Traffic Signs Machine Learning Report - Research Report Part 2 of 2

Dataset:

https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification/data?select=labels.csv

Chris Heimbuch: https://github.com/chrisheimbuch

Table of Contents

Overview

In the last notebook, I explored some of the qualities of the dataset. I explored

image classes and dimensionality exploration. I observed the height, widths, aspect ratios, and RGB intensity distribution of the image dataset. The majority of images seem to have a height and width of roughly 140 pixels, with many falling outside of that range. Additionally, after observing the RGB intensity of the images, it appears red has the highest intensity among the images. As I delve deeper into the machine learning approach and neural network approach, I expect classic machine learning will not be accurate at identifying the images correctly and will have pretty bad accuracy. Neural networks and deep learning will most likely be employed here to identify meaningful heurstics and patterns within the data.

In [2]:

```
import nbformat

def generate_toc(notebook_path):
    with open(notebook_path) as f:
        nb = nbformat.read(f, as_version=4)

    toc = []
    for cell in nb.cells:
        if cell.cell_type == 'markdown':
            lines = cell.source.split('\n')
            for line in lines:
                if line.startswith('#'):
                    header_level = line.count('#')
                    header_text = line.replace('#', '').strip()
                    toc.append((header_level, header_text))

    toc_md = ['## Table of Contents']
    for level, text in toc:
        toc_md.append(f"{'    ' * (level - 1)}- [{text}](#{text.replace(' ', '
```

```
    return '\n'.join(toc_md)

notebook_path = 'traffic_ml_report.ipynb'
toc_md = generate_toc(notebook_path)
```

Section 1: Data Preprocessing

In [13]:

```
#Importing in libraries to work with the image data.
import os
import shutil
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
from PIL import Image as Image
import matplotlib.pyplot as plt
```

```
from matplotlib.lines import Line2D

#Classic ML
from skimage.feature import hog
from skimage import color, exposure
from sklearn.model_selection import train_test_split, StratifiedKFold, Repeate
from sklearn.metrics import classification_report, confusion_matrix, accuracy_
from sklearn.model_selection import RandomizedSearchCV

# Machine Learning Algorithms
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, SVR
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, Gradi
from sklearn.metrics import roc_curve, RocCurveDisplay, roc_auc_score

# TensorFlow and deep learning libraries
import tensorflow as tf
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping, ModelCheckpoint

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

I have 57 different classes which will be tough for the machine learning models to work with. I am going to aggregate the images into three main classes.

In [5]:

```
#Main directory of my p4 project
main_dir = r"C:\Users\Chris\Documents\Flatiron\Course Materials\Phase_4\P4_Pro

#new variable to combine the directory and combined word
combined = os.path.join(main_dir, "Combined")

#create new folder called "combined" which will combine my 57 classes in the D
os.makedirs(combined, exist_ok=True)
```

In [6]:

```
#create a new directory for the image classification in newly created combined
output_dir = r"C:\Users\Chris\Documents\Flatiron\Course Materials\Phase_4\P4_P

#new directories
speed_limit_signs = os.path.join(output_dir, 'Speed_Limit')
directional_signs = os.path.join(output_dir, 'Directional_Signs')
warning_signs = os.path.join(output_dir, 'Warning_Signs')

# Create new directories in combined folder, if they don't exist
os.makedirs(speed_limit_signs, exist_ok=True)
os.makedirs(directional_signs, exist_ok=True)
os.makedirs(warning_signs, exist_ok=True)
```

In [8]:
```
#load the excel sheet into a DataFrame
path = r'C:\Users\Chris\Documents\Flatiron\Course Materials\Phase_4\P4_Project
df = pd.read_csv(path)
```

In [9]:
```
#Inspect to make sure it loaded correctly
df.head()
```

Out[9]:

	ClassId	Name
0	0	Speed limit (5km/h)
1	1	Speed limit (15km/h)
2	2	Speed limit (30km/h)
3	3	Speed limit (40km/h)
4	4	Speed limit (50km/h)

In [7]:
```
#Redefining my input directory of training data for the 3 classes we have just

input_dir_train = r"C:\Users\Chris\Documents\Flatiron\Course Materials\Phase_4
```

Here I will clean up the training data. Currently all images are in 57 different folders for each class.

In [8]:
```
# Loop through each class
for index, row in df.iterrows():
    class_id = row['ClassId']
    class_name = row['Name'].lower()

    # Path to the current folder (e.g., 0, 1, etc.)
    class_folder = os.path.join(input_dir_train, str(class_id))

    if not os.path.exists(class_folder):
        print(f"Class folder {class_folder} not found.")
        continue

    # Skip classes with "unknown" in their name
    # if 'unknown' in class_name:
    #     target_folder = directional_signs
    #     # print(f"Skipping class {class_name} (ID: {class_id})")
    #     continue

    # Determine the target folder based on the class name
    if 'speed limit' in class_name:
        target_folder = speed_limit_signs
    elif 'unknown' in class_name:
        target_folder = directional_signs
    elif any(direction in class_name for direction in ['go', 'left', 'right',
        target_folder = directional_signs
    elif any(warning in class_name for warning in ['danger', 'crossing', 'cons
        target_folder = warning_signs
```

```
    else:
        # If the class doesn't match any category, continue
        print(f"Class {class_name} doesn't fit into a known category.")
        continue

    # Move images from the current folder to the target folder
    for image_file in os.listdir(class_folder):
        image_path = os.path.join(class_folder, image_file)
        if os.path.isfile(image_path):
            shutil.move(image_path, os.path.join(target_folder, image_file))

    print(f"Moved images from class {class_name} (ID: {class_id}) to {target_f
```

```
Moved images from class speed limit (5km/h) (ID: 0) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (15km/h) (ID: 1) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (30km/h) (ID: 2) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (40km/h) (ID: 3) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (50km/h) (ID: 4) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (60km/h) (ID: 5) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (70km/h) (ID: 6) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (80km/h) (ID: 7) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class dont go straight or left (ID: 8) to C:\Users\Chris\Docum
ents\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directio
nal_Signs
Moved images from class dont go straight or right (ID: 9) to C:\Users\Chris\Docu
ments\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directi
onal_Signs
Moved images from class dont go straight (ID: 10) to C:\Users\Chris\Documents\Fl
atiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Sig
ns
Moved images from class dont go left (ID: 11) to C:\Users\Chris\Documents\Flatir
on\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class dont go left or right (ID: 12) to C:\Users\Chris\Documen
ts\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directiona
l_Signs
Moved images from class dont go right (ID: 13) to C:\Users\Chris\Documents\Flati
ron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class dont overtake from left (ID: 14) to C:\Users\Chris\Docum
ents\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directio
nal_Signs
Moved images from class no uturn (ID: 15) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class no car (ID: 16) to C:\Users\Chris\Documents\Flatiron\Cou
rse Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class no horn (ID: 17) to C:\Users\Chris\Documents\Flatiron\Co
urse Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class speed limit (40km/h) (ID: 18) to C:\Users\Chris\Document
s\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class speed limit (50km/h) (ID: 19) to C:\Users\Chris\Document
s\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Speed_Limit
Moved images from class go straight or right (ID: 20) to C:\Users\Chris\Document
```

```
s\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional
_Signs
Moved images from class go straight (ID: 21) to C:\Users\Chris\Documents\Flatiro
n\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class go left (ID: 22) to C:\Users\Chris\Documents\Flatiron\Co
urse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class go left or right (ID: 23) to C:\Users\Chris\Documents\Fl
atiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Sig
ns
Moved images from class go right (ID: 24) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class keep left (ID: 25) to C:\Users\Chris\Documents\Flatiron
\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class keep right (ID: 26) to C:\Users\Chris\Documents\Flatiron
\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class roundabout mandatory (ID: 27) to C:\Users\Chris\Document
s\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional
_Signs
Moved images from class watch out for cars (ID: 28) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Sign
s
Moved images from class horn (ID: 29) to C:\Users\Chris\Documents\Flatiron\Cours
e Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class bicycles crossing (ID: 30) to C:\Users\Chris\Documents\F
latiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class uturn (ID: 31) to C:\Users\Chris\Documents\Flatiron\Cour
se Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class road divider (ID: 32) to C:\Users\Chris\Documents\Flatir
on\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class traffic signals (ID: 33) to C:\Users\Chris\Documents\Fla
tiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class danger ahead (ID: 34) to C:\Users\Chris\Documents\Flatir
on\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class zebra crossing (ID: 35) to C:\Users\Chris\Documents\Flat
iron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class bicycles crossing (ID: 36) to C:\Users\Chris\Documents\F
latiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class children crossing (ID: 37) to C:\Users\Chris\Documents\F
latiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class dangerous curve to the left (ID: 38) to C:\Users\Chris\D
ocuments\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Dire
ctional_Signs
Moved images from class dangerous curve to the right (ID: 39) to C:\Users\Chris
\Documents\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Di
rectional_Signs
Moved images from class unknown1 (ID: 40) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class unknown2 (ID: 41) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class unknown3 (ID: 42) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class go right or straight (ID: 43) to C:\Users\Chris\Document
s\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional
_Signs
Moved images from class go left or straight (ID: 44) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_
Signs
Moved images from class unknown4 (ID: 45) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class zigzag curve (ID: 46) to C:\Users\Chris\Documents\Flatir
```

```
on\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class train crossing (ID: 47) to C:\Users\Chris\Documents\Flat
iron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class under construction (ID: 48) to C:\Users\Chris\Documents
\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Sign
s
Moved images from class unknown5 (ID: 49) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class fences (ID: 50) to C:\Users\Chris\Documents\Flatiron\Cou
rse Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class heavy vehicle accidents (ID: 51) to C:\Users\Chris\Docum
ents\Flatiron\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_
Signs
Moved images from class unknown6 (ID: 52) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class give way (ID: 53) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class no stopping (ID: 54) to C:\Users\Chris\Documents\Flatiro
n\Course Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class no entry (ID: 55) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Warning_Signs
Moved images from class unknown7 (ID: 56) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
Moved images from class unknown8 (ID: 57) to C:\Users\Chris\Documents\Flatiron\C
ourse Materials\Phase_4\P4_Project\traffic_Data\Combined\Directional_Signs
```

After running this code, instead of having 57 different classes (or folders), now we have an organized structure of 3 classes, more balanced than before, with a better understanding of the sign classification (directional signs, speed limit signs, and warning signs!)

The Test images have a naming convention of 000, 001, 002 to signify which class it belongs to. However, to accurately test the models on these, we need to aggregate them into similar folders, exactly like the training data so we can test if our models are accurate or not!

In [10]:
```
#Going to prepare image data to work with. I will first handle the training da

#Setting variable to directory for my TEST images.
input_dir_test = r"C:\Users\Chris\Documents\Flatiron\Course Materials\Phase_4\
```

In [11]:
```
#new directories
speed_limit_signs = os.path.join(input_dir_test, 'Speed_Limit')
directional_signs = os.path.join(input_dir_test, 'Directional_Signs')
warning_signs = os.path.join(input_dir_test, 'Warning_Signs')

#Create new directories in combined folder, if they don't exist
os.makedirs(speed_limit_signs, exist_ok=True)
os.makedirs(directional_signs, exist_ok=True)
os.makedirs(warning_signs, exist_ok=True)
```

In [12]:
```
#mapping for TEST images.
sign_map = df.set_index('ClassId')['Name'].to_dict()
```

In [45]:
```
#Iterate through each file in the test directory
for file_name in os.listdir(input_dir_test):

        #Extract the class ID from the file name (the naming convention for th
        class_id = file_name[:3]

        #this will get the class from the dictionary and convert the first par
        class_name = sign_map.get(int(class_id))

        #determine which folder the test image belongs to based on classificat
        if 'speed limit' in class_name.lower():
            target_folder = speed_limit_signs
        elif 'unknown' in class_name.lower():
            target_folder = directional_signs
        elif any(direction in class_name.lower() for direction in ['go', 'left
            target_folder = directional_signs
        elif any(warning in class_name.lower() for warning in ['danger', 'cros
            target_folder = warning_signs
        else:
            continue

        #move images
        src_path = os.path.join(input_dir_test, file_name)
        dest_path = os.path.join(target_folder, file_name)
        shutil.move(src_path, dest_path)

print("Test images sorted successfully!")
```

```
Test images sorted successfully!
```

Section 2: Classic Machine Learning Approach

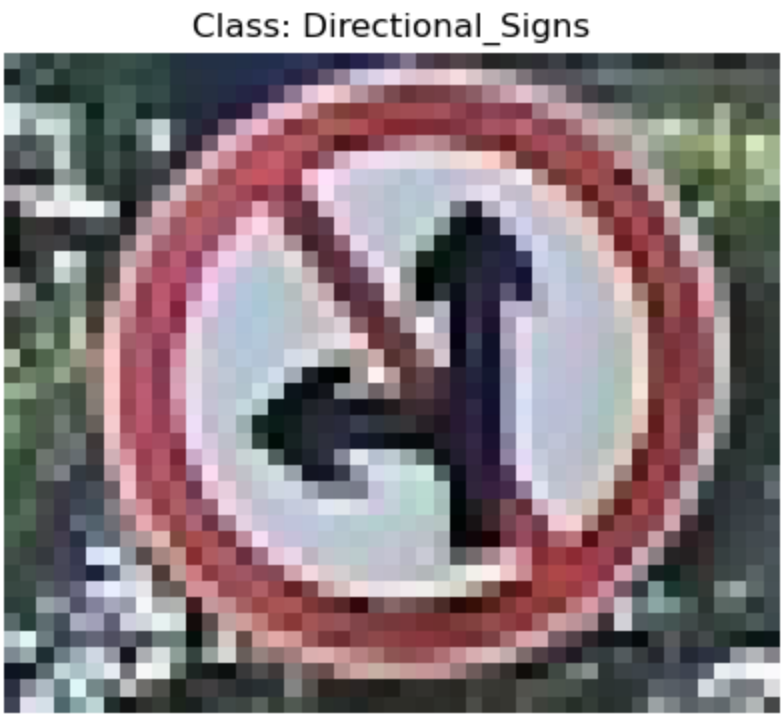
In [14]:
```
#Redefining my input directory of training data for the 3 classes we have just

input_dir_train = r"C:\Users\Chris\Documents\Flatiron\Course Materials\Phase_4
```

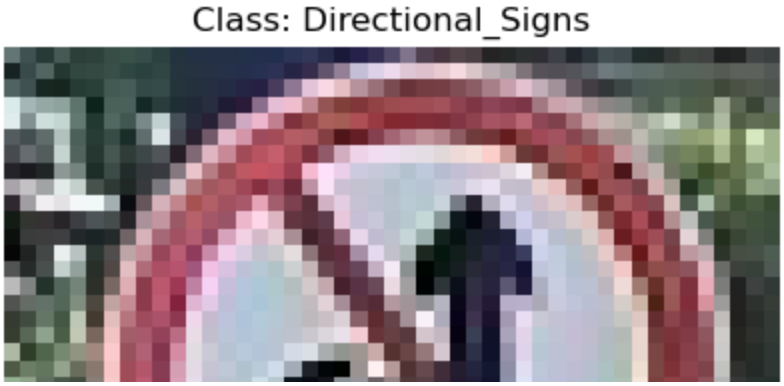
In [15]:
```
#Here I am extracting out the images from my combined folder, and adding them
#By default, cv2 stores image in BGR format, in which to display some images,

image_list = os.listdir(input_dir_train)
folders = ['Directional_Signs', 'Speed_Limit', 'Warning_Signs']

image = []
class_name = []

for folder_name in folders:
    folder_path = os.path.join(input_dir_train, folder_name)
```

