This Notebook shows the EDA performed on CIFAR-10 dataset

1. Load dependencies

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np


# ###############################
# from tensorflow.keras.applications import ResNet50
# from tensorflow.keras.layers import Dense, Flatten, Input
# from tensorflow.keras.models import Model
# from tensorflow.keras.utils import to_categorical
# from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

2. Load the CIFAR-10 dataset

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
```

Split Training Set into Training & Validation The original dataset has:

50,000 training images 10,000 test images We'll use 80% for training and 20% for validation.

```
# Define split ratio
validation_ratio = 0.2
# Compute validation set size
num_train = int(train_images.shape[0] * (1 - validation_ratio))
# Split training data into train & validation sets
train_images, val_images = train_images[:num_train], train_images[num_train:]
train_labels, val_labels = train_labels[:num_train], train_labels[num_train:]
```

Class names in CIFAR-10

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'horse', 'ship', 'truck']
```

3. Exploratory Data Analysis (EDA) on CIFAR10

3.1. Shape of the Data

```
print("Training data shape:",train_images.shape)
print("Training labels shape:", train_labels.shape)
print("Validation data shape", val_images.shape)
print("Validation labels shape", val_labels.shape)
print("Testing data shape:", test_images.shape)
print("Testing labels shape:", test_labels.shape)
```

```
Training data shape: (40000, 32, 32, 3)
Training labels shape: (40000, 1)
Validation data shape (10000, 32, 32, 3)
Validation labels shape (10000, 1)
Testing data shape: (10000, 32, 32, 3)
Testing labels shape: (10000, 1)
```

