COMP 576 Final Report

Road Signs Recognition Using Convolutional Neural Network

Member: Zhewei Su, Yi Zhao, Zheng Liang, Yifu Zhou

GitHub Link: https://github.com/SUZHEWEI/group22

Motivation:

With the development of the automotive industry, more and more vehicles are on the road. In recent years, the concept of autonomous driving has become more and more popular. At the national level, many countries have incorporated autonomous driving into their top-level planning. Take the United States as an example: the U.S. Department of Transportation issued the "Autonomous Vehicle Policy Guidelines" in 2016, which aims to promote the safety testing of autonomous driving technology; while Japan released the "2017 Official and Private ITS Concept and Roadmap" in 2017 ”, planning the development timetable for autonomous driving; China has also issued a number of documents such as the “Medium and Long-Term Development Plan for the Automobile Industry” to provide clear and specific guidance for the autonomous driving industry. From the perspective of enterprises, many domestic and foreign High-tech companies have also begun to focus on autonomous driving. For example, Didi in China and Waymo in America have set up special research departments. However, before automatic driving can actually enter the actual large-scale application, its safety needs to be ensured. Our team takes road sign recognition during driving as an example to explore difficulties in automotive industry.

Dataset:

The dataset: “German Traffic Sign Recognition Benchmark (GTSRB)” is available online. In The International Joint Conference on Neural Networks 2011 competition, this dataset was used as the official training set and test set of the competition. The dataset has 43 classes with a total of about 52,000 images. The training set has nearly 40,000 images, and the test set has about 12,000 images.

