Traffic Sign Classifier

Using Convolutional Neural Networks

William Galindez Arias

In this project I tried different experiments with convolutional Neural networks using the LeNet architecture as a starting point, and as a dataset, a collection German Traffic Signs images.

Rubric Points

1. Basic Summary of the dataset

Using pandas *dataframe* I read the labels and Class IDs for the 43 different classes in the dataset. With *numpy* and shape method the dimensions of the dataset were found, in addition:

```
In [5]: n_train = len(X_train)
        n_validation = len(X_valid)
        n_test = len(X_test)
        image_shape = X_train[0].shape
        n_classes = len(set(y_test))
```

Fig1. Lines of Code used to explore the dataset

Number of training examples	34799
Number of testing examples	12630
Image Data shape	(32, 32, 3)
Number of clases	43

Table 1. Summary of dataset

2. Exploratory Visualization of the dataset

I used the *matplotlib* histogram function in order to have a visual on how the distribution of the dataset was split, I could observe that there were more images in some classes than others with highly marked differences. 43 bins were used

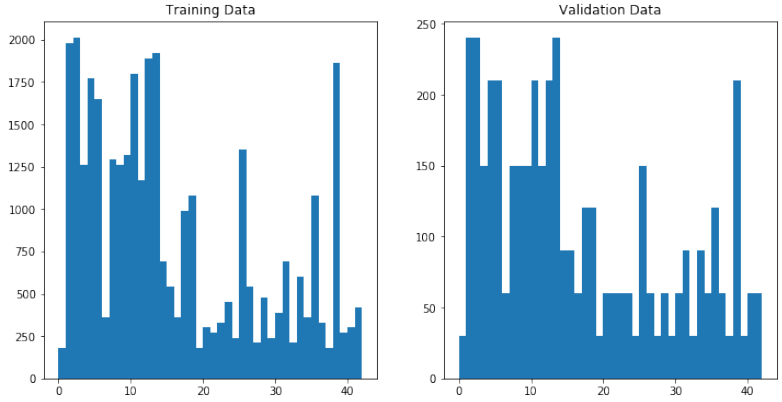


Fig2. Distribution of classes in training and validation data

Additionally, a visual inspection was carried out with random selection of images

Fig3. Visual inspection of the dataset (32,32,3)

3. Designing and test a Model Architecture

3.1 Preprocessing the image

- The images were standardized, making their pixels values be between [0-1] with mean zero, this was achieved by dividing the images by 255 using np.divide.
- A color selection was executed by using openCV and transforming the image to YUV color space, as mentioned in the original paper of LeNet (http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf)

Example of a preprocessed image:

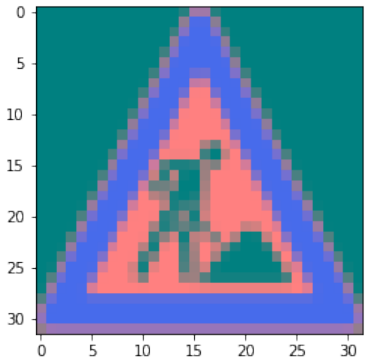


Fig4. Preprocessed image

3.2 Neural Network architecture

Consisted of the following layers, further details of implementation can be observed in the jupyter notebook.

Convolution	Relu	Dropout	MaxPooling	Convolution2	Relu	Dropout	MaxPooling	FullyCon

Input (32,32,3)

3.3 Model Training

The model was trained using similar approach as discussed in the LeNet architecture and lecture

```
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=logits)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)
prediction = tf.argmax(logits, 1)
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

fig5. Neural network pipeline

As optimizer, Adam was used, 50 epochs and a batch of 128 and a learning rate of 0.001, the choosing of this values came by iteration, different epochs where selected and batches, I could observe that more than 50 epochs wouldn't make any difference in the obtained accuracy likewise with the other hyperparameters.

I took advantage of the GPU in AWS to experiment with different architectures, I experimented by doing:

1. Adding more convolutional layers, the training time increased, nevertheless there was not significant improvement in the accuracy above 93%
2. Adding dropout, with the intent of regularize and avoid overfitting of that the neural network would memorize the dataset
3. MaxPooling to help in the processing task and improve generalization of the neural net.
4. I added a second fully connected layer just to observe if there was any significant improvement in the final step of classification, there was not meaningful increase in the accuracy

These are the final values obtained:

