Traffic Sign Classifier

資安學程 NQ6121070 周姵彣

```
import pickle


training_file = "C:/Users/Pei/train.p"
validation_file = "C:/Users/Pei/valid.p"
testing_file = "C:/Users/Pei/test.p"


with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)


x_train, y_train = train['features'], train['labels']
x_valid, y_valid = valid['features'], valid['labels']
x_test, y_test = test['features'], test['labels']


print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("x_valid shape:", x_valid.shape)
print("y_valid shape:", y_valid.shape)
print("x_test shape:", x_test.shape)
```

```
print("y_test shape:", y_test.shape)
```

```
x_train shape: (34799, 32, 32, 3)
y_train shape: (34799,)
x_valid shape: (4410, 32, 32, 3)
y_valid shape: (4410,)
x_test shape: (12630, 32, 32, 3)
y_test shape: (12630,)
```

這段程式碼載入了預先保存的訓練、驗證和測試數據，這些數據使用了 pickle 格式進行保存。

```
import numpy as np
n_train = len(x_train)
n_test = len(x_test)
image_shape = x_train[0].shape
n_classes = len(np.unique(y_train))
print("Number of training examples =" , n_train)
print("Number of testing examples =" , n_test)
print("Image data shape =" , image_shape)
print("Number of classes =" , n_classes)
```

```
Number of training examples = 34799
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

這段程式碼計算了訓練集和測試集的一些基本屬性，並對這些屬性

進行了打印。

輸出結果顯示了每個數據集的特徵和標籤的形狀

Step 1: Dataset Summary & Exploration

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt

# Assuming x_train, y_train are already loaded

index = random.randint(0, len(x_train))
image = x_train[index].squeeze()

plt.figure(figsize=(1, 1))
plt.imshow(image)
plt.show()

print(y_train[index])
```

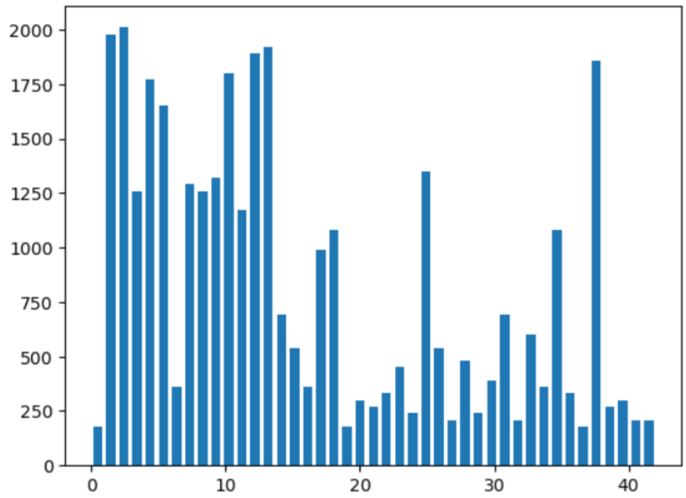
這段程式碼假設已經加載了訓練集 x_train 和其對應的標籤 y_train，執行後隨機展示了數據集中交通標誌的影像。

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Assuming y_train is defined

n_classes = len(np.unique(y_train))
hist, bins = np.histogram(y_train, bins=n_classes)
width = 0.7 * (bins[1] - bins[0])
center = (bins[:-1] + bins[1:]) / 2

