Collecting Data

```
#downloading data
!git clone https://github.com/rslim087a/track
```

```
Cloning into 'track'...
remote: Enumerating objects: 12163, done.ote: Total 12163 (delta 0),
reused 0 (delta 0), pack-reused 12163 (from 1)
```

```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from sklearn.model_selection import train_test_split
from imgaug import augmenters as iaa
import cv2
import pandas as pd
import ntpath
import random

datadir = 'track'
columns = ['center', 'left', 'right', 'steering', 'throttle',
'reverse', 'speed']
data = pd.read_csv(os.path.join(datadir, 'driving_log.csv'), names =
columns)
pd.set_option('display.max_colwidth', 1)
data.head()
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 4053,\n  \"fields\":
[\n    {\n      \"column\": \"center\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4053,\n
\"samples\": [\n          \"C:\\\\Users\\\\Amer\\\\Desktop\\\\
new_track\\\\IMG\\\\center_2018_07_16_17_12_53_121.jpg\",\n
\"C:\\\\Users\\\\Amer\\\\Desktop\\\\new_track\\\\IMG\\\\
center_2018_07_16_17_12_40_396.jpg\",\n          \"C:\\\\Users\\\\
Amer\\\\Desktop\\\\new_track\\\\IMG\\\\
center_2018_07_16_17_13_32_392.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"left\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4053,\n
\"samples\": [\n          \"C:\\\\Users\\\\Amer\\\\Desktop\\\\
new_track\\\\IMG\\\\left_2018_07_16_17_12_53_121.jpg\",\n
\"C:\\\\Users\\\\Amer\\\\Desktop\\\\new_track\\\\IMG\\\\
left_2018_07_16_17_12_40_396.jpg\",\n          \"C:\\\\Users\\\\
```

```
Amer\\\\Desktop\\\\new_track\\\\IMG\\\\
left_2018_07_16_17_13_32_392.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"right\",\n      \"properties\": {\
n        \"dtype\": \"string\",\n        \"num_unique_values\": 4053,\
n        \"samples\": [\n          \"C:\\\\Users\\\\Amer\\\\
Desktop\\\\new_track\\\\IMG\\\\right_2018_07_16_17_12_53_121.jpg\",\n
\"C:\\\\Users\\\\Amer\\\\Desktop\\\\new_track\\\\IMG\\\\
right_2018_07_16_17_12_40_396.jpg\",\n          \"C:\\\\Users\\\\
Amer\\\\Desktop\\\\new_track\\\\IMG\\\\
right_2018_07_16_17_13_32_392.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"steering\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.19054945053021108,\n        \"min\": -1.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 580,\n        \"samples\": [\n
0.1990736,\n          -0.2064387,\n          -0.009710268\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"throttle\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.18898062099972482,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 33,\n        \"samples\": [\n
0.3802254,\n          0.1444282,\n          0.5257462\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"reverse\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.1397309624883202,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 14,\n        \"samples\": [\n          1.0,\n
0.8578613,\n          0.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"speed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4.598975915203963,\n        \"min\":
0.0483394,\n        \"max\": 30.65264,\n        \"num_unique_values\":
1166,\n        \"samples\": [\n          30.15488,\n
30.19033,\n          30.17225\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data"}
```

1. `datadir = 'track'`

- **Explanation:**
  This line sets a variable `datadir` to the string `'track'`. This variable is intended to store the directory name where the data files are located. In this case, it is assumed that the `driving_log.csv` file is inside a folder named `track`.

2. `columns = ['center', 'left', 'right', 'steering', 'throttle', 'reverse', 'speed']`

- **Explanation:**
  This line creates a list called `columns` that contains the names of the columns in the CSV file. These names correspond to the data recorded by the self-driving car simulator:

  - `'center'`: The file path of the image taken by the center camera.
  - `'left'`: The file path of the image taken by the left camera.
  - `'right'`: The file path of the image taken by the right camera.
  - `'steering'`: The steering angle of the car at the time the image was taken.
  - `'throttle'`: The throttle position of the car (how much the accelerator is pressed).
  - `'reverse'`: Whether the car is in reverse gear.
  - `'speed'`: The speed of the car.

3. `data = pd.read_csv(os.path.join(datadir, 'driving_log.csv'), names=columns)`

- **Explanation:**
  This line reads the `driving_log.csv` file into a Pandas DataFrame, which is stored in the variable `data`. Let's break down the components:

  - `pd.read_csv(...)`: This is a Pandas function that reads a CSV file and returns its contents as a DataFrame.
  - `os.path.join(datadir, 'driving_log.csv')`: This function creates a path by joining the directory stored in `datadir` (`'track'`) with the file name `'driving_log.csv'`. This is useful for making your code platform-independent since it handles different path separators (e.g., `/` on Linux/Mac and `\` on Windows).
  - `names=columns`: This parameter tells Pandas to use the `columns` list as the names of the columns in the DataFrame.

4. `pd.set_option('display.max_colwidth', 1)`

- **Explanation:**
  This line sets an option in Pandas to control how wide the columns can be when displayed. The `'display.max_colwidth'` option limits the maximum number of characters that will be displayed in each column. Setting it to `1` means that only one character will be shown for each column entry when displaying the DataFrame. This is generally used to keep the output concise, especially when dealing with long strings like file paths.

5. `data.head()`

- **Explanation:**
  This line returns the first few rows of the DataFrame `data`. The `head()` function by default returns the first 5 rows, allowing you to quickly inspect the structure and contents of the DataFrame. This is typically used to verify that the data was loaded correctly.

```
def path_leaf(path):
    head, tail = ntpath.split(path)
    return tail

data['center'] = data['center'].apply(path_leaf)
data['left'] = data['left'].apply(path_leaf)
data['right'] = data['right'].apply(path_leaf)
data.head()
```

```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 4053,\n  \"fields\":
[\n    {\n      \"column\": \"center\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4053,\n
\"samples\": [\n          \"center_2018_07_16_17_12_53_121.jpg\",\n
\"center_2018_07_16_17_12_40_396.jpg\",\n
\"center_2018_07_16_17_13_32_392.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"left\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4053,\n
\"samples\": [\n          \"left_2018_07_16_17_12_53_121.jpg\",\n
\"left_2018_07_16_17_12_40_396.jpg\",\n
\"left_2018_07_16_17_13_32_392.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"right\",\n      \"properties\": {\
n        \"dtype\": \"string\",\n        \"num_unique_values\": 4053,\
n        \"samples\": [\n
\"right_2018_07_16_17_12_53_121.jpg\",\n
\"right_2018_07_16_17_12_40_396.jpg\",\n
\"right_2018_07_16_17_13_32_392.jpg\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"steering\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.19054945053021108,\n        \"min\": -1.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 580,\n        \"samples\": [\n
0.1990736,\n          -0.2064387,\n          -0.009710268\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"throttle\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.18898062099972482,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 33,\n        \"samples\": [\n
0.3802254,\n          0.1444282,\n          0.5257462\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"reverse\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.1397309624883202,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 14,\n        \"samples\": [\n          1.0,\n
0.8578613,\n          0.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"speed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4.598975915203963,\n        \"min\":
0.0483394,\n        \"max\": 30.65264,\n        \"num_unique_values\":
1166,\n        \"samples\": [\n          30.15488,\n
30.19033,\n          30.17225\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data"}
```

1. `def path_leaf(path):`

- **Explanation:**
  This line defines a function named `path_leaf` that takes one parameter called `path`. The purpose of this function is to extract the "leaf" or file name from a full file path.