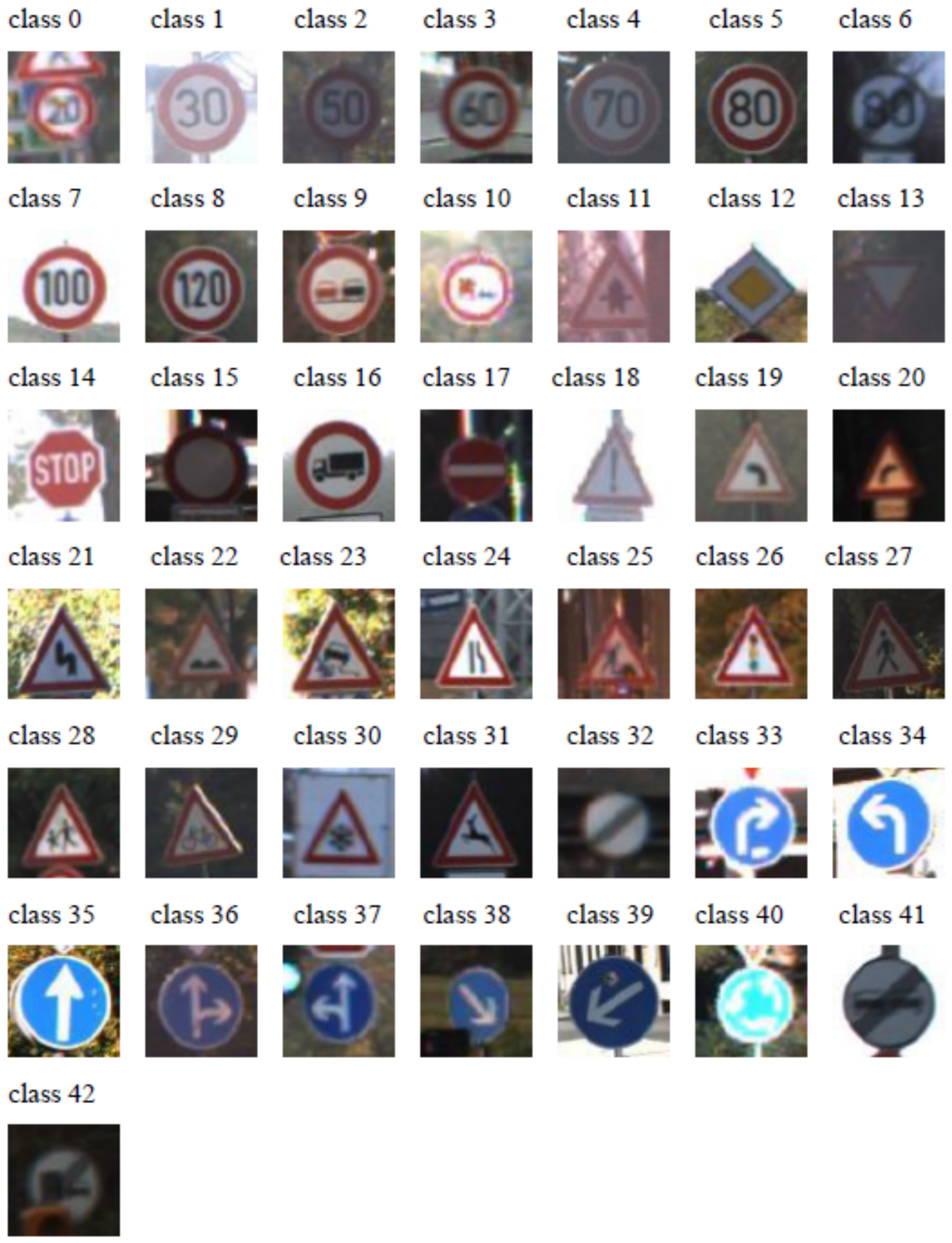
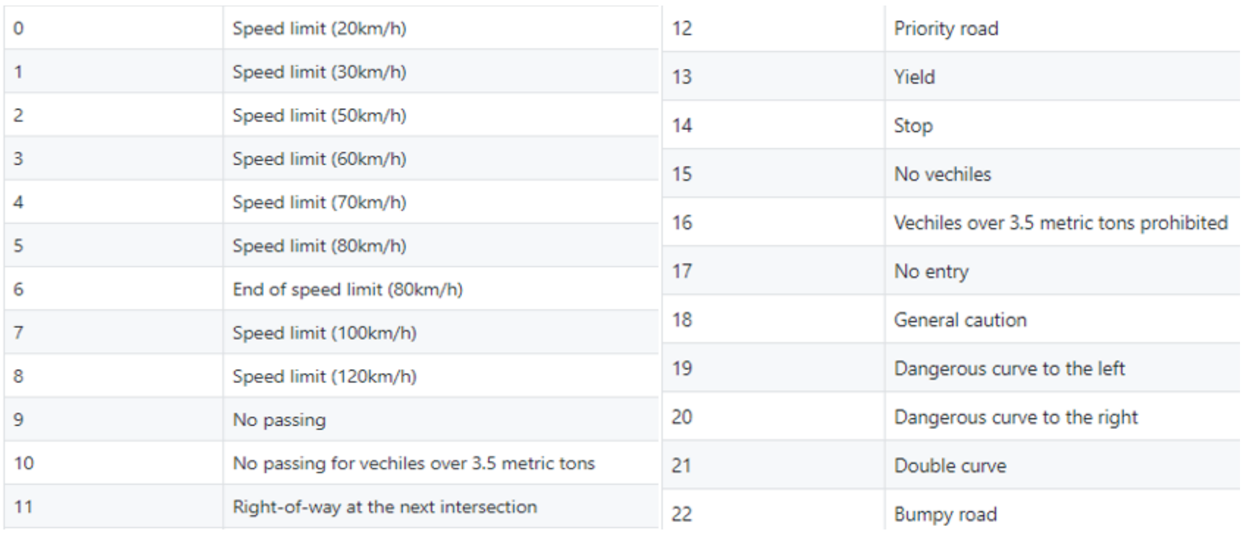
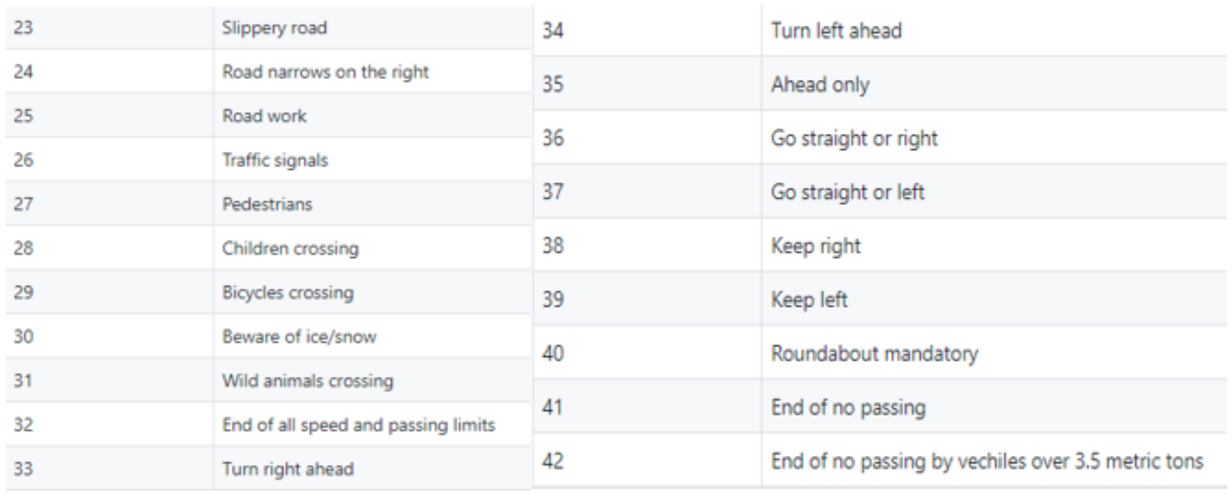


Fig1. Example for each class



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 for vechiles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road
13	Yield
14	Stop
15	No vechiles
16	Vechiles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road

Fig2. Meaning of class 0 – 22



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 of all speed and 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 of no passing by vechiles over 3.5 metric tons

Fig3. Meaning of class 23 – 42

Data Preprocessing:

Since the total amount of data in the data set is not very large, in order to expand the total amount of training data as much as possible, the existing training data pictures can be used to rotate, flip, etc., so as to obtain new pictures to expand the data set. Observing Figure 1, it can be seen that some road signs still belong to this class after the pictures of some classes are flipped horizontally, such as class 11, class 13, class 35, etc. Some road signs still belong to this class after vertical flipping, such as class 1, class 5; There are also some classes that are symmetrical to each other, such as class 19 and class 20, and the icon picture of class 20 can be obtained by flipping class 19 horizontally. Through this flip operation on the original picture, the amount of training picture data has increased from nearly 40,000 to more than 60,000.

