oqxafip01

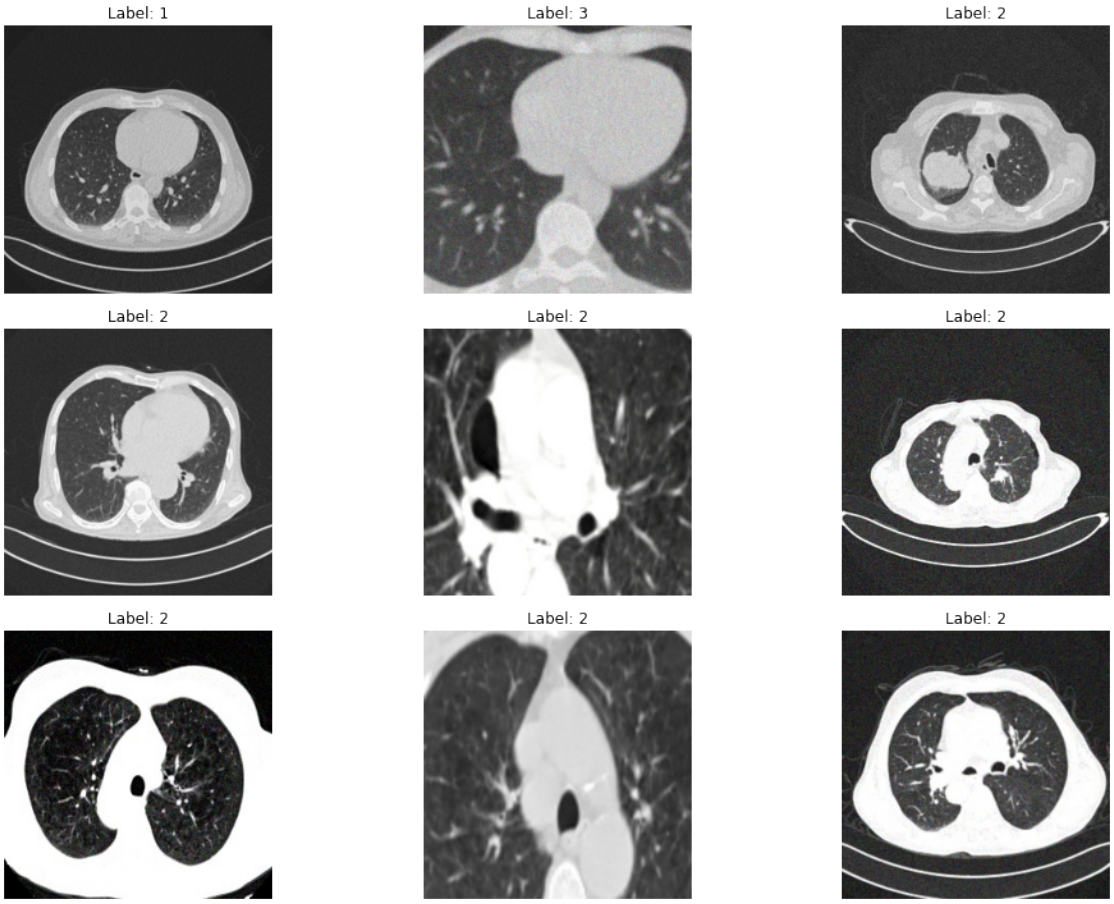
October 13, 2024

[17]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from skimage import measure
import numpy as np
import pandas as pd
import os
```

[6]:
```
# Specify the path to your train folder containing TFRecord files(Change folder␣
 ↪path based on where dataset folder is stored)
train_folder = r"C:
 ↪\Users\nikhi\Desktop\archive\dataset\drive\MyDrive\lungcancer\data\tfrecords\train"

# Get all TFRecord files in the train folder
tfrecord_files = [os.path.join(train_folder, f) for f in os.
 ↪listdir(train_folder) if f.endswith('.tfrecord')]