2. `head, tail = ntpath.split(path)`

- **Explanation:**
  Inside the `path_leaf` function, this line uses the `ntpath.split()` function to split the given `path` into two parts:
  - `head`: The directory path leading up to the file.
  - `tail`: The actual file name with its extension (this is what you want to extract).

  `ntpath` is a module in Python that is similar to `os.path`, but it's specialized for handling Windows-style paths. However, it works across platforms, making it a good choice for extracting file names in a platform-independent way.

3. `return tail`

- **Explanation:**
  This line returns the `tail`, which is the file name portion of the `path`. This means that when you pass a file path to the `path_leaf` function, it will return just the file name, stripping away the directory path.

4. `data['center'] = data['center'].apply(path_leaf)`

- **Explanation:**
  This line applies the `path_leaf` function to each entry in the `'center'` column of the `data` DataFrame. The `apply()` function is a Pandas method that allows you to apply a function to each element of a DataFrame column. Here, it replaces each full file path in the `'center'` column with just the file name.

5. `data['left'] = data['left'].apply(path_leaf)`

- **Explanation:**
  Similarly, this line applies the `path_leaf` function to each entry in the `'left'` column of the `data` DataFrame, extracting just the file names from the paths.

6. `data['right'] = data['right'].apply(path_leaf)`

- **Explanation:**
  This line does the same for the `'right'` column, again applying the `path_leaf` function to extract only the file names.

7. `data.head()`

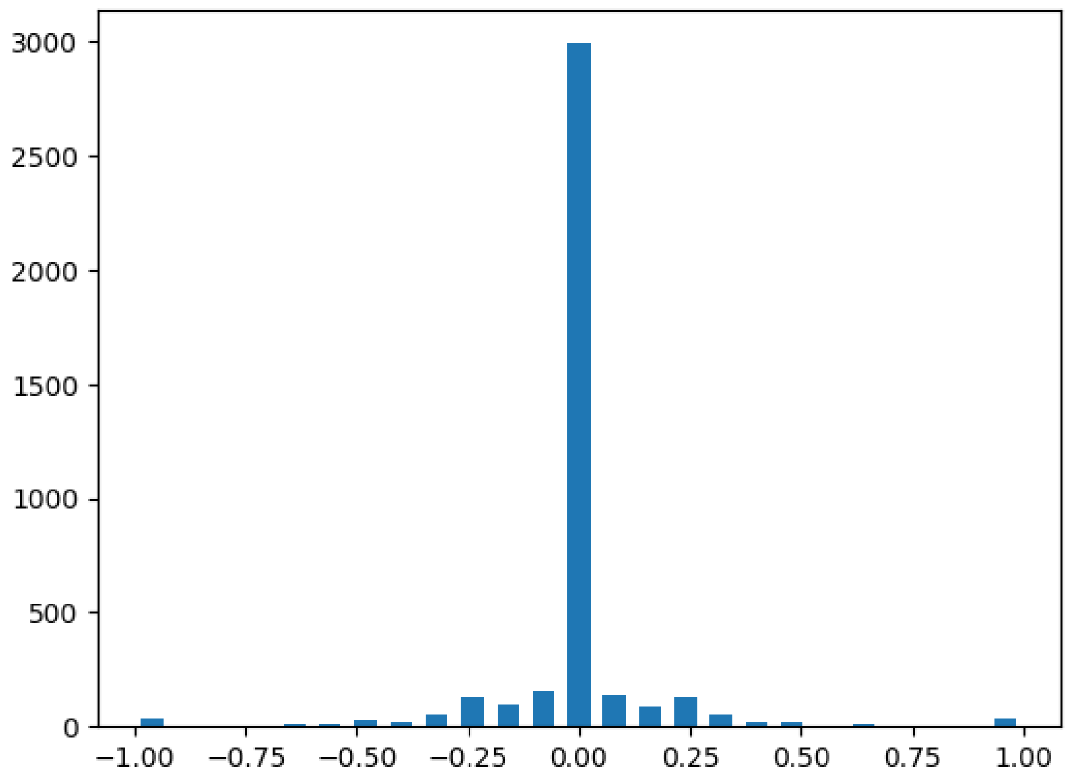
- **Explanation:**
  This line displays the first few rows of the `data` DataFrame again, allowing you to verify that the paths have been successfully converted to just the file names.

Summary

This block of code is used to clean up the `'center'`, `'left'`, and `'right'` columns in your DataFrame by stripping away the directory paths and keeping only the file names of the images. This can be useful if you only need the file names for further processing or to avoid redundancy when dealing with long file paths.

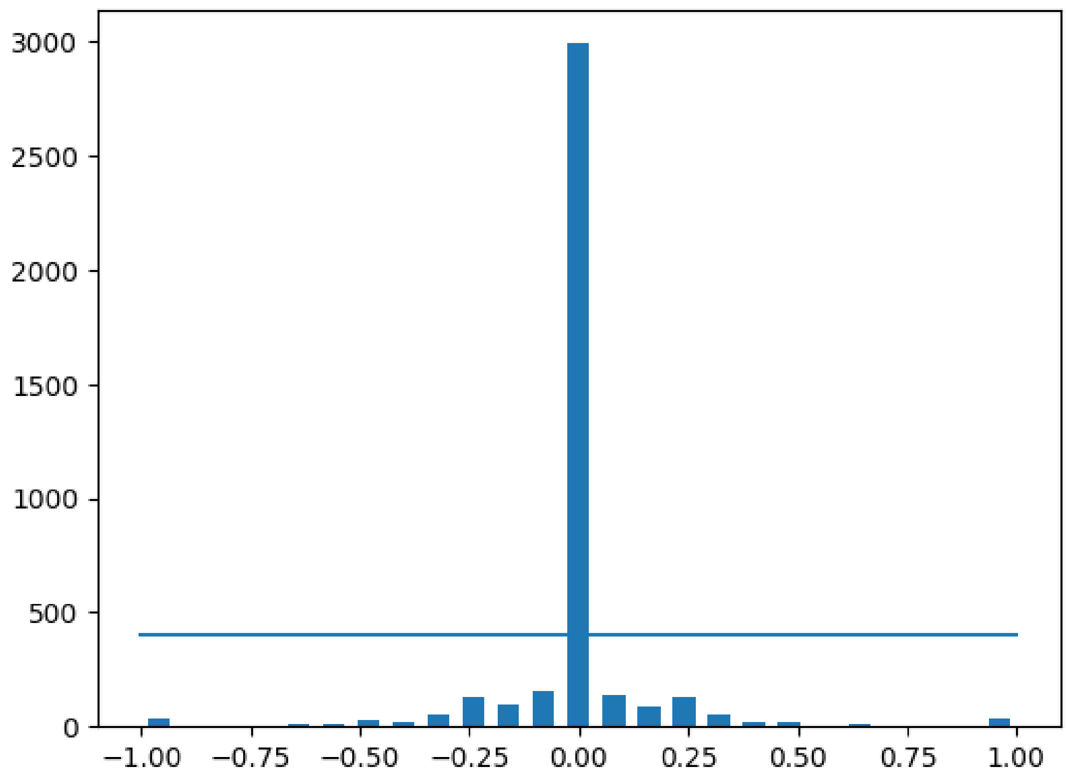
```python
num_bins=25
hist, bins = np.histogram(data['steering'], num_bins)
center = (bins[:-1]+ bins[1:]) * 0.5
plt.bar(center, hist, width=0.05)
```

```
<BarContainer object of 25 artists>
```