Pictures before expanding	After
39209	63538

In addition to expanding the amount of data, the quality of pictures should also be improved. Many of the original pictures are blurry, and the brightness and contrast of the pictures vary greatly. After investigating the commonly used image quality improvement algorithms, it is found that The Contrast-limited adaptive histogram equalization (CLAHE) algorithm can make the hidden features of the image more obvious and improve the quality of the image. OpenCV also comes with the call interface of the algorithm, which makes it more convenient to process image data.

By analyzing the distribution of the number of pictures in the data set, it can be observed that there are not small differences in the amount of data among the various classes.

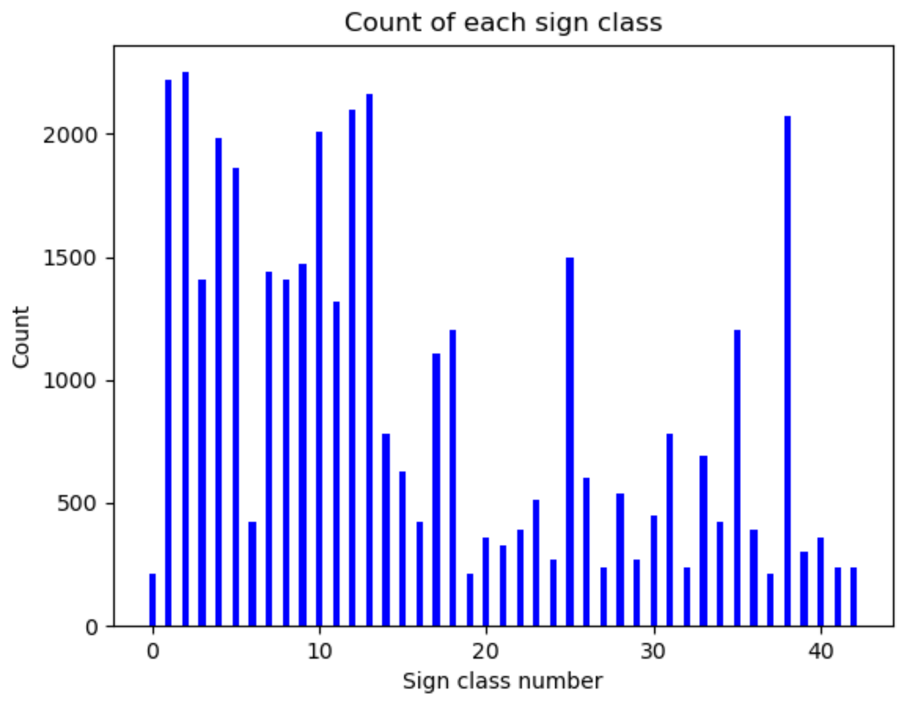
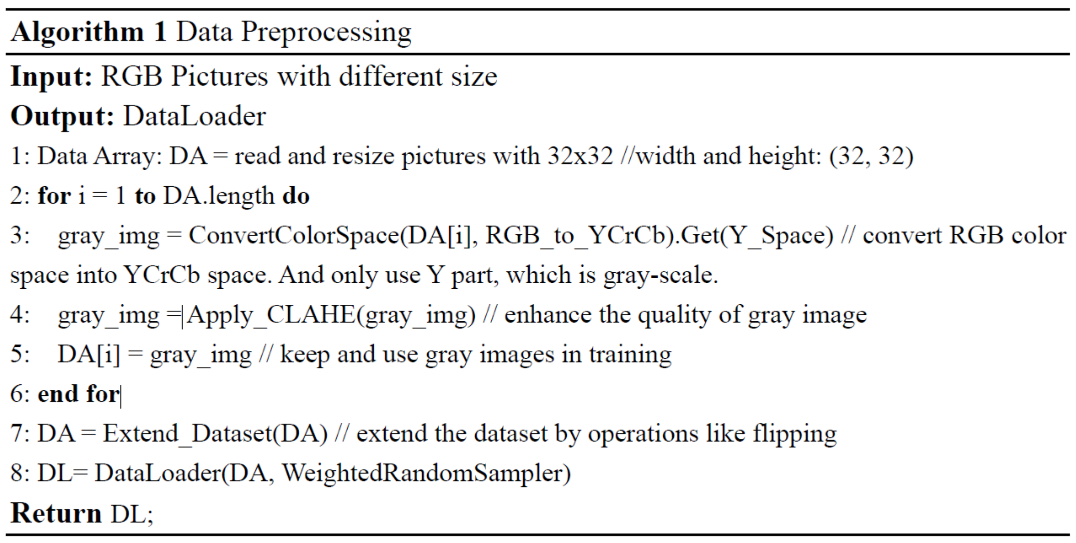