# Create a dataset from multiple TFRecord files
dataset = tf.data.TFRecordDataset(tfrecord_files)

# Define a function to parse your TFRecord examples
def parse_tfrecord_fn(example):
    feature_description = {
        'image': tf.io.FixedLenFeature([], tf.string),
        'label': tf.io.FixedLenFeature([], tf.int64)
    }
    parsed_example = tf.io.parse_single_example(example, feature_description)
    image = tf.io.decode_jpeg(parsed_example['image'], channels=3)
    label = parsed_example['label']
    return image, label

# Apply the parsing function to the dataset
parsed_dataset = dataset.map(parse_tfrecord_fn)

parsed_dataset
```

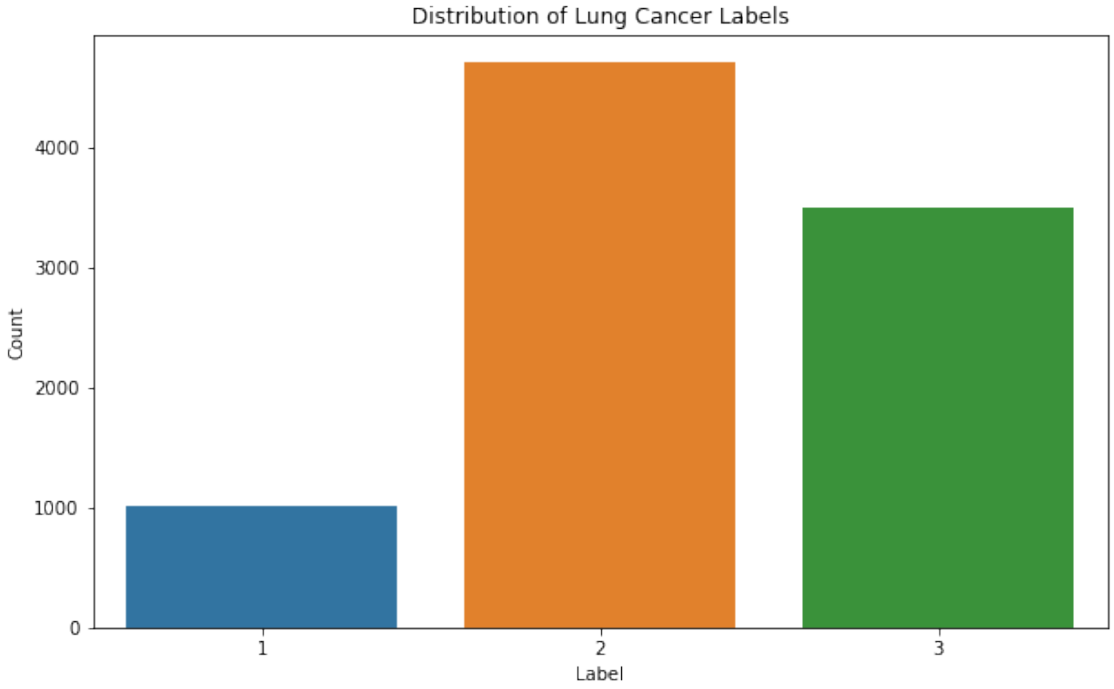
[6]:
```
<_MapDataset element_spec=(TensorSpec(shape=(None, None, 3), dtype=tf.uint8,
name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>
```

[7]:
```
plt.figure(figsize=(15, 10))
for i, (image, label) in enumerate(parsed_dataset.take(9)):
    plt.subplot(3, 3, i+1)
    plt.imshow(image.numpy().astype('uint8'))
    plt.title(f'Label: {label.numpy()}')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

