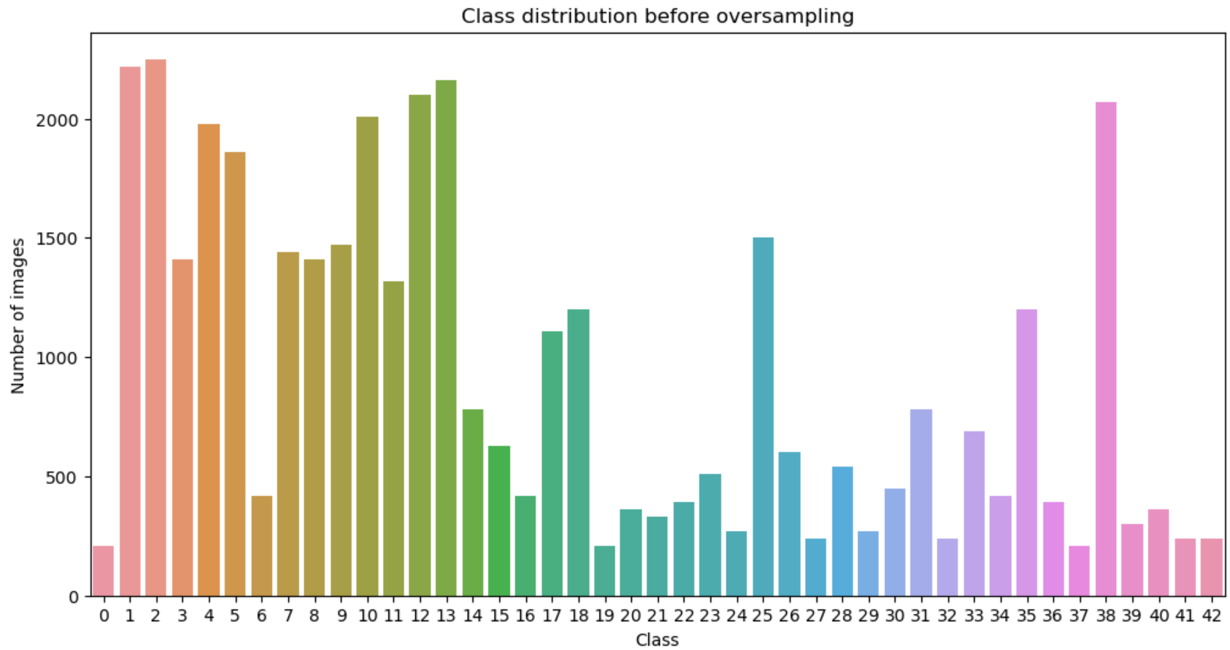
In [1]:
```
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import classification_report

# Load the data
train_data = pd.read_excel("/Users/wkammerait/Desktop/ML Data Sets/data/Tra
test_data = pd.read_excel("/Users/wkammerait/Desktop/ML Data Sets/data/Test
folder_path = "/Users/wkammerait/Desktop/ML Data Sets/data/"
train_data['Path'] = folder_path + train_data['Path']
test_data['Path'] = folder_path + test_data['Path']
```

```
2023-08-08 08:47:39.386673: I tensorflow/core/platform/cpu_feature_guard.
cc:182] This TensorFlow binary is optimized to use available CPU instruct
ions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebu
ild TensorFlow with the appropriate compiler flags.
```

In [2]:
```
# Load and resize images
def load_and_resize_images(data):
    images = []
    for path in data['Path']:
        img = cv2.imread(path)
        img = cv2.resize(img, (30, 30))
        images.append(img)
    return np.array(images)

train_images = load_and_resize_images(train_data)
test_images = load_and_resize_images(test_data)

train_images_gray = np.array([cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img
test_images_gray = np.array([cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img

# Normalize the pixel values to be in the range [0, 1]
train_images = train_images.astype('float32') / 255.0
train_images_gray = train_images_gray.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
test_images_gray = test_images_gray.astype('float32') / 255.0

# Convert the labels to categorical format
y_train = np.array(train_data['ClassId'])
y_test = np.array(test_data['ClassId'])
y_train_cat = to_categorical(y_train, num_classes=43)
y_test_cat = to_categorical(y_test, num_classes=43)
```

