**Classification de Panneaux de Signalisation Routière**

1. **Data exploration**

**Training dataset**

* Number of images: 39209
* Number of classes: 43
* Maximum Width in Pixel: 243
* Maximum Height in Pixel: 225

**Testing dataset**

* Number of images: 39209
* Number of classes: 43

**Have a look on random image of each class**

A collage of different road signs

Description automatically generated

**Define the label dictionary**

Based on the clear image of ./Meta folder, we can find out the class name corresponding to the classID as follows.

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' }

**Showing the number of Training Images for each ClassID.**

A graph of blue bars

Description automatically generated with medium confidence

The number of images for each classID is not balanced. This can cause model bias, when it is more likely to learn features from the dominant classID, and as a result make more correct predictions for dominant classes than the minority classes.

**Showing distributions in Image sizes, including Width and Height**

A graph of a graph of a graph

Description automatically generated with medium confidence

We see that most of the images have less than 50 pixel in width and height.

1. Data Loading

class GTSRBDataset(Dataset):

    def \_\_init\_\_(self, csv\_file, root\_dir, transform=None):

        self.data = pd.read\_csv(csv\_file)

        self.root\_dir = root\_dir

        self.transform = transform

    def \_\_len\_\_(self):

        return len(self.data)

    def \_\_getitem\_\_(self, idx):

        img\_path = os.path.join(self.root\_dir, self.data.iloc[idx, 7]) *# Path is in column 7*

        image = Image.open(img\_path).convert('RGB')

        label = self.data.iloc[idx, 6] *# ClassId is in column 6*

        if self.transform:

            image = self.transform(image)

        return image, label

1. Data processing and preparation

*# Define data augmentation and normalization transformations*

transform = transforms.Compose([

    transforms.Resize(image\_size),

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

*# Load datasets*

train\_dataset = GTSRBDataset('./data/Train.csv', './data', transform=transform)

test\_dataset = GTSRBDataset('./data/Test.csv', './data', transform=transform)

*# Split train dataset into train and validation*

train\_size = int(0.8 \* len(train\_dataset))

val\_size = len(train\_dataset) - train\_size

train\_data, val\_data = random\_split(train\_dataset, [train\_size, val\_size])

*# Create data loaders*

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

val\_loader = DataLoader(val\_data, batch\_size=batch\_size, shuffle=False)

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

1. Building base CNN model

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Hyperparameter configuration

image\_size = (32, 32)

batch\_size = 16

num\_epochs = 10

validation\_split = 0.2

lr=0.001

1. Experimentation to improve model performance

More complex CNN

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Data Augmentation

During training we apply several transformation techniques on the fly to increase the variety of training image, that help model more robust and against model overfitting.

transform = transforms.Compose([

    transforms.Resize(image\_size),

    transforms.RandomHorizontalFlip(),

    transforms.RandomRotation(10),

    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),

    transforms.ToTensor(),

    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

])

We show here several examples of image after applying the data augmentation techniques.



There are several options we can experiment with

|  |  |  |
| --- | --- | --- |
| **Experiments** | **Configuration** | **Performance (F1)** |
| Experiment 1 | Base model: 3 Conv layers, batch=16, epoch=10 | 0.94 |
| Experiment 2 | Experiment 1 with more complex model: 5 Conv layers with normalization and dropout | 0.96 |
| Experiment 3 | Experiment 2 with batch size 32 | 0.96 |
| Experiment 4 | Experiment 3 with Data augmentation | 0.94 |

The training performance is logged like this

A screenshot of a computer program

Description automatically generated

1. Performance analysis

Showing the accuracy and loss log of training process

A comparison of graphs with numbers

Description automatically generated with medium confidence

The confusion matrix is then calculated.

A graph of numbers and a graph of numbers

Description automatically generated with medium confidence

Print also the classification report

A screenshot of a graph

Description automatically generated

With the best model, we can plot several predictions. The correct predictions in blue and incorrect predictions in red.

A collage of different road signs

Description automatically generated

We can show also the incorrect predictions to analyse more why the predictions is incorrectA collage of different road signs

Description automatically generated

1. CNN visualization

We can also show the activation layer of Conv layers in CNN model like this

A screenshot of a green and blue image

Description automatically generated

A screenshot of a computer generated image

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

1. Transfer learning with VGG

*# Define transfer learning model*

class TransferLearningVGG(nn.Module):

    def \_\_init\_\_(self, num\_classes=43):

        super(TransferLearningVGG, self).\_\_init\_\_()

*# Load the pre-trained VGG model*

        self.vgg = models.vgg16\_bn(pretrained=True)

*# Freeze the pre-trained layers*

        for param in self.vgg.parameters():

            param.requires\_grad = False

*# Modify the classifier to match the number of classes*

        self.vgg.classifier[6] = nn.Linear(4096, num\_classes)

    def forward(self, x):

        x = self.vgg(x)

        return F.log\_softmax(x, dim=1)

A graph of different colored lines

Description automatically generated with medium confidence

As the expriments are only set for 10 epoche, we see that the VGG model using transfer learning with pretrained VGG weight converge lower than the above small CNN models. That may be because that the VGG take input size as (224x224), while our training images are as small as around (50 x 50). The resize process cause the quality of image worse, therefore the model take more time to converge.