plt.bar(center, hist, align='center', width=width)
plt.show()
```



這段程式碼用於繪製類別標籤的直方圖，以可視化每個類別在訓練集中的分佈情況。

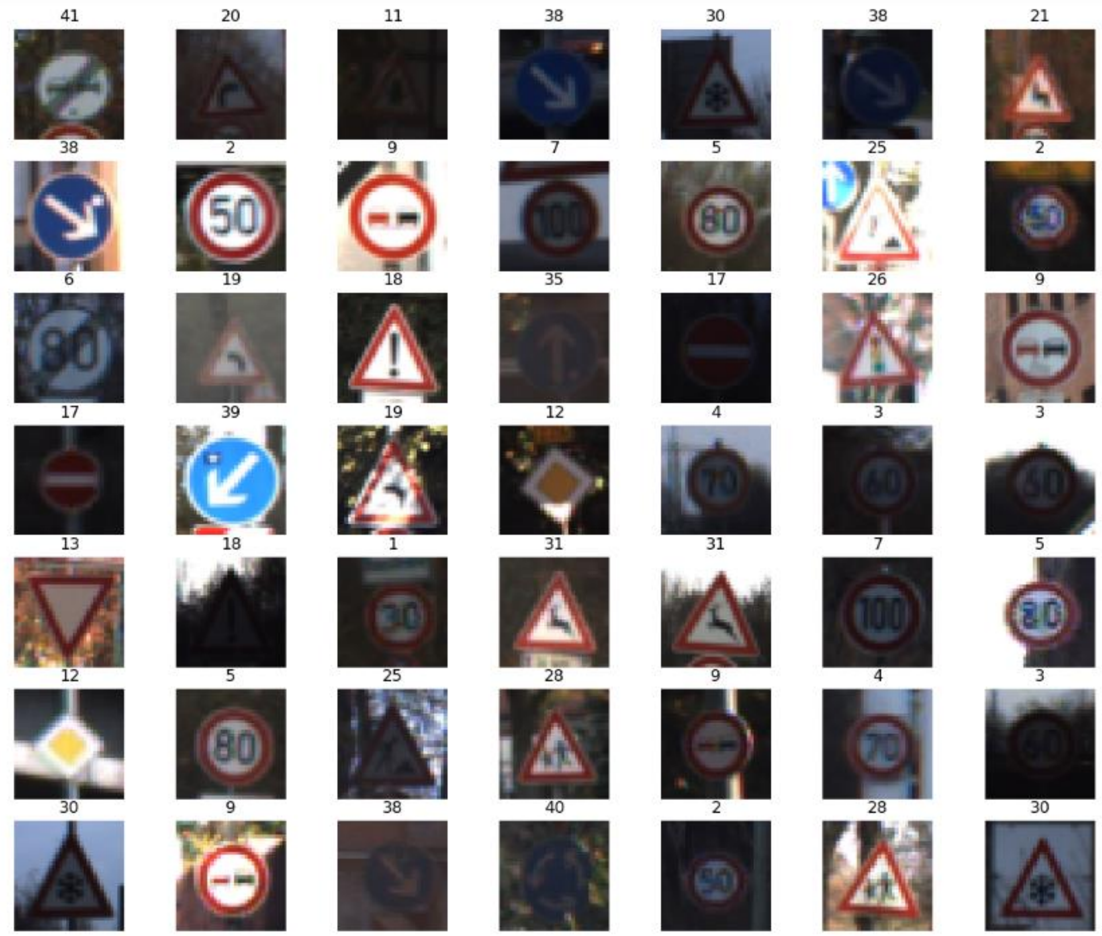
Step 2: Design and Test a Model Architecture

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt

# Assuming x_train, y_train are defined

fig, axs = plt.subplots(7, 7, figsize=(15, 12))
fig.subplots_adjust(hspace=0.2, wspace=0.001)
axs = axs.ravel()
for i in range(49):
    index = random.randint(0, len(x_train))
```

```
    image = x_train[index].squeeze()    # Add .squeeze() to remove
singleton dimension
    axs[i].axis('off')
    axs[i].imshow(image, cmap='jet')    # Use 'jet' colormap for rich
tones
    axs[i].set_title(y_train[index])
plt.show()
```



這段程式碼用於顯示訓練集中隨機選取的 49 張圖像，每行顯示 7 張，每張圖像的標題顯示了對應的類別標籤。

```
import tensorflow as tf

# Assuming x_train is defined somewhere before

def gray_and_equalize_hist(img):
    gray = tf.image.rgb_to_grayscale(img)
    equ = tf.image.adjust_contrast(gray, 2.0) # Adjust contrast as an alternative
    return equ

x_train = tf.convert_to_tensor([gray_and_equalize_hist(img) for img in x_train])
x_test = tf.convert_to_tensor([gray_and_equalize_hist(img) for img in x_test])

print('Preprocessed the data')
```

```
Preprocessed the data
```