```
    #list of images in newly created folders
    pictures = os.listdir(folder_path)

    for img in pictures:
        img_path = os.path.join(folder_path, img)
        current = cv2.imread(img_path)
        #Convert to RGB for plotting for matplotlib
        current_rgb = cv2.cvtColor(current, cv2.COLOR_BGR2RGB)

        image.append(current_rgb)
        class_name.append(folder_name)
```

In [16]:

```
#Inspect a few images in the image list

# Display one image from each folder
for i in range(3):
    plt.figure(figsize=(5, 5))
    plt.imshow(image[i])
    plt.title(f"Class: {class_name[i]}")
    plt.axis('off')
    plt.show()
```





In [17]:

```
#Going to prepare image data to work with. I will first handle the training da

image_data = []
labels = []
folders = ['Directional_Signs', 'Speed_Limit', 'Warning_Signs']

# Create a mapping from folder names to numeric labels
label_mapping = {'Directional_Signs': 0, 'Speed_Limit': 1, 'Warning_Signs': 2}

#Create a loop to go through each class folder
for folder_name in folders:
    folder_path = os.path.join(input_dir_train, folder_name)

    #Check to see if the folder is an actual directory.
    if os.path.isdir(folder_path):
        #here we will loop through each image in the given class folder (direc
        for image in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image)

            #read the image in color.
            img = cv2.imread(image_path, cv2.IMREAD_COLOR)
```

```
            #this will only proceed if the image exists
            if img is not None:
                #resize image to fixed size so its uniform ( as most images ha
                img_resized = cv2.resize(img, (32,32))

                #make image interpretable for machine models by flattening it
                img_flattened = img_resized.flatten()

                #add the flattened image and the label to the lists above that
                image_data.append(img_flattened)
                labels.append(label_mapping[folder_name])
            else:
                print(f"The image could not be read {image_path}")

#convert the lists into numpy arrays for efficiency
image_data = np.array(image_data)
labels = np.array(labels)
```

In [18]:

```
#Convert image data to a DataFrame
df = pd.DataFrame(image_data)

#Scale data so it is more managable for machine models.
df = df / 255

# Add the labels as the target column
df['label'] = labels

df
```

Out[18]:

	0	1	2	3	4	5	6	7
0	0.392157	0.392157	0.329412	0.254902	0.266667	0.196078	0.298039	0.317647
1	0.392157	0.392157	0.329412	0.254902	0.266667	0.196078	0.298039	0.317647
2	0.843137	0.874510	0.870588	0.811765	0.878431	0.858824	0.843137	0.886275
3	0.890196	0.913725	0.949020	0.909804	0.917647	0.964706	0.933333	0.933333
4	0.392157	0.392157	0.329412	0.254902	0.266667	0.196078	0.298039	0.317647
...	...	...	...	...	...	...	...	...
4165	0.188235	0.247059	0.254902	0.686275	0.764706	0.858824	0.545098	0.643137
4166	0.823529	0.788235	0.752941	0.533333	0.517647	0.494118	0.254902	0.266667
4167	0.592157	0.623529	0.486275	0.603922	0.619608	0.490196	0.584314	0.611765
4168	0.254902	0.313725	0.333333	0.552941	0.552941	0.505882	0.549020	0.537255
4169	0.643137	0.423529	0.223529	0.619608	0.364706	0.160784	0.607843	0.360784

4170 rows × 3073 columns

Now that we have our class data organized into a dataframe with RGB values per image

(images were converted to 32x32=1024, 1024x3=3072 features to represent RGB values, and 4072 total images), lets split out data up into training and test splits to try a shotgun approach, where we will have classic machine learning models predict on images and testing data.

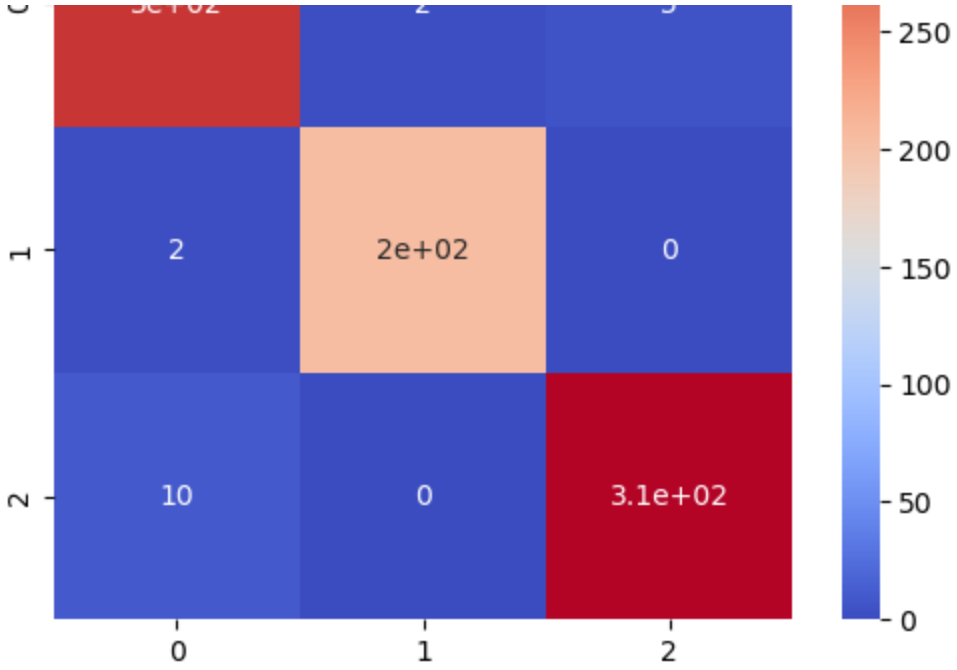
In [19]:
```
#Going to create a function for "Shotgun approach" for machine learning modeli

def classification_model_test(model, X_train, y_train, X_test, y_test):
    # Fit the model with the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate and return the accuracy score
    accuracy = accuracy_score(y_true=y_test, y_pred=y_pred)

    # Generate heatmap of confusion matrix
    sns.heatmap(confusion_matrix(y_true=y_test, y_pred=y_pred),
            annot=True,
            cmap="coolwarm",
            square=True)

    # Print classification report
    print(classification_report(y_true=y_test, y_pred=y_pred))

    return f"Accuracy Score: {accuracy:.2f}"
```

In [20]:
```
X = df.drop(columns=['label'])
y = df['label']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test
```

In [21]:
```
log_reg = LogisticRegression()
classification_model_test(log_reg, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       304
           1       0.99      0.99      0.99       206
           2       0.98      0.97      0.98       324

    accuracy                           0.98       834
   macro avg       0.98      0.98      0.98       834
weighted avg       0.98      0.98      0.98       834
```

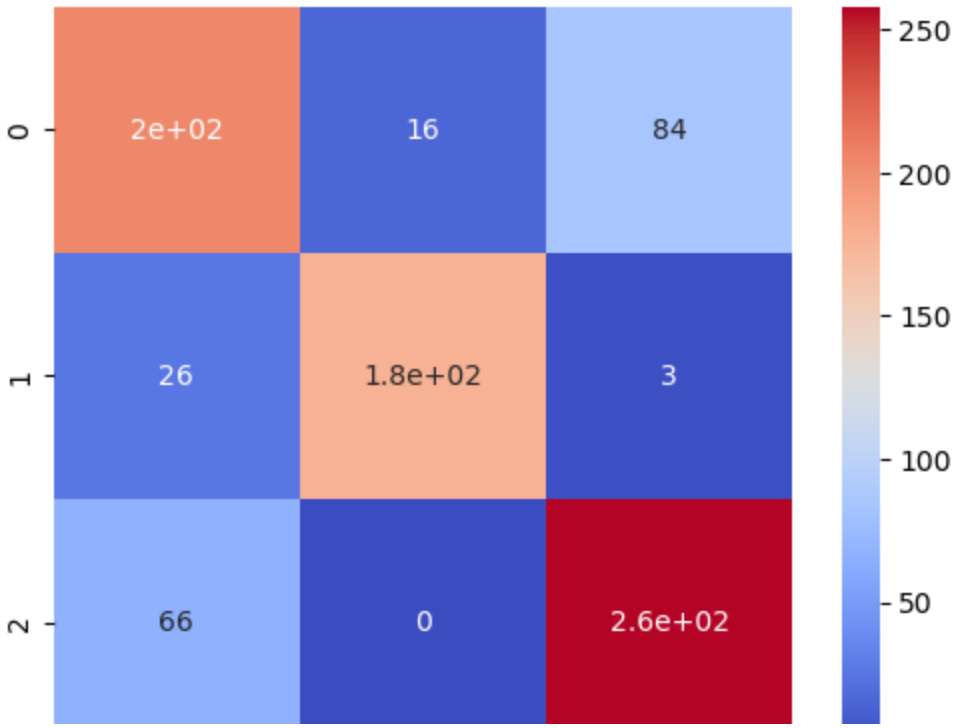
Out[21]: 'Accuracy Score: 0.98'



In [22]:
```
ada_model = AdaBoostClassifier()
classification_model_test(ada_model, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       0.69      0.67      0.68       304
           1       0.92      0.86      0.89       206
           2       0.75      0.80      0.77       324

    accuracy                           0.77       834
   macro avg       0.78      0.78      0.78       834
weighted avg       0.77      0.77      0.77       834
```

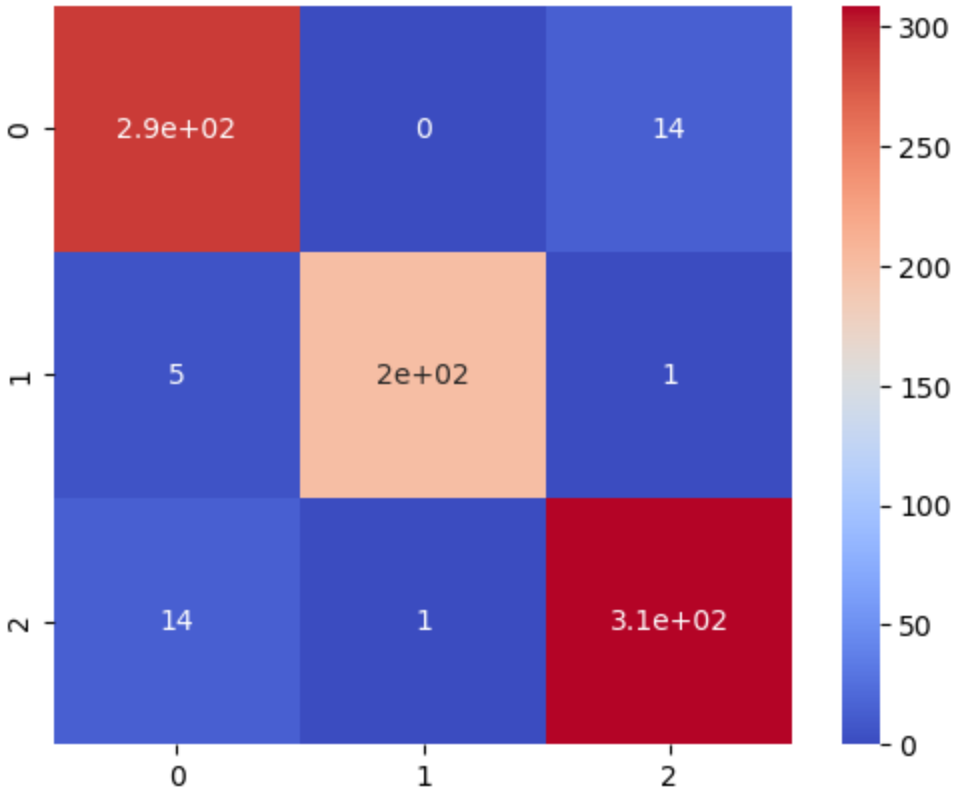
Out[22]: 'Accuracy Score: 0.77'



In [23]:
```
svc_model = SVC()
classification_model_test(svc_model, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       304
           1       1.00      0.97      0.98       206
           2       0.95      0.95      0.95       324

    accuracy                           0.96       834
   macro avg       0.96      0.96      0.96       834
weighted avg       0.96      0.96      0.96       834
```

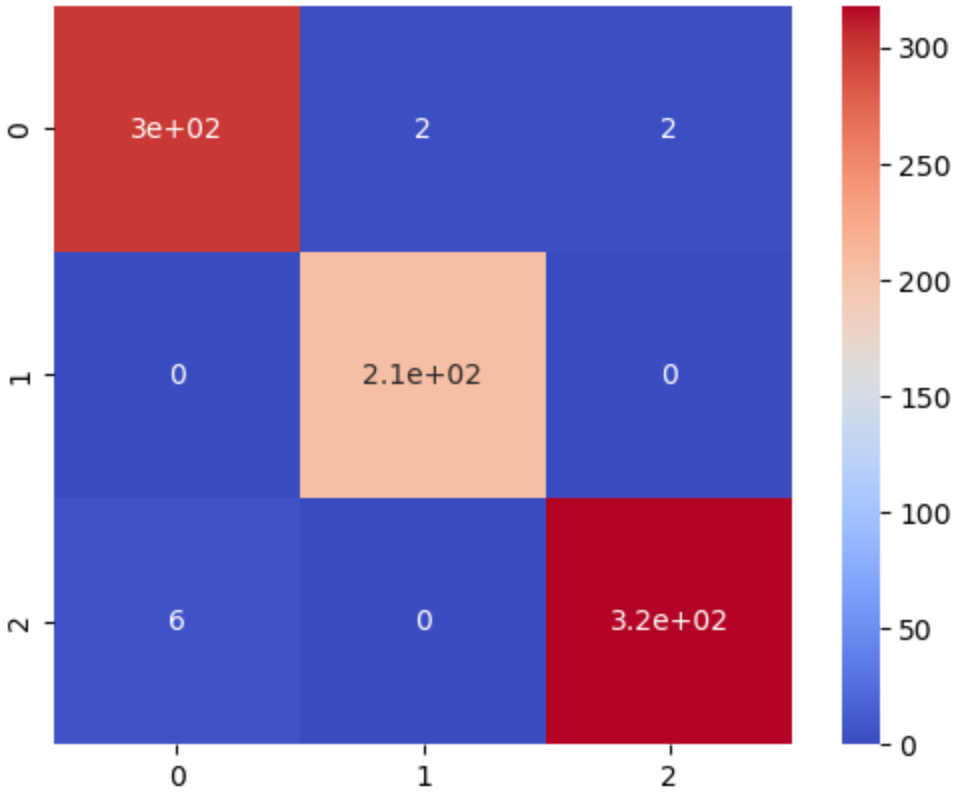
Out[23]: 'Accuracy Score: 0.96'



In [24]:
```
gbc_model = GradientBoostingClassifier()
classification_model_test(gbc_model, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       304
           1       0.99      1.00      1.00       206
           2       0.99      0.98      0.99       324