Fig 4. Num of pictures in each class of trainset

As shown in the figure above, the class with a large amount of data has more than 2000 pictures, while the class with a small amount of data has only about 200 pictures. The imbalance of data volume between classes will also affect the test results, and in order to solve this problem, the WeightedRandomSampler that comes with PyTorch is used in the implementation. The sampler assigns a weight to each class, and the higher the weight, the higher the probability of being sampled, and allows repeated sampling. Through this operation, the amount of data in each class is roughly equal during training, and the amount of data in large classes is not reduced.

The following figure shows the pseudocode of the data preprocessing process:



Algorithm 1 Data Preprocessing

Input: RGB Pictures with different size

Output: DataLoader

```
1: Data Array: DA = read and resize pictures with 32x32 //width and height: (32, 32)
2: for i = 1 to DA.length do
3:     gray_img = ConvertColorSpace(DA[i], RGB_to_YCrCb).Get(Y_Space) // convert RGB color
       space into YCrCb space. And only use Y part, which is gray-scale.
4:     gray_img =|Apply_CLAHE(gray_img) // enhance the quality of gray image
5:     DA[i] = gray_img // keep and use gray images in training
6: end for|
7: DA = Extend_Dataset(DA) // extend the dataset by operations like flipping
8: DL= DataLoader(DA, WeightedRandomSampler)
Return DL;
```

Fig 5. Pseudocode of data preprocessing

In the above algorithm, in the third step, the image is converted from RGB space to YCrCb space, and only the Y part of it is used, that is, the grayscale image part. In steps 4-5, use the CLAHE algorithm to improve the grayscale image, and replace the original RGB image with a grayscale image for subsequent use. In step 7-8, pack the processed dataset into the container interface, and use the Weighted Random Sampler to balance the number of different categories of data.

Network Structure:

Our team chooses the base model mentioned in the paper: “Multi-Column Deep Neural Network for Traffic Sign Classification”. We implement one column of it, which is basically a CNN model. We revise the model with Dropout layer and Batch Normalization layer to make the model more generalizable and alleviate the problem of overfitting.

Below shows the CNN model:

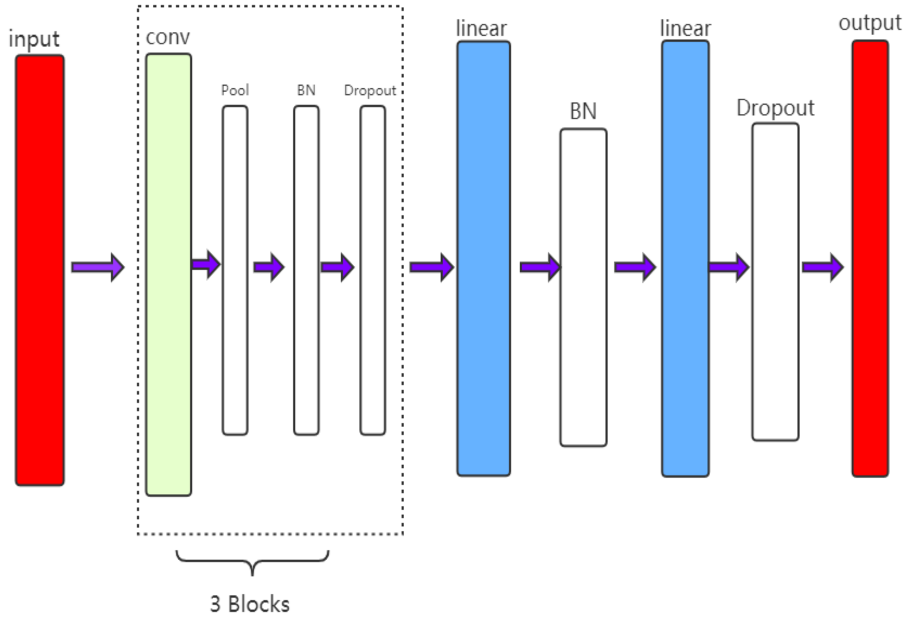


Fig6. The structure of CNN model

However, in the online papers and resources, it mentions that some pictures with bad angles or noises may negatively affect the result. Hence, our team consider adding extra advanced module into our base CNN model. We choose to add Space Transformer Network (STN) that is proposed by Google's Deep Mind team into our network structure. STN can provide more invariance to the original network. It can also be treated as a single layer to the network.