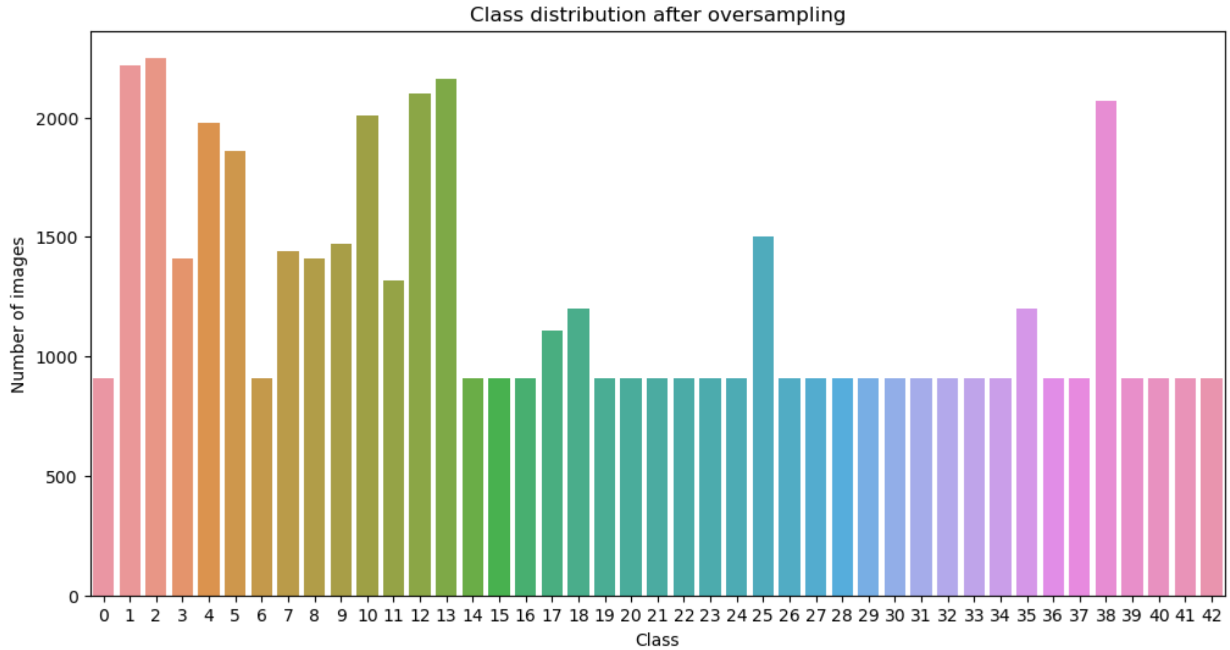
In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Prior to oversampling
class_counts = pd.Series(y_train).value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=class_counts.index, y=class_counts.values)
plt.title("Class distribution before oversampling")
plt.xlabel("Class")
plt.ylabel("Number of images")
plt.show()
```



In [4]:
```
# Handle class imbalance with oversampling
counter = Counter(y_train)
mean_count = int(np.mean([count for label, count in counter.items()]))
strategy = {label: max(count, mean_count) for label, count in counter.items
oversample = RandomOverSampler(sampling_strategy=strategy)
train_images_res, y_train_res = oversample.fit_resample(train_images.reshap
train_images_gray_res, y_train_gray_res = oversample.fit_resample(train_ima
```

In [5]:
```
from keras.preprocessing.image import ImageDataGenerator

