In [5]:

```python
import os
import cv2
import numpy as np
import tensorflow as tf
import xml.etree.ElementTree as ET
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

In [6]:

```python
# 📂 Dataset paths
annotation_dir = "/kaggle/input/road-sign-detection/annotations"
image_dir = "/kaggle/input/road-sign-detection/images"

# Function to parse a single XML file
def parse_annotation(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    filename = root.find("filename").text
    objects = []

    for obj in root.findall("object"):
        label = obj.find("name").text
        bbox = obj.find("bndbox")
        xmin = int(bbox.find("xmin").text)
        ymin = int(bbox.find("ymin").text)
        xmax = int(bbox.find("xmax").text)
        ymax = int(bbox.find("ymax").text)
        objects.append((label, xmin, ymin, xmax, ymax))

    return filename, objects
```

In [7]:

```python
# 📥 Load images & labels
images = []
labels = []
target_size = (64, 64)  # Resize all crops to this size

for xml_file in os.listdir(annotation_dir):
    if xml_file.endswith(".xml"):
        file_path = os.path.join(annotation_dir, xml_file)
        filename, objects = parse_annotation(file_path)

        img_path = os.path.join(image_dir, filename)
        if os.path.exists(img_path):
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            for label, xmin, ymin, xmax, ymax in objects:
                cropped = img[ymin:ymax, xmin:xmax]
                cropped = cv2.resize(cropped, target_size)
                cropped = cropped / 255.0  # Normalize
                images.append(cropped)
                labels.append(label)
```

In [8]:

```python
images = np.array(images, dtype=np.float32)
labels = np.array(labels)
```

In [9]:

```python
# 🔤 Encode labels
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
labels_categorical = to_categorical(labels_encoded)
```

In [10]:

```python
# 📊 Split into train & validation sets
X_train, X_val, y_train, y_val = train_test_split(
    images, labels_categorical, test_size=0.2, random_state=42
)

print("Train:", X_train.shape, y_train.shape)
print("Validation:", X_val.shape, y_val.shape)
print("Classes:", label_encoder.classes_)
```

```
Train: (995, 64, 64, 3) (995, 4)
Validation: (249, 64, 64, 3) (249, 4)
Classes: ['crosswalk' 'speedlimit' 'stop' 'trafficlight']
```

In [11]:

```python
# 🧠 CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(64, 64, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(label_encoder.classes_), activation='softmax')
])

# ⚙️ Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:10
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
ng Sequential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-08-14 05:31:09.443533: E external/local_xla/xla/stream_executor/cuda/cuda_driver.
cc:152] failed call to cuInit: INTERNAL: CUDA error: Failed call to cuInit: UNKNOWN ER
ROR (303)
```

Model: "sequential"

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                 │ (None, 62, 62, 32)     │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 31, 31, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 29, 29, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 14, 14, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 12, 12, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 6, 6, 128)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 4608)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 128)            │       589,952 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 4)              │           516 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 683,716 (2.61 MB)
 Trainable params: 683,716 (2.61 MB)
 Non-trainable params: 0 (0.00 B)
```

In [12]:

```
# 🚀 Train
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=15,
    batch_size=32
)
```

```
Epoch 1/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 111ms/step - accuracy: 0.5748 - loss: 0.9475 - val_accur
acy: 0.8675 - val_loss: 0.3697
Epoch 2/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.8751 - loss: 0.3805 - val_accura
cy: 0.9598 - val_loss: 0.1473
Epoch 3/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 102ms/step - accuracy: 0.9519 - loss: 0.1706 - val_accur
acy: 0.9759 - val_loss: 0.0747
Epoch 4/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 109ms/step - accuracy: 0.9644 - loss: 0.0984 - val_accur
acy: 0.9839 - val_loss: 0.0631
Epoch 5/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 104ms/step - accuracy: 0.9849 - loss: 0.0577 - val_accur
acy: 0.9839 - val_loss: 0.0746
Epoch 6/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - accuracy: 0.9808 - loss: 0.0695 - val_accur
acy: 0.9839 - val_loss: 0.0442
Epoch 7/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.9887 - loss: 0.0584 - val_accura
cy: 0.9880 - val_loss: 0.0305
Epoch 8/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.9882 - loss: 0.0361 - val_accura
cy: 0.9880 - val_loss: 0.0510
Epoch 9/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 103ms/step - accuracy: 0.9879 - loss: 0.0335 - val_accur
acy: 0.9880 - val_loss: 0.0585
Epoch 10/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 103ms/step - accuracy: 0.9881 - loss: 0.0286 - val_accur
acy: 0.9920 - val_loss: 0.0225
Epoch 11/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 103ms/step - accuracy: 0.9916 - loss: 0.0210 - val_accur
acy: 0.9799 - val_loss: 0.0643
Epoch 12/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - accuracy: 0.9980 - loss: 0.0145 - val_accur
acy: 0.9960 - val_loss: 0.0113
Epoch 13/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.9998 - loss: 0.0064 - val_accura
cy: 0.9880 - val_loss: 0.0318
Epoch 14/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 114ms/step - accuracy: 0.9927 - loss: 0.0233 - val_accur
acy: 0.9880 - val_loss: 0.0555
Epoch 15/15
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 102ms/step - accuracy: 0.9925 - loss: 0.0160 - val_accur
acy: 0.9920 - val_loss: 0.0574
```

In [14]:

```
# 💾 Save model & label encoder
model.save("traffic_sign_cnn.h5")

import pickle
with open("label_encoder.pkl", "wb") as f:
    pickle.dump(label_encoder, f)
```