This code generates a histogram of the steering angle data. It divides the steering angles into 25 bins, calculates how many data points fall into each bin, and then creates a bar plot showing the distribution of the steering angles. The `center` array ensures that the bars are centered correctly over each bin, and the `width` parameter controls the visual width of the bars in the plot.

```
#0 more because we drive our car at the middle of the road
#but if i will pass this data to our model then our model will be
baised towards 0
#so i have to remove some 0

num_bins=25
samples_per_bin = 400
hist, bins = np.histogram(data['steering'], num_bins)
center = (bins[:-1]+ bins[1:]) * 0.5
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])),
(samples_per_bin, samples_per_bin))

[<matplotlib.lines.Line2D at 0x79ad85468040>]
```



This block of code visualizes the distribution of steering angles in your dataset and adds a reference line at a height of 400. The histogram shows how many samples fall into each bin, and the horizontal line at 400 serves as a visual guide or threshold. This could be useful if you're planning to balance the dataset by ensuring that no bin has more than 400 samples, potentially indicating that you might want to downsample bins with more than 400 samples or consider how to deal with underrepresented bins.

```
#till 400 we will keep the zero and will removes the zero
#so that our data will be balanced

print('total data :'),len(data)
print(data.shape)

total data :
(4053, 7)
```

Balancing Data

```
from random import shuffle

remove_list = []
for j in range(num_bins):
  list_ = []
```

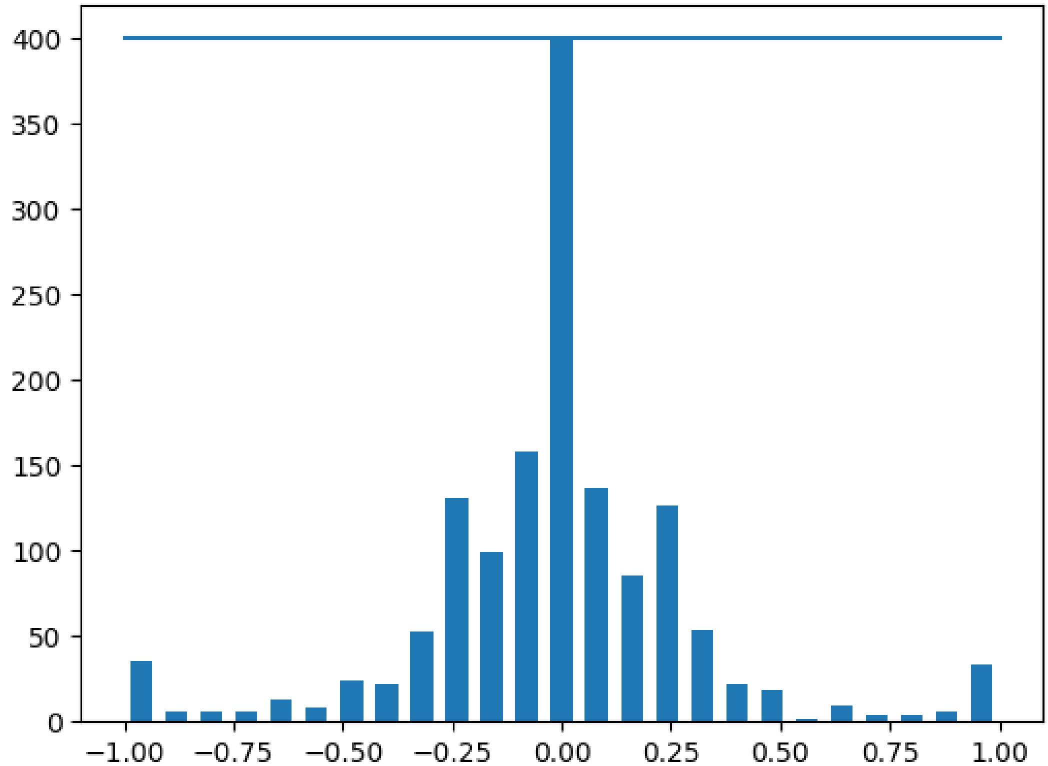
```
  for i in range(len(data['steering'])):
    if data['steering'][i] >= bins[j] and data['steering'][i] <=
bins[j+1]:
      list_.append(i)
  shuffle(list_)
  list_ = list_[samples_per_bin:]
  remove_list.extend(list_)

print('removed:', len(remove_list))
data.drop(data.index[remove_list], inplace=True)
print('remaining:', len(data))
```

```
removed: 2590
remaining: 1463
```

This code segment balances the dataset by ensuring that no bin in the histogram has more than 400 samples. It works by iterating over each bin, collecting the indices of the samples in that bin, shuffling them randomly, and then removing the excess samples. After running this code, your dataset should have a more uniform distribution of steering angles, with each bin containing at most 400 samples. This is often done to prevent certain steering angles from being overrepresented in the dataset, which could bias the model during training.

```
hist, _ = np.histogram(data['steering'], (num_bins))
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])),
(samples_per_bin, samples_per_bin))
```

```
[<matplotlib.lines.Line2D at 0x79ad853314b0>]
```




```
#now as you can see this data is normally distributed
```

Training & Validation Split

```
def load_ima_steering(datadir, df):
  image_path = []
  steering = []
  for i in range(len(data)):
    indexed_data = data.iloc[i]
    center, left, right = indexed_data[0], indexed_data[1],
indexed_data[2]
    image_path.append(os.path.join(datadir, center.strip()))
    steering.append(float(indexed_data[3]))
    # left image append
    image_path.append(os.path.join(datadir, left.strip()))
    steering.append(float(indexed_data[3])+0.15)
    #right image append
    image_path.append(os.path.join(datadir, right.strip()))
    steering.append(float(indexed_data[3])-0.15)

  image_paths = np.asarray(image_path)
  steerings = np.asarray(steering)
  return image_paths, steerings
```

The `load_ima_steering` function loads the image paths and corresponding steering angles from the dataset, generating additional data points by including the left and right camera images with slightly adjusted steering angles. This technique is often used in training models for self-driving cars to increase the diversity of the training data and help the model generalize better. The function returns two NumPy arrays: one containing the paths to the images and the other containing the associated steering angles.

```
image_paths, steerings = load_ima_steering(datadir + '/IMG', data)
```

```
<ipython-input-13-157f4b23b916>:6: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer
keys will always be treated as labels (consistent with DataFrame
behavior). To access a value by position, use `ser.iloc[pos]`
  center, left, right = indexed_data[0], indexed_data[1],
indexed_data[2]
<ipython-input-13-157f4b23b916>:8: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer
keys will always be treated as labels (consistent with DataFrame
behavior). To access a value by position, use `ser.iloc[pos]`
  steering.append(float(indexed_data[3]))
<ipython-input-13-157f4b23b916>:11: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer
keys will always be treated as labels (consistent with DataFrame
behavior). To access a value by position, use `ser.iloc[pos]`
  steering.append(float(indexed_data[3])+0.15)
<ipython-input-13-157f4b23b916>:14: FutureWarning: Series.__getitem__
treating keys as positions is deprecated. In a future version, integer
keys will always be treated as labels (consistent with DataFrame
behavior). To access a value by position, use `ser.iloc[pos]`
  steering.append(float(indexed_data[3])-0.15)
```