這段程式碼進行了數據預處理。

```
import tensorflow as tf
from tensorflow.keras.layers import Flatten

def LeNet(x):
    mu = 0
    sigma = 0.1

    conv1_W = tf.Variable(tf.random.truncated_normal(shape=(5, 5, 1, 6), mean=mu, stddev=sigma))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1 = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b

    conv1 = tf.nn.relu(conv1)
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    conv2_W = tf.Variable(tf.random.truncated_normal(shape=(5, 5, 6, 16), mean=mu, stddev=sigma))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2 = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b

    conv2 = tf.nn.relu(conv2)
```

```
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2,
1], padding='VALID')

    fc0 = Flatten()(conv2)

    fc1_W = tf.Variable(tf.random.truncated_normal(shape=(400, 120),
mean=mu, stddev=sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1 = tf.matmul(fc0, fc1_W) + fc1_b

    fc1 = tf.nn.relu(fc1)

    fc2_W = tf.Variable(tf.random.truncated_normal(shape=(120, 84),
mean=mu, stddev=sigma))
    fc2_b = tf.Variable(tf.zeros(84))
    fc2 = tf.matmul(fc1, fc2_W) + fc2_b

    fc2 = tf.nn.relu(fc2)

    fc3_W = tf.Variable(tf.random.truncated_normal(shape=(84, 10),
mean=mu, stddev=sigma))
    fc3_b = tf.Variable(tf.zeros(10))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits
```

這段程式碼定義了一個簡單的 LeNet 網絡模型，其中包含了卷積層、池化層和全連接層。

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pickle


with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

x_train, y_train = train['features'], train['labels']
```

```
x_valid, y_valid = valid['features'], valid['labels']

x_test, y_test = test['features'], test['labels']


# 將 RGB 圖像轉換為灰度圖像

x_train_gray = np.dot(x_train[...,:3], [0.2989, 0.5870, 0.1140])

x_valid_gray = np.dot(x_valid[...,:3], [0.2989, 0.5870, 0.1140])

x_test_gray = np.dot(x_test[...,:3], [0.2989, 0.5870, 0.1140])


# 將通道維度添加到灰度圖像中

x_train_gray = np.expand_dims(x_train_gray, axis=-1)

x_valid_gray = np.expand_dims(x_valid_gray, axis=-1)

x_test_gray = np.expand_dims(x_test_gray, axis=-1)


# 確定圖像形狀和類別數量

image_shape = x_train_gray[0].shape

n_classes = len(np.unique(y_train))


# 定義 LeNet 模型

def LeNet(input_shape, num_classes):

    model = tf.keras.models.Sequential([

        tf.keras.layers.Conv2D(6, (5, 5), activation='relu',
input_shape=input_shape),
```

```
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D(16, (5, 5), activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(120, activation='relu'),
        tf.keras.layers.Dense(84, activation='relu'),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])
    return model

# 創建 LeNet 模型
model = LeNet(input_shape=image_shape, num_classes=n_classes)

# 編譯模型
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 訓練模型
history = model.fit(x_train_gray, y_train, epochs=30, batch_size=128,
validation_data=(x_valid_gray, y_valid))
```

評估模型

test_loss, test_accuracy = model.evaluate(x_test_gray, y_test)

print(f"Test Accuracy = {test_accuracy:.3f}")