# 1. Define the ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=15,      # Random rotations between -15 to 15 degrees
    width_shift_range=0.1,  # Random shift in width by 10%
    height_shift_range=0.1, # Random shift in height by 10%
    shear_range=0.1,        # Shear transformations
    zoom_range=0.1,         # Random zooming up to 10%
    #horizontal_flip=True,   # Randomly flip images horizontally
    fill_mode='nearest'     # Fill missing pixels using nearest neighbours
)

# 2. Apply Augmentation to Specific Classes
under_represented_classes = [20, 27, 40]
augmented_images = []
augmented_labels = []

for class_id in under_represented_classes:
    # Filter images of the specific class
    mask = y_train == class_id
    class_images = train_images[mask]

    # Augment the images. This will generate batches, so we loop through th
    # until we've produced a desired number of augmented samples for the cl
    # For instance, you could double the number of samples with num_augment
    num_augmented = 0
    for x_batch, y_batch in datagen.flow(class_images, np.zeros(len(class_i
        augmented_images.extend(x_batch)
        augmented_labels.extend([class_id]*len(x_batch))
        num_augmented += len(x_batch)
        if num_augmented >= len(class_images):
            break

# Convert augmented data to numpy arrays
augmented_images = np.array(augmented_images)
augmented_labels = np.array(augmented_labels)

# Add augmented data to the original data
train_images = np.concatenate([train_images, augmented_images], axis=0)
y_train = np.concatenate([y_train, augmented_labels], axis=0)

# Update the y_train_cat with the new y_train data
y_train_cat = to_categorical(y_train, num_classes=43)
```

In [6]:
```
# After oversampling
class_counts_res = pd.Series(y_train_res).value_counts()
plt.figure(figsize=(12,6))
sns.barplot(x=class_counts_res.index, y=class_counts_res.values)
plt.title("Class distribution after oversampling")
plt.xlabel("Class")
plt.ylabel("Number of images")
plt.show()
```



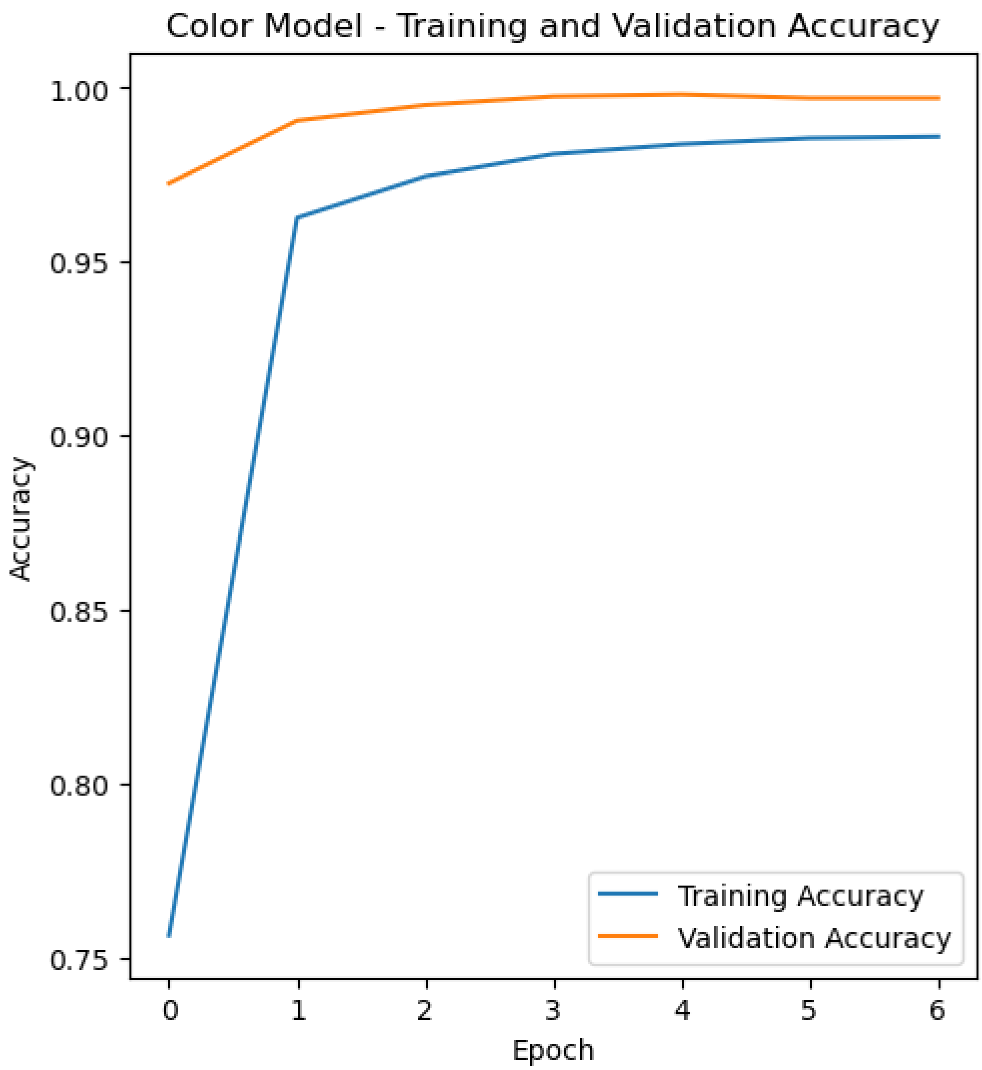
In [9]:
```
# Reshape data
train_images_res = train_images_res.reshape(-1, 30, 30, 3)
train_images_gray_res = train_images_gray_res.reshape(-1, 30, 30, 1)
y_train_res_cat = to_categorical(y_train_res, num_classes=43)
y_train_gray_res_cat = to_categorical(y_train_gray_res, num_classes=43)
```

In [16]:
```
from tensorflow.keras.layers import Dropout
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=2)