This line initializes the `image_paths` and `steerings` arrays, which you will likely use for training your self-driving car model. The `image_paths` array contains paths to the image files, and the `steerings` array contains the associated steering angles for each image.

```
image_paths
```

```
array(['track/IMG/center_2018_07_16_17_11_44_413.jpg',
       'track/IMG/left_2018_07_16_17_11_44_413.jpg',
       'track/IMG/right_2018_07_16_17_11_44_413.jpg', ...,
       'track/IMG/center_2018_07_16_17_16_32_161.jpg',
       'track/IMG/left_2018_07_16_17_16_32_161.jpg',
       'track/IMG/right_2018_07_16_17_16_32_161.jpg'], dtype='<U44')
```

```
len(image_paths)
```

```
4389
```

```
steerings
```

```
array([-0.05,  0.1 , -0.2 , ...,  0.  ,  0.15, -0.15])
```

```
len(steerings)
```

```
4389
```

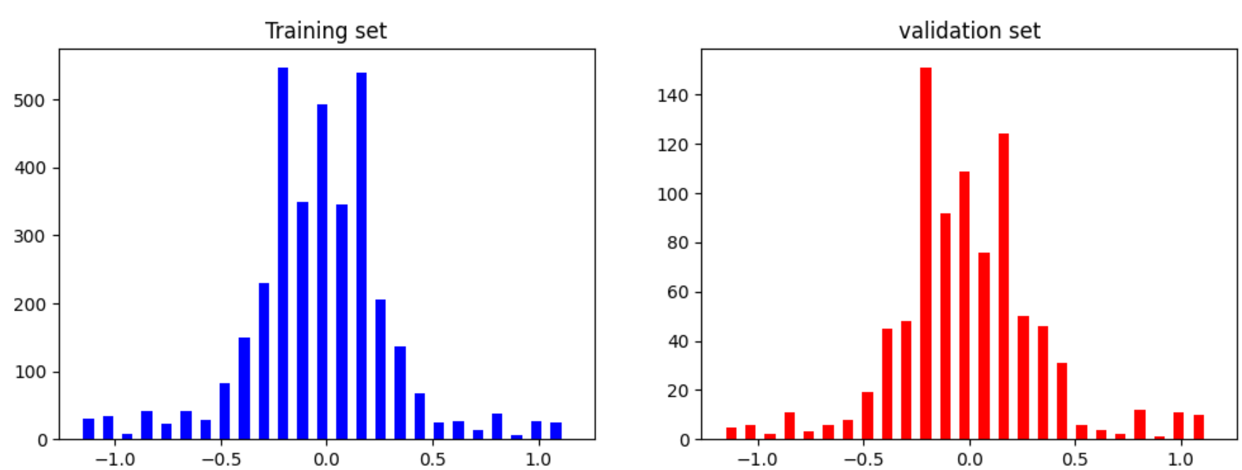
```
X_train, X_valid, y_train, y_valid = train_test_split(image_paths,
steerings, test_size=0.2, random_state=6)
print('Training Samples: {}\nValid Samples: {}'.format(len(X_train),
len(X_valid)))
```

```
Training Samples: 3511
Valid Samples: 878
```

This code splits the dataset into training and validation sets, with 80% of the data used for training and 20% for validation. It then prints out the number of samples in each set. This step is crucial for model evaluation, as it allows you to assess the model's performance on unseen data (the validation set) after training.

```
fig ,axes = plt.subplots(1,2, figsize=(12,4))
axes[0].hist(y_train, bins=num_bins, width=0.05, color='blue')
axes[0].set_title('Training set')
axes[1].hist(y_valid, bins=num_bins, width=0.05, color='red')
axes[1].set_title('validation set')
```

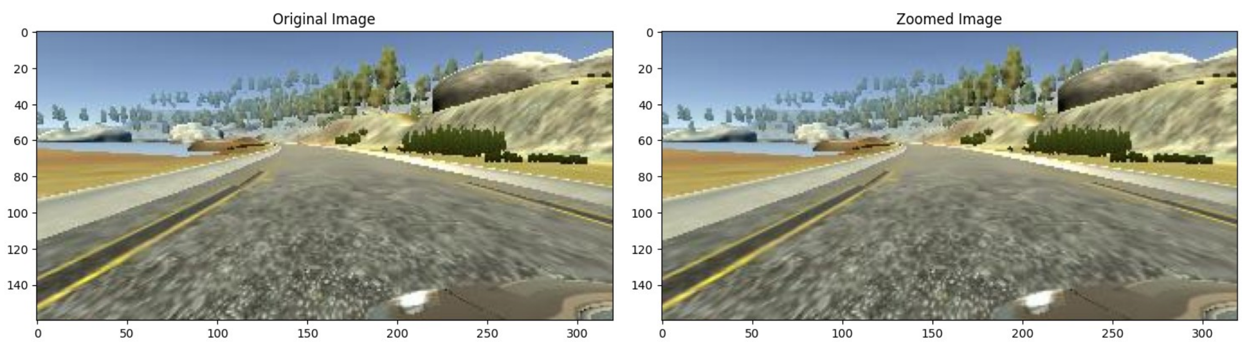
```
Text(0.5, 1.0, 'validation set')
```



This code creates a side-by-side comparison of the distribution of steering angles in the training and validation sets. The left subplot shows the histogram for the training set (`y_train`), and the right subplot shows the histogram for the validation set (`y_valid`). This visualization helps you verify that the distribution of steering angles in the training and validation sets is similar, which is important for ensuring that the model generalizes well from training to validation (and eventually test) data. The blue and red colors visually distinguish between the two datasets.

Preprocessing Images

```
# zoom augmentation
def zoom(image):
  zoom = iaa.Affine(scale=(1, 1.3)) # max 30% zoom
  image = zoom.augment_image(image)
  return image
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
zoomed_image = zoom(original_image)


fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')
axs[1].imshow(zoomed_image)
axs[1].set_title('Zoomed Image')

Text(0.5, 1.0, 'Zoomed Image')
```

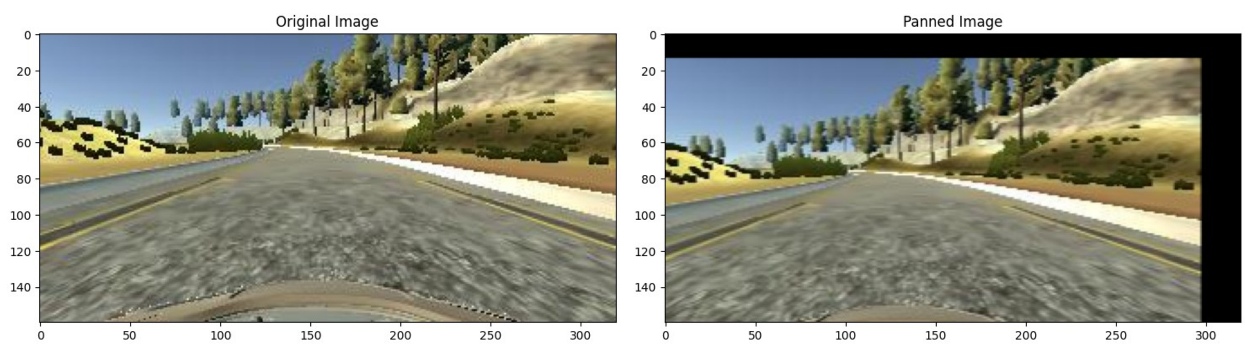


This code demonstrates how to perform and visualize a zoom augmentation on an image. The `zoom` function scales an image by up to 30%, and then the code displays the original and zoomed images side by side for comparison. This type of augmentation is useful in machine learning to help

train models that are more robust to variations in the input data, such as different perspectives or scales of the objects in the images. By augmenting the data with zoomed versions of the images, the model can learn to better generalize to new data that may have similar variations.

```
# pan augmentation
def pan(image):
  pan = iaa.Affine(translate_percent= {"x" : (-0.1, 0.1), "y": (-0.1,
0.1)}) # +-10% hor+vert pan
  image = pan.augment_image(image)
  return image


image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
panned_image = pan(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')
axs[1].imshow(panned_image)
axs[1].set_title('Panned Image')
```

```
Text(0.5, 1.0, 'Panned Image')
```

