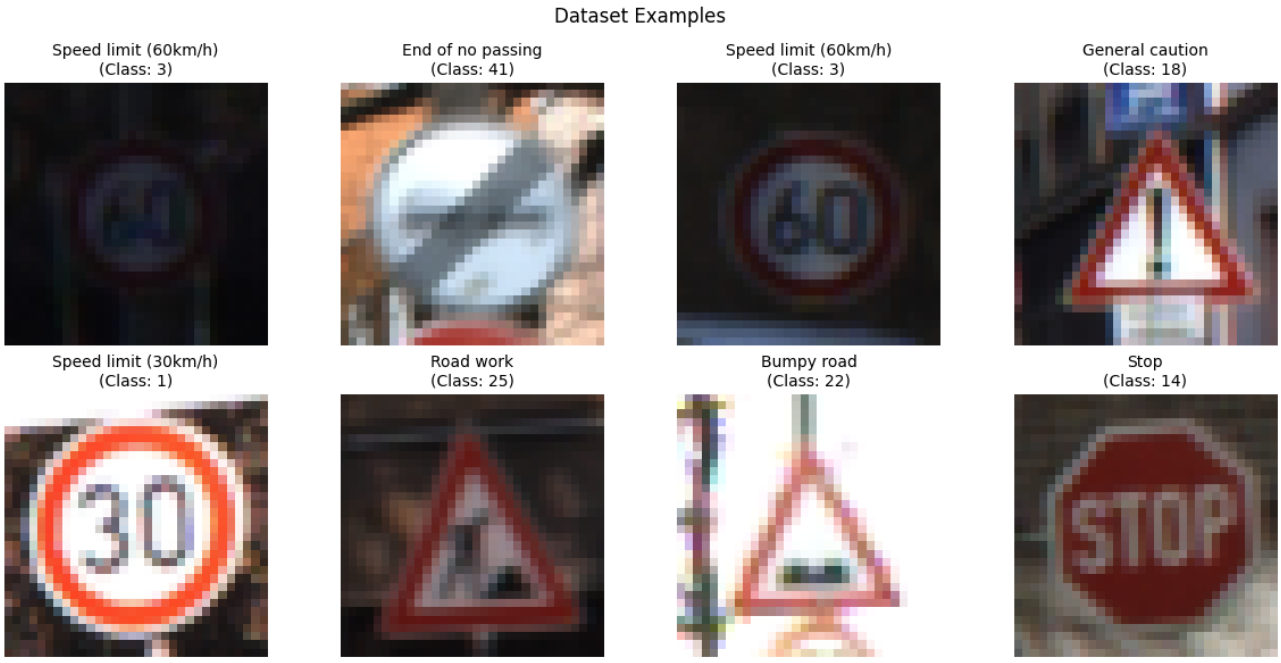
Pytorch + Klasifikace dopravních značek (dataset GTSRB)

Úkolem je navrhnout, natrénovat a porovnat alespoň dvě různé architektury neuronových sítí pro klasifikaci dopravních značek z datasetu GTSRB (German Traffic Sign Recognition Benchmark).

Dataset obsahuje reálné fotografie dopravních značek rozdělených do 43 tříd (např. omezení rychlosti, dej přednost v jízdě, zákaz předjíždění atd.).



1. Příprava dat

K načtení datasetu použijte `torchvision.datasets.GTSRB`. *Důležité upozornění:* Obrázky v datasetu GTSRB na rozdíl od FashionMNIST nemají jednotnou velikost a jsou barevné (3 kanály - RGB). Než je předáte síti, musíte je sjednotit na stejnou velikost (např. 32x32 pixelů) pomocí `transforms.Resize()`.