    accuracy                           0.99       834
   macro avg       0.99      0.99      0.99       834
weighted avg       0.99      0.99      0.99       834
```

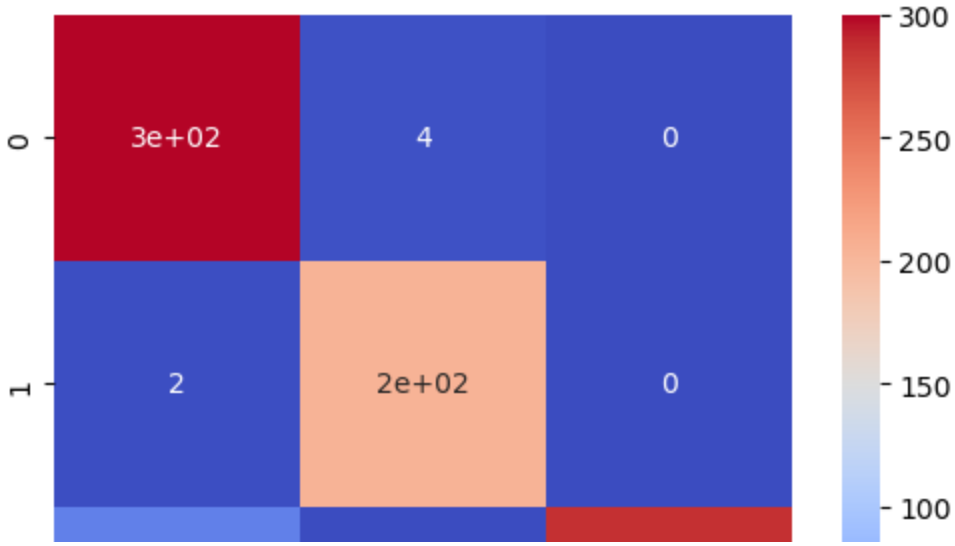
Out[24]: 'Accuracy Score: 0.99'

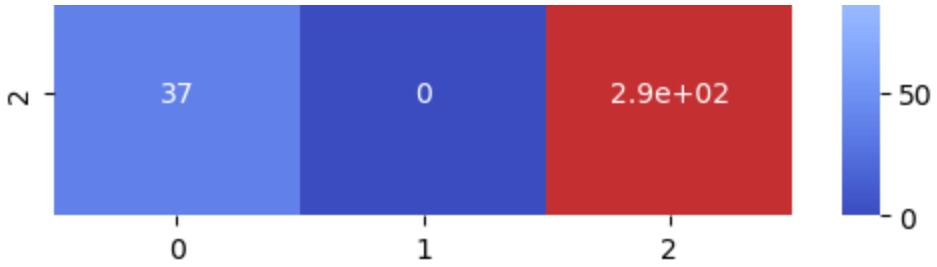


In [25]:
```
sgd_model = SGDClassifier()
classification_model_test(sgd_model, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93       304
           1       0.98      0.99      0.99       206
           2       1.00      0.89      0.94       324

    accuracy                           0.95       834
   macro avg       0.96      0.95      0.95       834
weighted avg       0.95      0.95      0.95       834
```

Out[25]: 'Accuracy Score: 0.95'

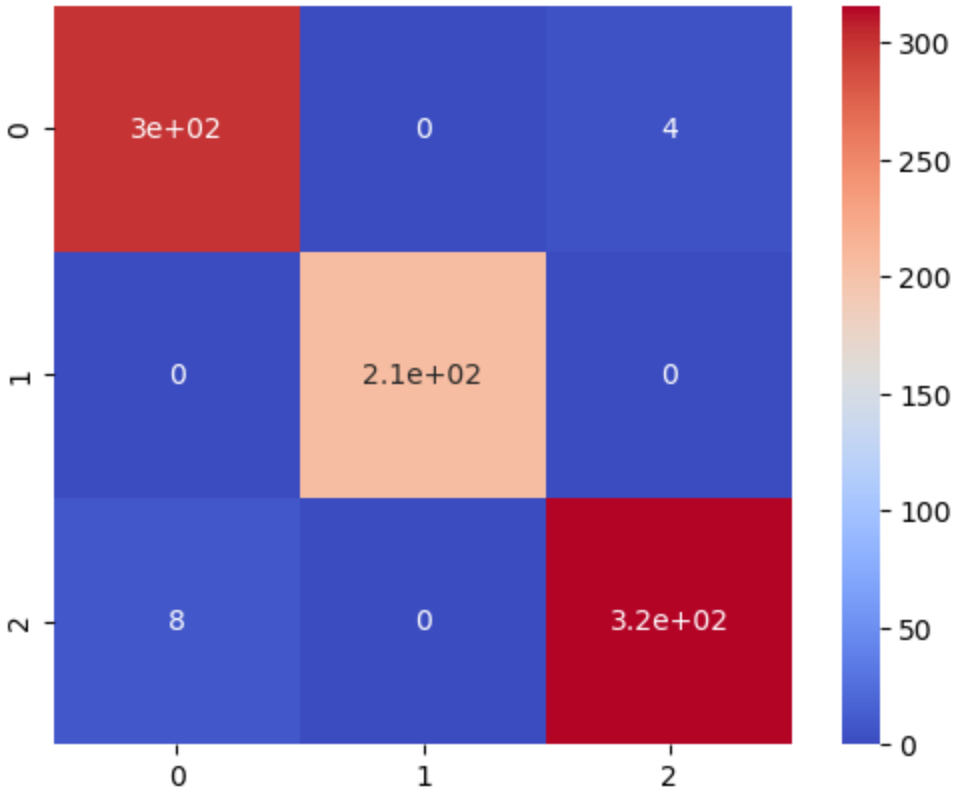




In [26]:
```
rfc_model = RandomForestClassifier()
classification_model_test(rfc_model, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       304
           1       1.00      1.00      1.00       206
           2       0.99      0.98      0.98       324

    accuracy                           0.99       834
   macro avg       0.99      0.99      0.99       834
weighted avg       0.99      0.99      0.99       834
```

Out[26]: 'Accuracy Score: 0.99'



My classic ML approach actually worked better than I had expected after normalizing and preprocessing my image data. Prior to aggregating the images into 3 classes, it had some trouble identifying 57 classes. This has helped and proved to be much better. I may run a random search classifier to try and improve the model accuracy for SGD and SVC models.

In [27]:

In [30]:
```
#Setup cross-validation for hyper parameter tuning and grids for hyperparamete

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=42)

#Create grid search for SGD and .

grid_sgd = {
    "penalty": ["l2", "l1", "elasticnet"],
    "alpha": [1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100],
    "max_iter": [100, 200, 300, 400, 500],
    "learning_rate": ["constant", "optimal", "invscaling", "adaptive"],
    "eta0": [0.01, 0.1, 1],
    "class_weight": [None, 'balanced'],
    'average': [False, True]
}

grid_svc = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'poly', 'sigmoid'],
    'degree': [2, 3, 4],
    'coef0': [0, 0.1, 1],
    'tol': [1e-3, 1e-4],
    'max_iter': [10, 20, 30],
    'class_weight': [None, 'balanced']
}

grid_gbc = {
    'n_estimators': [5, 10, 15],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 3],
    'min_samples_leaf': [1, 2]
}
```

In [31]:
```
search = RandomizedSearchCV(
    estimator=sgd_model,
    param_distributions=grid_sgd,
    n_iter=10,
    cv=cv,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)
```

In [32]:
```
#Create result variable to fit the search gridsearch with x train and y train
result = search.fit(X_train, y_train)
```

In [33]:
```
#Print best score and optimal parameters.
print("> BEST SCORE: \t\t{}".format(result.best_score_))
print("> OPTIMAL PARAMETERS: \t{}".format(result.best_params_))
```

```
> BEST SCORE:           0.9616331900763038
> OPTIMAL PARAMETERS:   {'penalty': 'l1', 'max_iter': 100, 'learning_rate': 'ada
ptive', 'eta0': 0.1, 'class_weight': None, 'average': True, 'alpha': 1e-07}
```

In [46]:
```
search_svc = RandomizedSearchCV(
    estimator=svc_model,
    param_distributions=grid_svc,
    n_iter=10,
    cv=cv,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42)
```

In [47]:
```
#Create result variable to fit the search gridsearch with x train and y train
result_svc = search_svc.fit(X_train, y_train)
```

In [48]:
```
#Print best score and optimal parameters.
print("> BEST SCORE: \t\t{}".format(result_svc.best_score_))
print("> OPTIMAL PARAMETERS: \t{}".format(result_svc.best_params_))
```

```
> BEST SCORE:           0.6026259792726859
> OPTIMAL PARAMETERS:   {'tol': 0.0001, 'max_iter': 30, 'kernel': 'poly', 'degre
e': 2, 'coef0': 1, 'class_weight': None, 'C': 10}
```

Interestingly enough, the stochastic gradient descent model performed very well with tuning, and the support vector machine model performed worse with hyper parameter tuning. So for a classic machine learning approach, it appears that the SGD model is the best one!

Next, lets look into histogram of oriented gradients (or HOG) to see if we get any improvements, before we dive into deep learning.

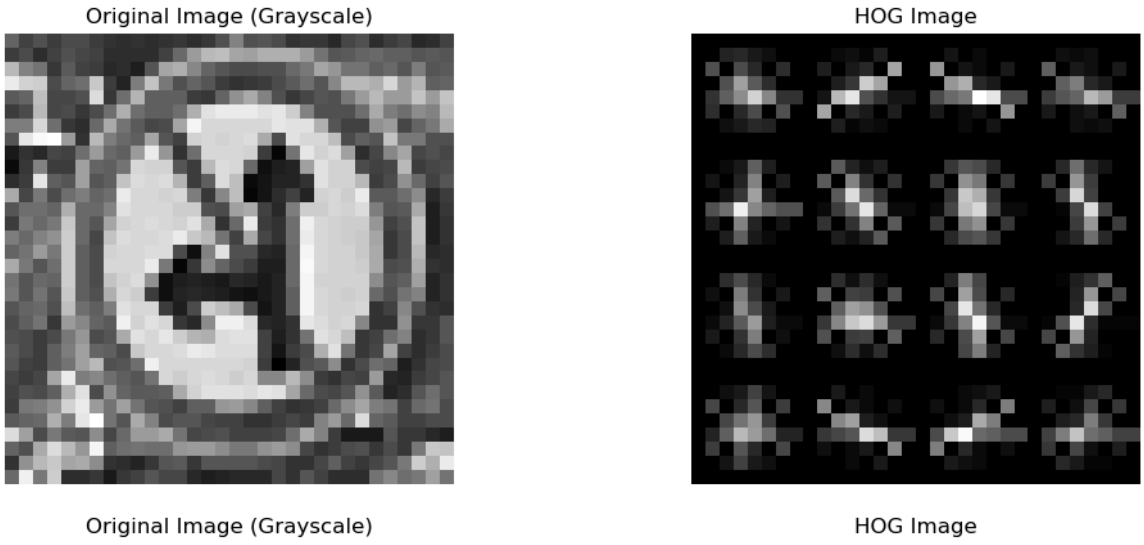
Section 3: Histogram of Oriented Gradients Approach

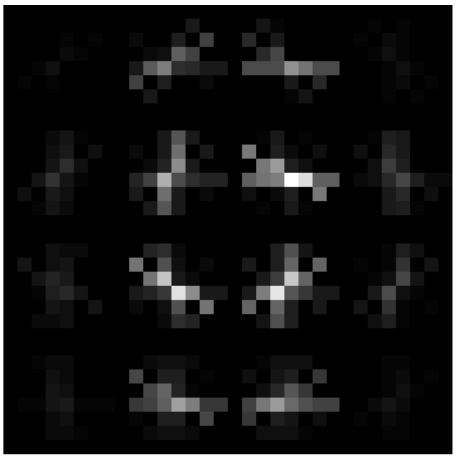
One method that one can use when working with image data is called the HOG (Histogram of Oriented Gradients) which is a feature descriptor used in computer vision and image processing for object detection. It works using Gradient computation among many things, to uncover edge and shape detection among complex images.

I will first use some code to display a before and after image of each class, in black and white, and after HOG has been applied. We can get an idea of what this does and then I will apply it to all my images.

In [27]:
```
#Loop through each folder
for folder_name in folders:
    folder_path = os.path.join(input_dir_train, folder_name)
```

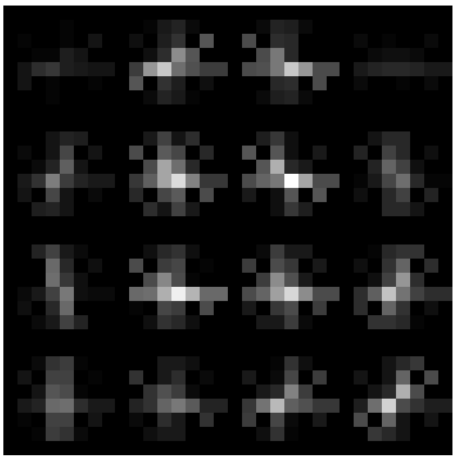
```
    #Check if the folder is an actual directory
    if os.path.isdir(folder_path):
        #Loop through each image in the class folder
        for image in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image)

            #Read the image in color
            img = cv2.imread(image_path, cv2.IMREAD_COLOR)

            #This will only proceed if the image exists
            if img is not None:
                # Resize image to fixed size so it's uniform
                img_resized = cv2.resize(img, (32,32))

                #Convert image to grayscale
                img_gray = color.rgb2gray(img_resized)

                # Display original image (grayscale)
                plt.figure(figsize=(12, 4))
                plt.subplot(1, 2, 1)
                plt.imshow(img_gray, cmap='gray')
                plt.title('Original Image (Grayscale)')
                plt.axis('off')

                #Apply HOG descriptor
                hog_features, hog_image = hog(img_gray,
                                              orientations=9,
                                              pixels_per_cell=(8, 8),
                                              cells_per_block=(2, 2),
                                              block_norm='L2-Hys',
                                              visualize=True)

                # Display HOG image
                plt.subplot(1, 2, 2)
                plt.imshow(hog_image, cmap='gray')
                plt.title('HOG Image')
                plt.axis('off')

                plt.tight_layout()
                plt.show()

                break
```





Original Image (Grayscale)

HOG Image



In [53]:

```
#this will hold the class and image data once HOG is applied
image_data_hog = []
labels = []

#Loop through each folder
for folder_name in folders:
    folder_path = os.path.join(input_dir_train, folder_name)

    #Check if the folder is an actual directory
    if os.path.isdir(folder_path):
        #Loop through each image in the class folder
        for image in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image)

            #Read the image in color
            img = cv2.imread(image_path, cv2.IMREAD_COLOR)

            #This will only proceed if the image exists
            if img is not None:


                #resize image to fixed size so its uniform ( as most images ha
                img_resized = cv2.resize(img, (32,32))

                #convert image to grayscale
                img_gray = color.rgb2gray(img_resized)

                #apply HOG descriptor
                hog_features, hog_image = hog(img_gray,
```

```
                                                orientations=9,
                                                pixels_per_cell=(8, 8),
                                                cells_per_block=(2, 2),
                                                block_norm='L2-Hys',
                                                visualize=True)

                        #flatten the HOG features into 1D array for machine learning
                        hog_features_flattened = hog_features.flatten()

                        #Add HOG features to the list
                        image_data_hog.append(hog_features_flattened)

