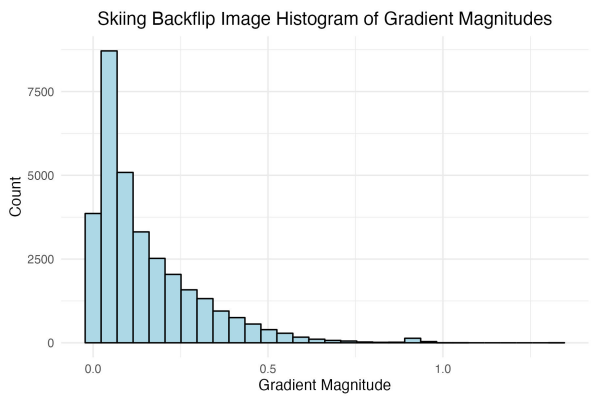


Figure 6.3: Skiing Backflip Magnitudes and Angles

6.6 Build New Distributed Histogram Data Frame for Backflip Image

```
# Define the number of bins
num_bins <- 9

# function to calculate the contributions to neighboring bins
calculate_bin_contributions <- function(angle, magnitude, num_bins) {
  bin_width <- 180 / num_bins
  contributions <- numeric(num_bins)

  # get the central bin
  central_bin <- floor(angle / bin_width) %% num_bins
  next_bin <- (central_bin + 1) %% num_bins

  # get contributions to neighboring bins
  weight <- (1 - abs((angle %% bin_width) / bin_width)) * magnitude

  contributions[central_bin + 1] <- weight
  contributions[next_bin + 1] <- magnitude - weight

  return(list(contributions[1],
         contributions[2],
```

```
          contributions[3],
          contributions[4],
          contributions[5],
          contributions[6],
          contributions[7],
          contributions[8],
          contributions[9])
          )
}

filtered_flip_standard_df_list <-list(backflip_hog_df %>%
                                        filter(mag >= flip_mag_filter))

flip_contribution_df_list <- list()


for (i in 1:length(filtered_flip_standard_df_list)){

  flip_contribution_hog_df <-
    filtered_flip_standard_df_list[[i]] %>%
    rowwise() %>%
    mutate(`0` = calculate_bin_contributions(theta, mag, 9)[[1]],
           `20` = calculate_bin_contributions(theta, mag, 9)[[2]],
           `40` = calculate_bin_contributions(theta, mag, 9)[[3]],
           `60` = calculate_bin_contributions(theta, mag, 9)[[4]],
           `80` = calculate_bin_contributions(theta, mag, 9)[[5]],
           `100` = calculate_bin_contributions(theta, mag, 9)[[6]],
           `120` = calculate_bin_contributions(theta, mag, 9)[[7]],
           `140` = calculate_bin_contributions(theta, mag, 9)[[8]],
           `160` = calculate_bin_contributions(theta, mag, 9)[[9]],
           )

  flip_split_histo_df <-
    flip_contribution_hog_df %>%
    pivot_longer(names_to = "bin",
                 values_to = "contribution",
                 cols = 4:ncol(flip_contribution_hog_df)) %>%
    mutate(bin = as.numeric(bin)) %>%
    group_by(bin) %>%
    summarise(contribution_sum = sum(contribution))
```

```
  flip_contribution_df_list[[i]] <- flip_split_histo_df

}
```

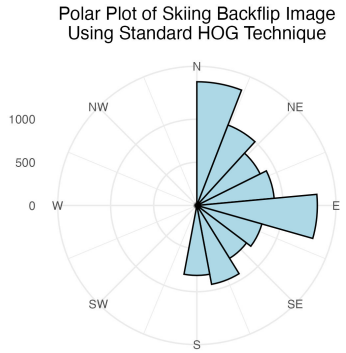
6.7 Generate Polar Plots for Standard Historgrams for Backflip Image

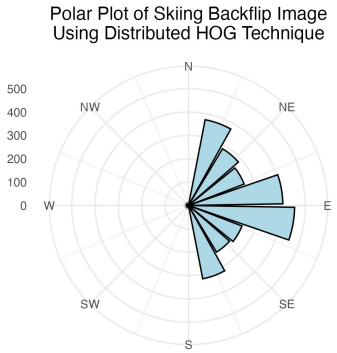
```
flip_plot <-
  ggplot(filtered_flip_standard_df_list[[1]],
         aes(x = theta)) +
  geom_histogram(colour = "black",
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Skiing Backflip Image\nUsing Standard HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/backflip/backflip_standard_polar_plot.jpg", flip_plot, width = 6, height
```

6.8 Generate Polar Plots for Distributed Historgrams of Diagonal Image

```
flip_split_plot <-
  ggplot(flip_contribution_df_list[[1]],
         aes(x = bin, y = contribution_sum)) +
  geom_histogram(stat = "identity",
                 colour = "black",
```

```
                 fill = "lightblue",
                 breaks = seq(0, 360, length.out = 17.5),
                 bins = 9) +
  coord_polar(
    theta = "x",
    start = 0,
    direction = 1) +
  scale_x_continuous(limits = c(0,360),
    breaks = c(0, 45, 90, 135, 180, 225, 270, 315),
    labels = c("N", "NE", "E", "SE", "S", "SW", "W", "NW")
  )+
  labs(title = "Polar Plot of Skiing Backflip Image\nUsing Distributed HOG Technique") +
  theme_minimal() +
  labs(x = "") +
  theme(axis.title.y = element_blank(),
        plot.title = element_text(hjust = 0.5))

ggsave("images/plots/backflip/backflip_contribution_polar_plot.jpg", flip_split_plot, width =
```





(a) Skiing Backflip Image Standard HOG Method (a) Skiing Backflip Image Distributed HOG Method

Figure 6.5: Skiing Backflip Image Distributed Method Polar Plot

7 Conclusion