```
INFO:tensorflow:Restoring parameters from ./lenet

  Training set accuracy = 0.999
Validation set accuracy = 0.951
      Test set accuracy = 0.946
```

Fig6. Evaluation values obtained

As can be derived from the values obtained, the accuracy is slightly above 93% and after having tried different architectures without getting a significant improvement, I came to the conclusion that an educated guess could be to do Data Augmentation, by using different techniques learned in previous projects, such as: Distortion correction, cropping and detecting in regions of interest, blur, rotate and transform the images to help the model generalize better and fill the gap in the unbalanced training test, as could be seen from the histogram plotting.

5. **Test on new images**

Images from Wikipedia German Traffic Signs were downloaded
https://en.wikipedia.org/wiki/Road_signs_in_Germany

The accuracy was 100% given the fact that new images that were utilized had even better quality than the ones provided in the dataset, which in fact was the expected result, the usage of these images should not fail, and if it would be the case, it could prove that something really wrong was in the model.

The new images, were preprocessed and resized to meet the requirements of the model,

```
:  for image in (x_new):
       plt.figure()
       plt.imshow(image)
```

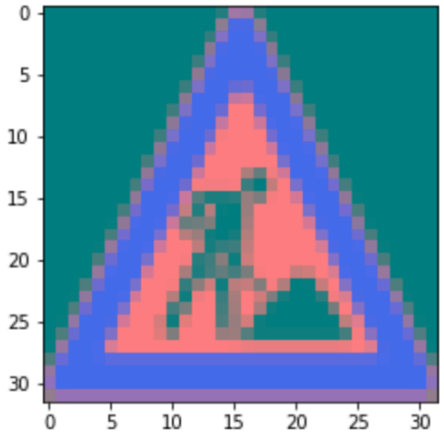


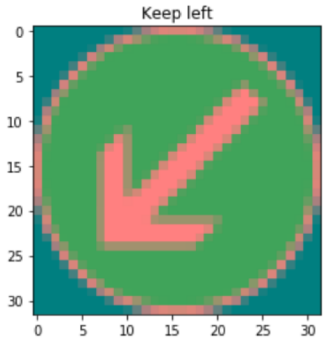
Fig7. New image preprocessed fed into the model

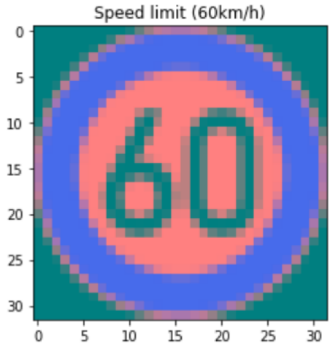
6. **Predictions:**

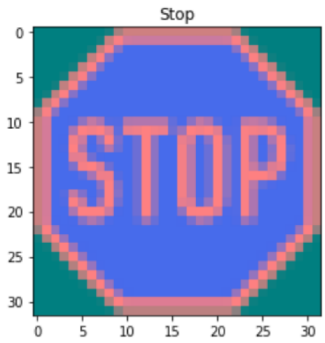
All the predictions were correct, by combining the class labels in the csv file, the image obtained and the output from the model: ndarray:

```
[25  3 39 14]
```

we get the predictions shown in the images below and that can be observed in detail in the jupy ter notebook







Softmax output

```
INFO:tensorflow:Restoring parameters from ./lenet

[[  9.99998e-01   8.35876e-07   6.00434e-07   1.88250e-07   4.99211e-08]
 [  9.63001e-01   3.69991e-02   1.93807e-12   1.78132e-14   7.49293e-17]
 [  1.00000e+00   5.13522e-09   1.51031e-09   6.02147e-11   3.42353e-15]
 [  9.99938e-01   4.23082e-05   1.94350e-05   1.55136e-07   8.94959e-08]]
```

As it can be seen, it coincides with the values obtained and show the application of the softmax function as a distribution of probabilities for a multiclass selection task