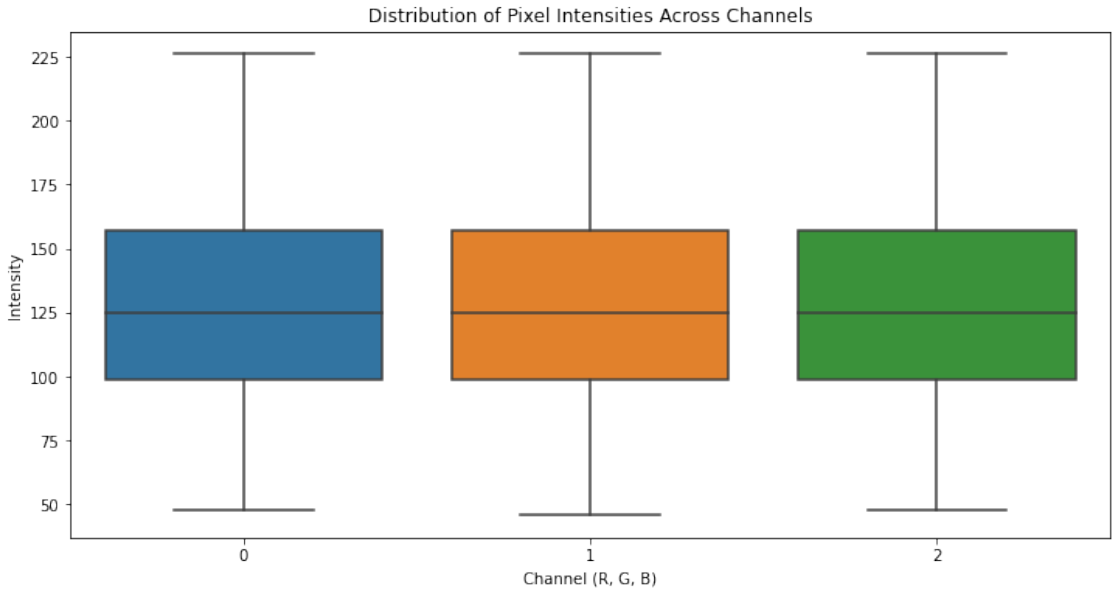


[8]:
```
labels = [label.numpy() for _, label in parsed_dataset]
label_counts = pd.Series(labels).value_counts().sort_index()

plt.figure(figsize=(10, 6))
sns.barplot(x=label_counts.index, y=label_counts.values)
plt.title('Distribution of Lung Cancer Labels')
plt.xlabel('Label')
```

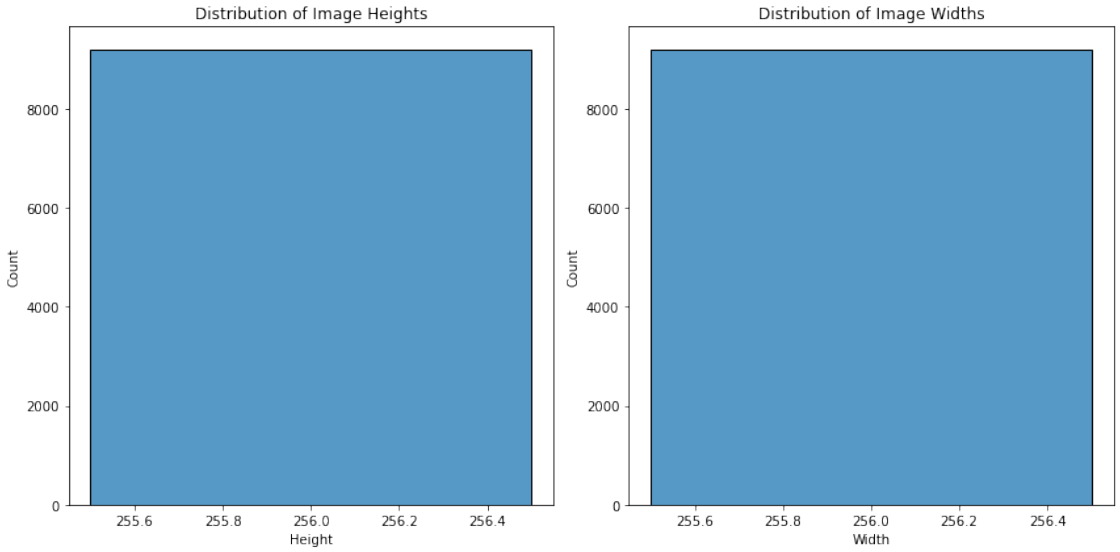
```
plt.ylabel('Count')
plt.show()
```



[9]:
```
def get_image_intensities(image):
    return tf.reduce_mean(image, axis=[0, 1])

intensities = [get_image_intensities(image).numpy() for image, _ in␣
 ↪parsed_dataset]
intensities = np.array(intensities)

plt.figure(figsize=(12, 6))
sns.boxplot(data=intensities)
plt.title('Distribution of Pixel Intensities Across Channels')
plt.xlabel('Channel (R, G, B)')
plt.ylabel('Intensity')
plt.show()
```