Below is the structure of STN:

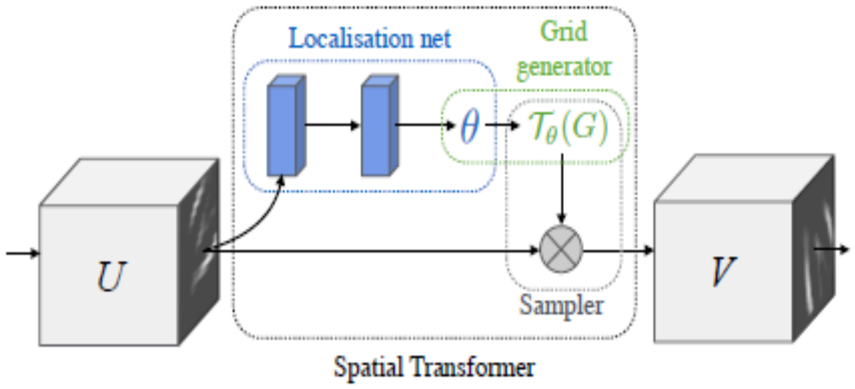


Fig7. The structure of STN model

So, our finalized model structure is shown below:

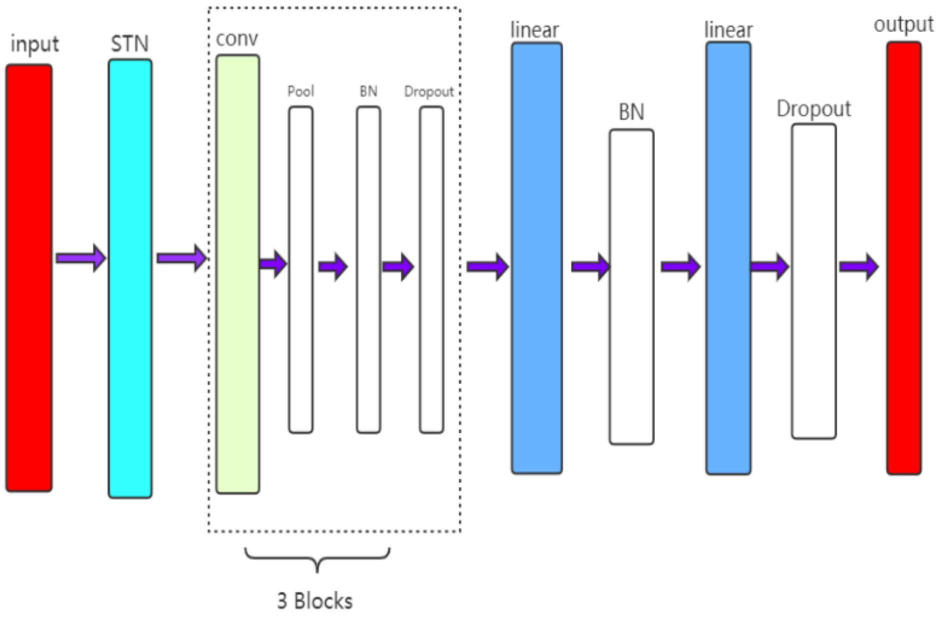


Fig8. The structure of finalized model

Experiments:

The training process uses the GTSRB dataset introduced above. After preprocessing the data, 80% of the training set data (about 50,000 data) is used as the final training data, and the remaining 20% of the training set data (about 13,000 data) is used as the verification set during the training process.

Both training and testing are carried out in the Linux system, and the GPU used is 2080TI.

We adopt CrossEntropyLoss Function as our training loss function.

Batch Size	64
Drop Out	0.5
Learning Rate	0.0001
Epoch	100
Early Stopping Patience	10

The training process adopts an early stopping mechanism, that is, if the validation loss does not decrease after a certain number of generations, the training will be terminated directly. At the 21st epoch, it is stopped by the early stopping mechanism. The final saved training model has an accuracy rate of 99.388% on the validation set.