                        #Assign labels based on the folder (0, 1, or 2)
                        if folder_name == 'Directional_Signs':
                            labels.append(0)
                        elif folder_name == 'Speed_Limit':
                            labels.append(1)
                        elif folder_name == 'Warning_Signs':
                            labels.append(2)
                    else:
                        print(f"Image could not be read: {image_path}")

# Convert lists to numpy arrays
image_data_hog = np.array(image_data_hog)
labels = np.array(labels)

print("Finished getting HOG features")
```

```
Finished getting HOG features
```

In [71]:

```
#Convert image data to a DataFrame
df_hog = pd.DataFrame(image_data_hog)

#Scale data so it is more managable for machine models.
# df = df / 255

# Add the labels as the target column
df_hog['label'] = labels

df
```

Out[71]:

	0	1	2	3	4	5	6	7
0	0.392157	0.392157	0.329412	0.254902	0.266667	0.196078	0.298039	0.317647
1	0.392157	0.392157	0.329412	0.254902	0.266667	0.196078	0.298039	0.317647
2	0.843137	0.874510	0.870588	0.811765	0.878431	0.858824	0.843137	0.886275
3	0.890196	0.913725	0.949020	0.909804	0.917647	0.964706	0.933333	0.933333
4	0.392157	0.392157	0.329412	0.254902	0.266667	0.196078	0.298039	0.317647
...	...	...	...	...	...	...	...	...
4165	0.188235	0.247059	0.254902	0.686275	0.764706	0.858824	0.545098	0.643137
4166	0.823529	0.788235	0.752941	0.533333	0.517647	0.494118	0.254902	0.266667

4167	0.592157	0.623529	0.486275	0.603922	0.619608	0.490196	0.584314	0.611765
4168	0.254902	0.313725	0.333333	0.552941	0.552941	0.505882	0.549020	0.537255
4169	0.643137	0.423529	0.223529	0.619608	0.364706	0.160784	0.607843	0.360784

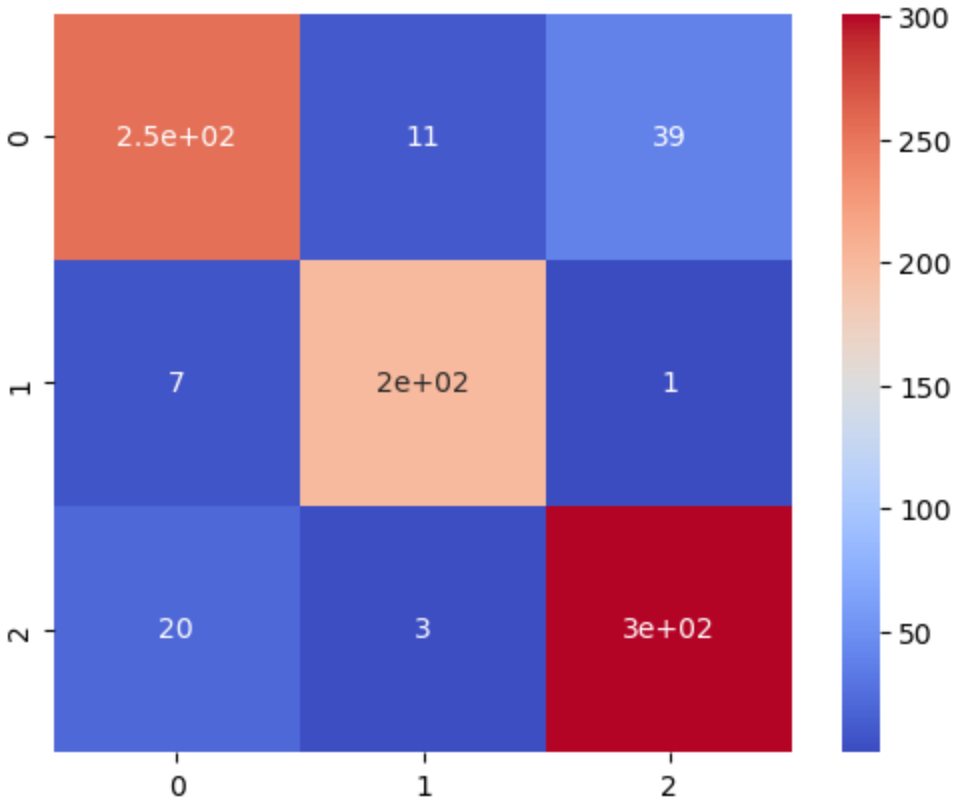
4170 rows × 3073 columns

In [72]:
```
X_hog = df_hog.drop(columns=['label'])
y_hog = df_hog['label']

# Split the data into training and test sets
X_train_hog, X_test_hog, y_train_hog, y_test_hog = train_test_split(X_hog, y_h
```

In [73]:
```
log_reg_hog = LogisticRegression()
classification_model_test(log_reg_hog, X_train_hog, y_train_hog, X_test_hog, y
```

```
              precision    recall  f1-score   support

           0       0.90      0.84      0.87       304
           1       0.93      0.96      0.95       206
           2       0.88      0.93      0.91       324

    accuracy                           0.90       834
   macro avg       0.91      0.91      0.91       834
weighted avg       0.90      0.90      0.90       834
```

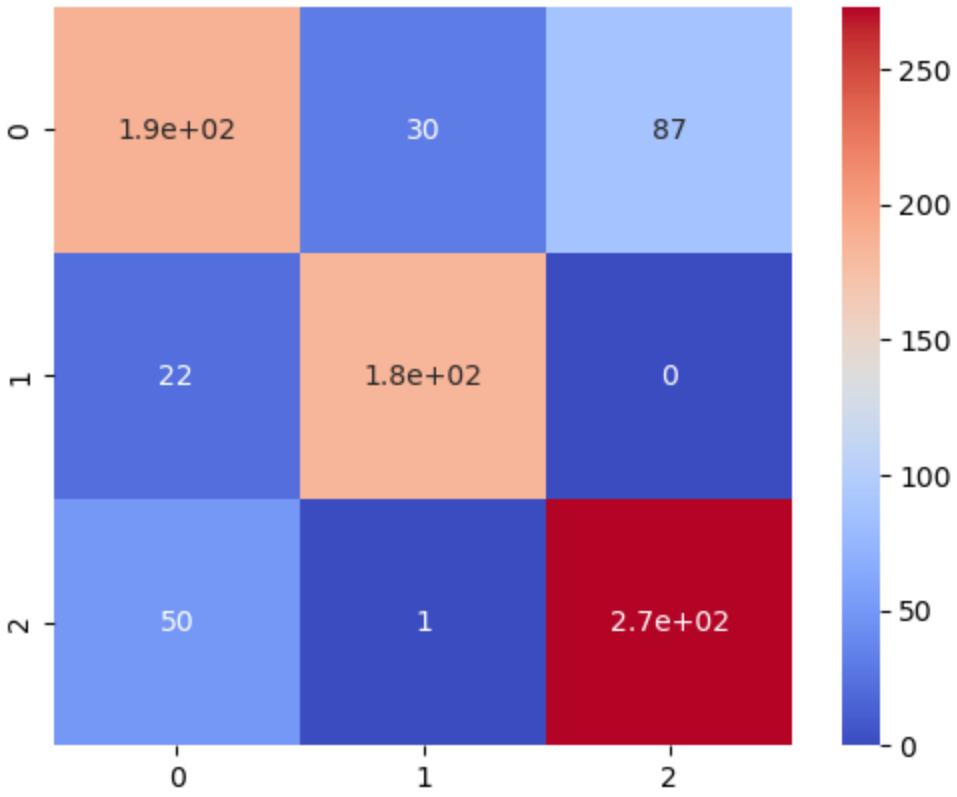
Out[73]: 'Accuracy Score: 0.90'



In [75]:
```
ada_model_hog = AdaBoostClassifier()
classification_model_test(ada_model_hog, X_train_hog, y_train_hog, X_test_hog,
```

	precision	recall	f1-score	support
0	0.72	0.62	0.66	304
1	0.86	0.89	0.87	206
2	0.76	0.84	0.80	324
accuracy			0.77	834
macro avg	0.78	0.78	0.78	834
weighted avg	0.77	0.77	0.77	834

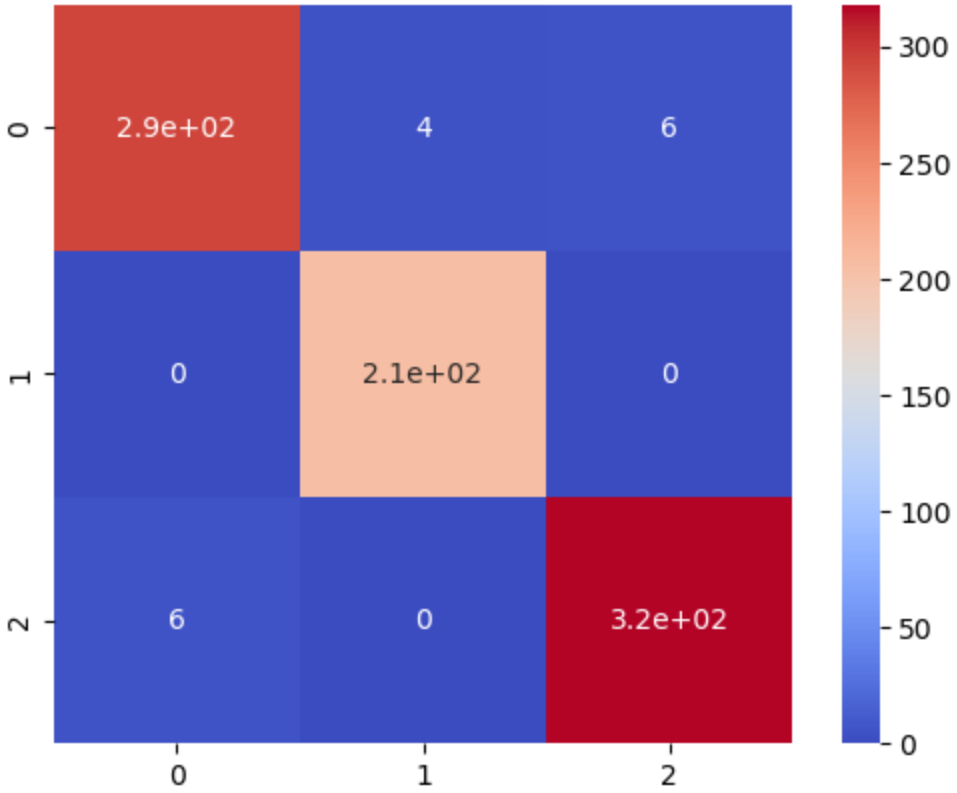
Out[75]: 'Accuracy Score: 0.77'



In [76]:
```
svc_model_hog = SVC()
classification_model_test(svc_model_hog, X_train_hog, y_train_hog, X_test_hog,
```

	precision	recall	f1-score	support
0	0.98	0.97	0.97	304
1	0.98	1.00	0.99	206
2	0.98	0.98	0.98	324
accuracy			0.98	834
macro avg	0.98	0.98	0.98	834
weighted avg	0.98	0.98	0.98	834

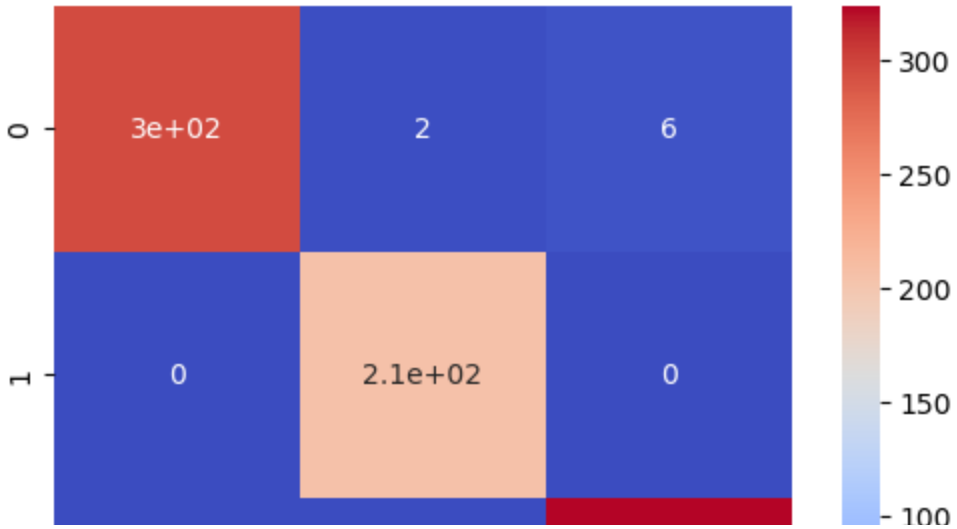
Out[76]: 'Accuracy Score: 0.98'

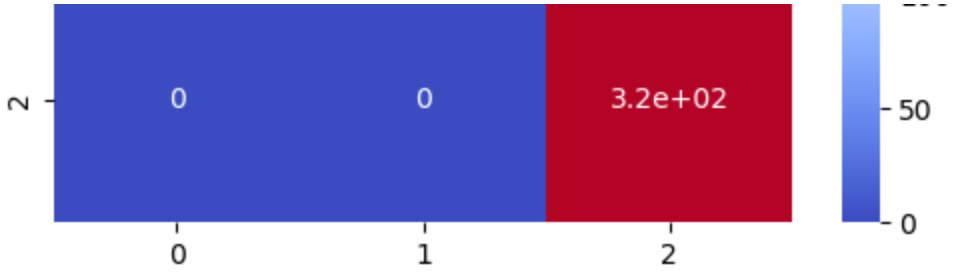


In [77]:
```
gbc_model_hog = GradientBoostingClassifier()
classification_model_test(gbc_model_hog, X_train_hog, y_train_hog, X_test_hog,
```

	precision	recall	f1-score	support
0	1.00	0.97	0.99	304
1	0.99	1.00	1.00	206
2	0.98	1.00	0.99	324
accuracy			0.99	834
macro avg	0.99	0.99	0.99	834
weighted avg	0.99	0.99	0.99	834

Out[77]: 'Accuracy Score: 0.99'

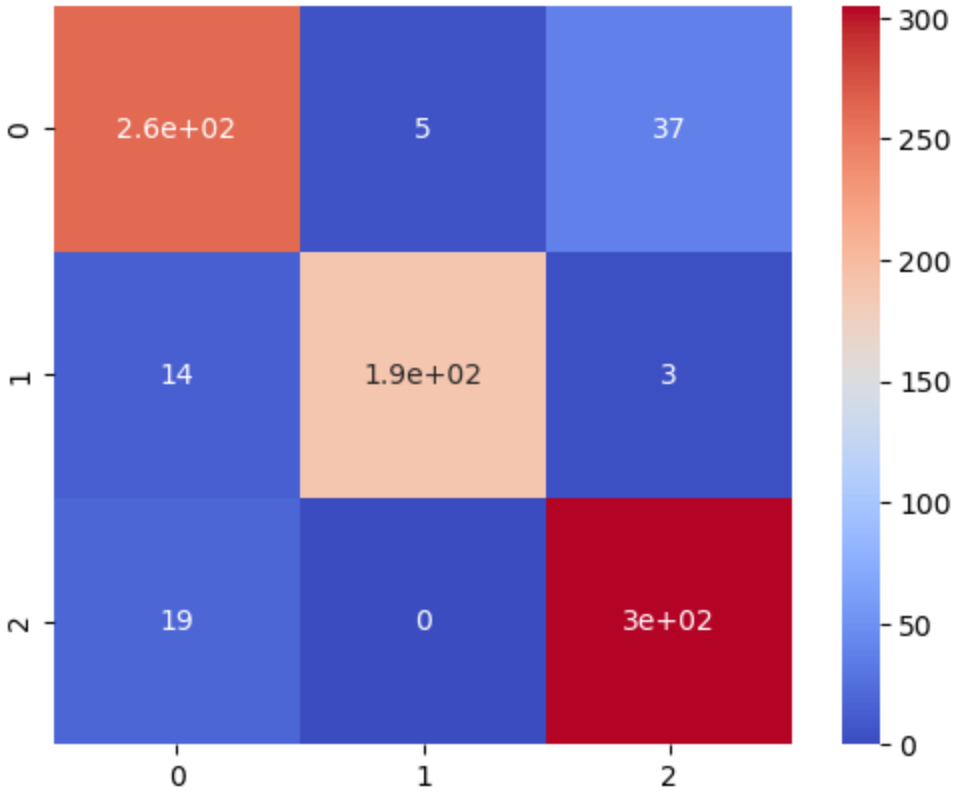




In [78]:
```
sgd_model_hog = SGDClassifier()
classification_model_test(sgd_model_hog, X_train_hog, y_train_hog, X_test_hog,
```

	precision	recall	f1-score	support
0	0.89	0.86	0.87	304
1	0.97	0.92	0.94	206
2	0.88	0.94	0.91	324
accuracy			0.91	834
macro avg	0.92	0.91	0.91	834
weighted avg	0.91	0.91	0.91	834

Out[78]: 'Accuracy Score: 0.91'

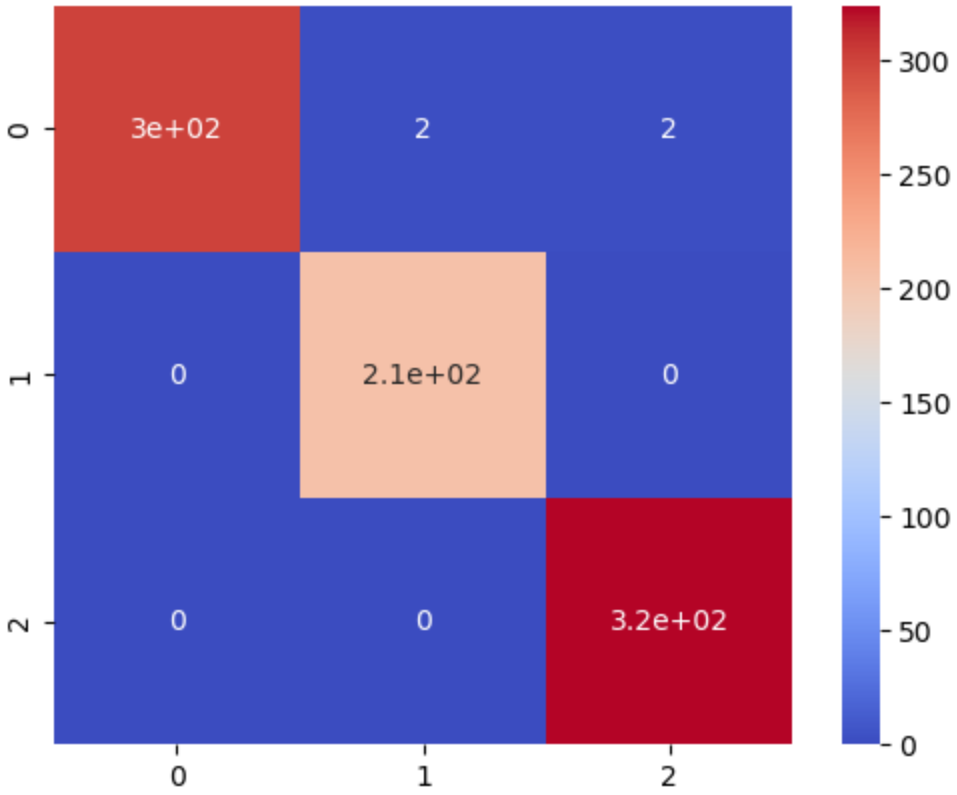


In [79]:
```
rfc_model_hog = RandomForestClassifier()
classification_model_test(rfc_model_hog, X_train_hog, y_train_hog, X_test_hog,
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	304

```
              0       1.00      0.99      0.99       304
              1       0.99      1.00      1.00       206
              2       0.99      1.00      1.00       324

       accuracy                           1.00       834
      macro avg       0.99      1.00      1.00       834
   weighted avg       1.00      1.00      1.00       834
```