Kroky:

1. Definujte transformace (změna velikosti na 32x32 a převod na tenzor).
2. Načtěte trénovací (`split='train'`) a testovací (`split='test'`) data.
3. Vytvořte pro ně `DataLoader` se zvolenou velikostí dávky (batch size), například 64. *(Tip pro pomalejší PC: Pokud trénování trvá dlouho, můžete dataset zmenšit na 5000 náhodných obrázků. Nejprve si vygenerujte těmito obrázky indexy* `indices = torch.randperm(len(train_data))[:5000]` *a použijte* `Subset` *z* `torch.utils.data`*:* `train_data = Subset(train_data, indices)` *)*

2. Návrh neuronových sítí

Vytvořte dvě různé třídy dědící z `nn.Module`. Cílem je porovnat tyto dvě architektury na stejných datech.

Část A: Klasická (plně propojená) neuronová síť (MLP)

1. Vstupní obrázek (3 barevné kanály, 32x32 pixelů) rozbalte na 1D vektor pomocí `nn.Flatten`. Vektor bude mít délku 3 * 32 * 32 = 3072.
2. Přidejte 1 až 2 skryté plně propojené vrstvy (`nn.Linear`) oddělené aktivační funkcí (`nn.ReLU`). (např. velikosti 512, 128).
3. Výstupní lineární vrstva (`nn.Linear`) se **43 výstupy** (bez softmax, jelikož použijete `CrossEntropyLoss`).

Část B: Konvoluční neuronová síť (CNN) Pro zpracování snímků bývají konvoluce výrazně efektivnější než plně propojené vrstvy.

1. Přidejte konvoluční blok (např. `nn.Conv2d` se 16 výstupními kanály a kernelem 3), po němž následuje aktivace (`nn.ReLU`) a vrstva sdružování (`nn.MaxPool2d` s kernelem 2).
2. Přidejte druhý konvoluční blok (Conv2d se 32 kanály -> ReLU -> MaxPool2d).
3. Zarovnejte tenzor na 1D vektor (`nn.Flatten`). *Tip: Pokud v konvolučních vrstvách (`Conv2d`) použijete `kernel_size=3` společně s `padding=1`, výstupní rozměry příznakových map zůstanou stejné jako původní vstup. K redukci prostorových rozměrů pak dochází výhradně ve vrstvách `MaxPool2d` (při `kernel_size=2` se rozměry zmenší na polovinu). Po dvou takových konvolučních blocích klesne rozměr z 32x32 na 16x16 a poté na 8x8. Počet parametrů pro vrstvu `Linear` pak bude `32 kanálů * 8 * 8 = 2048`.*
4. Přidejte plně propojenou skrytou vrstvu a nakonec výstupní vrstvu (`nn.Linear`) se **43 výstupy**.

3. Trénování a evaluace

Chcete-li předejít duplicitnímu kódu pro každou síť, můžete si napsat pomocnou funkci pro trénování a evaluaci: `def train_and_evaluate(model, train_loader, test_loader, epochs=5):`

1. Inicializujte oba modely a přesuňte je na grafickou kartu (GPU, pokud máte k dispozici).
2. Zvolte ztrátovou funkci (`nn.CrossEntropyLoss`) a optimalizátor (např. `optim.SGD` s parametry `lr=0.01` a `momentum=0.9`).
3. Natrénujte obě sítě po stejný počet epoch (např. 5 až 8 epoch). Nezapomínejte na standardní kroky (`zero_grad`, `backward`, `step`).
4. Vyhodnoťte úspěšnost obou modelů na testovací sadě. Získejte predikce a určete maximální hodnotu přes dimenzi tříd (`torch.max(..., dim=1)`).

4. Analýza výsledků

Po dokončení porovnejte výsledky klasické sítě a konvoluční sítě:

- Rychlost (čas potřebný k trénování po stejný počet epoch).
- Zrychlení konvergence (hodnota loss funkce na konci trénování).
- **Finální procentuální úspěšnost na testovacích datech** – o kolik je CNN lepší? Proč myslíte, že k tomu došlo?

