```
import kagglehub
from kagglehub import KaggleDatasetAdapter
```

```
import kagglehub
import os
# Download latest version
path = kagglehub.dataset_download("meowmeowmeowmeowmeow/gtsrb-german-traffic-sign")

print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/gtsrb-german-traffic-sign
```

```
print(os.listdir('/kaggle/input/gtsrb-german-traffic-sign/'))
```

```
['Meta', 'meta', 'Meta.csv', 'Train.csv', 'Test.csv', 'Test', 'test', 'Train', 'train']
```

```
import pandas as pd

# Патека до CSV фајловите
train_csv = "/kaggle/input/gtsrb-german-traffic-sign/Train.csv"
test_csv = "/kaggle/input/gtsrb-german-traffic-sign/Test.csv"

# Прочитај ги CSV фајловите
train_df = pd.read_csv(train_csv)
test_df = pd.read_csv(test_csv)

# Прикажи неколку реда од секој CSV
print("Train CSV Sample:")
print(train_df.head())

print("\nTest CSV Sample:")
print(test_df.head())
```

```
Train CSV Sample:
   Width  Height  Roi.X1  Roi.Y1  Roi.X2  Roi.Y2  ClassId  \
0     27      26       5       5      22      20       20
1     28      27       5       6      23      22       20
2     29      26       6       5      24      21       20
3     28      27       5       6      23      22       20
4     28      26       5       5      23      21       20

                            Path
0  Train/20/00020_00000_00000.png
1  Train/20/00020_00000_00001.png
2  Train/20/00020_00000_00002.png
3  Train/20/00020_00000_00003.png
4  Train/20/00020_00000_00004.png

Test CSV Sample:
   Width  Height  Roi.X1  Roi.Y1  Roi.X2  Roi.Y2  ClassId            Path
0     53      54       6       5      48      49       16  Test/00000.png
1     42      45       5       5      36      40        1  Test/00001.png
2     48      52       6       6      43      47       38  Test/00002.png
3     27      29       5       5      22      24       33  Test/00003.png
4     60      57       5       5      55      52       11  Test/00004.png
```

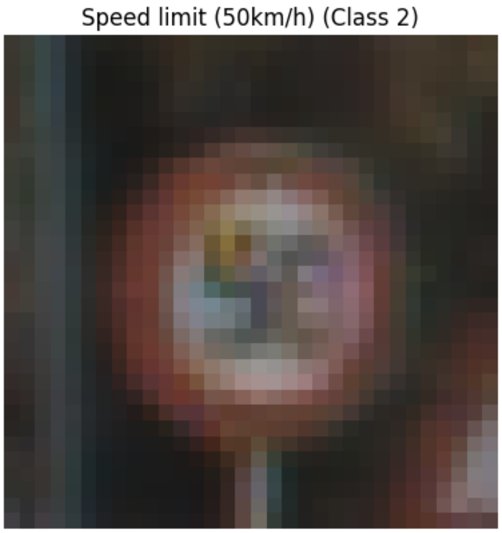
```
from google.colab import drive
drive.mount('/content/drive')
```

```
# Label Overview
classes = { 0:'Speed limit (20km/h)',
            1:'Speed limit (30km/h)',
            2:'Speed limit (50km/h)',
            3:'Speed limit (60km/h)',
            4:'Speed limit (70km/h)',
            5:'Speed limit (80km/h)',
            6:'End of speed limit (80km/h)',
            7:'Speed limit (100km/h)',
            8:'Speed limit (120km/h)',
            9:'No passing',
            10:'No passing veh over 3.5 tons',
            11:'Right-of-way at intersection',
            12:'Priority road',
            13:'Yield',
            14:'Stop',
            15:'No vehicles',
            16:'Veh > 3.5 tons prohibited',
            17:'No entry',
            18:'General caution',
            19:'Dangerous curve left',
            20:'Dangerous curve right',
            21:'Double curve',
            22:'Bumpy road',
            23:'Slippery road',
            24:'Road narrows on the right',
            25:'Road work',
            26:'Traffic signals',
            27:'Pedestrians',
            28:'Children crossing',
            29:'Bicycles crossing',
            30:'Beware of ice/snow',
            31:'Wild animals crossing',
            32:'End speed + passing limits',
            33:'Turn right ahead',
            34:'Turn left ahead',
            35:'Ahead only',
            36:'Go straight or right',
            37:'Go straight or left',
            38:'Keep right',
            39:'Keep left',
            40:'Roundabout mandatory',
            41:'End of no passing',
            42:'End no passing veh > 3.5 tons' }
```

```
import os
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
from sklearn.metrics import classification_report

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# =====================
# 1. Data Preparation
# =====================

class TrafficSignDataset(Dataset):
    def __init__(self, csv_file, root_dir, transform=None):
        self.dataframe = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        relative_path = self.dataframe.iloc[idx, 7]  # Path is column 7
        img_path = os.path.join(self.root_dir, relative_path)
        image = Image.open(img_path).convert("RGB")
        label = int(self.dataframe.iloc[idx, 6])  # ClassId is column 6

        if self.transform:
            image = self.transform(image)

        return image, label

# Transformations
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Create datasets
```

```
train_dataset = TrafficSignDataset(
    csv_file="/kaggle/input/gtsrb-german-traffic-sign/Train.csv",
    root_dir="/kaggle/input/gtsrb-german-traffic-sign/",
    transform=transform
)

test_dataset = TrafficSignDataset(
    csv_file="/kaggle/input/gtsrb-german-traffic-sign/Test.csv",
    root_dir="/kaggle/input/gtsrb-german-traffic-sign/",
    transform=transform
)

# Create dataloaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

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

# Земи една слика и нејзината етикета
image, label = train_dataset[4234]

# Враќање од нормализација (0 до 1)
image = image.permute(1, 2, 0).numpy() * 0.5 + 0.5

# Добиј го името на знакот
sign_name = classes.get(label, "Unknown Sign")

# Прикажи ја сликата со името на знакот
plt.imshow(image)
plt.title(f"{sign_name} (Class {label})")
plt.axis("off")
plt.show()
```