# Define and compile the color model
model_color = Sequential()
model_color.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_sha
model_color.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model_color.add(MaxPooling2D(pool_size=(2, 2)))
model_color.add(Dropout(0.25)) # Dropout layer added
model_color.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model_color.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model_color.add(MaxPooling2D(pool_size=(2, 2)))
model_color.add(Dropout(0.25)) # Dropout layer added
model_color.add(Flatten())
model_color.add(Dense(256, activation='relu'))
model_color.add(Dropout(0.5)) # Dropout layer added
model_color.add(Dense(43, activation='softmax'))
model_color.compile(optimizer=Adam(), loss='categorical_crossentropy', metr

# Train the color model
history_color = model_color.fit(train_images_res, y_train_res_cat, batch_si

# Define and compile the grayscale model
model_gray = Sequential()
model_gray.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shap
model_gray.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model_gray.add(MaxPooling2D(pool_size=(2, 2)))
model_gray.add(Dropout(0.25)) # Dropout layer added
model_gray.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model_gray.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model_gray.add(MaxPooling2D(pool_size=(2, 2)))
model_gray.add(Dropout(0.25)) # Dropout layer added
model_gray.add(Flatten())
model_gray.add(Dense(256, activation='relu'))
model_gray.add(Dropout(0.5)) # Dropout layer added
model_gray.add(Dense(43, activation='softmax'))
model_gray.compile(optimizer=Adam(), loss='categorical_crossentropy', metri