3.2. Visualize Images

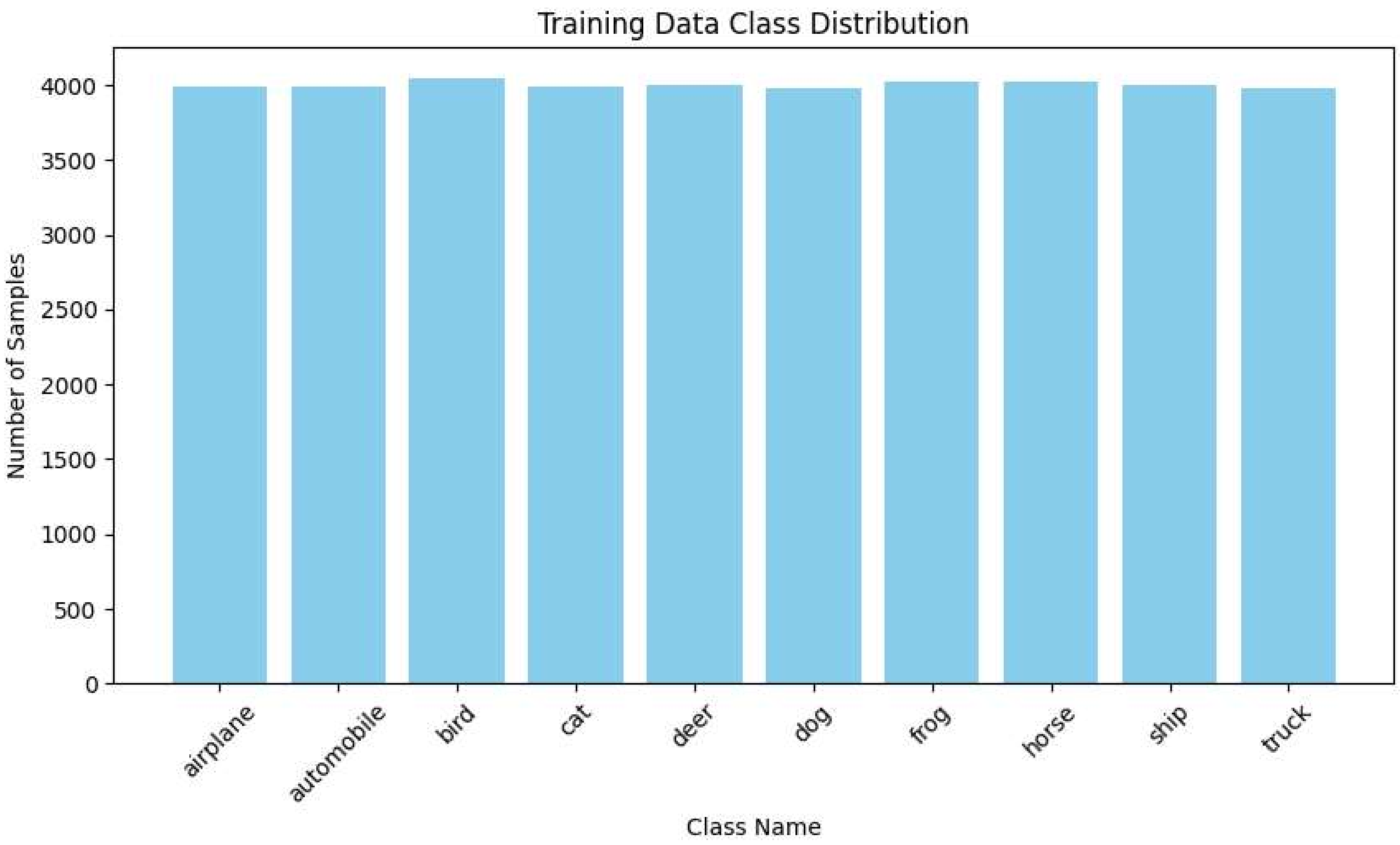
```
plt.figure(figsize=(10,10))
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

frog
truck