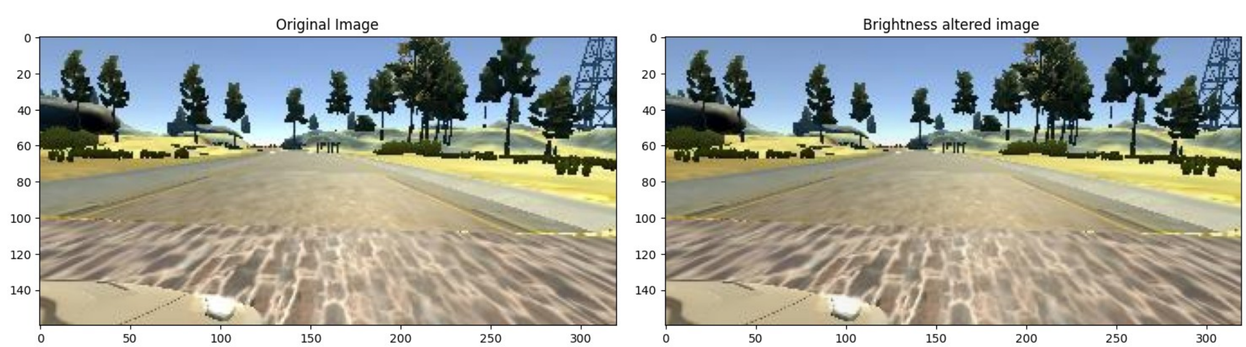


This code applies a panning augmentation to an image to simulate shifts in the camera's viewpoint. The `pan` function translates the image horizontally and vertically by up to 10% of its width and height, respectively. The original and panned images are displayed side by side to illustrate the effect of the augmentation. This type of transformation helps the model become more robust to variations in the position of objects within the frame.

In essence, panning augmentation helps create a more robust and adaptable model by exposing it to a wider range of image variations. This makes the model more effective at handling real-world conditions where objects might not always be perfectly centered or positioned as in the training data.

```
# brightness augmentation
def img_random_brightness(image):
    brightness = iaa.Multiply((0.2, 1.2))
    image = brightness.augment_image(image)
    return image


image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
brightness_altered_image = img_random_brightness(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')
axs[1].imshow(brightness_altered_image)
axs[1].set_title('Brightness altered image ')
```

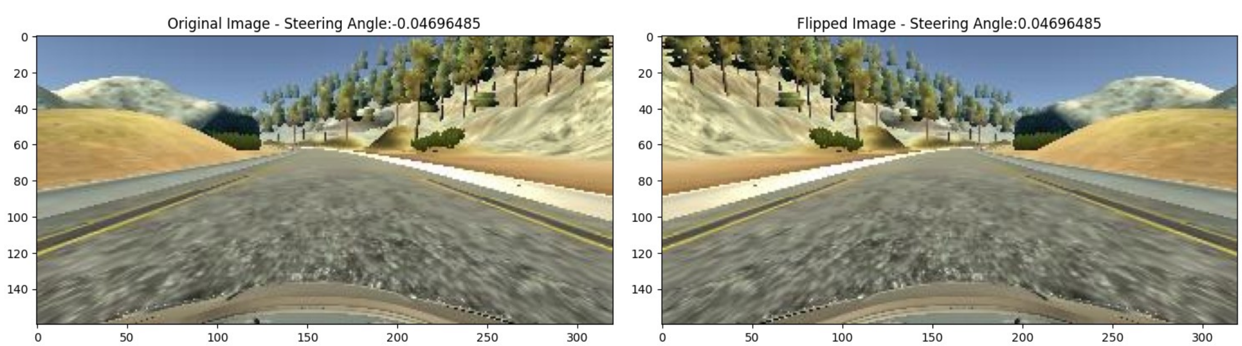
```
Text(0.5, 1.0, 'Brightness altered image ')
```



This code demonstrates how to apply brightness augmentation to an image and visualize the result. The `img_random_brightness` function adjusts the brightness of the image randomly within a specified range. The original and brightness-altered images are displayed side by side for comparison. This type of augmentation helps the model learn to handle variations in lighting conditions, making it more robust to changes in brightness that may occur in real-world scenarios.