Out[79]: 'Accuracy Score: 1.00'



Section 4: Deep Learning

The basic ML models seemed to have predicted classes very well. I will construct a CNN (Convolutional Neural Network) and train it to also learn on the image data!

Before I get started with training the network, I will set up a frame work. The machine learning models took in data as a 1D array. For a Convolutional Neural Network to work, it looks at height and width of images (2D). Therefore, I will need to preprocess my data before sending the image data to the network.

In [21]:
```
#This will be a sequential CNN network

#Instance a sequential layer architecture
Sequential = tf.keras.models.Sequential

#CNN network layers
Dense = tf.keras.layers.Dense
Dropout = tf.keras.layers.Dropout
```

```
Flatten = tf.keras.layers.Flatten
Conv2D = tf.keras.layers.Conv2D
MaxPool2D = tf.keras.layers.MaxPool2D

#Optimizer
Adam = tf.keras.optimizers.Adam

#Image preprocessing for better learning
ImageDataGenerator =  tf.keras.preprocessing.image.ImageDataGenerator
```

Like before, I need to access my directories and the images within them. I then need to do some preprocessing, like reshaping and scaling. I will do that now.

In [22]:

```
cnn_image_data = []
cnn_labels = []
folders = ['Directional_Signs', 'Speed_Limit', 'Warning_Signs']

# reate a mapping from folder names to numeric labels
label_mapping = {'Directional_Signs': 0, 'Speed_Limit': 1, 'Warning_Signs': 2}

#Loop to go through each class folder
for folder_name in folders:
    folder_path = os.path.join(input_dir_train, folder_name)

    #Check if the folder is an actual directory
    if os.path.isdir(folder_path):
        #Loop through each image in the given class folder
        for image in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image)

            #Read the image in color (height, width, channels)
            img = cv2.imread(image_path, cv2.IMREAD_COLOR)

            #This will only proceed if the image exists
            if img is not None:
                #Resize image to fixed size (32x32x3) for CNN input
                img_resized = cv2.resize(img, (32, 32))

                #Add the resized image and the label to the lists above
                cnn_image_data.append(img_resized)
                cnn_labels.append(label_mapping[folder_name])
            else:
                print(f"The image could not be read: {image_path}")

#Convert the lists into numpy arrays for efficiency
cnn_image_data = np.array(cnn_image_data)  # This will have shape (num_images,
cnn_labels = np.array(cnn_labels)
```

In [23]:

```
#I am going to scale the image data to 0-1 range for the model.

cnn_image_data = cnn_image_data / 255.0

cnn_image_data
```

Out[23]:

```
array([[[[0.39215686, 0.39215686, 0.32941176],
```

```
       [0.25490196, 0.26666667, 0.19607843],
       [0.29803922, 0.31764706, 0.24705882],
       ...,
       [0.17254902, 0.18039216, 0.16470588],
       [0.19607843, 0.20392157, 0.19215686],
       [0.2627451 , 0.2627451 , 0.25098039]],

      [[0.24313725, 0.24705882, 0.18823529],
       [0.29019608, 0.30588235, 0.23137255],
       [0.17647059, 0.20392157, 0.13333333],
       ...,
       [0.29411765, 0.33333333, 0.30588235],
       [0.32941176, 0.36078431, 0.3372549 ],
       [0.3254902 , 0.34509804, 0.32156863]],

      [[0.68235294, 0.68235294, 0.61960784],
       [0.48627451, 0.49803922, 0.42745098],
       [0.41568627, 0.43921569, 0.36470588],
       ...,
       [0.41960784, 0.49019608, 0.45490196],
       [0.29019608, 0.34901961, 0.31372549],
       [0.31764706, 0.36078431, 0.32941176]],

      ...,

      [[0.41960784, 0.38431373, 0.34117647],
       [0.34117647, 0.32156863, 0.2627451 ],
       [0.65490196, 0.63137255, 0.55686275],
       ...,
       [0.48235294, 0.44313725, 0.41960784],
       [0.75686275, 0.72941176, 0.71764706],
       [0.72941176, 0.70980392, 0.71372549]],

      [[0.25098039, 0.19607843, 0.17647059],
       [0.57254902, 0.5372549 , 0.49803922],
       [0.68627451, 0.67058824, 0.58039216],
       ...,
       [0.76862745, 0.74901961, 0.72156863],
       [0.6627451 , 0.64313725, 0.62352941],
       [0.85882353, 0.84705882, 0.84705882]],

      [[0.45098039, 0.38823529, 0.37647059],
       [0.89019608, 0.85098039, 0.81960784],
       [0.47843137, 0.45882353, 0.36078431],
       ...,
       [0.4627451 , 0.44705882, 0.41568627],
       [0.58823529, 0.58039216, 0.55686275],
       [0.43137255, 0.42745098, 0.42352941]]],


     [[[0.39215686, 0.39215686, 0.32941176],
       [0.25490196, 0.26666667, 0.19607843],
       [0.29803922, 0.31764706, 0.24705882],
       ...,
       [0.17254902, 0.18039216, 0.16470588],
       [0.19607843, 0.20392157, 0.19215686],
       [0.2627451 , 0.2627451 , 0.25098039]],

      [[0.24313725, 0.24705882, 0.18823529],
       [0.29019608, 0.30588235, 0.23137255],
       [0.17647059, 0.20392157, 0.13333333],
```

```
       [0.17647059, 0.20392157, 0.13333333],
       ...,
       [0.29411765, 0.33333333, 0.30588235],
       [0.32941176, 0.36078431, 0.3372549 ],
       [0.3254902 , 0.34509804, 0.32156863]],

      [[0.68235294, 0.68235294, 0.61960784],
       [0.48627451, 0.49803922, 0.42745098],
       [0.41568627, 0.43921569, 0.36470588],
       ...,
       [0.41960784, 0.49019608, 0.45490196],
       [0.29019608, 0.34901961, 0.31372549],
       [0.31764706, 0.36078431, 0.32941176]],

      ...,

      [[0.41960784, 0.38431373, 0.34117647],
       [0.34117647, 0.32156863, 0.2627451 ],
       [0.65490196, 0.63137255, 0.55686275],
       ...,
       [0.48235294, 0.44313725, 0.41960784],
       [0.75686275, 0.72941176, 0.71764706],
       [0.72941176, 0.70980392, 0.71372549]],

      [[0.25098039, 0.19607843, 0.17647059],
       [0.57254902, 0.5372549 , 0.49803922],
       [0.68627451, 0.67058824, 0.58039216],
       ...,
       [0.76862745, 0.74901961, 0.72156863],
       [0.6627451 , 0.64313725, 0.62352941],
       [0.85882353, 0.84705882, 0.84705882]],

      [[0.45098039, 0.38823529, 0.37647059],
       [0.89019608, 0.85098039, 0.81960784],
       [0.47843137, 0.45882353, 0.36078431],
       ...,
       [0.4627451 , 0.44705882, 0.41568627],
       [0.58823529, 0.58039216, 0.55686275],
       [0.43137255, 0.42745098, 0.42352941]]],


     [[[0.84313725, 0.8745098 , 0.87058824],
       [0.81176471, 0.87843137, 0.85882353],
       [0.84313725, 0.88627451, 0.87843137],
       ...,
       [0.54509804, 0.63137255, 0.61176471],
       [0.55294118, 0.64313725, 0.62352941],
       [0.53333333, 0.63529412, 0.61176471]],

      [[0.84705882, 0.88235294, 0.87058824],
       [0.81568627, 0.88235294, 0.8627451 ],
       [0.83137255, 0.88235294, 0.8745098 ],
       ...,
       [0.56078431, 0.65098039, 0.63529412],
       [0.55686275, 0.64705882, 0.62352941],
       [0.53333333, 0.64313725, 0.61960784]],

      [[0.85098039, 0.88627451, 0.88235294],
       [0.82745098, 0.89019608, 0.87058824],
       [0.85098039, 0.89019608, 0.87843137],
       ...,
```

```
       [0.53333333, 0.62745098, 0.62352941],
       [0.5254902 , 0.63137255, 0.61960784],
       [0.5372549 , 0.63529412, 0.62352941]],

      ...,

      [[0.85490196, 0.90588235, 0.90196078],
       [0.83921569, 0.89019608, 0.88235294],
       [0.85490196, 0.90196078, 0.89019608],
       ...,
       [0.40784314, 0.37254902, 0.99215686],
       [0.32941176, 0.33333333, 0.96862745],
       [0.37254902, 0.2745098 , 0.92156863]],

      [[0.83137255, 0.88235294, 0.8745098 ],
       [0.83529412, 0.88235294, 0.8745098 ],
       [0.85098039, 0.89411765, 0.88627451],
       ...,
       [0.34117647, 0.25882353, 0.98823529],
       [0.46666667, 0.30196078, 0.99607843],
       [0.31764706, 0.25882353, 0.99607843]],

      [[0.82352941, 0.87843137, 0.85882353],
       [0.84313725, 0.89803922, 0.88235294],
       [0.83921569, 0.87843137, 0.87843137],
       ...,
       [0.28627451, 0.14901961, 0.94117647],
       [0.20784314, 0.17254902, 0.94509804],
       [0.09411765, 0.1254902 , 0.95686275]]],


     ...,


     [[[0.59215686, 0.62352941, 0.48627451],
       [0.60392157, 0.61960784, 0.49019608],
       [0.58431373, 0.61176471, 0.45490196],
       ...,
       [0.45098039, 0.47843137, 0.47843137],
       [0.39215686, 0.39215686, 0.41568627],
       [0.60784314, 0.61960784, 0.59607843]],

      [[0.64313725, 0.61960784, 0.59607843],
       [0.4745098 , 0.45882353, 0.44313725],
       [0.60784314, 0.60784314, 0.51764706],
       ...,
       [0.4627451 , 0.48235294, 0.48627451],
       [0.43529412, 0.44313725, 0.45098039],
       [0.56470588, 0.58039216, 0.56078431]],

      [[0.61568627, 0.61960784, 0.58431373],
       [0.37647059, 0.34509804, 0.33333333],
       [0.58431373, 0.60392157, 0.43529412],
       ...,
       [0.49019608, 0.49411765, 0.49411765],
       [0.39607843, 0.40392157, 0.40392157],
       [0.56078431, 0.57647059, 0.55686275]],

      ...,
```

```
       [[0.31764706, 0.32941176, 0.3372549 ],
        [0.30980392, 0.32156863, 0.3254902 ],
        [0.32156863, 0.33333333, 0.3372549 ],
        ...,
        [0.32156863, 0.27843137, 0.2627451 ],
        [0.4627451 , 0.48627451, 0.48627451],
        [0.16470588, 0.16862745, 0.15686275]],

       [[0.30196078, 0.31764706, 0.33333333],
        [0.3254902 , 0.33333333, 0.33333333],
        [0.31764706, 0.33333333, 0.33333333],
        ...,
        [0.48235294, 0.45882353, 0.43529412],
        [0.46666667, 0.4745098 , 0.4745098 ],
        [0.15294118, 0.15686275, 0.14901961]],

       [[0.29803922, 0.30980392, 0.30980392],
        [0.14901961, 0.15686275, 0.15686275],
        [0.17254902, 0.18823529, 0.16862745],
        ...,
        [0.4745098 , 0.4627451 , 0.42352941],
        [0.45490196, 0.47058824, 0.47058824],
        [0.19215686, 0.19607843, 0.18039216]]],


      [[[0.25490196, 0.31372549, 0.33333333],
        [0.55294118, 0.55294118, 0.50588235],
        [0.54901961, 0.5372549 , 0.49411765],
        ...,
        [0.35686275, 0.38039216, 0.39215686],
        [0.62352941, 0.63137255, 0.59215686],
        [0.62745098, 0.62352941, 0.59215686]],

       [[0.27058824, 0.31372549, 0.34901961],
        [0.56470588, 0.59607843, 0.45882353],
        [0.56470588, 0.59215686, 0.44313725],
        ...,
        [0.41960784, 0.43137255, 0.43137255],
        [0.61568627, 0.62352941, 0.58823529],
        [0.61568627, 0.61568627, 0.59215686]],

       [[0.2627451 , 0.32156863, 0.34509804],
        [0.52941176, 0.51372549, 0.49411765],
        [0.5254902 , 0.50980392, 0.48627451],
        ...,
        [0.43137255, 0.45882353, 0.45882353],
        [0.58431373, 0.58431373, 0.56470588],
        [0.58431373, 0.58431373, 0.56078431]],

       ...,

       [[0.30980392, 0.34117647, 0.34901961],
        [0.27843137, 0.29411765, 0.29019608],
        [0.29803922, 0.31372549, 0.31372549],
        ...,
        [0.19215686, 0.2       , 0.19215686],
        [0.29019608, 0.32156863, 0.28235294],
        [0.28627451, 0.30588235, 0.25882353]],

       [[0.32156863, 0.34901961, 0.36078431],
        [0.29019608, 0.29411765, 0.30196078]
```

```
       [0.29019608, 0.29411765, 0.30196078],
       [0.12941176, 0.13333333, 0.13333333],
       ...,
       [0.17647059, 0.18431373, 0.19215686],
       [0.2745098 , 0.29803922, 0.25490196],
       [0.25490196, 0.27058824, 0.21568627]],

      [[0.18039216, 0.2       , 0.2       ],
       [0.16470588, 0.16862745, 0.16078431],
       [0.29019608, 0.30196078, 0.30588235],
       ...,
       [0.18431373, 0.18431373, 0.18431373],
       [0.25490196, 0.28235294, 0.25490196],
       [0.2627451 , 0.2745098 , 0.23921569]]],


     [[[0.64313725, 0.42352941, 0.22352941],
       [0.61960784, 0.36470588, 0.16078431],
       [0.60784314, 0.36078431, 0.15686275],
       ...,
       [0.24313725, 0.24705882, 0.24705882],
       [0.16862745, 0.16470588, 0.16470588],
       [0.25882353, 0.25882353, 0.25098039]],

      [[0.51372549, 0.38431373, 0.29019608],
       [0.50196078, 0.4       , 0.32156863],
       [0.49411765, 0.40392157, 0.34901961],
       ...,
       [0.25098039, 0.24705882, 0.25098039],
       [0.17647059, 0.17647059, 0.17647059],
       [0.25098039, 0.25490196, 0.25098039]],

      [[0.18823529, 0.17647059, 0.17647059],
       [0.18823529, 0.17254902, 0.17647059],
       [0.17647059, 0.16078431, 0.16862745],
       ...,
       [0.24313725, 0.24705882, 0.25490196],
       [0.17254902, 0.17647059, 0.17254902],
       [0.25490196, 0.25490196, 0.24705882]],

      ...,

      [[0.35294118, 0.34117647, 0.36862745],
       [0.40392157, 0.40392157, 0.40392157],
       [0.34509804, 0.34117647, 0.34509804],
       ...,
       [0.29019608, 0.30588235, 0.3372549 ],
       [0.18431373, 0.18431373, 0.18431373],
       [0.15294118, 0.14901961, 0.14901961]],

      [[0.32941176, 0.32941176, 0.32941176],
       [0.3254902 , 0.3254902 , 0.3254902 ],
       [0.34117647, 0.3372549 , 0.34117647],
       ...,
       [0.29411765, 0.30196078, 0.3372549 ],
       [0.19215686, 0.18823529, 0.19215686],
       [0.1372549 , 0.1372549 , 0.1372549 ]],

      [[0.32941176, 0.32941176, 0.32941176],
       [0.35686275, 0.35686275, 0.36470588],
       [0.36078431, 0.35686275, 0.36078431],
```

```
        ...,
        [0.29803922, 0.31764706, 0.32941176],
        [0.19215686, 0.19215686, 0.19215686],
        [0.14901961, 0.14117647, 0.14117647]]])
```