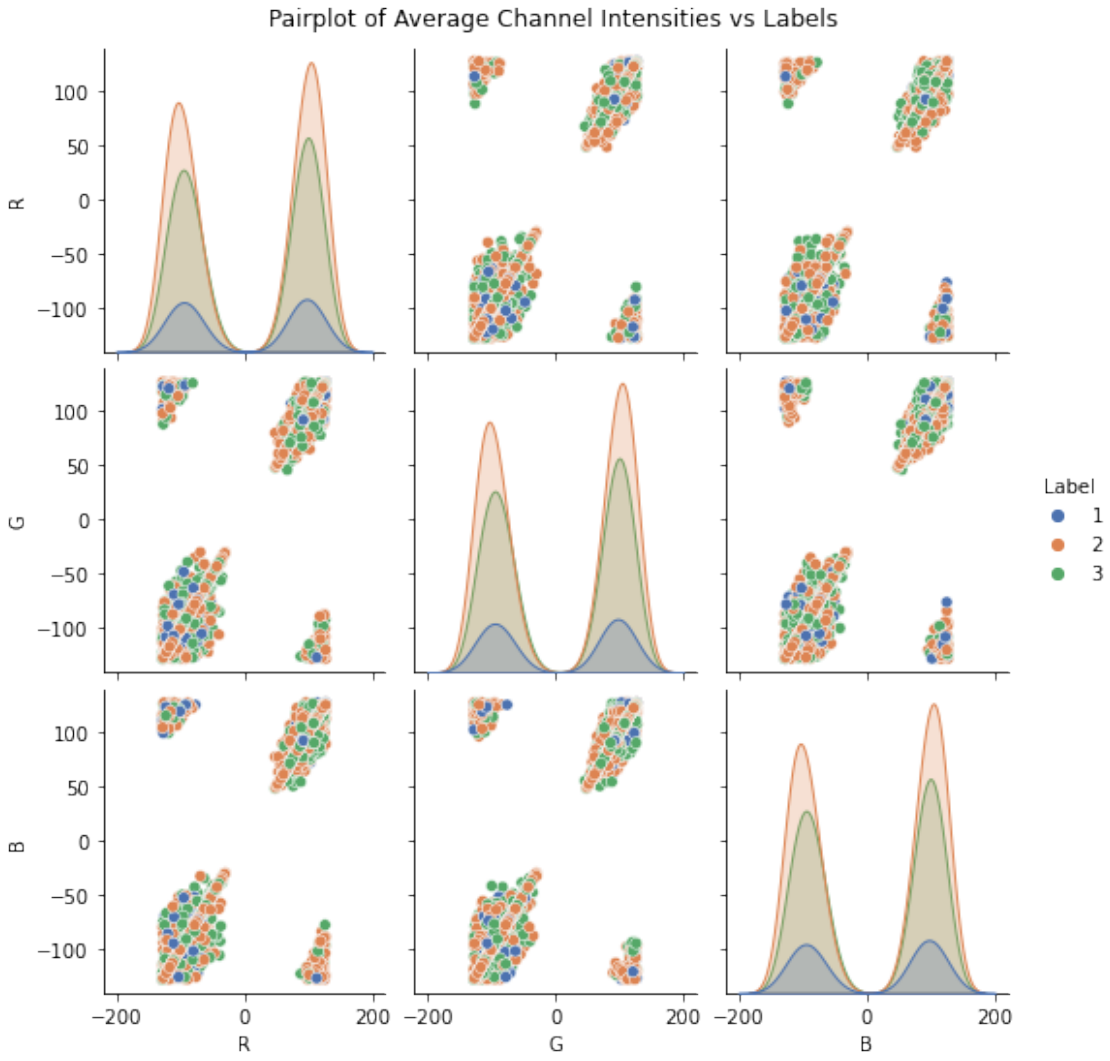


[10]:
```
image_sizes = [tf.shape(image).numpy()[:2] for image, _ in parsed_dataset]
heights, widths = zip(*image_sizes)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(heights)
plt.title('Distribution of Image Heights')
plt.xlabel('Height')

plt.subplot(1, 2, 2)
sns.histplot(widths)
plt.title('Distribution of Image Widths')
plt.xlabel('Width')

plt.tight_layout()
plt.show()
```