```
Train on 34799 samples, validate on 4410 samples
Epoch 1/30
34799/34799 [==============================] - 2s 69us/sample - loss: 3.0843 - accuracy: 0.4435 - val_loss: 1.3450 - val_accura
cy: 0.6776
Epoch 2/30
34799/34799 [==============================] - 2s 51us/sample - loss: 0.6494 - accuracy: 0.8317 - val_loss: 0.9255 - val_accura
cy: 0.7923
Epoch 3/30
34799/34799 [==============================] - 2s 51us/sample - loss: 0.3684 - accuracy: 0.9052 - val_loss: 0.7305 - val_accura
cy: 0.8372
Epoch 4/30
34799/34799 [==============================] - 2s 59us/sample - loss: 0.2531 - accuracy: 0.9352 - val_loss: 0.5866 - val_accura
cy: 0.8830
Epoch 5/30
34799/34799 [==============================] - 2s 61us/sample - loss: 0.1785 - accuracy: 0.9538 - val_loss: 0.6634 - val_accura
cy: 0.8771
Epoch 6/30
34799/34799 [==============================] - 2s 52us/sample - loss: 0.1289 - accuracy: 0.9679 - val_loss: 0.6149 - val_accura
cy: 0.8855
Epoch 7/30
34799/34799 [==============================] - 2s 53us/sample - loss: 0.1091 - accuracy: 0.9716 - val_loss: 0.7190 - val_accura
cy: 0.8830
Epoch 8/30
34799/34799 [==============================] - 3s 77us/sample - loss: 0.0954 - accuracy: 0.9752 - val_loss: 0.6509 - val_accura
cy: 0.8914
Epoch 9/30
34799/34799 [==============================] - 3s 89us/sample - loss: 0.0824 - accuracy: 0.9776 - val_loss: 0.7269 - val_accura
cy: 0.8891
Epoch 10/30
34799/34799 [==============================] - 3s 73us/sample - loss: 0.0786 - accuracy: 0.9788 - val_loss: 0.6185 - val_accura
cy: 0.9007
Epoch 11/30
34799/34799 [==============================] - 2s 66us/sample - loss: 0.0676 - accuracy: 0.9817 - val_loss: 0.6747 - val_accura
cy: 0.9007
Epoch 12/30
34799/34799 [==============================] - 2s 65us/sample - loss: 0.0586 - accuracy: 0.9848 - val_loss: 0.6953 - val_accura
cy: 0.9107
```

```
Epoch 13/30
34799/34799 [==============================] - 2s 62us/sample - loss: 0.0581 - accuracy: 0.9844 - val_loss: 0.6313 - val_accura
cy: 0.9082
Epoch 14/30
34799/34799 [==============================] - 2s 62us/sample - loss: 0.0497 - accuracy: 0.9865 - val_loss: 0.7422 - val_accura
cy: 0.9120
Epoch 15/30
34799/34799 [==============================] - 2s 62us/sample - loss: 0.0439 - accuracy: 0.9878 - val_loss: 0.7015 - val_accura
cy: 0.9095
Epoch 16/30
34799/34799 [==============================] - 2s 60us/sample - loss: 0.0438 - accuracy: 0.9885 - val_loss: 0.5721 - val_accura
cy: 0.9122
Epoch 17/30
34799/34799 [==============================] - 2s 53us/sample - loss: 0.0538 - accuracy: 0.9863 - val_loss: 0.6117 - val_accura
cy: 0.9145
Epoch 18/30
34799/34799 [==============================] - 2s 52us/sample - loss: 0.0461 - accuracy: 0.9876 - val_loss: 0.7062 - val_accura
cy: 0.9045
Epoch 19/30
34799/34799 [==============================] - 2s 54us/sample - loss: 0.0449 - accuracy: 0.9888 - val_loss: 0.6033 - val_accura
cy: 0.9186
Epoch 20/30
34799/34799 [==============================] - 2s 53us/sample - loss: 0.0381 - accuracy: 0.9901 - val_loss: 0.7746 - val_accura
cy: 0.9109
Epoch 21/30
34799/34799 [==============================] - 2s 60us/sample - loss: 0.0369 - accuracy: 0.9900 - val_loss: 0.5891 - val_accura
cy: 0.9231
Epoch 22/30
34799/34799 [==============================] - 2s 53us/sample - loss: 0.0369 - accuracy: 0.9905 - val_loss: 0.7045 - val_accura
cy: 0.9147
Epoch 23/30
34799/34799 [==============================] - 2s 60us/sample - loss: 0.0219 - accuracy: 0.9943 - val_loss: 0.9204 - val_accura
cy: 0.9039
Epoch 24/30
34799/34799 [==============================] - 2s 60us/sample - loss: 0.0497 - accuracy: 0.9875 - val_loss: 1.0775 - val_accura
cy: 0.8980
```