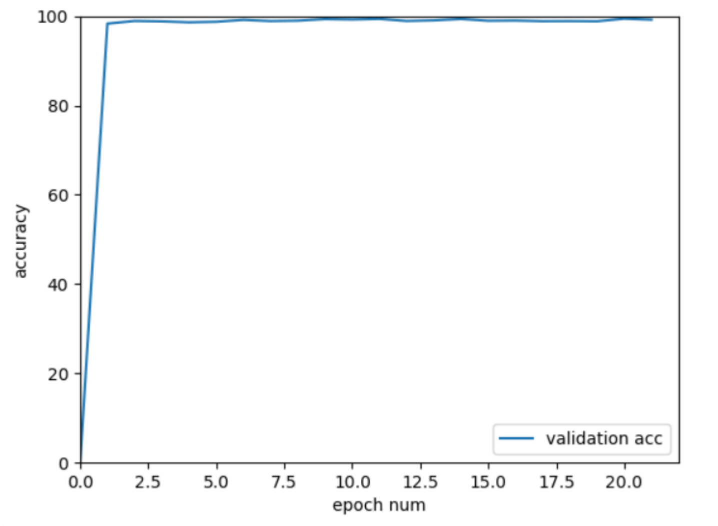


Fig9. Acc of validation set while training

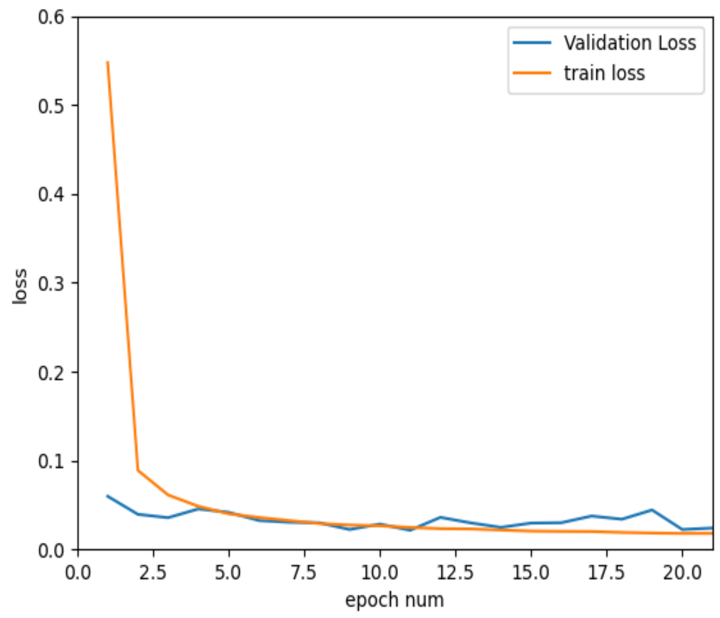


Fig10. Train loss and validation loss during training

The test uses the test set part of the GTSRB dataset, which has 12630 pictures. The test process is carried out on the Linux system, and the GPU used is 2080TI.

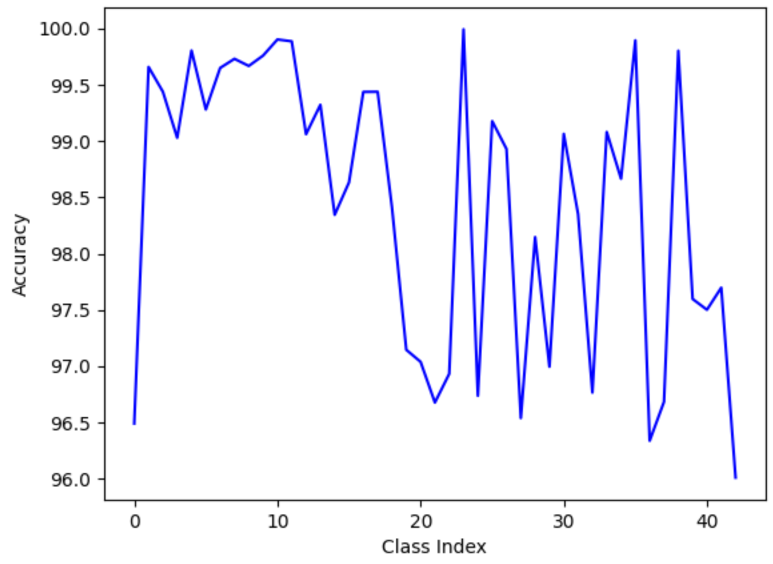


Fig11. Accuracy across different classes in the dataset

Fig.11 shows the test accuracy across the 43 classes of the dataset, it could be observed all the classes get an accuracy higher then 96% and the best class get an accuracy at around 100%.

For the whole dataset, we get a final test accuracy of 98.95%.

Challenges and Conclusion:

In this project, our team explore network design to tackle with the road sign recognition problem. The model reaches an accuracy about 99%. However, in real life, more complex network can be too large to be deployed on edge-devices with low computing power and limited storage place. It would be meaningful to explore how to compress the size of the network while maintaining a high recognition accuracy.

As for the challenges we met in this project, it comes mostly from two parts: 1. We heard PyTorch is also a popular machine learning framework used in industry. We took time to learn it and used it to complete this project. 2. We adopted two approaches to optimize our original network design, which were elaborated in Network Design Section.

References:

Dan C, Ueli M, Jonathan M, Jurgen S. Multi-Column Deep Neural Network for Traffic Sign Classification. [J]. Neural Networks, Aug 2012, Volume 32, 333-338.