# Train the grayscale model
history_gray = model_gray.fit(train_images_gray_res, y_train_gray_res_cat,
```

```
Epoch 1/10
1310/1310 [==============================] - 128s 97ms/step - loss: 0.866
0 - accuracy: 0.7563 - val_loss: 0.1172 - val_accuracy: 0.9726
Epoch 2/10
1310/1310 [==============================] - 166s 126ms/step - loss: 0.12
63 - accuracy: 0.9627 - val_loss: 0.0349 - val_accuracy: 0.9906
Epoch 3/10
1310/1310 [==============================] - 171s 131ms/step - loss: 0.08
77 - accuracy: 0.9745 - val_loss: 0.0182 - val_accuracy: 0.9950
Epoch 4/10
1310/1310 [==============================] - 142s 108ms/step - loss: 0.06
32 - accuracy: 0.9810 - val_loss: 0.0125 - val_accuracy: 0.9975
Epoch 5/10
1310/1310 [==============================] - 176s 135ms/step - loss: 0.05
64 - accuracy: 0.9838 - val_loss: 0.0080 - val_accuracy: 0.9981
Epoch 6/10
1310/1310 [==============================] - 168s 128ms/step - loss: 0.04
87 - accuracy: 0.9855 - val_loss: 0.0135 - val_accuracy: 0.9970
Epoch 7/10
1310/1310 [==============================] - 143s 109ms/step - loss: 0.04
84 - accuracy: 0.9859 - val_loss: 0.0116 - val_accuracy: 0.9970
Epoch 1/10
1310/1310 [==============================] - 135s 102ms/step - loss: 1.08
58 - accuracy: 0.6995 - val_loss: 0.1822 - val_accuracy: 0.9479
Epoch 2/10
1310/1310 [==============================] - 132s 101ms/step - loss: 0.14
97 - accuracy: 0.9549 - val_loss: 0.0481 - val_accuracy: 0.9887
Epoch 3/10
1310/1310 [==============================] - 157s 120ms/step - loss: 0.08
89 - accuracy: 0.9733 - val_loss: 0.0171 - val_accuracy: 0.9961
Epoch 4/10
1310/1310 [==============================] - 156s 119ms/step - loss: 0.07
23 - accuracy: 0.9776 - val_loss: 0.0144 - val_accuracy: 0.9968
Epoch 5/10
1310/1310 [==============================] - 160s 122ms/step - loss: 0.05
