**Training Documentation**

**🚦Traffic Sign Classification using ResNet-18 on GTSRB Dataset**

**📄 Project Overview:**

This project involves training a **ResNet-18** model to classify German traffic signs using the **GTSRB (German Traffic Sign Recognition Benchmark)** dataset.  
The system processes traffic sign images, learns patterns, and predicts the correct traffic sign category with high accuracy.

**📁 Dataset Information:**

* **Dataset Name:** GTSRB - German Traffic Sign Recognition Benchmark
* **Files Used:**
  + Train.csv — Training data with image paths and labels
  + Test.csv — Validation data with image paths and labels
* **Image Labels:** Traffic sign class IDs ranging from 0 to 42 (43 classes total).

**⚙️ Project Dependencies:**

* **Libraries Used:**
  + torch
  + torchvision
  + PIL (Pillow)
  + pandas
* **Pretrained Model:**
  + ResNet-18 (from torchvision.models)

**📜 Training Steps Explained:**

**1. Device Configuration**

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

* Checks for GPU availability (CUDA).
* Falls back to CPU if a GPU is not found.

**2. Image Transformations**

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225]),

])

* **Resize:** All images are resized to 224×224 pixels (ResNet-18 input requirement).
* **Normalization:** Images are normalized with ImageNet dataset means and standard deviations (since the pretrained ResNet-18 was trained on ImageNet).

**3. Custom Dataset Class**

class TrafficDataset(Dataset):

...

* Custom PyTorch Dataset to load the GTSRB dataset.
* Reads the CSV files (Train.csv, Test.csv), loads corresponding images, and applies transformations.
* Returns a tuple (image\_tensor, label) for each sample.

**4. Data Loading**

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

valid\_loader = DataLoader(valid\_dataset, batch\_size=32, shuffle=False)

* **Batch size:** 32 images per batch.
* **Shuffle:** Enabled for training data to improve generalization.

**5. Model Preparation**

model = models.resnet18(pretrained=True)

num\_classes = len(set(train\_dataset.data['ClassId']))

model.fc = nn.Linear(model.fc.in\_features, num\_classes)

model = model.to(device)

* **Base Model:** Pretrained ResNet-18.
* **Modification:** Final fully connected layer (fc) adjusted for 43 output classes (instead of default 1000).
* **Transfer Learning:** We leverage pretrained features and fine-tune the model for traffic signs.

**6. Loss Function & Optimizer**

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=1e-4)

* **Loss Function:** Cross Entropy Loss (standard for multi-class classification).
* **Optimizer:** Adam Optimizer with a learning rate of 0.0001.

**7. Training Loop**

for epoch in range(3):

...

* **Epochs:** 3 (each epoch represents a full pass over the training set).
* **Steps Per Epoch:**
  + Set model to training mode.
  + Forward pass: Images → Model → Predictions.
  + Loss computation.
  + Backward pass: Gradients are computed.
  + Parameters updated via the optimizer.
  + Metrics collected: Running Loss, Accuracy.

**Training Results:**

* **Epoch 1:** Loss: 0.1907 | Accuracy: 95.86%
* **Epoch 2:** Loss: 0.0089 | Accuracy: 99.82%
* **Epoch 3:** Loss: 0.0077 | Accuracy: 99.82%

**8. Validation Loop**

model.eval()

with torch.no\_grad():

...

* Model is switched to **evaluation mode** (no dropout, no batch norm update).
* No gradient calculation (torch.no\_grad() improves speed and reduces memory usage).
* Predictions are made on the validation dataset.

**Validation Result:**

* **Validation Accuracy:** 99.14%

**📈 Final Model Performance:**

| **Phase** | **Accuracy** |
| --- | --- |
| Training (Epoch 3) | 99.82% |
| Validation | 99.14% |

**🔥 Key Highlights:**

* **Pretrained Model:** Leveraging ResNet-18 significantly boosts training speed and accuracy.
* **Custom Dataset Loader:** Flexibility to work with CSV file structures.
* **Excellent Accuracy:** >99% accuracy in just 3 epochs due to fine-tuning on powerful ResNet features.

**🚀 Potential Improvements:**

* Fine-tune more layers of ResNet-18 (not just fc layer).
* Use data augmentation (random rotation, horizontal flip, color jitter) for better robustness.
* Try advanced schedulers like CosineAnnealingLR for dynamic learning rate adjustments.
* Train for more epochs for even finer learning.
* Save the best model using torch.save(model.state\_dict(), 'traffic\_sign\_model.pth').