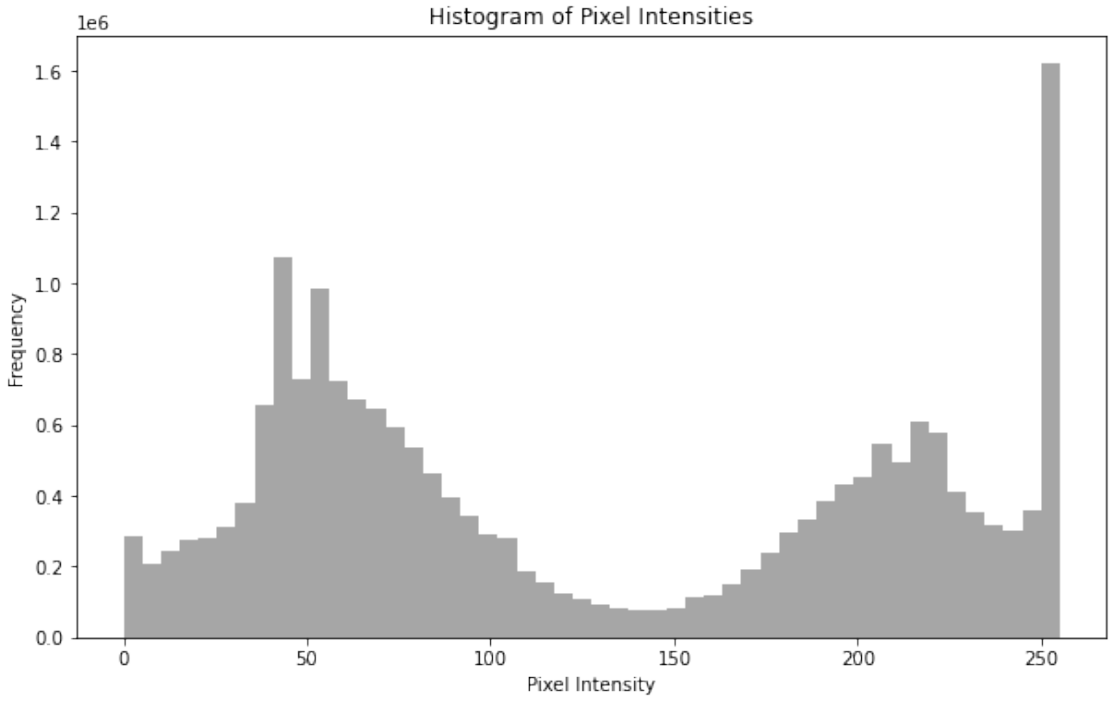


[12]:
```
features = [tf.reduce_mean(image, axis=[0, 1]).numpy() for image, _ in␣
 ↪parsed_dataset]
labels = [label.numpy() for _, label in parsed_dataset]

df = pd.DataFrame(features, columns=['R', 'G', 'B'])
df['Label'] = labels

plt.figure(figsize=(12, 10))
sns.pairplot(df, hue='Label', palette='deep')
plt.suptitle('Pairplot of Average Channel Intensities vs Labels', y=1.02)
plt.show()
```