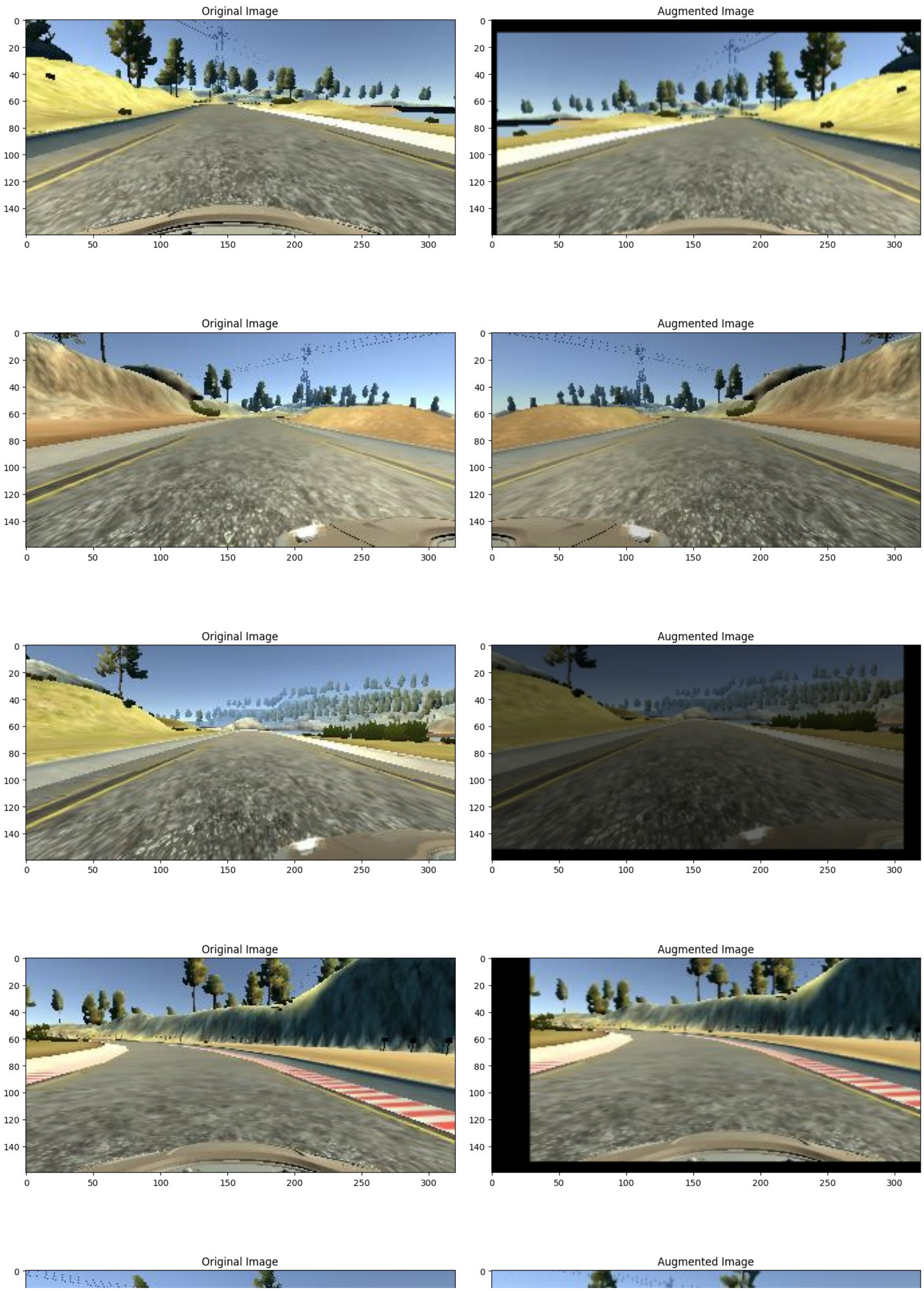
```python
# flip augmentation
def img_random_flip(image, steering_angle):
    image = cv2.flip(image,1)
    steering_angle = -steering_angle
    return image, steering_angle


random_index = random.randint(0, 1000)
image = image_paths[random_index]
steering_angle = steerings[random_index]


original_image = mpimg.imread(image)
flipped_image, flipped_steering_angle =
img_random_flip(original_image, steering_angle)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image - ' + 'Steering Angle:' +
str(steering_angle))
axs[1].imshow(flipped_image)
axs[1].set_title('Flipped Image - ' + 'Steering Angle:' +
str(flipped_steering_angle))
```

```
Text(0.5, 1.0, 'Flipped Image - Steering Angle:0.04696485')
```



This code illustrates how to apply horizontal flip augmentation to an image and adjust its associated steering angle. The original and flipped images, along with their respective steering angles, are displayed side by side. This augmentation technique is useful for training models in tasks like self-driving car simulations, where the model needs to handle various orientations and directions.

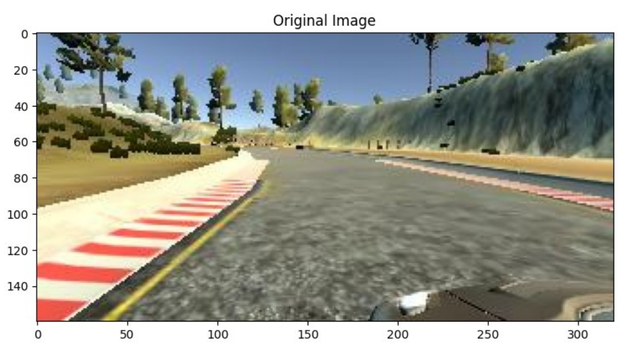
```
# augmentation chances
def random_augment(image, steering_angle):
    image = mpimg.imread(image)
    if np.random.rand() < 0.5:
      image = pan(image)
    if np.random.rand() < 0.5:
      image = zoom(image)
    if np.random.rand() < 0.5:
      image = img_random_brightness(image)
    if np.random.rand() < 0.5:
      image, steering_angle = img_random_flip(image, steering_angle)
    return image, steering_angle

ncol = 2
nrow = 10
fig, axs = plt.subplots(nrow, ncol, figsize=(15, 50))
fig.tight_layout()
for i in range(10):
  randnum = random.randint(0, len(image_paths) - 1)
  random_image = image_paths[randnum]
  random_steering = steerings[randnum]

  original_image = mpimg.imread(random_image)
  augmented_image, steering = random_augment(random_image,
random_steering)

  axs[i][0].imshow(original_image)
  axs[i][0].set_title("Original Image")
  axs[i][1].imshow(augmented_image)
  axs[i][1].set_title("Augmented Image")
```

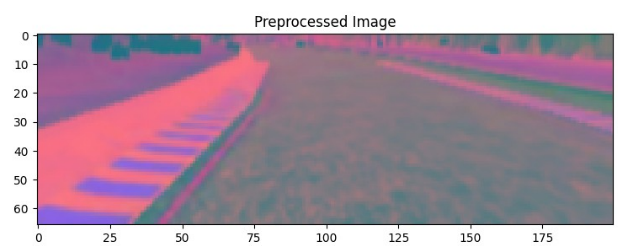

Original Image
Augmented Image
Original Image
Augmented Image
Original Image
Augmented Image
Original Image
Augmented Image
Original Image
Augmented Image

This code snippet visualizes 10 pairs of images where each pair consists of an original image and its augmented version. It arranges the images in a grid with 10 rows and 2 columns, allowing for a side-by-side comparison of the original and augmented images. This is useful for inspecting the effects of various augmentations applied to the images and ensuring that the augmentations are performed correctly.

```python
# delete unimportant image data
def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV) # for nvidia model
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255 # normalization
    return img


image = image_paths[100]
original_image = mpimg.imread(image)
preprocessed_image = img_preprocess(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axs[0].imshow(original_image)
axs[0].set_title('Original Image')
axs[1].imshow(preprocessed_image)
axs[1].set_title('Preprocessed Image')
```

```
Text(0.5, 1.0, 'Preprocessed Image')
```





This code snippet visualizes the effect of preprocessing on an image. The preprocessing steps include cropping, color space conversion, blurring, resizing, and normalization. By comparing the original and preprocessed images side by side, you can see how these transformations prepare the image for input into a neural network, potentially improving model performance by reducing noise and standardizing the input.

```
def batch_generator(image_paths, steering_ang, batch_size,
istraining):
  while True:
    batch_img = []
    batch_steering = []
    for i in range(batch_size):
      random_index = random.randint(0, len(image_paths) - 1)
      if istraining:
        im, steering = random_augment(image_paths[random_index],
steering_ang[random_index])
      else:
        im = mpimg.imread(image_paths[random_index])
        steering = steering_ang[random_index]
      im = img_preprocess(im)
      batch_img.append(im)
      batch_steering.append(steering)
    yield (np.asarray(batch_img), np.asarray(batch_steering))
```

The `**batch_generator**` function is designed to generate batches of images and their corresponding steering angles for training a neural network. This is particularly useful for training deep learning models where data is fed into the model in batches. Here's a detailed explanation of how the function works:

Summary

- **Data Generation:** Continuously generates batches of images and steering angles.
- **Augmentation:** Applies random augmentations during training to improve model robustness.
- **Preprocessing:** Ensures images are preprocessed consistently before being fed into the model.
- **Yielding Batches:** Efficiently yields batches of data in a format suitable for training neural networks.

This setup is typical for training deep learning models where data is large and needs to be processed in manageable chunks.

