Traffic Sign Recognition

Name: Ahmad Kaddoura

The goal of this project was to build a neural network model capable of classifying German traffic signs using a convolutional network built on the Lenet-5 architecture. This is achieved by having a dataset of labeled traffic signs and preprocessing the training set to prepare the model to be trained. The model is then cross tested on a dev or valid set and tuning hyperparameters until a prediction accuracy of over 93% was achieved. When I was satisfied with the model's performance, I tested it again on the test set to confirm it's capable of the desired performance parameter.

While the lectures introduced us to Tensorflow, I've chosen to use Keras to complete this project instead as it's simpler to build the model and I had already completed a previous course which introduced me to both frameworks.

Dataset Exploration

The data consists of about 35k photos of German traffic signs resized to 32x32 pixels with accompanying labels for each which are stored in a csv file. It also contains about 4.5k images in a dev set and 12.5k images functioning as a test set.

I started by building a visualization of the data set which displays a random set of images from the training dataset along with their labels. The image quality differed as some were clearly visible while others suffered from blurriness or dark shadows which in theory is a good thing as it allows the model to be trained more efficiently on imperfect images. However, some might have been too blurry as in one instance I was unable to identify the sign which made me wonder how close the human level performance would be.

Design and Model Architecture

Preprocessing

I started off the design by creating a quick softmax model using only three layers of neurons to get a feeling for the models performance. This allowed me to quickly gauge the effect of some preprocessing techniques without wasting a lot of time or computational power as I test image processing techniques.

The effect of gray scaling the images was immediately evident as it creates an easier contrast between the white space on signs or the symbols themselves. Furthermore, I performed a normalization across the pixel values of each image by dividing by 255. In truth, I tested many different normalization values and in different ways including diving by 128 or taking a normalized mean around 128 and 255 and having no normalization at all.

In the end, dividing by 255 yielded the best result in my numerous tests even if the difference was at times minimal. Moreover, after building the Lenet model I was still getting a performance output of around 91% and shuffling the dataset prior to training finally pushed me towards the 94% mark which is acceptable.

Model Architecture

The architecture used in this model is based on the Lenet-5 model first proposed by Yann LeCun et al. It functions on an input of 32x32 pixel images in grayscale. The image is fed into the first conv layer of 6 5x5 filters and (1,1) stride. This is followed by three steps which will be repeated later in the following order:

1. Batch Normalization: A layer to normalize the parameter values around a mean of zero. This makes weighs deeper in the network more robust to changes in earlier layers. However, this network is rather small compared to other architectures like ResNet or VGG so the effect might not be as apparent.
2. A relu activation layer to obtain output information from the layer's computation.
3. A 2x2 Max pooling layer to control the number of channels and filters.

This is fed into another conv layer of 16 5x5 filters which is then fed into the same three layers. Before the output of the max pool is flattened in preparation to be connected to the fully connected layers. The first consists of 120 layers with relu activations which is then fed into 84 layers.

Finally, the output is fed into a soft-max layer which will give us an array of probabilities for each label. The highest one will be taken as the predicted result.

Model Training

The model was trained by feeding the X_train dataset into the neural network and using an adam optimizer which is short for Adaptive Moment Estimation. This acts as a combination of two other optimizers: gradient momentum and RMS Prop and is a good fit for this sort of problem.

Other parameters of training the system was using a batch size of 128 along 30 epochs. A batch size of 128 is higher than some other applications which typically use a batch size of 64 but I chose 128 to reduce the covariant shift effect of the parameters from layer to layer.

Solution Approach

As mentioned before, I managed to reach a 0.94 accuracy by adjusting my preprocessing step by converting images to grayscale, shuffling the data set and adding normalization.

Test on New Images

For this step I found five random German traffic sign images and rescaled them to 32x32 to fit into the classifier. At a glance they looked rather similar to some of the other images I've seen in the dataset and it was rather simple to preprocess using the same steps and run them through the model.

In particular, I chose two which might be tricky for the model such as the plain white circle "No vehicle" sign and the bumpy road. I was interest in the first image's complete white space and how the model might react to it as the neurons might struggle to pinpoint interesting features while the other image was a triangular sign with a pixelated sign at the bottom.

Performance on New Images

On the new five images, the model got only three out of five images correctly which gives an accuracy of 0.6 which is very low compared to the model's performance earlier. This however can be attributed to the low number of images and perhaps its poor performance on one of the particular images.

I also output the soft max probabilities of the model's predications to get a sense of how close it might have been to the desired result. In the case of the first incorrectly classified image, the correct output was the third best prediction while in the other incorrect test image, the correct output was the fourth best prediction. So while it did not present the correct prediction, it was somewhat close and some further tweaking or additional layers might assist in reaching a more favorable outcome.

Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         (None, 32, 32, 1)         0         
_________________________________________________________________
conv1 (Conv2D)               (None, 28, 28, 6)         156       
_________________________________________________________________
bn_conv1 (BatchNormalization (None, 28, 28, 6)         24        
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 6)         0         
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 6)         0         
_________________________________________________________________
conv2 (Conv2D)               (None, 10, 10, 16)        2416      
_________________________________________________________________
bn_conv2 (BatchNormalization (None, 10, 10, 16)        64        
_________________________________________________________________
activation_4 (Activation)    (None, 10, 10, 16)        0         
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0         
_________________________________________________________________
fc1 (Dense)                  (None, 120)               48120     
_________________________________________________________________
fc2 (Dense)                  (None, 84)                10164     
_________________________________________________________________
fc3 (Dense)                  (None, 43)                3655      
=================================================================
Total params: 64,599
Trainable params: 64,555
Non-trainable params: 44
```