Data Mining Project Final Report

Vishwanath Venkataraman

31600783

Project Title: Traffic Sign Recognition

Link to Dataset: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

Data Exploration:

The data set contains 3 folders and 3 csvs each named Meta, train and test.

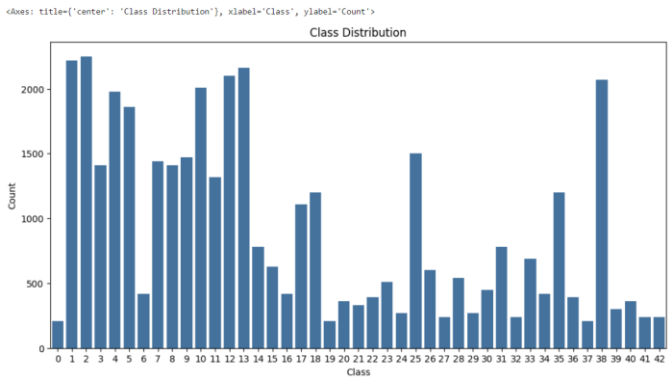
The Meta folder contains clear images of the 43 different road signs and are named according to their class ID.

The Test folder has a collection of images of traffic signs which don't seem to be ordered in any way.

The train folder has subfolders separated by class id and each folder has 200-2500 images.

In total the test folder contains 12630 samples while the train folder contains 39209 samples.

The meta csv contains the path for the image along with the class Id, shape id and color id of each image. In total there are 43 class ids, 5 shape ids and 4 color ids.



The train and test csvs contain the paths to the images, their width, height and their class Id. Along with this we also have coordinates marking the Roi (Region of interest)

5]:

	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

]:

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	27	26	5	5	22	20	20	Train/20/00020_00000_00000.png
1	28	27	5	6	23	22	20	Train/20/00020_00000_00001.png
2	29	26	6	5	24	21	20	Train/20/00020_00000_00002.png
3	28	27	5	6	23	22	20	Train/20/00020_00000_00003.png
4	28	26	5	5	23	21	20	Train/20/00020_00000_00004.png

Data Preparation:

The images of varying sizes were resized to a 50x50 size. As an alternative I did try to resize the images to 224x224 but when the training the model the accuracy did not go above 5 percent.

As mentioned above the train folder has folders of different classes with each folder containing images of the class. After scouring through plenty of resources online, I found a method that didn't use os commands to iterate through each folder to extract the images.

I used the ImageDataGenerator library from tensorflow to prepare the train dataset for training and split it into a train and validation set in one go.

[8]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   validation_split = 0.2)
train_data = train_datagen.flow_from_directory(train_path,
                                               target_size = (height,width),
                                               batch_size = batch_size,
                                               seed = seed,
                                               shuffle = True,
                                               class_mode = 'categorical',
                                               color_mode = 'rgb',
                                               interpolation = 'hamming',
                                               subset = 'training')
```

```
Found 31368 images belonging to 43 classes.
```

[9]:
```
test_datagen = ImageDataGenerator(rescale = 1./255,
                                  validation_split = 0.2)
test_data = test_datagen.flow_from_directory(train_path,
                                             target_size = (height,width),
                                             batch_size = batch_size,
                                             seed = seed,
                                             shuffle = True,
                                             class_mode = 'categorical',
                                             color_mode = 'rgb',
                                             interpolation = 'hamming',
                                             subset = 'validation')
```

```
Found 7841 images belonging to 43 classes.
```

Models:

VGG16:

VGG16 (Visual Geometry Group 16) is a CNN architecture with 16 convolutional and fully connected layers. It has a simple and uniform architecture where the convolutional layers have 3x4 filters and the max pooling layers have 2x2 windows. Each convolutional block has multiple convolutional layers followed by a max pooling layer and each hidden layer has a relu activation function.

VGG16 is a popular choice for image classification tasks owing to its simplicity and effectiveness.

ResNet50V2:

ResNet50V2 is the improved version of ResNet50. The V2 model introduced skip connections to address the vanishing gradient problem in neural networks.

As mentioned above the V2 model uses skip connections or residual connection to enable the flow of gradients during back propagation. The basic building block here is a residual block which includes the skip connections.

It follows a bottleneck architecture where the number of filters is reduced before the spatial dimensions are reduced and then expanded again.

Batch normalization and Global Average pooling are applied at the end to improve training stability and reduce the chance of overfitting.

Compared to VGG16 ResNet is a more modern and resilient model providing. The V2 especially provides better convergence and generalization when compared to its earlier architectures.

Training:

To detect and classify the signs I used the VGG16 model and compared its performance to that of the Resnet50V2 model. To the both the above models I added a Dense relu layer for detection and a Dense Softmax layer to classify it into the appropriate class. A popular choice instead of the relu layer was the sigmoid layer when it comes to binary classification because it gives the output between 0 to 1. I used relu for its simplicity and its ability to avoid the vanishing gradient issue.

I have attached the model summary below for a better idea of the layers involved.

```
[10]: vgg_model = Sequential([VGG16(weights = 'imagenet',
                                   include_top = False,
                                   input_shape = (height,width,3)),
                              keras.layers.BatchNormalization(),
                              Flatten(),
                              Dense(512, activation = 'relu'),
                              Dense(43, activation = 'softmax')
                              ])
      vgg_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 batch_normalization (Batch  (None, 1, 1, 512)         2048
 Normalization)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 512)               262656

 dense_1 (Dense)             (None, 43)                22059

=================================================================
Total params: 15001451 (57.23 MB)
Trainable params: 15000427 (57.22 MB)
Non-trainable params: 1024 (4.00 KB)
_________________________________________________________________
```

```
: resnet_model = Sequential([ResNet50V2(weights = 'imagenet',
                                        include_top = False,
                                        input_shape = (height,width,3)),
                             keras.layers.BatchNormalization(),
                             keras.layers.GlobalAveragePooling2D(),
                             Dense(512, activation = 'relu'),
                             Dense(43, activation = 'softmax')
                             ])
  resnet_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras
94668760/94668760 [==============================] - 1s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 2, 2, 2048)        23564800

 batch_normalization_1 (Bat  (None, 2, 2, 2048)        8192
 chNormalization)

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 512)               1049088

 dense_3 (Dense)             (None, 43)                22059

=================================================================
Total params: 24644139 (94.01 MB)
Trainable params: 24594603 (93.82 MB)
Non-trainable params: 49536 (193.50 KB)
_________________________________________________________________
```

I ran both models for 25 epochs at a learning rate of 0.001.

At the end of training the VGG16 model gave us an accuracy of 0.9964 and a loss value of 0.2792 while the ResNet model gave us an accuracy of 0.9974 and loss value of 0.2351.

From the graph we can see the model hasn't overfit.

Now that training is done, it is time to test this model.

I used a for loop to pull out the images from the test folder and put it into a list. And created a function to interpret the model's prediction according to class labels we have.

Result:

After Testing the models, the VGG model gave an accuracy of 69.15% while the ResNet model gave a prediction accuracy of 80.36%.

VGG:

ResNet:

Challenges Faced:

- It took a while to figure out a method to extract images from the train and test folder without having to iterate over each image. The problem being here that the train data has over 30,000 images.
- I could not avoid having to iterate over each image for the test folder but there was the only way to extract the image while maintaining its relation to its ClassID in the test CSV.
- Since it is a DL model, I ran it on Kaggle which albeit being fast because of the accelerators provided, would start hanging after 15 minutes and will forcibly close after. Using the joblib library to save the model and loading it later was the only way to get the accuracy and loss values per epoch.