Speed limit (50km/h) (Class 2)

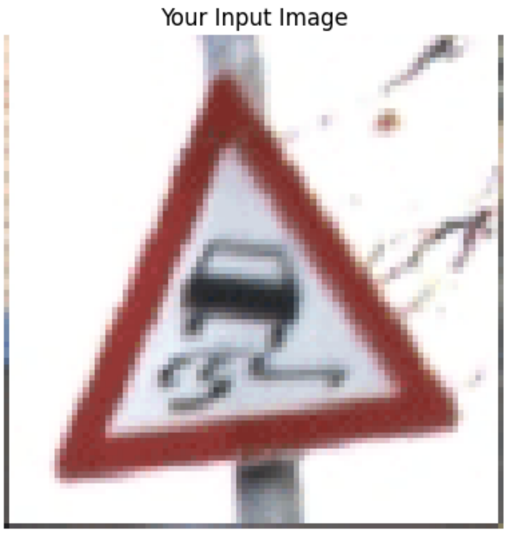
```
# =====================
# 2. CNN Model Definition
# =====================
# Define the EXACT same model architecture used during training
class TrafficSignCNN(nn.Module):
    def __init__(self, num_classes=43):
        super(TrafficSignCNN, self).__init__()

        # Feature extractor as sequential
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )

        # Classifier as sequential
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Dropout(0.5),
            nn.Linear(128 * 4 * 4, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.feature_extractor(x)
        x = self.classifier(x)
        return x



model = TrafficSignCNN(num_classes=43).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

num_epochs = 3
best_accuracy = 0.0

for epoch in range(num_epochs):
    # Training
    model.train()
    train_loss = 0.0
    correct_train = 0
    total_train = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    # Validation
    model.eval()
    val_loss = 0.0
    correct_val = 0
    total_val = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)

            val_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted == labels).sum().item()
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # Update learning rate
    scheduler.step()

    # Calculate metrics
    train_accuracy = 100 * correct_train / total_train
    val_accuracy = 100 * correct_val / total_val

    print(f"Epoch [{epoch+1}/{num_epochs}]")
    print(f"Train Loss: {train_loss/len(train_loader):.4f} | Train Acc: {train_accuracy:.2f}%")
    print(f"Val Loss: {val_loss/len(test_loader):.4f} | Val Acc: {val_accuracy:.2f}%")
    print("-" * 50)

    # Save best model
    if val_accuracy > best_accuracy:
        best_accuracy = val_accuracy
        torch.save(model.state_dict(), "best_model.pth")
        print("Saved new best model!")
```

```
Epoch [1/3]
Train Loss: 0.8918 | Train Acc: 73.21%
Val Loss: 0.3336 | Val Acc: 89.79%
--------------------------------------------------
Saved new best model!
Epoch [2/3]
Train Loss: 0.1618 | Train Acc: 94.77%
Val Loss: 0.1822 | Val Acc: 94.54%
--------------------------------------------------
Saved new best model!
Epoch [3/3]
Train Loss: 0.1039 | Train Acc: 96.72%
Val Loss: 0.1328 | Val Acc: 96.35%
--------------------------------------------------
Saved new best model!
```

```
# ====================
# 🔷 4. Тестирање на моделот
# ====================

model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Точност на тест податоци: {accuracy:.2f}%")
```

```
Точност на тест податоци: 96.35%
```

```
import torch
from PIL import Image
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

# Load the trained model
model = TrafficSignCNN(num_classes=43).to(device)
model.load_state_dict(torch.load("best_model.pth", map_location=device))
model.eval()

# Define transformations
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

def predict_custom_image(image_path):
    # Open and display the image
    image = Image.open(image_path).convert("RGB")
    plt.imshow(image)
    plt.axis('off')
    plt.title("Your Input Image")
    plt.title("Your Input Image")
    plt.show()

    # Preprocess the image
    image_tensor = transform(image).unsqueeze(0).to(device)

    # Make prediction
    with torch.no_grad():
        outputs = model(image_tensor)
        _, predicted = torch.max(outputs.data, 1)
        predicted_class = predicted.item()

    # Get probabilities
    probabilities = torch.nn.functional.softmax(outputs, dim=1)[0] * 100

    # Display results
    print("\nPrediction Results:")
    print(f"Predicted Class: {predicted_class} - {classes[predicted_class]}")
    print("\nTop 5 Predictions:")

    # Get top 5 predictions
    top5_prob, top5_classes = torch.topk(probabilities, 5)
    for i in range(5):
        print(f"{i+1}. {classes[top5_classes[i].item()]}: {top5_prob[i].item():.2f}%")

    return predicted_class

# Example usage
custom_image_path = "test.png"  # Replace with your image path
prediction = predict_custom_image(custom_image_path)
```



Your Input Image

```
Prediction Results:
Predicted Class: 23 - Slippery road

Top 5 Predictions:
1. Slippery road: 100.00%
2. Dangerous curve right: 0.00%
3. Beware of ice/snow: 0.00%
4. Bicycles crossing: 0.00%
5. Dangerous curve left: 0.00%
```