In [24]:

```
# # Split the data into training and test sets
X_train_cnn, X_test_cnn, y_train_cnn, y_test_cnn = train_test_split(cnn_image_

# Image Type Forcing
X_train_cnn = X_train_cnn.astype("float32"); X_test_cnn = X_test_cnn.astype("f
```

In [25]:

```
#One-hot encode the labels
y_train_cnn = to_categorical(y_train_cnn, num_classes=3)
y_test_cnn = to_categorical(y_test_cnn, num_classes=3)
```

Great! Our data now looks ready for our CNN network. Let's get the architecture set up.

In [26]:

```
# Here I will define the network layers.
convolutional_layer_1 = Conv2D(50,
                               kernel_size=(3, 3),
                               strides=(1, 1),
                               padding="same",
                               activation="relu",
                               input_shape=(32, 32, 3))
convolutional_layer_2 = Conv2D(75,
                               kernel_size=(3, 3),
                               strides=(1, 1),
                               padding="same",
                               activation="relu")
convolutional_layer_3 = Conv2D(125,
                               kernel_size=(3, 3),
                               strides=(1, 1),
                               padding="same",
                               activation="relu")

# Two corresponding pooling layers to reduce convolved dimensionality
pooling_layer_1 =         MaxPool2D(pool_size=(2, 2))
pooling_layer_2 =         MaxPool2D(pool_size=(2, 2))

# Four dropout layers: two for the convolutions and two for the ANN
dropout_layer_1 =         Dropout(0.25)
dropout_layer_2 =         Dropout(0.25)
dropout_layer_3 =         Dropout(0.4)
dropout_layer_4 =         Dropout(0.3)

# A flattening layer for ingestion into the ANN
flattening_layer_1 =      Flatten()

# Three dense layers to make up the significant ANN architecture
connective_layer_1 =      Dense(500, activation="relu")
connective_layer_2 =      Dense(250, activation="relu")
output_layer =            Dense(3, activation="softmax")
```

In [27]:

```
# Initialize sequential model schema
```

```
model = Sequential()

# Add first convolutional feature mapping process layers
model.add(convolutional_layer_1)

# Add second convolutional feature mapping process layers
model.add(convolutional_layer_2)
model.add(pooling_layer_1)
model.add(dropout_layer_1)

# Add third convolutional feature mapping process layers
model.add(convolutional_layer_3)
model.add(pooling_layer_2)
model.add(dropout_layer_2)

# Add image vectorization process layer
model.add(flattening_layer_1)

# Add connective ANN process layers
model.add(connective_layer_1)
model.add(dropout_layer_3)
model.add(connective_layer_2)
model.add(dropout_layer_4)
model.add(output_layer)

# Summarize model layering setup
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 50)        1400

 conv2d_1 (Conv2D)           (None, 32, 32, 75)        33825

 max_pooling2d (MaxPooling2D  (None, 16, 16, 75)       0
 )

 dropout (Dropout)           (None, 16, 16, 75)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 125)       84500

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 125)        0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 125)         0

 flatten (Flatten)           (None, 8000)              0

 dense (Dense)               (None, 500)               4000500

 dropout_2 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 250)               125250

 dropout_3 (Dropout)         (None, 250)               0

 dense_2 (Dense)             (None, 3)                 753

_________________________________________________________________
```

```
Total params: 4,246,228
Trainable params: 4,246,228
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
# Define Adam optimization
optimizer = Adam(learning_rate=0.001)
```

In [29]:
```
# Set compilation properties
model.compile(optimizer=optimizer,
              loss="categorical_crossentropy",
              metrics=["accuracy"])

# Set epochs and batch size
epochs, batch_size = 40, 32
```

In [30]:
```
# Create image augmentation engine as generator-like object
generator = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=5,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False,
)

# Fit training data to augmentation generator
generator.fit(X_train_cnn)
```

In [62]:
```
#created this only if necessary to use. Doesn't seem like I will need this.

#Define EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,          #this is the number of epochs with no improvement aft
    restore_best_weights=True  # Rstores the model weights from the best epoch
)

#Define ModelCheckpoint callback to save the best model
model_checkpoint = ModelCheckpoint(
    filepath='best_model_traffic_images.h5',  #path where model is saved
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)
```

In [32]:
```
print("Independent training set size:\t\t{}".format(X_train_cnn.shape))
print("Independent validation set size:\t{}".format(X_test_cnn.shape))
```

```
print("Target training set size:\t\t{}".format(y_train_cnn.shape))
print("Target validation set size:\t\t{}".format(y_test_cnn.shape))
```

```
Independent training set size:          (3336, 32, 32, 3)
Independent validation set size:        (834, 32, 32, 3)
Target training set size:               (3336, 3)
Target validation set size:             (834, 3)
```

In [33]:

```
# Fit model using generator-augmented dataset and mini-batch ingestion
history = model.fit(
    generator.flow(X_train_cnn,
                   y_train_cnn,
                   batch_size=batch_size),
    epochs=epochs,
    validation_data=(X_test_cnn, y_test_cnn),
    batch_size=batch_size,
    #callbacks=[early_stopping, model_checkpoint] #This is only if the model i
)
```

```
Epoch 1/40
105/105 [==============================] - 11s 46ms/step - loss: 0.9802 - accura
cy: 0.4814 - val_loss: 0.8319 - val_accuracy: 0.5839
Epoch 2/40
105/105 [==============================] - 55s 532ms/step - loss: 0.6232 - accur
acy: 0.7071 - val_loss: 0.4348 - val_accuracy: 0.7962
Epoch 3/40
105/105 [==============================] - 200s 2s/step - loss: 0.4062 - accurac
y: 0.8150 - val_loss: 0.2703 - val_accuracy: 0.8897
Epoch 4/40
105/105 [==============================] - 176s 2s/step - loss: 0.2850 - accurac
y: 0.8792 - val_loss: 0.1669 - val_accuracy: 0.9257
Epoch 5/40
105/105 [==============================] - 84s 809ms/step - loss: 0.2234 - accur
acy: 0.9062 - val_loss: 0.1576 - val_accuracy: 0.9329
Epoch 6/40
105/105 [==============================] - 125s 1s/step - loss: 0.1850 - accurac
y: 0.9260 - val_loss: 0.1011 - val_accuracy: 0.9556
Epoch 7/40
105/105 [==============================] - 196s 2s/step - loss: 0.1646 - accurac
y: 0.9362 - val_loss: 0.1448 - val_accuracy: 0.9496
Epoch 8/40
105/105 [==============================] - 211s 2s/step - loss: 0.1365 - accurac
y: 0.9430 - val_loss: 0.0976 - val_accuracy: 0.9520
Epoch 9/40
105/105 [==============================] - 9s 84ms/step - loss: 0.1298 - accurac
y: 0.9502 - val_loss: 0.0934 - val_accuracy: 0.9640
Epoch 10/40
105/105 [==============================] - 3s 32ms/step - loss: 0.1113 - accurac
y: 0.9577 - val_loss: 0.0683 - val_accuracy: 0.9736
Epoch 11/40
105/105 [==============================] - 3s 28ms/step - loss: 0.1037 - accurac
y: 0.9619 - val_loss: 0.1005 - val_accuracy: 0.9556
Epoch 12/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0838 - accurac
y: 0.9676 - val_loss: 0.0540 - val_accuracy: 0.9844
Epoch 13/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0875 - accurac
y: 0.9670 - val_loss: 0.0783 - val_accuracy: 0.9760
Epoch 14/40
```

```
Epoch 14/40
105/105 [==============================] - 3s 27ms/step - loss: 0.0846 - accurac
y: 0.9676 - val_loss: 0.0546 - val_accuracy: 0.9844
Epoch 15/40
105/105 [==============================] - 3s 27ms/step - loss: 0.0784 - accurac
y: 0.9718 - val_loss: 0.0452 - val_accuracy: 0.9904
Epoch 16/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0580 - accurac
y: 0.9784 - val_loss: 0.0681 - val_accuracy: 0.9796
Epoch 17/40
105/105 [==============================] - 3s 27ms/step - loss: 0.0758 - accurac
y: 0.9745 - val_loss: 0.0376 - val_accuracy: 0.9904
Epoch 18/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0450 - accurac
y: 0.9850 - val_loss: 0.0360 - val_accuracy: 0.9916
Epoch 19/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0667 - accurac
y: 0.9736 - val_loss: 0.0364 - val_accuracy: 0.9892
Epoch 20/40
105/105 [==============================] - 3s 28ms/step - loss: 0.0626 - accurac
y: 0.9775 - val_loss: 0.0324 - val_accuracy: 0.9976
Epoch 21/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0425 - accurac
y: 0.9847 - val_loss: 0.0503 - val_accuracy: 0.9844
Epoch 22/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0484 - accurac
y: 0.9832 - val_loss: 0.0586 - val_accuracy: 0.9808
Epoch 23/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0590 - accurac
y: 0.9739 - val_loss: 0.0673 - val_accuracy: 0.9808
Epoch 24/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0457 - accurac
y: 0.9829 - val_loss: 0.0440 - val_accuracy: 0.9892
Epoch 25/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0412 - accurac
y: 0.9838 - val_loss: 0.0556 - val_accuracy: 0.9820
Epoch 26/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0478 - accurac
y: 0.9850 - val_loss: 0.0291 - val_accuracy: 0.9904
Epoch 27/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0373 - accurac
y: 0.9853 - val_loss: 0.0521 - val_accuracy: 0.9928
Epoch 28/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0462 - accurac
y: 0.9832 - val_loss: 0.0250 - val_accuracy: 0.9976
Epoch 29/40
105/105 [==============================] - 3s 27ms/step - loss: 0.0371 - accurac
y: 0.9862 - val_loss: 0.0294 - val_accuracy: 0.9952
Epoch 30/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0363 - accurac
y: 0.9874 - val_loss: 0.0383 - val_accuracy: 0.9928
Epoch 31/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0423 - accurac
y: 0.9823 - val_loss: 0.0393 - val_accuracy: 0.9892
Epoch 32/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0592 - accurac
y: 0.9814 - val_loss: 0.0452 - val_accuracy: 0.9868
Epoch 33/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0223 - accurac
y: 0.9925 - val_loss: 0.0455 - val_accuracy: 0.9880
Epoch 34/40
```

```
105/105 [==============================] - 3s 25ms/step - loss: 0.0337 - accurac
y: 0.9907 - val_loss: 0.0335 - val_accuracy: 0.9952
Epoch 35/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0273 - accurac
y: 0.9892 - val_loss: 0.0295 - val_accuracy: 0.9952
Epoch 36/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0383 - accurac
y: 0.9862 - val_loss: 0.0239 - val_accuracy: 0.9928
Epoch 37/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0366 - accurac
y: 0.9892 - val_loss: 0.0359 - val_accuracy: 0.9916
Epoch 38/40
105/105 [==============================] - 3s 25ms/step - loss: 0.0380 - accurac
y: 0.9874 - val_loss: 0.0352 - val_accuracy: 0.9904
Epoch 39/40
105/105 [==============================] - 3s 26ms/step - loss: 0.0302 - accurac
y: 0.9898 - val_loss: 0.0315 - val_accuracy: 0.9916
Epoch 40/40
105/105 [==============================] - 3s 27ms/step - loss: 0.0245 - accurac
y: 0.9913 - val_loss: 0.0154 - val_accuracy: 0.9952
```