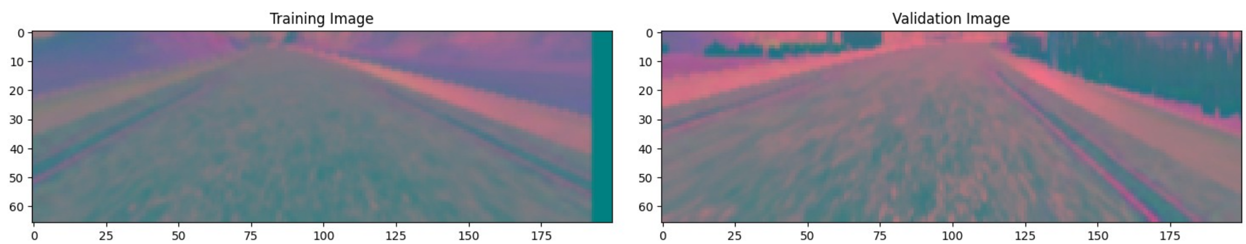
```
x_train_gen, y_train_gen = next(batch_generator(X_train, y_train, 1,
1))
x_valid_gen, y_valid_gen = next(batch_generator(X_valid, y_valid, 1,
0))
```

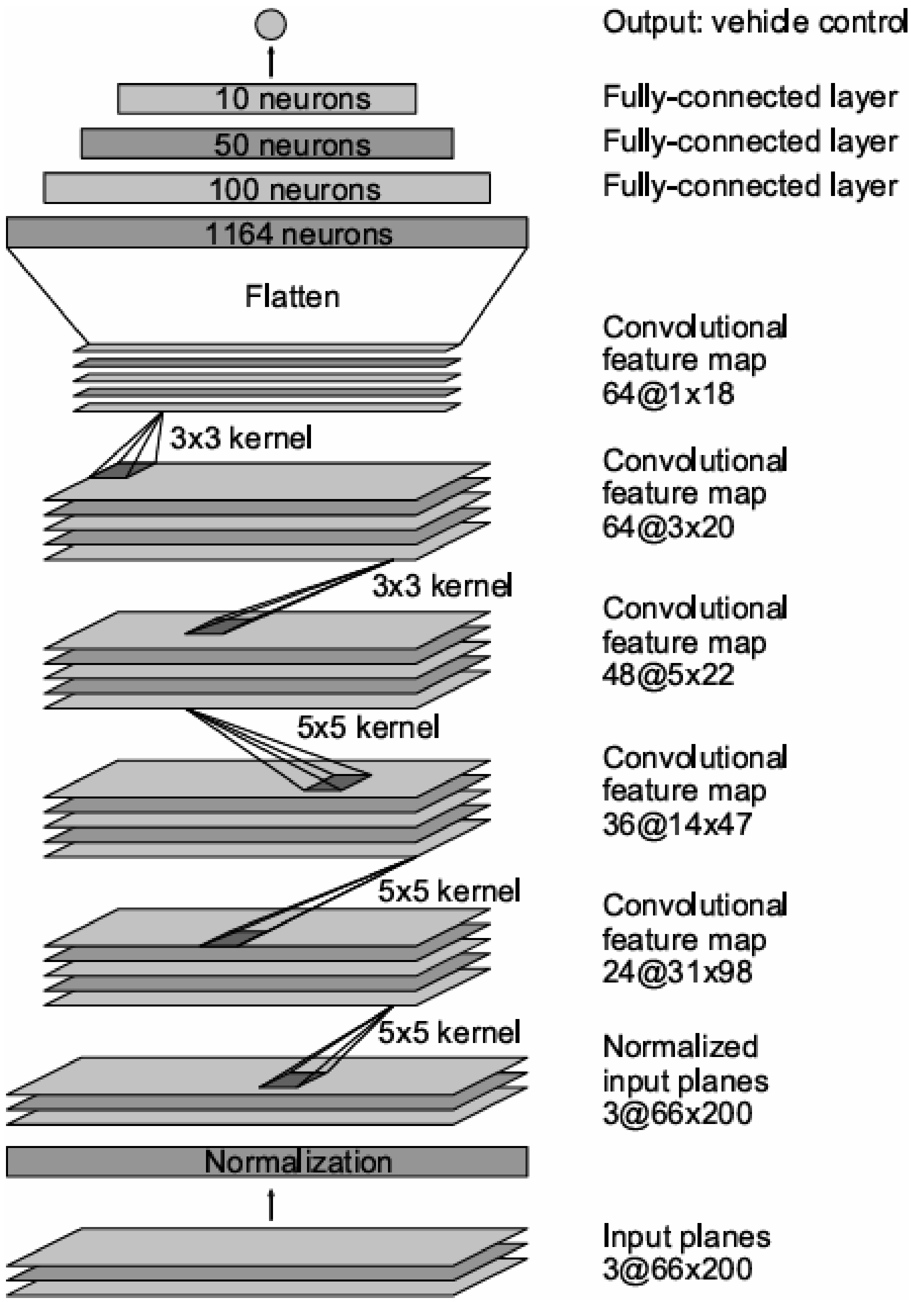
- **Training Data:** `x_train_gen` and `y_train_gen` contain a single batch of training data. Since the batch size is 1, this will be one image and one steering angle, both preprocessed and augmented.
- **Validation Data:** `x_valid_gen` and `y_valid_gen` contain a single batch of validation data. The data is preprocessed but not augmented.

This approach allows you to inspect and verify the data produced by your `batch_generator` function, ensuring that it is correctly preprocessed and augmented. For debugging or visualization purposes, retrieving and inspecting a single batch can be very helpful.

```
fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(x_train_gen[0])
axs[0].set_title('Training Image')
axs[1].imshow(x_valid_gen[0])
axs[1].set_title('Validation Image')
```

```
Text(0.5, 1.0, 'Validation Image')
```





Defining Nvidia Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten
from tensorflow.keras.optimizers import Adam
from keras.losses import mean_squared_error

def nvidia_model():
    model = Sequential()
```

```
    model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200,
3), activation='elu'))
    model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='elu'))
    model.add(Conv2D(48, (5, 5), strides=(2, 2), activation='elu'))
    #model.add(Conv2D(64, (3, 3), activation='elu'))
    #model.add(Conv2D(64, (3, 3), activation='elu'))


    model.add(Flatten())

    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=1e-3)
    model.compile(loss='mse', optimizer=optimizer)
    return model