truck
deer
automobile
automobile

3.3 Class Distribution

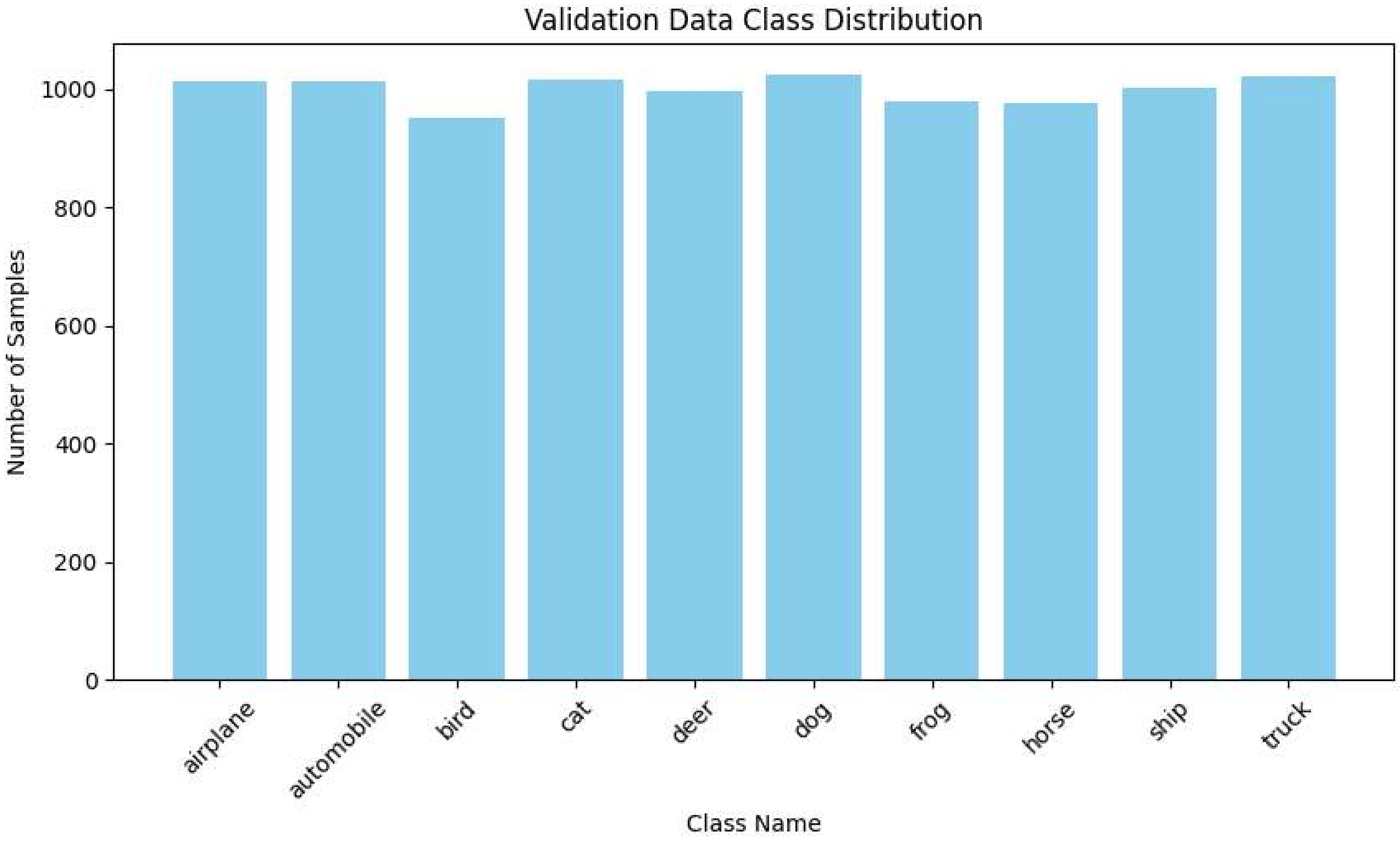
Check the class the distribution the training and test data