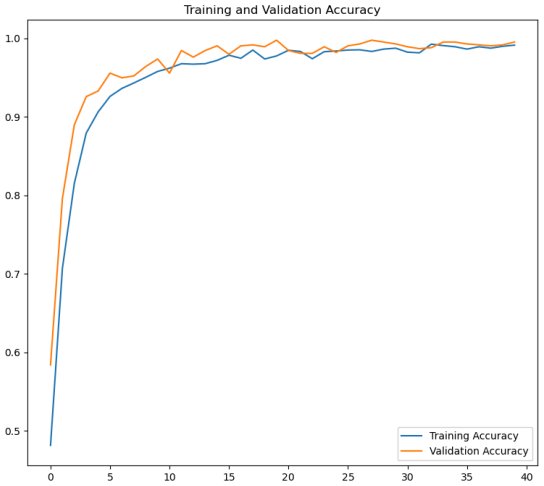
In [34]:

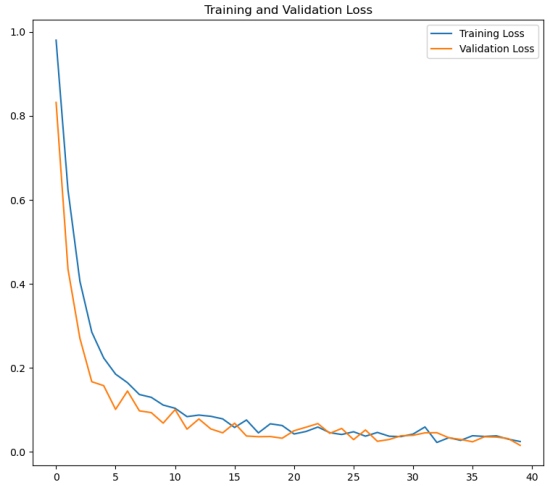
```python
#Let's visualize the results
def plot_training_results(history):
    """
    Visualize results of the model training using `matplotlib`.

    The visualization will include charts for accuracy and loss,
    on the training and as well as validation data sets.

    INPUTS:
        history(tf.keras.callbacks.History):
            Contains data on how the model metrics changed
            over the course of training.

    OUTPUTS:
        None.
    """
    # Get accuracy for training and validation sets
    accuracy = history.history['accuracy']
    validation_accuracy = history.history['val_accuracy']

    # Get loss for training and validation sets
    loss = history.history['loss']
    validation_loss = history.history['val_loss']

    # Get range of epochs to produce common plotting range
    epochs_range = range(epochs)

    # Instantiate plotting figure space
    plt.figure(figsize=(20, 8))

    # Create training/validation accuracy subplot
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, accuracy, label='Training Accuracy')
    plt.plot(epochs_range, validation_accuracy, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    # Create training/validation loss subplot
    plt.subplot(1, 2, 2)
```

```
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, validation_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')

    # Render visualization
    plt.show()
```

In [35]:

```
# Visualize accuracy and loss for training and validation datasets
plot_training_results(history)
```

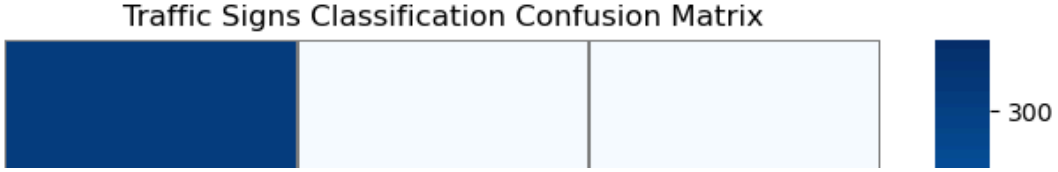


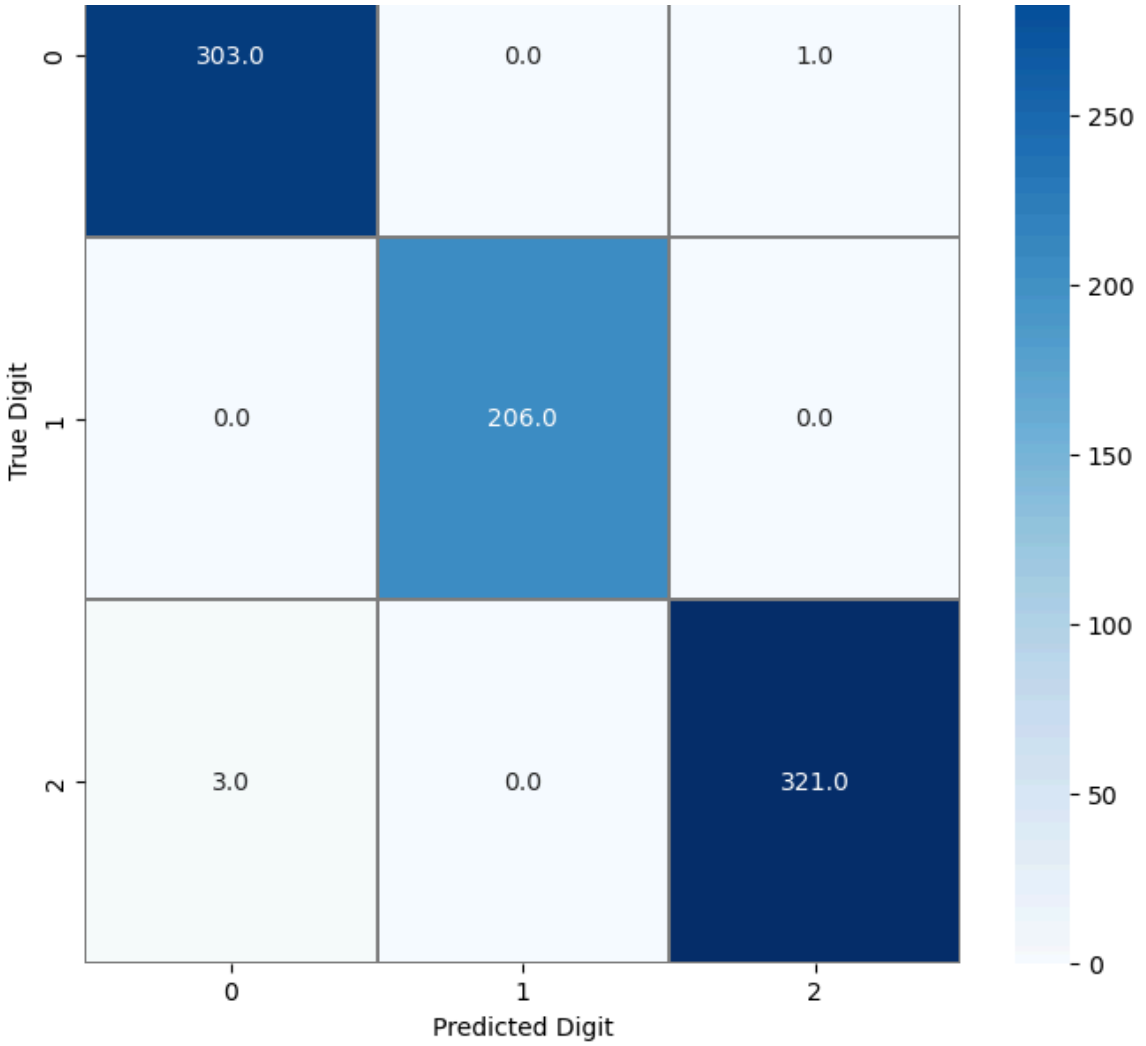


In [36]:

```
# Get predicted class values from fitted model
y_pred = model.predict(X_test_cnn)

# Get class distributions for predicted and true class values
y_pred_classes =  np.argmax(y_pred, axis=1)
y_true =          np.argmax(y_test_cnn, axis=1)

# Create confusion matrix object from class distributions
cmat = confusion_matrix(y_true, y_pred_classes)

# Render confusion matrix as heatmap visualization
figure, axis = plt.subplots(figsize=(8, 8))
sns.heatmap(cmat,
            annot=True,
            linewidths=0.01,
            cmap="Blues",
            linecolor="gray",
            fmt=".1f",
            ax=axis)
plt.xlabel("Predicted Digit")
plt.ylabel("True Digit")
plt.title("Traffic Signs Classification Confusion Matrix")
plt.show()
```

```
27/27 [==============================] - 9s 321ms/step
```





Section 5: Test Data Images (Model has Never Seen These)

In [52]:

```
test_image_data = []
test_labels = []
folders = ['Directional_Signs', 'Speed_Limit', 'Warning_Signs']

# reate a mapping from folder names to numeric labels
label_mapping = {'Directional_Signs': 0, 'Speed_Limit': 1, 'Warning_Signs': 2}

#Loop to go through each class folder
for folder_name in folders:
    folder_path = os.path.join(input_dir_test, folder_name)

    #Check if the folder is an actual directory
    if os.path.isdir(folder_path):
        #Loop through each image in the given class folder
        for image in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image)
```

```
            #Read the image in color (height, width, channels)
            img = cv2.imread(image_path, cv2.IMREAD_COLOR)

            #This will only proceed if the image exists
            if img is not None:
                #Resize image to fixed size (32x32x3) for CNN input
                img_resized = cv2.resize(img, (32, 32))

                #Add the resized image and the label to the lists above
                test_image_data.append(img_resized)
                test_labels.append(label_mapping[folder_name])
            else:
                print(f"The image could not be read: {image_path}")

#Convert the lists into numpy arrays for efficiency
test_image_data = np.array(test_image_data)  # This will have shape (num_image
test_labels = np.array(test_labels)
```

In [53]:

```
#scaling image data to 0-1 range for the model.

test_image_data = test_image_data / 255.0

test_image_data
```