55 - accuracy: 0.9831 - val_loss: 0.0127 - val_accuracy: 0.9965
Epoch 6/10
1310/1310 [==============================] - 166s 127ms/step - loss: 0.05
09 - accuracy: 0.9841 - val_loss: 0.0056 - val_accuracy: 0.9986
Epoch 7/10
1310/1310 [==============================] - 156s 119ms/step - loss: 0.04
11 - accuracy: 0.9872 - val_loss: 0.0040 - val_accuracy: 0.9993
Epoch 8/10
1310/1310 [==============================] - 154s 118ms/step - loss: 0.03
98 - accuracy: 0.9876 - val_loss: 0.0014 - val_accuracy: 0.9999
Epoch 9/10
1310/1310 [==============================] - 155s 118ms/step - loss: 0.03
55 - accuracy: 0.9889 - val_loss: 0.0015 - val_accuracy: 0.9994
Epoch 10/10
1310/1310 [==============================] - 160s 122ms/step - loss: 0.03
43 - accuracy: 0.9898 - val_loss: 0.0025 - val_accuracy: 0.9996
```

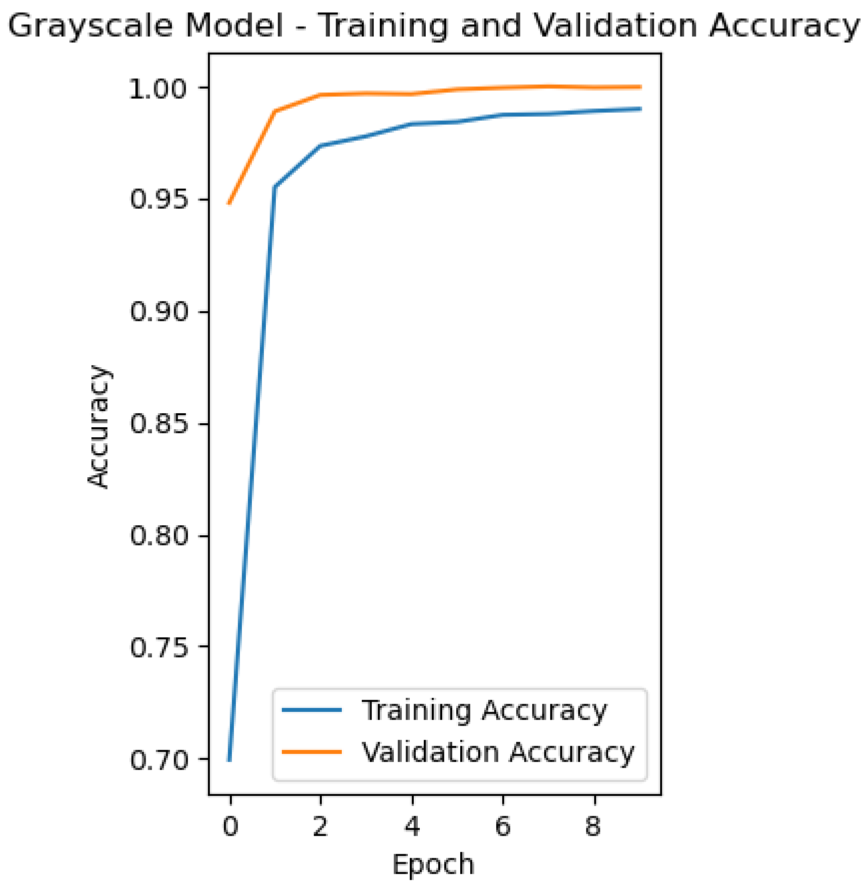
In [17]:
```
# Plot the training and validation accuracy
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history_color.history['accuracy'], label='Training Accuracy')
plt.plot(history_color.history['val_accuracy'], label='Validation Accuracy'
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Color Model - Training and Validation Accuracy')
```

Out[17]:
```
Text(0.5, 1.0, 'Color Model - Training and Validation Accuracy')
```



In [18]:
```
plt.subplot(1, 2, 2)
plt.plot(history_gray.history['accuracy'], label='Training Accuracy')
plt.plot(history_gray.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Grayscale Model - Training and Validation Accuracy')
```

Out[18]:
```
Text(0.5, 1.0, 'Grayscale Model - Training and Validation Accuracy')
```



In [19]:
```
# Evaluate the color model on test data
test_loss, test_accuracy = model_color.evaluate(test_images, y_test_cat)
print('Color Model Test Accuracy:', test_accuracy)