```
# Count occurrences of each class
unique_classes, class_counts = np.unique(train_labels, return_counts=True)
# Plot using Matplotlib (much faster than Seaborn)
plt.figure(figsize=(10, 5))
plt.bar(class_names, class_counts, color='skyblue')
plt.xlabel("Class Name")
plt.ylabel("Number of Samples")
plt.title("Training Data Class Distribution")
plt.xticks(rotation=45)
plt.show()
```



Check the class the distribution of the Validation data

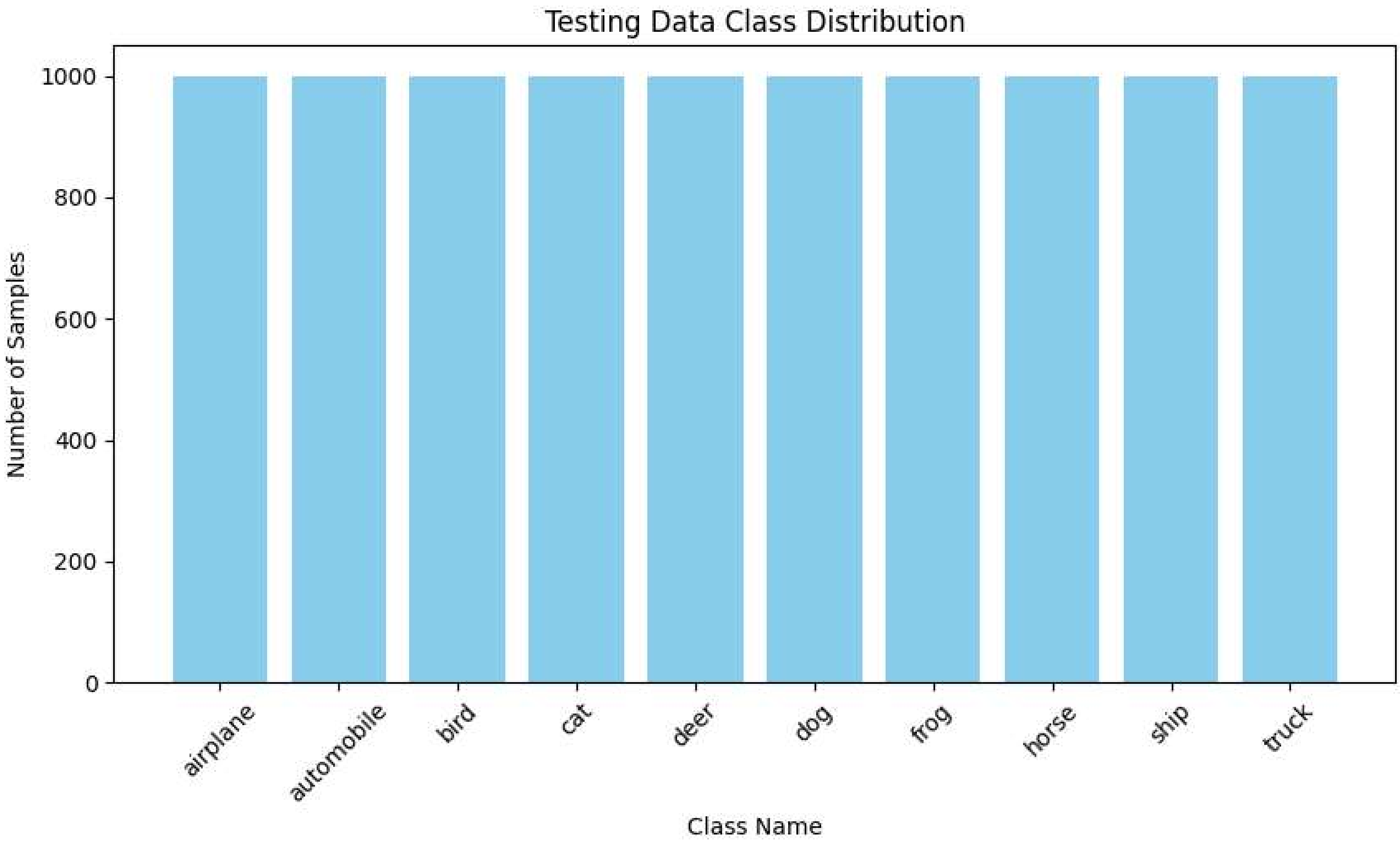
```
# Count occurrences of each class
unique_classes, class_counts = np.unique(val_labels, return_counts=True)
# Plot using Matplotlib (much faster than Seaborn)
plt.figure(figsize=(10, 5))
plt.bar(class_names, class_counts, color='skyblue')
plt.xlabel("Class Name")
plt.ylabel("Number of Samples")
plt.title("Validation Data Class Distribution")
plt.xticks(rotation=45)
plt.show()
```



Check the class the distribution of the Test data