🌟 BONUS - 1: Vliv barvy na úspěšnost (RGB vs. Grayscale)

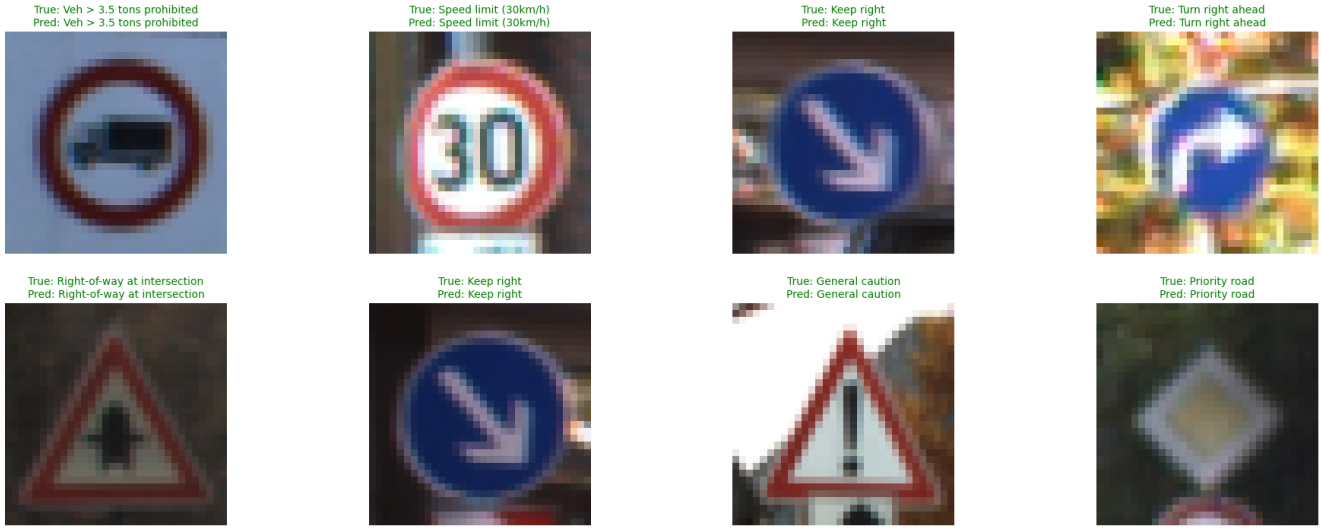
Jak moc se sítě spoléhají při rozpoznávání značek na barvu?

1. V části **Příprava dat** si vytvořte novou transformaci, která kromě `Resize` obsahuje i `transforms.Grayscale(num_output_channels=1)`. Tím odstraníte z obrázků barvu.

2. Načtěte si dataset znovu a vytvořte si pro tyto černobílé obrázky nové `DataLoadery`.
3. Upravte obě své sítě (MLP i CNN) (in_channels=1 vs. in_channels=3)
    - U MLP se změní první vrstva na `nn.Linear(in_channels * 32 * 32, ...)`.
    - U CNN se u první konvoluce změní parametr z `in_channels=3` na `in_channels=1`.
4. Spusťte trénink MLP i CNN znovu, tentokrát nad černobílými daty a porovnejte, o kolik klesla přesnost oproti barevné verzi.

🌟 BONUS - 2: Vizualizace predikcí

Ověřte si, že se síť opravdu naučila poznávat značky tím, že si vykreslíte několik testovacích obrázků i s jejich predikovanou a skutečnou třídou. Zkuste si vytáhnout jeden batch dat (např. pomocí `iter(test_loader)`) a zobrazit obrázky pomocí `matplotlib.pyplot.imshow()`. Inspirovat se můžete přímo v oficiálních PyTorch tutoriálech.



Užitečné zdroje a tutoriály (PyTorch):

- Studijní materiál ZAO
- Training a Classifier (CIFAR-10 tutorial)
- Build the Neural Network