model = nvidia_model()
print(model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 31, 98, 24)          │
1,824 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 14, 47, 36)          │
21,636 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 5, 22, 48)           │
43,248 │
├──────────────────────────────────────┼─────────────────────────────┼
─────────────────┤
```

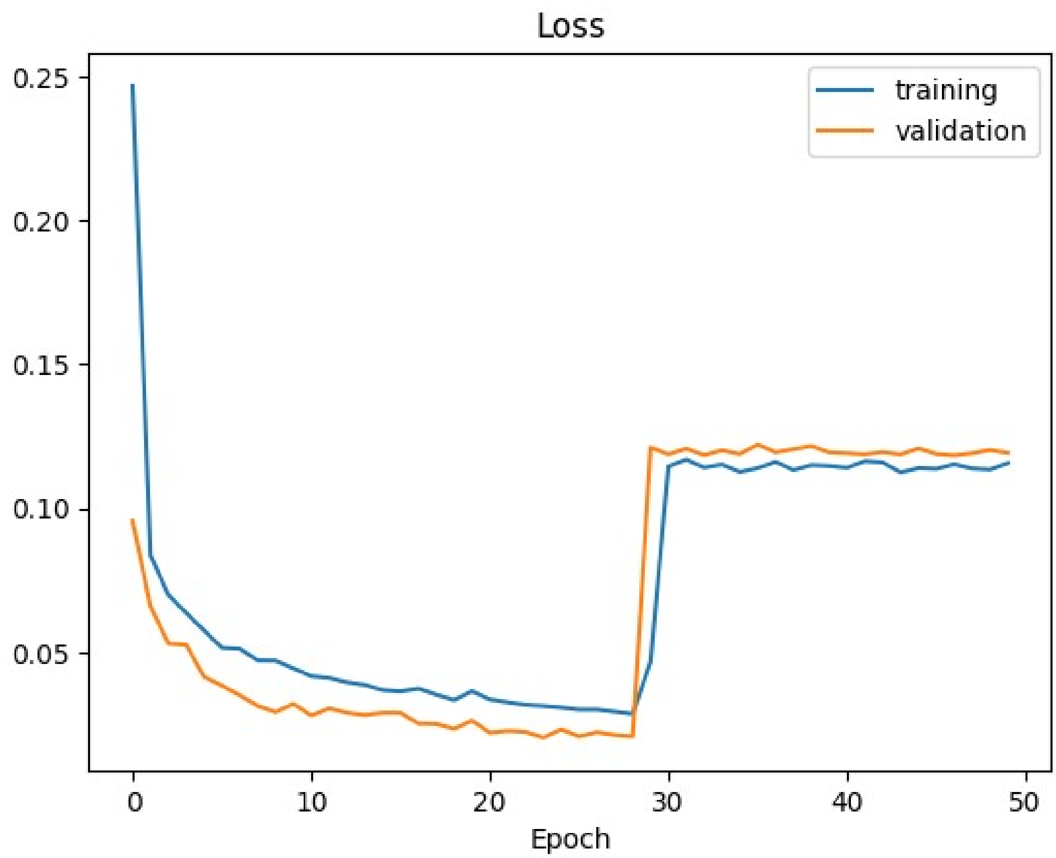
```
│ flatten (Flatten)                      │ (None, 5280)                 │
0 │
├────────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense (Dense)                          │ (None, 100)                  │
528,100 │
├────────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_1 (Dense)                        │ (None, 50)                   │
5,050 │
├────────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_2 (Dense)                        │ (None, 10)                   │
510 │
├────────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense_3 (Dense)                        │ (None, 1)                    │
11 │
└────────────────────────────────────────┴──────────────────────────────┴
─────────────────┘

 Total params: 600,379 (2.29 MB)

 Trainable params: 600,379 (2.29 MB)

 Non-trainable params: 0 (0.00 B)

None

history = model.fit(
    batch_generator(X_train, y_train, 100, 1),
    steps_per_epoch=300,
    epochs=5,
    validation_data=batch_generator(X_valid, y_valid, 100, 0),
    validation_steps=200,
    verbose=1,
    shuffle=True
)

Epoch 1/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 327s 1s/step - loss: 0.8145 - val_loss:
0.0955
Epoch 2/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 337s 1s/step - loss: 0.0873 - val_loss:
0.0658
Epoch 3/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 326s 1s/step - loss: 0.0735 - val_loss:
0.0530
Epoch 4/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 328s 1s/step - loss: 0.0646 - val_loss:
0.0525
```

```
Epoch 5/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 325s 1s/step - loss: 0.0584 - val_loss:
0.0414
Epoch 6/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 326s 1s/step - loss: 0.0532 - val_loss:
0.0383
Epoch 7/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 331s 1s/step - loss: 0.0520 - val_loss:
0.0350
Epoch 8/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.0478 - val_loss:
0.0313
Epoch 9/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.0481 - val_loss:
0.0291
Epoch 10/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.0444 - val_loss:
0.0320
Epoch 11/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 319s 1s/step - loss: 0.0419 - val_loss:
0.0280
Epoch 12/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - loss: 0.0423 - val_loss:
0.0304
Epoch 13/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 328s 1s/step - loss: 0.0390 - val_loss:
0.0289
Epoch 14/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 319s 1s/step - loss: 0.0386 - val_loss:
0.0281
Epoch 15/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 322s 1s/step - loss: 0.0361 - val_loss:
0.0289
Epoch 16/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.0379 - val_loss:
0.0289
Epoch 17/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 326s 1s/step - loss: 0.0365 - val_loss:
0.0252
Epoch 18/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.0353 - val_loss:
0.0250
Epoch 19/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.0333 - val_loss:
0.0233
Epoch 20/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 325s 1s/step - loss: 0.0353 - val_loss:
0.0262
Epoch 21/50
```

```
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.0344 - val_loss:
0.0219
Epoch 22/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - loss: 0.0327 - val_loss:
0.0226
Epoch 23/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 320s 1s/step - loss: 0.0317 - val_loss:
0.0222
Epoch 24/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - loss: 0.0311 - val_loss:
0.0203
Epoch 25/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 326s 1s/step - loss: 0.0307 - val_loss:
0.0231
Epoch 26/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 327s 1s/step - loss: 0.0304 - val_loss:
0.0206
Epoch 27/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - loss: 0.0299 - val_loss:
0.0222
Epoch 28/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 325s 1s/step - loss: 0.0310 - val_loss:
0.0211
Epoch 29/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 325s 1s/step - loss: 0.0281 - val_loss:
0.0207
Epoch 30/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.0300 - val_loss:
0.1211
Epoch 31/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 327s 1s/step - loss: 0.1141 - val_loss:
0.1187
Epoch 32/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 325s 1s/step - loss: 0.1142 - val_loss:
0.1206
Epoch 33/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 325s 1s/step - loss: 0.1155 - val_loss:
0.1184
Epoch 34/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.1149 - val_loss:
0.1201
Epoch 35/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.1117 - val_loss:
0.1188
Epoch 36/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 326s 1s/step - loss: 0.1137 - val_loss:
0.1221
Epoch 37/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.1175 - val_loss:
```

```
0.1194
Epoch 38/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.1125 - val_loss:
0.1205
Epoch 39/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.1135 - val_loss:
0.1216
Epoch 40/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - loss: 0.1145 - val_loss:
0.1194
Epoch 41/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 322s 1s/step - loss: 0.1137 - val_loss:
0.1191
Epoch 42/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.1143 - val_loss:
0.1187
Epoch 43/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 324s 1s/step - loss: 0.1170 - val_loss:
0.1195
Epoch 44/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 322s 1s/step - loss: 0.1113 - val_loss:
0.1186
Epoch 45/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 322s 1s/step - loss: 0.1146 - val_loss:
0.1208
Epoch 46/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - loss: 0.1141 - val_loss:
0.1188
Epoch 47/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 321s 1s/step - loss: 0.1160 - val_loss:
0.1183
Epoch 48/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 323s 1s/step - loss: 0.1148 - val_loss:
0.1190
Epoch 49/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 320s 1s/step - loss: 0.1141 - val_loss:
0.1202
Epoch 50/50
300/300 ━━━━━━━━━━━━━━━━━━━━ 319s 1s/step - loss: 0.1171 - val_loss:
0.1192


model.save('model.h5')
from google.colab import files
files.download('model.h5')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('Loss')
plt.xlabel('Epoch')
```

```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

Text(0.5, 0, 'Epoch')
```




```
model.save('model.h5')
from google.colab import files
files.download('model.h5')

WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```