```
# Count occurrences of each class
unique_classes, class_counts = np.unique(test_labels, return_counts=True)
```

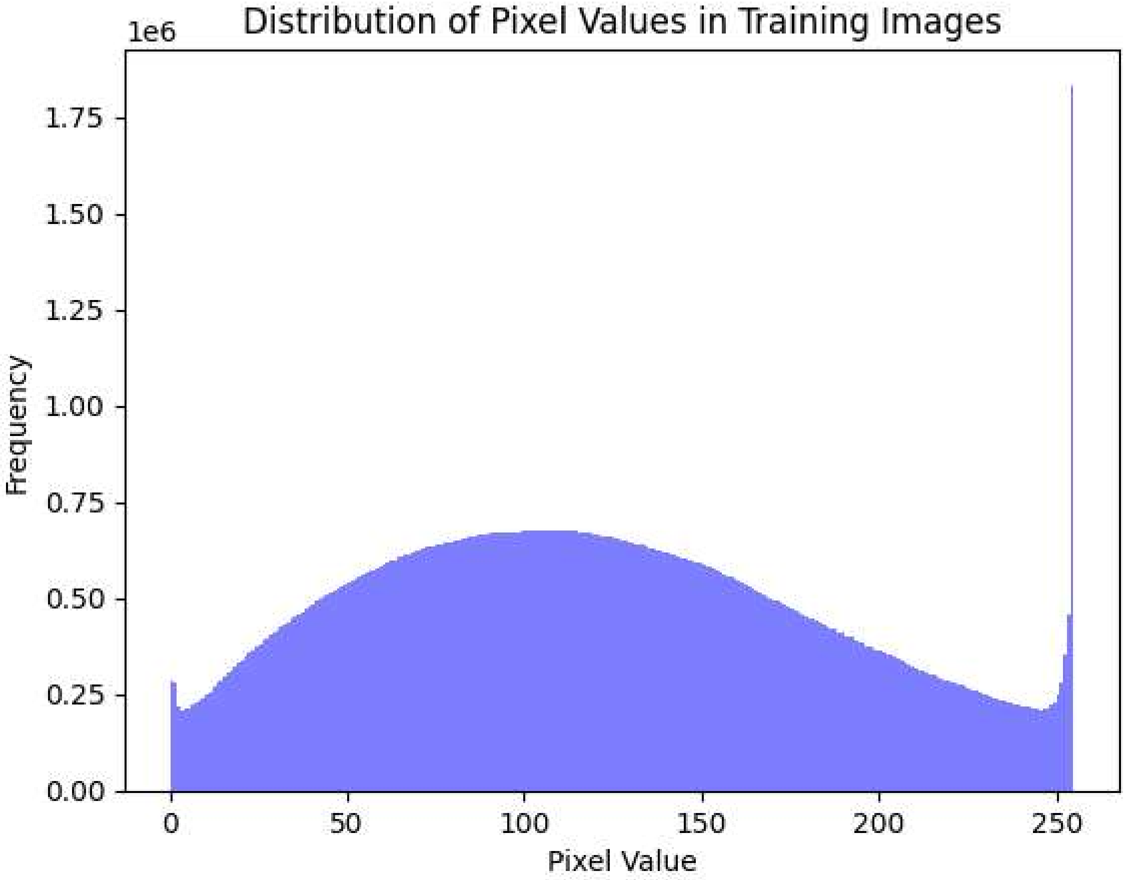
```
# Plot using Matplotlib (much faster than Seaborn)
plt.figure(figsize=(10, 5))
plt.bar(class_names, class_counts, color='skyblue')
plt.xlabel("Class Name")
plt.ylabel("Number of Samples")
plt.title("Testing Data Class Distribution")
plt.xticks(rotation=45)
plt.show()
```



3.4. Pixel Value Distribution

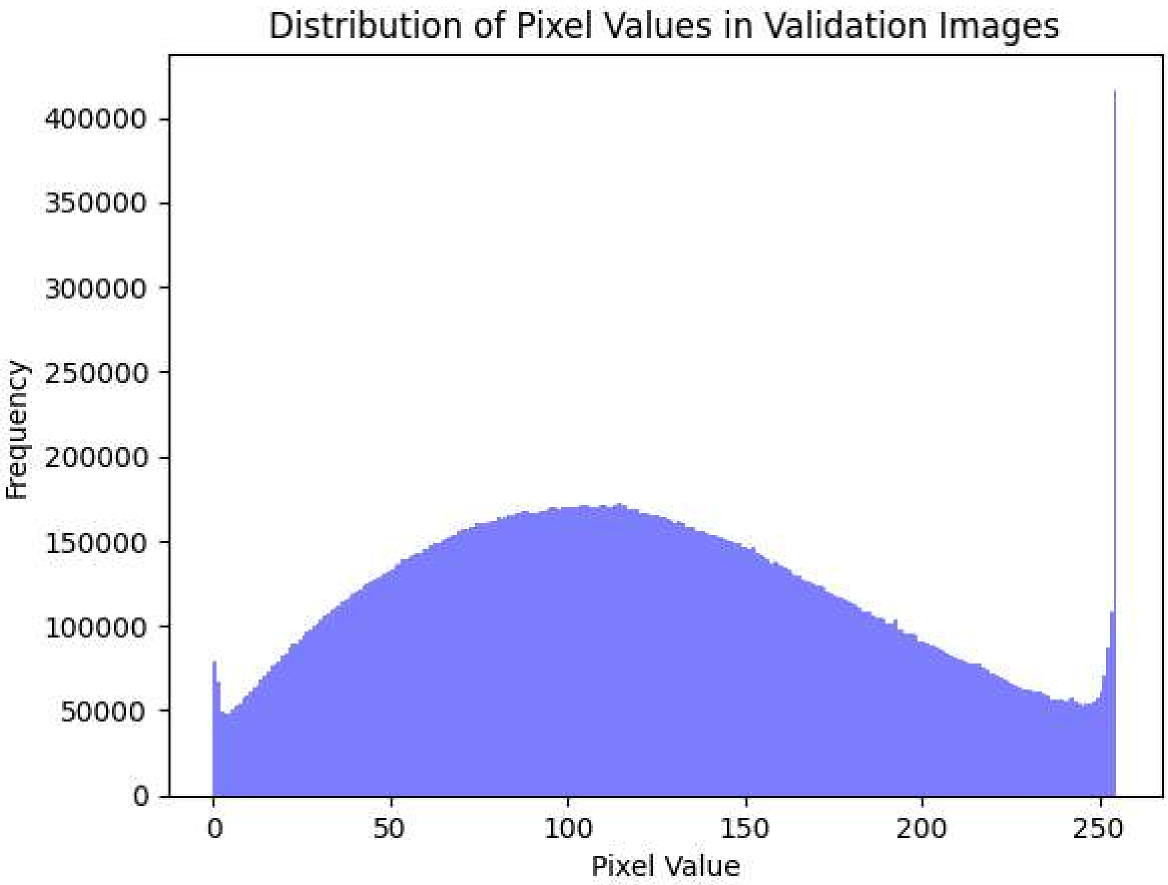
For training data