```
Epoch 25/30
34799/34799 [==============================] - 2s 56us/sample - loss: 0.0360 - accuracy: 0.9910 - val_loss: 0.6875 - val_accura
cy: 0.9222
Epoch 26/30
34799/34799 [==============================] - 2s 55us/sample - loss: 0.0178 - accuracy: 0.9951 - val_loss: 0.8162 - val_accura
cy: 0.9177
Epoch 27/30
34799/34799 [==============================] - 2s 53us/sample - loss: 0.0290 - accuracy: 0.9922 - val_loss: 0.9093 - val_accura
cy: 0.9163
Epoch 28/30
34799/34799 [==============================] - 2s 53us/sample - loss: 0.0443 - accuracy: 0.9891 - val_loss: 0.8066 - val_accura
cy: 0.9234
Epoch 29/30
34799/34799 [==============================] - 2s 53us/sample - loss: 0.0330 - accuracy: 0.9920 - val_loss: 0.8720 - val_accura
cy: 0.9220
Epoch 30/30
34799/34799 [==============================] - 2s 54us/sample - loss: 0.0386 - accuracy: 0.9906 - val_loss: 0.8094 - val_accura
cy: 0.9229
Test Accuracy = 0.906
```

這段程式碼將經典的 LeNet 模型應用於灰度圖像數據，並使用 TensorFlow 2.x 進行模型的定義、編譯、訓練和評估。

Step 3: Test a Model on New Images

import cv2

import numpy as np

import matplotlib.pyplot as plt

image_paths = ["C:/Users/Pei/image1.png", "C:/Users/Pei/image2.png", "C:/Users/Pei/image3.png", "C:/Users/Pei/image4.png", "C:/Users/Pei/image5.png"]

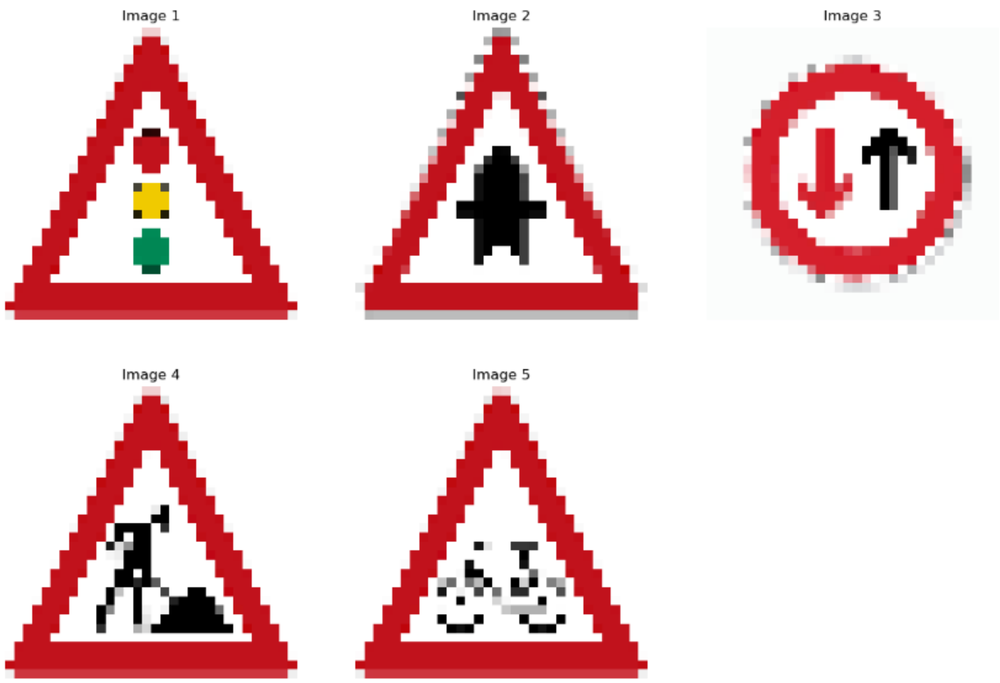
images = [cv2.imread(image_path) for image_path in image_paths]