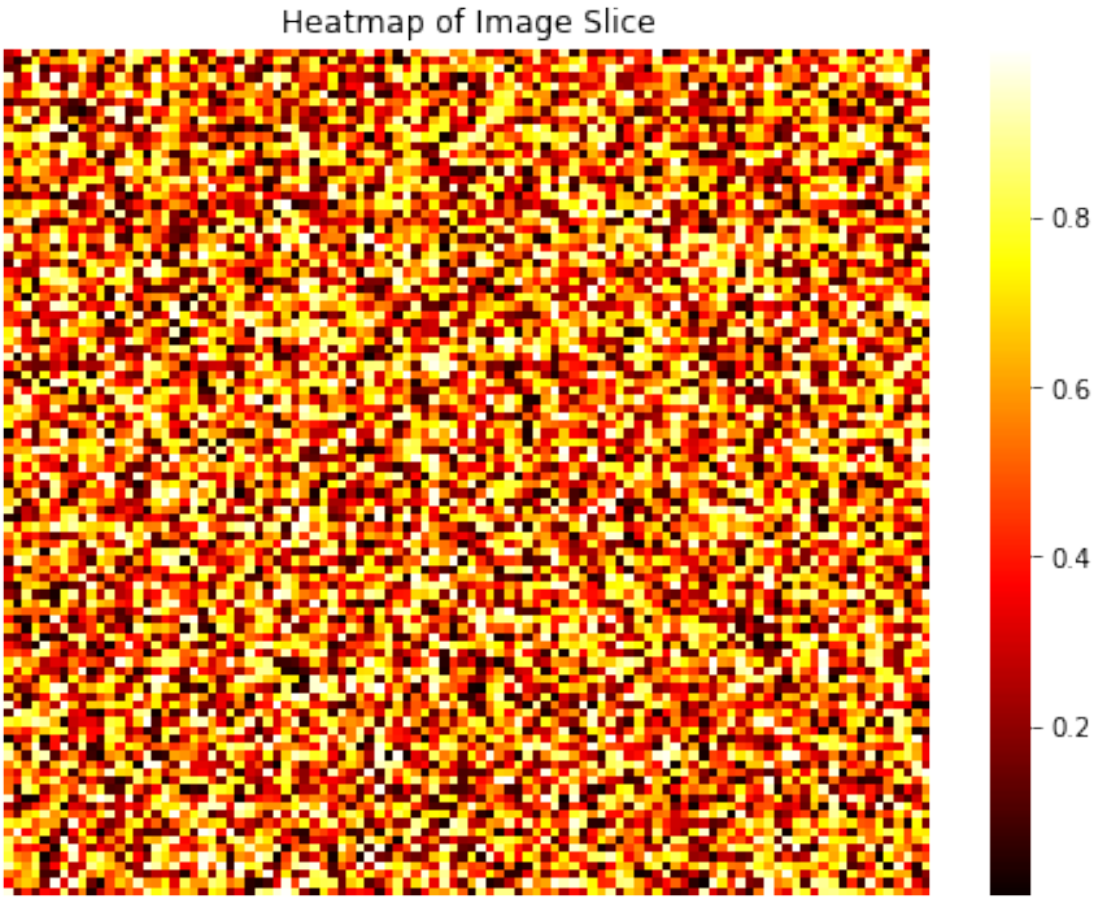
```
<Figure size 864x720 with 0 Axes>
```



[14]:
```python
# Function to plot histograms of pixel intensities
def plot_histogram(dataset):
    pixel_values = []
    for image, _ in dataset.take(100):  # Adjust the number of samples as needed
        pixel_values.extend(image.numpy().flatten())

    plt.figure(figsize=(10, 6))
    plt.hist(pixel_values, bins=50, color='gray', alpha=0.7)
    plt.title('Histogram of Pixel Intensities')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.show()

plot_histogram(parsed_dataset)
```



[15]:
```
# Assuming 'image' is a 2D numpy array representing a single slice from your dataset
def plot_heatmap(image):
    plt.figure(figsize=(8, 6))
    sns.heatmap(image, cmap='hot', cbar=True)
    plt.title('Heatmap of Image Slice')
    plt.axis('off')
    plt.show()

# Example usage with a random image slice
random_image_slice = np.random.rand(100, 100)  # Replace with an actual image slice
plot_heatmap(random_image_slice)
```


Heatmap of Image Slice
0.8
0.6
0.4
0.2