```
plt.hist(train_images.flatten(), bins=255, color='blue', alpha=0.5)
plt.title("Distribution of Pixel Values in Training Images")
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.show()
```



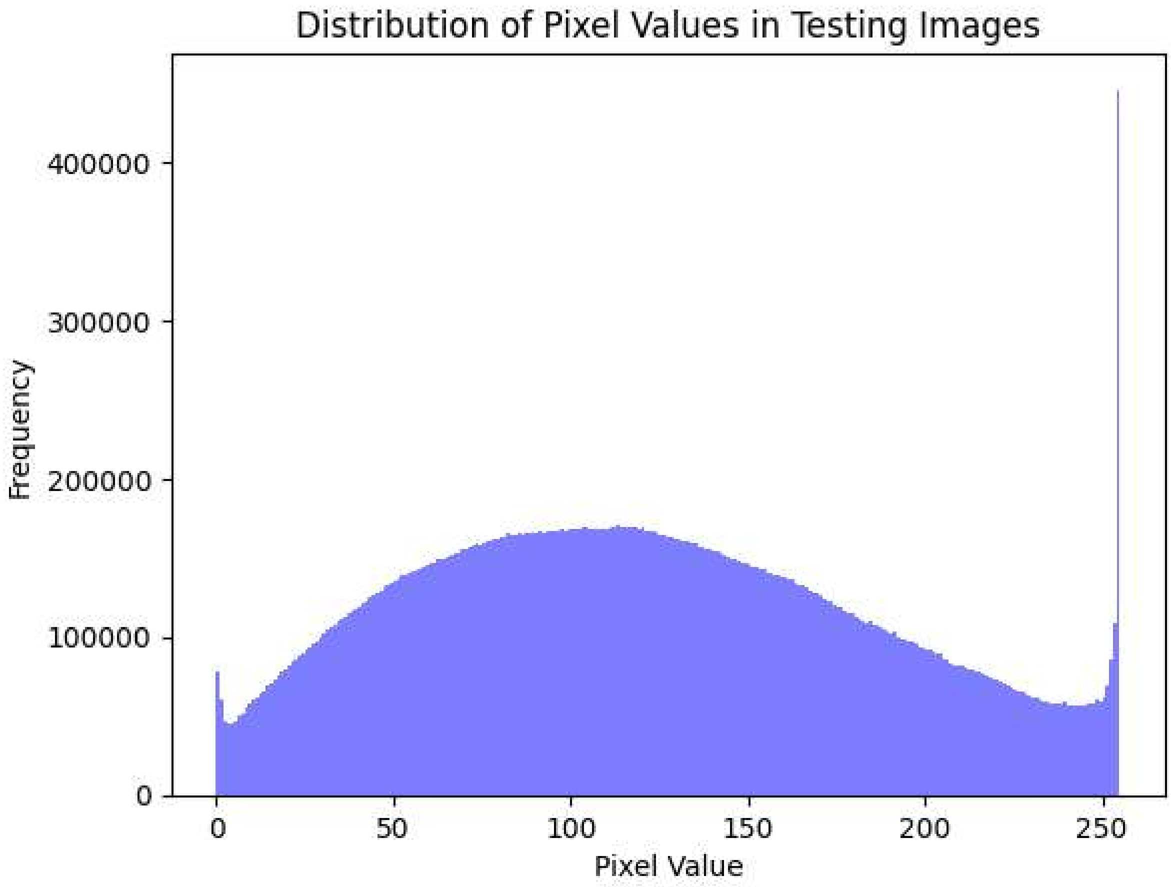
For Validation data