```
# 從 BGR 轉 RGB

images_rgb = [cv2.cvtColor(image, cv2.COLOR_BGR2RGB) for image in images]


resized_images = [cv2.resize(image, (32, 32)) for image in images_rgb]


plt.figure(figsize=(15, 10))
for i, image in enumerate(resized_images):
    plt.subplot(2, 3, i+1)
    plt.imshow(image)
    plt.title(f"Image {i+1}")
    plt.axis('off')
plt.show()
```



```
# 將影像轉為灰階並進行正規化

def preprocess_image(image):
    image_gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    image_normalized = (image_gray - 128.0) / 128.0
    return image_normalized[..., np.newaxis]


preprocessed_images = np.array([preprocess_image(image) for image in resized_images])


# 使用模型進行預測

predictions = model.predict(preprocessed_images)


# 找到每個預測的類別

predicted_classes = np.argmax(predictions, axis=1)


# 載入標籤名稱

import pandas as pd
sign_names = pd.read_csv('signnames.csv')
class_names = sign_names['SignName'].values


# 輸出預測結果

for i, predicted_class in enumerate(predicted_classes):
```

    print(f"Image {i+1} is predicted as: {class_names[predicted_class]}")

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
Image 1 is predicted as: General caution
Image 2 is predicted as: Road work
Image 3 is predicted as: Speed limit (70km/h)
Image 4 is predicted as: No passing
Image 5 is predicted as: No entry
```

這段程式碼用於對一組影像進行預測並輸出預測結果，可應用於圖像分類等任務，然而由結果可知此預測的準確率有 80%。

Analyze Performance

actual_classes = [18, 25, 4, 9, 38]

比較預測結果和實際結果

correct_predictions = sum([1 for i in range(len(actual_classes)) if actual_classes[i] == predicted_classes[i]])

total_images = len(actual_classes)

accuracy = correct_predictions / total_images

print(f"Model accuracy on new images: {accuracy:.2f}")

這段程式碼有助於評估模型在新影像上的表現，並提供了一個量化

的指標來衡量模型的準確性。

Output Top 5 Softmax Probabilities For Each Image Found on the Web

```
top_k = 5
for i, prediction in enumerate(predictions):
    top_k_indices = np.argsort(prediction)[-top_k:][::-1]
    top_k_probabilities = prediction[top_k_indices]
    top_k_classes = [class_names[idx] for idx in top_k_indices]

    print(f"Image {i+1} top {top_k} predictions:")
    for prob, cls in zip(top_k_probabilities, top_k_classes):
        print(f"Probability: {prob:.2f} - Prediction: {cls}")
    print()
```

```
Image 1 top 5 predictions:
Probability: 1.00 - Prediction: General caution
Probability: 0.00 - Prediction: Traffic signals
Probability: 0.00 - Prediction: End of no passing by vehicles over 3.5 metric tons
Probability: 0.00 - Prediction: Dangerous curve to the right
Probability: 0.00 - Prediction: No entry

Image 2 top 5 predictions:
Probability: 1.00 - Prediction: Road work
Probability: 0.00 - Prediction: Bicycles crossing
Probability: 0.00 - Prediction: Keep right
Probability: 0.00 - Prediction: Keep left
Probability: 0.00 - Prediction: Bumpy road

Image 3 top 5 predictions:
Probability: 0.96 - Prediction: Speed limit (70km/h)
Probability: 0.03 - Prediction: Speed limit (50km/h)
Probability: 0.00 - Prediction: Speed limit (30km/h)
Probability: 0.00 - Prediction: Speed limit (60km/h)
Probability: 0.00 - Prediction: Speed limit (20km/h)

Image 4 top 5 predictions:
Probability: 1.00 - Prediction: No passing
Probability: 0.00 - Prediction: Vehicles over 3.5 metric tons prohibited
Probability: 0.00 - Prediction: No passing for vehicles over 3.5 metric tons
Probability: 0.00 - Prediction: End of no passing
Probability: 0.00 - Prediction: General caution

Image 5 top 5 predictions:
Probability: 1.00 - Prediction: No entry
Probability: 0.00 - Prediction: Keep right
Probability: 0.00 - Prediction: Stop
Probability: 0.00 - Prediction: Ahead only
Probability: 0.00 - Prediction: Turn right ahead
```