Out[53]:

```
array([[[[0.39215686, 0.39215686, 0.32941176],
         [0.25490196, 0.26666667, 0.19607843],
         [0.29803922, 0.31764706, 0.24705882],
         ...,
         [0.17254902, 0.18039216, 0.16470588],
         [0.19607843, 0.20392157, 0.19215686],
         [0.2627451 , 0.2627451 , 0.25098039]],

        [[0.24313725, 0.24705882, 0.18823529],
         [0.29019608, 0.30588235, 0.23137255],
         [0.17647059, 0.20392157, 0.13333333],
         ...,
         [0.29411765, 0.33333333, 0.30588235],
         [0.32941176, 0.36078431, 0.3372549 ],
         [0.3254902 , 0.34509804, 0.32156863]],

        [[0.68235294, 0.68235294, 0.61960784],
         [0.48627451, 0.49803922, 0.42745098],
         [0.41568627, 0.43921569, 0.36470588],
         ...,
         [0.41960784, 0.49019608, 0.45490196],
         [0.29019608, 0.34901961, 0.31372549],
         [0.31764706, 0.36078431, 0.32941176]],

        ...,

        [[0.41960784, 0.38431373, 0.34117647],
         [0.34117647, 0.32156863, 0.2627451 ],
         [0.65490196, 0.63137255, 0.55686275],
         ...,
         [0.48235294, 0.44313725, 0.41960784],
         [0.75686275, 0.72941176, 0.71764706],
         [0.72941176, 0.70980392, 0.71372549]],
```

```
       [[0.25098039, 0.19607843, 0.17647059],
        [0.57254902, 0.5372549 , 0.49803922],
        [0.68627451, 0.67058824, 0.58039216],
        ...,
        [0.76862745, 0.74901961, 0.72156863],
        [0.6627451 , 0.64313725, 0.62352941],
        [0.85882353, 0.84705882, 0.84705882]],

       [[0.45098039, 0.38823529, 0.37647059],
        [0.89019608, 0.85098039, 0.81960784],
        [0.47843137, 0.45882353, 0.36078431],
        ...,
        [0.4627451 , 0.44705882, 0.41568627],
        [0.58823529, 0.58039216, 0.55686275],
        [0.43137255, 0.42745098, 0.42352941]]],


      [[[0.39215686, 0.39215686, 0.32941176],
        [0.25490196, 0.26666667, 0.19607843],
        [0.29803922, 0.31764706, 0.24705882],
        ...,
        [0.17254902, 0.18039216, 0.16470588],
        [0.19607843, 0.20392157, 0.19215686],
        [0.2627451 , 0.2627451 , 0.25098039]],

       [[0.24313725, 0.24705882, 0.18823529],
        [0.29019608, 0.30588235, 0.23137255],
        [0.17647059, 0.20392157, 0.13333333],
        ...,
        [0.29411765, 0.33333333, 0.30588235],
        [0.32941176, 0.36078431, 0.3372549 ],
        [0.3254902 , 0.34509804, 0.32156863]],

       [[0.68235294, 0.68235294, 0.61960784],
        [0.48627451, 0.49803922, 0.42745098],
        [0.41568627, 0.43921569, 0.36470588],
        ...,
        [0.41960784, 0.49019608, 0.45490196],
        [0.29019608, 0.34901961, 0.31372549],
        [0.31764706, 0.36078431, 0.32941176]],

       ...,

       [[0.41960784, 0.38431373, 0.34117647],
        [0.34117647, 0.32156863, 0.2627451 ],
        [0.65490196, 0.63137255, 0.55686275],
        ...,
        [0.48235294, 0.44313725, 0.41960784],
        [0.75686275, 0.72941176, 0.71764706],
        [0.72941176, 0.70980392, 0.71372549]],

       [[0.25098039, 0.19607843, 0.17647059],
        [0.57254902, 0.5372549 , 0.49803922],
        [0.68627451, 0.67058824, 0.58039216],
        ...,
        [0.76862745, 0.74901961, 0.72156863],
        [0.6627451 , 0.64313725, 0.62352941],
        [0.85882353, 0.84705882, 0.84705882]],

       [[0.45098039, 0.38823529, 0.37647059],
```

```
       [0.89019608, 0.85098039, 0.81960784],
       [0.47843137, 0.45882353, 0.36078431],
       ...,
       [0.4627451 , 0.44705882, 0.41568627],
       [0.58823529, 0.58039216, 0.55686275],
       [0.43137255, 0.42745098, 0.42352941]]],


     [[[0.85098039, 0.88627451, 0.86666667],
       [0.82745098, 0.88235294, 0.86666667],
       [0.83137255, 0.89019608, 0.86666667],
       ...,
       [0.56862745, 0.65490196, 0.63529412],
       [0.55686275, 0.65098039, 0.63137255],
       [0.52941176, 0.64705882, 0.61960784]],

      [[0.84705882, 0.89019608, 0.8745098 ],
       [0.83921569, 0.89411765, 0.87843137],
       [0.83921569, 0.89411765, 0.87058824],
       ...,
       [0.54117647, 0.64313725, 0.63137255],
       [0.5372549 , 0.64313725, 0.62745098],
       [0.54117647, 0.63921569, 0.62352941]],

      [[0.84705882, 0.89019608, 0.88235294],
       [0.84313725, 0.89019608, 0.88235294],
       [0.85882353, 0.90196078, 0.87843137],
       ...,
       [0.49803922, 0.61960784, 0.61568627],
       [0.49411765, 0.61176471, 0.60392157],
       [0.5254902 , 0.60784314, 0.6       ]],

      ...,

      [[0.82745098, 0.88235294, 0.86666667],
       [0.82745098, 0.87843137, 0.87058824],
       [0.85098039, 0.89803922, 0.88627451],
       ...,
       [0.34509804, 0.35686275, 0.98039216],
       [0.39607843, 0.30196078, 0.95686275],
       [0.31764706, 0.29803922, 0.88235294]],

      [[0.83137255, 0.88627451, 0.87058824],
       [0.84705882, 0.89803922, 0.89019608],
       [0.85098039, 0.89411765, 0.88627451],
       ...,
       [0.4627451 , 0.31764706, 1.        ],
       [0.38823529, 0.27843137, 1.        ],
       [0.26666667, 0.19215686, 0.94901961]],

      [[0.83529412, 0.89803922, 0.87058824],
       [0.84313725, 0.89803922, 0.88235294],
       [0.82745098, 0.8745098 , 0.86666667],
       ...,
       [0.29019608, 0.18431373, 0.94509804],
       [0.11372549, 0.16470588, 0.96862745],
       [0.21960784, 0.18431373, 0.99215686]]],


     ...,
```

```
[[[0.49803922, 0.48235294, 0.43529412],
  [0.67843137, 0.68627451, 0.6       ],
  [0.78823529, 0.80784314, 0.69803922],
  ...,
  [0.45882353, 0.5254902 , 0.50980392],
  [0.30980392, 0.38039216, 0.36470588],
  [0.55294118, 0.65490196, 0.64705882]],

 [[0.40392157, 0.45882353, 0.42352941],
  [0.47843137, 0.56862745, 0.49411765],
  [0.45882353, 0.56862745, 0.4627451 ],
  ...,
  [0.50588235, 0.6       , 0.58823529],
  [0.48627451, 0.57647059, 0.56470588],
  [0.65098039, 0.76078431, 0.74901961]],

 [[0.47843137, 0.57254902, 0.5254902 ],
  [0.43137255, 0.5372549 , 0.47843137],
  [0.46666667, 0.56862745, 0.49803922],
  ...,
  [0.48235294, 0.59215686, 0.58431373],
  [0.52156863, 0.63137255, 0.62352941],
  [0.50588235, 0.61176471, 0.6       ]],

 ...,

 [[0.13333333, 0.14509804, 0.14509804],
  [0.1372549 , 0.14901961, 0.14509804],
  [0.11372549, 0.12941176, 0.11372549],
  ...,
  [0.1254902 , 0.12156863, 0.10196078],
  [0.17647059, 0.17254902, 0.15294118],
  [0.18039216, 0.19607843, 0.15294118]],

 [[0.1372549 , 0.14901961, 0.14901961],
  [0.14901961, 0.16078431, 0.15686275],
  [0.16078431, 0.18039216, 0.16078431],
  ...,
  [0.09803922, 0.10588235, 0.08627451],
  [0.23529412, 0.24313725, 0.22352941],
  [0.25490196, 0.2627451 , 0.24313725]],

 [[0.15294118, 0.16470588, 0.16470588],
  [0.15686275, 0.16862745, 0.16470588],
  [0.18823529, 0.20392157, 0.18823529],
  ...,
  [0.10588235, 0.11764706, 0.09803922],
  [0.10980392, 0.11764706, 0.09803922],
  [0.11372549, 0.10980392, 0.12156863]]],


[[[0.23921569, 0.29803922, 0.25490196],
  [0.4       , 0.45490196, 0.41176471],
  [0.4627451 , 0.5254902 , 0.44705882],
  ...,
  [0.45882353, 0.63529412, 0.61568627],
  [0.38039216, 0.54901961, 0.53333333],
  [0.22352941, 0.38039216, 0.36470588]],
```

```
       [[0.2       , 0.2627451 , 0.21960784],
        [0.23137255, 0.28627451, 0.24313725],
        [0.30196078, 0.36862745, 0.29019608],
        ...,
        [0.50980392, 0.68627451, 0.66666667],
        [0.52156863, 0.69019608, 0.67058824],
        [0.29803922, 0.45098039, 0.43529412]],

       [[0.34117647, 0.40392157, 0.35686275],
        [0.23529412, 0.29019608, 0.24705882],
        [0.20784314, 0.28627451, 0.21568627],
        ...,
        [0.58431373, 0.75294118, 0.72941176],
        [0.45490196, 0.61960784, 0.59607843],
        [0.14117647, 0.28627451, 0.26666667]],

       ...,

       [[0.12156863, 0.1254902 , 0.11372549],
        [0.12156863, 0.1254902 , 0.11372549],
        [0.12941176, 0.13333333, 0.1254902 ],
        ...,
        [0.11764706, 0.11764706, 0.09411765],
        [0.12156863, 0.11764706, 0.09803922],
        [0.12156863, 0.11764706, 0.09803922]],

       [[0.10980392, 0.11372549, 0.10196078],
        [0.11372549, 0.11764706, 0.10588235],
        [0.14509804, 0.14901961, 0.1372549 ],
        ...,
        [0.11764706, 0.11764706, 0.09803922],
        [0.1254902 , 0.12156863, 0.10196078],
        [0.12941176, 0.1254902 , 0.10196078]],

       [[0.12156863, 0.1254902 , 0.11372549],
        [0.14117647, 0.14509804, 0.13333333],
        [0.11372549, 0.11764706, 0.10588235],
        ...,
        [0.10196078, 0.10196078, 0.07843137],
        [0.10980392, 0.10588235, 0.08627451],
        [0.11372549, 0.10980392, 0.09019608]]],


      [[[0.21960784, 0.42352941, 0.36078431],
        [0.29019608, 0.51372549, 0.44705882],
        [0.29019608, 0.52941176, 0.45882353],
        ...,
        [0.68235294, 0.72156863, 0.78431373],
        [0.70196078, 0.7372549 , 0.80784314],
        [0.74901961, 0.81960784, 0.85098039]],

       [[0.36862745, 0.52941176, 0.4745098 ],
        [0.32156863, 0.5254902 , 0.45882353],
        [0.2745098 , 0.50588235, 0.43921569],
        ...,
        [0.74117647, 0.81568627, 0.83921569],
        [0.68627451, 0.76470588, 0.77647059],
        [0.71372549, 0.78823529, 0.81176471]],

       [[0.48627451, 0.61960784, 0.56862745],
```

```
       [0.21568627, 0.4       , 0.34117647],
       [0.3254902 , 0.55294118, 0.48627451],
       ...,
       [0.6627451 , 0.7372549 , 0.76078431],
       [0.69803922, 0.77647059, 0.78823529],
       [0.76470588, 0.83921569, 0.8627451 ]],

      ...,

      [[0.27058824, 0.51372549, 0.47058824],
       [0.30588235, 0.49019608, 0.45882353],
       [0.36078431, 0.50196078, 0.47843137],
       ...,
       [0.55294118, 0.67843137, 0.75686275],
       [0.56862745, 0.6627451 , 0.70196078],
       [0.2       , 0.22745098, 0.22352941]],

      [[0.37254902, 0.63529412, 0.58823529],
       [0.18431373, 0.34117647, 0.31764706],
       [0.39215686, 0.50588235, 0.48627451],
       ...,
       [0.64313725, 0.76470588, 0.85882353],
       [0.65490196, 0.75294118, 0.80392157],
       [0.24313725, 0.27058824, 0.26666667]],

      [[0.55686275, 0.67843137, 0.67843137],
       [0.0627451 , 0.15686275, 0.16470588],
       [0.36470588, 0.46666667, 0.47058824],
       ...,
       [0.63529412, 0.74901961, 0.79215686],
       [0.63137255, 0.7254902 , 0.75686275],
       [0.16078431, 0.20392157, 0.23921569]]]])
```

In [54]:

```
#one hot encode the test labels so model understands

test_labels_one_hot = to_categorical(test_labels)
```

In [58]:

```
evaluation = model.evaluate(test_image_data, test_labels_one_hot, batch_size=b

# Print the results
print(f"Test loss: {evaluation[0]}")
print(f"Test accuracy: {evaluation[1]}")
```

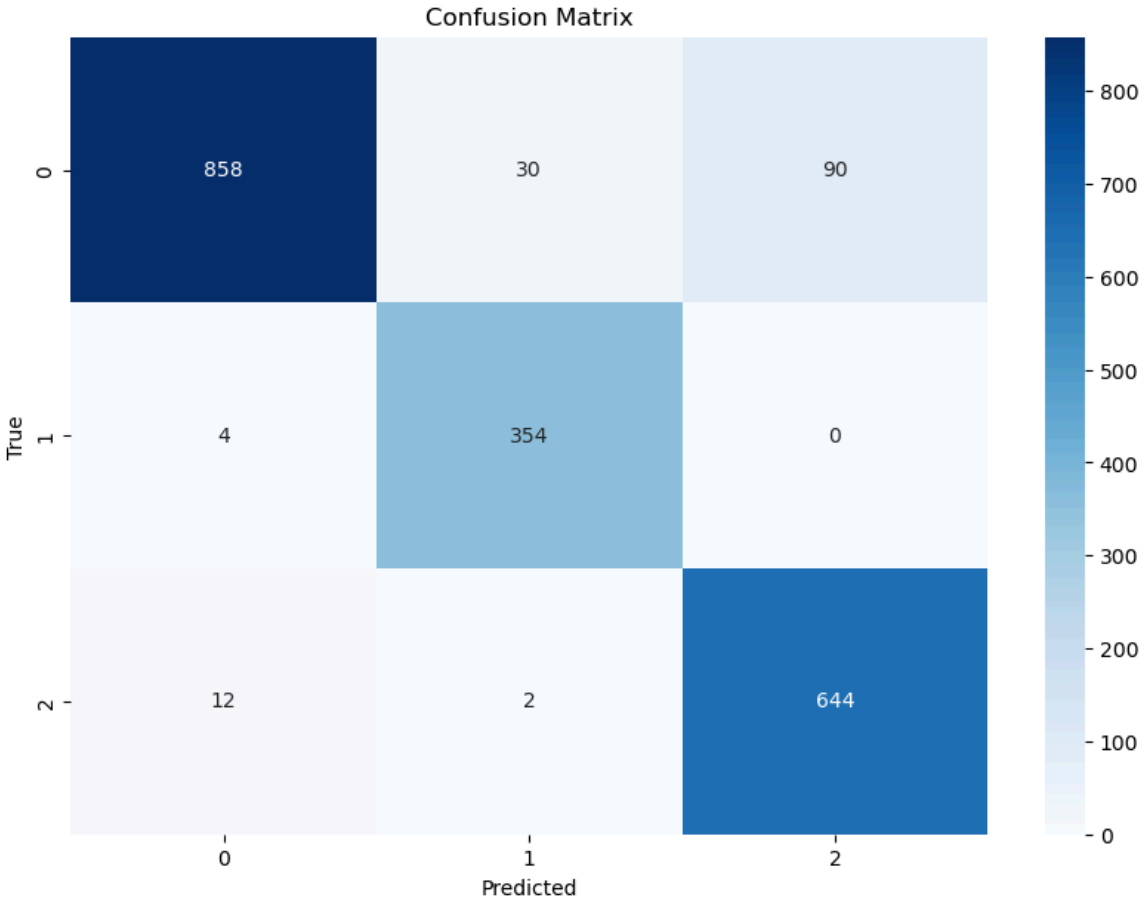
```
63/63 [==============================] - 13s 207ms/step - loss: 0.2576 - accurac
y: 0.9308
Test loss: 0.2575570344924927
Test accuracy: 0.9307923913002014
```

In [56]:

```
# Step 1: Predict on Test Set
predictions = model.predict(test_image_data)

# Step 2: Convert Predictions to Class Labels
# If your model outputs probabilities, use argmax to get the class label
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_labels  # Assuming test_labels are already in integer form

# Step 3: Compute Confusion Matrix
```

```
cm = confusion_matrix(true_classes, predicted_classes)

# Step 4: Plot Confusion Matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
63/63 [==============================] - 1s 8ms/step
```



Section 6: Analysis & Conclusion

Analysis

The image data that I worked with required some extensive preprocessing. Going back to the very beginning where I performed the EDA on my image data, it was revealed that there were 57 classes, with a very heavy class imbalance. Also, it appeared the images height and width dimensions were fairly equal, in the sense the the majority of the pictures in the dataset were around 140x125 pixels. I also looked into the intensities of the RGB channels of all images, which revealed that red had the highest intensity of all the images. The aspect ratios were almost normally distributed among the 4170

images total used for training.

When I started this extensive machine learning notebook, I first started using every class. I deployed a shotgun approach after preprocessing my image data. I looped through my images, resized them to a fixed size of 32x32, and then flattened them into a 1D array for a classic machine learning model to learn. Once this step was done and in an array, I converted the array into a pandas dataframe, where I further normalized the data by dividing the pixel range by 255 to get the pixel data into a range of 0 - 1, which would be more manageable for a machine learning model. I did train test splits, and began a functionalized shot gun approach on machine learning classifier models. The result I first achieved was not good at all. I was getting 0 accuracy among 57 classes. It also appeared there many have been an error with one of my for loops which I later rectified, but I was not getting much success on classic machine learning. However, I did not stop there. I went back and recategorized all of my classes, so instead of having 57 classes, I had 3 classes total, for speed limit signs, for directional signs, and for warning signs.

I had to create new directories in my project path, which required some research and trial and error. After successfully writing a code block to reorganize my images into the new classification, I reapplied the same preprocessing and normalization to my images, this time with adjusted classes. After doing this and running my shot gun approach, I received incredible accuracy right off the bat with my test models, as high as 96 percent! I also employed the Histogram of Oriented Gradients method to extract some heuristics on the images for the classic ML approach. This was interesting and did prove some models to train even better than the classic ML approach which was interesting to see. I was interested in employing a CNN so I moved on to do that.

I had to preprocess my data again as a CNN takes image data as 2D, instead of 1D which classic machine models do. I essentially applied the same preprocessing as before, resizing images to 32x32, appending them to a new list with the classifier, one hot encoding the classifiers, and normalizing the pixel values again by dividing the images by 255, to get the pixels in a range of 0 - 1. I then created my CNN neural network architecture. I trained the model over 40 epochs, and ended with an accuracy of 99.13%. This was really great! Next, I also had to preprocess some test images, which