```
plt.hist(val_images.flatten(), bins=255, color='blue', alpha=0.5)
plt.title("Distribution of Pixel Values in Validation Images")
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.show()
```



For testing data

```
plt.hist(test_images.flatten(), bins=255, color='blue', alpha=0.5)
plt.title("Distribution of Pixel Values in Testing Images")
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.show()
```



3.5. Check Image Size and Aspect Ratio

```
# Function to get unique image shapes
def get_unique_shapes(images):
    image_shapes = np.array([img.shape for img in images])
    return np.unique(image_shapes, axis=0)
```

```
# Function to plot aspect ratio distribution
def plot_aspect_ratios(images, title):
    aspect_ratios = [img.shape[1] / img.shape[0] for img in images]
    plt.figure(figsize=(8, 5))
    sns.histplot(aspect_ratios, bins=20, kde=True)
    plt.xlabel("Aspect Ratio (Width/Height)")
    plt.ylabel("Frequency")
    plt.title(title)
    plt.show()
```

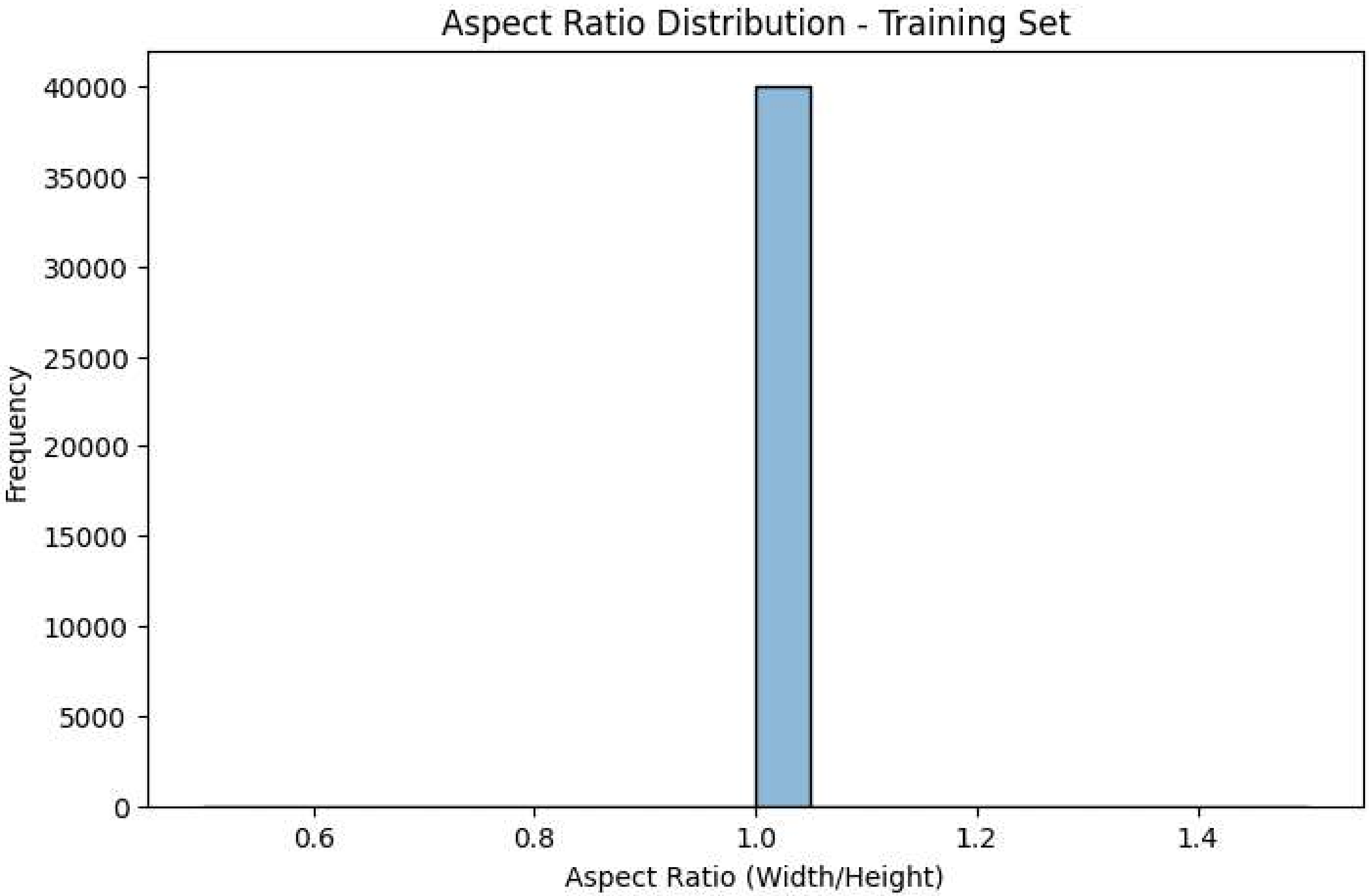