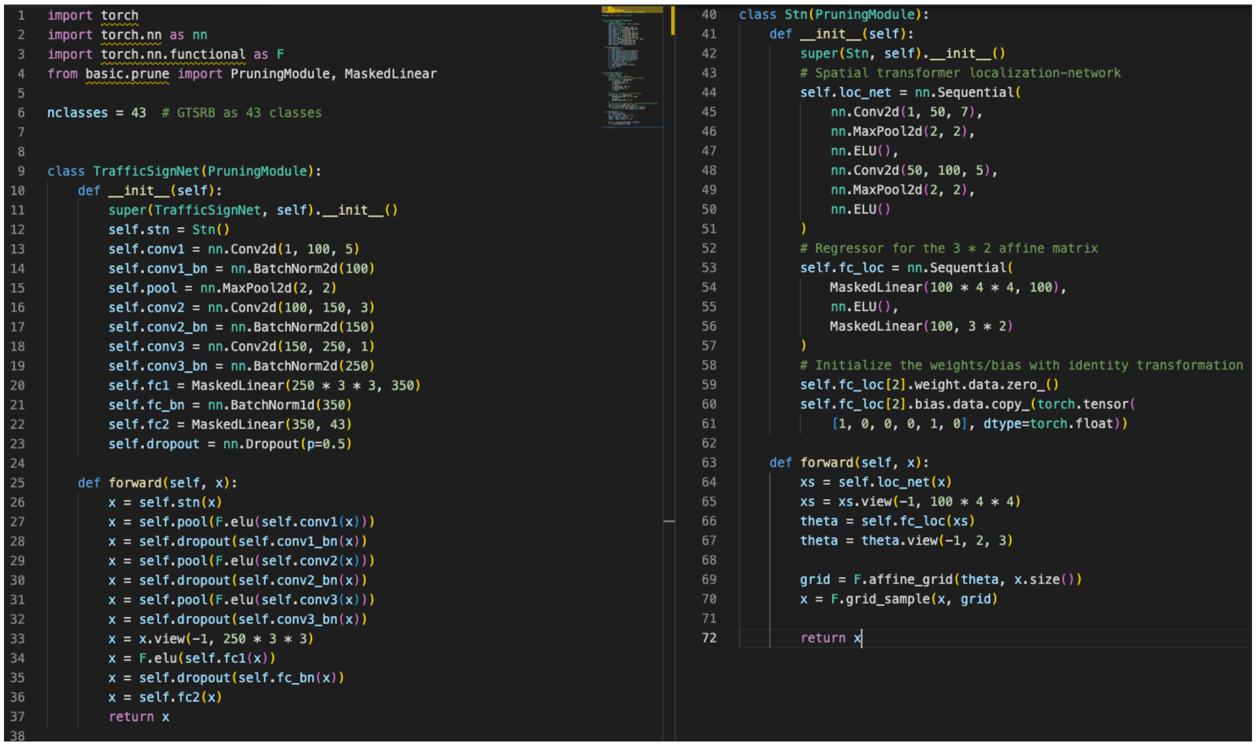
INI Benchmark Website. The German Traffic Sign Recognition Benchmark [EB/OL]. (2011-01-17)[2020- 05-13]. http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

Pierre S, Yann L. Traffic sign recognition with multi-scale Convolutional Networks[C]//IEEE. The 2011 International Joint Conference on Neural Networks, San Jose, USA: IEEE, Jul 2011:2809-2813

Torch. The power of Spatial Transformer Networks [EB/OL]. (2015-09-07)[2020-05-13]. http://torch.ch/blog/2015/09/07/spatial_transformers.html

Appendix:

Network Design Code Snapshot:



```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from basic.prune import PruningModule, MaskedLinear

nclasses = 43  # GTSRB as 43 classes


class TrafficSignNet(PruningModule):
    def __init__(self):
        super(TrafficSignNet, self).__init__()
        self.stn = Stn()
        self.conv1 = nn.Conv2d(1, 100, 5)
        self.conv1_bn = nn.BatchNorm2d(100)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(100, 150, 3)
        self.conv2_bn = nn.BatchNorm2d(150)
        self.conv3 = nn.Conv2d(150, 250, 1)
        self.conv3_bn = nn.BatchNorm2d(250)
        self.fc1 = MaskedLinear(250 * 3 * 3, 350)
        self.fc_bn = nn.BatchNorm1d(350)
        self.fc2 = MaskedLinear(350, 43)
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = self.stn(x)
        x = self.pool(F.elu(self.conv1(x)))
        x = self.dropout(self.conv1_bn(x))
        x = self.pool(F.elu(self.conv2(x)))
        x = self.dropout(self.conv2_bn(x))
        x = self.pool(F.elu(self.conv3(x)))
        x = self.dropout(self.conv3_bn(x))
        x = x.view(-1, 250 * 3 * 3)
        x = F.elu(self.fc1(x))
        x = self.dropout(self.fc_bn(x))
        x = self.fc2(x)
        return x