# Evaluate the grayscale model on test data
test_images_gray = test_images_gray.reshape(-1, 30, 30, 1)
test_loss, test_accuracy = model_gray.evaluate(test_images_gray, y_test_cat
print('Grayscale Model Test Accuracy:', test_accuracy)
```

```
395/395 [==============================] - 8s 20ms/step - loss: 0.1006 -
accuracy: 0.9740
Color Model Test Accuracy: 0.9739509224891663
395/395 [==============================] - 8s 19ms/step - loss: 0.1334 -
accuracy: 0.9711
Grayscale Model Test Accuracy: 0.9711005687713623
```

In [20]:
```
# Get predictions for both models
y_pred_color = np.argmax(model_color.predict(test_images), axis=-1)
y_pred_gray = np.argmax(model_gray.predict(test_images_gray), axis=-1)

# Print the classification report for both models
print("Classification report for color model:")
print(classification_report(y_test, y_pred_color))

print("Classification report for grayscale model:")
print(classification_report(y_test, y_pred_gray))
```

```
395/395 [==============================] - 8s 20ms/step
395/395 [==============================] - 8s 19ms/step
Classification report for color model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        60
           1       0.96      0.99      0.97       720
           2       0.98      0.99      0.99       750
           3       0.98      0.95      0.96       450
           4       0.99      1.00      0.99       660
           5       0.93      0.99      0.96       630
           6       1.00      0.87      0.93       150
           7       1.00      0.98      0.99       450
           8       0.97      0.99      0.98       450
           9       0.98      1.00      0.99       480
          10       1.00      0.98      0.99       660
          11       0.96      0.99      0.97       420
          12       0.98      0.99      0.99       690
          13       0.98      1.00      0.99       720
          14       1.00      1.00      1.00       270
          15       0.95      1.00      0.97       210
          16       1.00      1.00      1.00       150
          17       1.00      0.96      0.98       360
          18       0.98      0.92      0.95       390
          19       1.00      1.00      1.00        60
          20       0.98      1.00      0.99        90
          21       0.97      0.71      0.82        90
          22       1.00      0.95      0.97       120
          23       0.85      0.99      0.91       150
          24       1.00      0.97      0.98        90
          25       0.98      0.99      0.98       480
          26       0.93      0.89      0.91       180
          27       0.89      0.52      0.65        60
          28       1.00      0.96      0.98       150
          29       0.95      1.00      0.97        90
          30       1.00      0.69      0.81       150
          31       0.92      0.99      0.95       270
          32       1.00      1.00      1.00        60
          33       1.00      0.99      0.99       210
          34       0.89      0.99      0.94       120
          35       1.00      0.99      0.99       390
          36       0.99      0.95      0.97       120
          37       1.00      0.98      0.99        60
          38       0.97      0.99      0.98       690
          39       0.92      0.98      0.95        90
          40       0.97      0.99      0.98        90
          41       1.00      0.87      0.93        60
          42       1.00      1.00      1.00        90

    accuracy                           0.97     12630
   macro avg       0.97      0.95      0.96     12630
weighted avg       0.97      0.97      0.97     12630

Classification report for grayscale model:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        60
```

```
           1       0.96      0.99      0.97       720
           2       0.96      0.99      0.97       750
           3       0.95      0.98      0.97       450
           4       0.98      0.98      0.98       660
           5       0.97      0.93      0.95       630
           6       1.00      0.85      0.92       150
           7       0.98      0.98      0.98       450
           8       1.00      0.94      0.97       450
           9       0.98      1.00      0.99       480
          10       1.00      0.99      0.99       660
          11       0.97      0.99      0.98       420
          12       0.96      0.97      0.96       690
          13       1.00      1.00      1.00       720
          14       0.99      1.00      1.00       270
          15       0.86      1.00      0.93       210
          16       0.99      1.00      0.99       150
          17       1.00      0.97      0.98       360
          18       0.99      0.93      0.96       390
          19       0.98      1.00      0.99        60
          20       0.78      1.00      0.88        90
          21       1.00      0.79      0.88        90
          22       1.00      0.86      0.92       120
          23       0.98      1.00      0.99       150
          24       0.95      0.96      0.95        90
          25       0.98      0.98      0.98       480
          26       0.88      0.97      0.92       180
          27       0.86      0.52      0.65        60
          28       0.96      0.99      0.98       150
          29       1.00      0.99      0.99        90
          30       1.00      0.87      0.93       150
          31       0.99      0.99      0.99       270
          32       0.97      1.00      0.98        60
          33       0.91      1.00      0.95       210
          34       0.93      1.00      0.96       120
          35       1.00      0.98      0.99       390
          36       0.98      1.00      0.99       120
          37       0.98      1.00      0.99        60
          38       0.99      0.98      0.99       690
          39       0.99      0.98      0.98        90
          40       0.83      0.88      0.85        90
          41       0.95      0.87      0.90        60
          42       0.97      1.00      0.98        90

    accuracy                           0.97     12630
   macro avg       0.96      0.95      0.96     12630
weighted avg       0.97      0.97      0.97     12630
```

Looking at the classification reports, the color model performed with slightly better macro precision and otherwise performed the same (for overall metrics). Increasing the patience would further separate the color model from the grayscale model (meaning I tried this and the model performed better), but the project requires we set the patience = 2 for early stopping.

Additionally, while both models poorly identified pedestrians with excessive false negatives, the color model performed much better at identifying "Dangerous Curve Right" signs and roundabout mandatory signs.

Relatively strong F1 scores (> 0.9, as specified in the instructions) indicate a relatively healthy balance between precision and recall.