Check unique image shapes

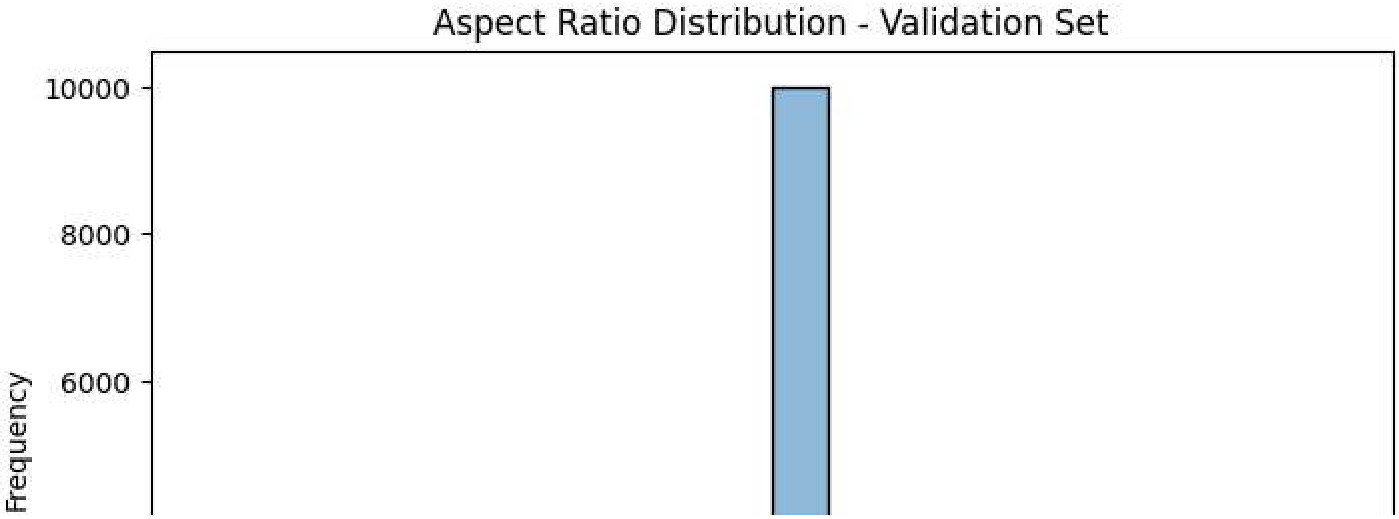
```
print("Unique Image Shapes in Training Set:", get_unique_shapes(train_images))
print("Unique Image Shapes in Validation Set:", get_unique_shapes(val_images))
print("Unique Image Shapes in Testing Set:", get_unique_shapes(test_images))
```

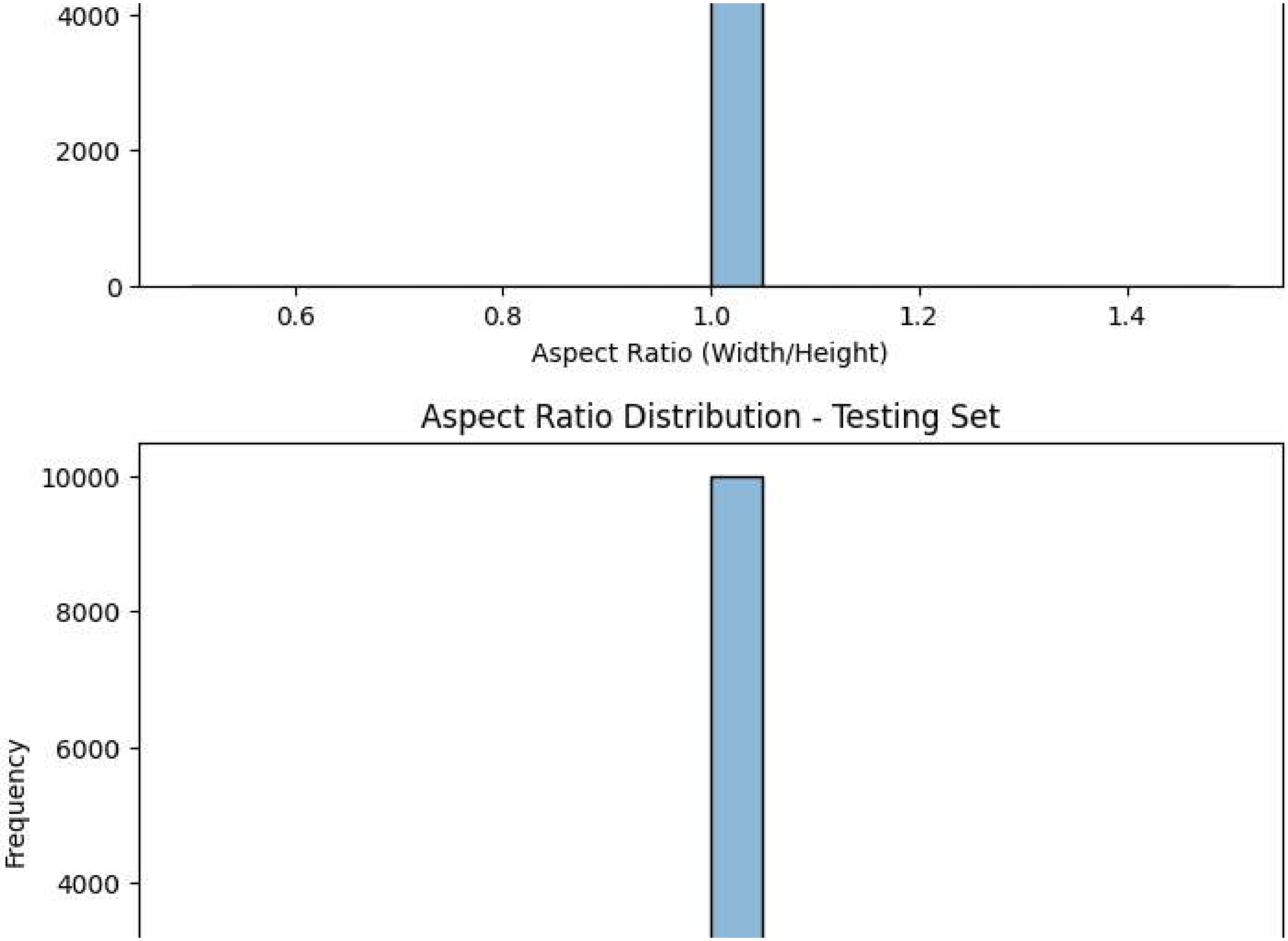
```
Unique Image Shapes in Training Set: [[32 32  3]]
Unique Image Shapes in Validation Set: [[32 32  3]]
Unique Image Shapes in Testing Set: [[32 32  3]]
```

Plot aspect ratio distributions

```
plot_aspect_ratios(train_images, "Aspect Ratio Distribution - Training Set")
plot_aspect_ratios(val_images, "Aspect Ratio Distribution - Validation Set")
plot_aspect_ratios(test_images, "Aspect Ratio Distribution - Testing Set")
```


Aspect Ratio Distribution - Training Set
40000
35000
30000
25000
20000
15000
10000
5000
0
Frequency
0.6
0.8
1.0
1.2
1.4
Aspect Ratio (Width/Height)


Aspect Ratio Distribution - Validation Set
10000
8000
6000
Frequency


Aspect Ratio Distribution - Testing Set