```

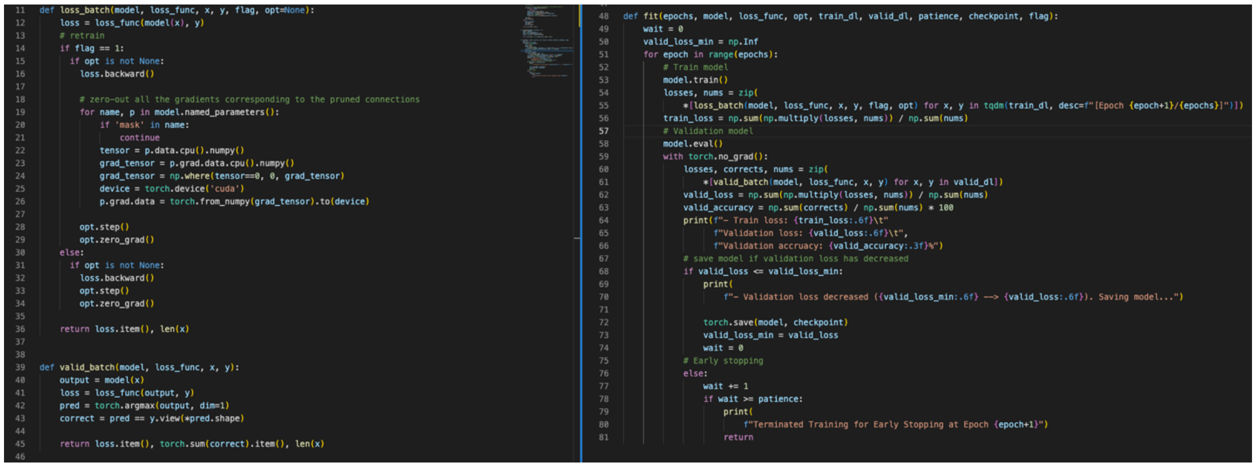
```python
class Stn(PruningModule):
    def __init__(self):
        super(Stn, self).__init__()
        # Spatial transformer localization-network
        self.loc_net = nn.Sequential(
            nn.Conv2d(1, 50, 7),
            nn.MaxPool2d(2, 2),
            nn.ELU(),
            nn.Conv2d(50, 100, 5),
            nn.MaxPool2d(2, 2),
            nn.ELU()
        )
        # Regressor for the 3 * 2 affine matrix
        self.fc_loc = nn.Sequential(
            MaskedLinear(100 * 4 * 4, 100),
            nn.ELU(),
            MaskedLinear(100, 3 * 2)
        )
        # Initialize the weights/bias with identity transformation
        self.fc_loc[2].weight.data.zero_()
        self.fc_loc[2].bias.data.copy_(torch.tensor(
            [1, 0, 0, 0, 1, 0], dtype=torch.float))

    def forward(self, x):
        xs = self.loc_net(x)
        xs = xs.view(-1, 100 * 4 * 4)
        theta = self.fc_loc(xs)
        theta = theta.view(-1, 2, 3)

        grid = F.affine_grid(theta, x.size())
        x = F.grid_sample(x, grid)

        return x
```

Model Training Code Snapshot:



```python
def loss_batch(model, loss_func, x, y, flag, opt=None):
    loss = loss_func(model(x), y)
    # retrain
    if flag == 1:
      if opt is not None:
        loss.backward()

        # zero-out all the gradients corresponding to the pruned connections
        for name, p in model.named_parameters():
            if 'mask' in name:
                continue
            tensor = p.data.cpu().numpy()
            grad_tensor = p.grad.data.cpu().numpy()
            grad_tensor = np.where(tensor==0, 0, grad_tensor)
            device = torch.device('cuda')
            p.grad.data = torch.from_numpy(grad_tensor).to(device)

        opt.step()
        opt.zero_grad()
    else:
      if opt is not None:
        loss.backward()
        opt.step()
        opt.zero_grad()

    return loss.item(), len(x)


def valid_batch(model, loss_func, x, y):
    output = model(x)
    loss = loss_func(output, y)
    pred = torch.argmax(output, dim=1)
    correct = pred == y.view(*pred.shape)

    return loss.item(), torch.sum(correct).item(), len(x)

```

```python
def fit(epochs, model, loss_func, opt, train_dl, valid_dl, patience, checkpoint, flag):
    wait = 0
    valid_loss_min = np.Inf
    for epoch in range(epochs):
        # Train model
        model.train()
        losses, nums = zip(
            *[loss_batch(model, loss_func, x, y, flag, opt) for x, y in tqdm(train_dl, desc=f"[Epoch {epoch+1}/{epochs}]")])
        train_loss = np.sum(np.multiply(losses, nums)) / np.sum(nums)
        # Validation model
        model.eval()
        with torch.no_grad():
            losses, corrects, nums = zip(
                *[valid_batch(model, loss_func, x, y) for x, y in valid_dl])
            valid_loss = np.sum(np.multiply(losses, nums)) / np.sum(nums)
            valid_accuracy = np.sum(corrects) / np.sum(nums) * 100
            print(f"- Train loss: {train_loss:.6f}\t"
                  f"Validation loss: {valid_loss:.6f}\t",
                  f"Validation accruacy: {valid_accuracy:.3f}%")
            # save model if validation loss has decreased
            if valid_loss <= valid_loss_min:
                print(
                    f"- Validation loss decreased ({valid_loss_min:.6f} --> {valid_loss:.6f}). Saving model...")

                torch.save(model, checkpoint)
                valid_loss_min = valid_loss
                wait = 0
            # Early stopping
            else:
                wait += 1
                if wait >= patience:
                    print(
                        f"Terminated Training for Early Stopping at Epoch {epoch+1}")
                    return
```