Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

The goal of this project is to design and implement the convolutional neural networks for traffic signs recognition.
Built model is trained on images from the German Traffic Sign Database.

The steps of this project are the following:

1. Load the data set. Explore, summarize and visualize the data set
2. Design, train and test a model architecture
3. Use the model to make predictions on new images
4. Analyze the softmax probabilities of the new images

Step 1: Dataset Summary & Exploration ¶

There are three datasets provided: training, validation and testing. The pickled data is a dictionary with 4 key/value pairs:

- `'features'` is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- `'labels'` is a 1D array containing the label/class id of the traffic sign. The file `signnames.csv` contains id -> name mappings for each id.
- `'sizes'` is a list containing tuples, (width, height) representing the original width and height the image.
- `'coords'` is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image.

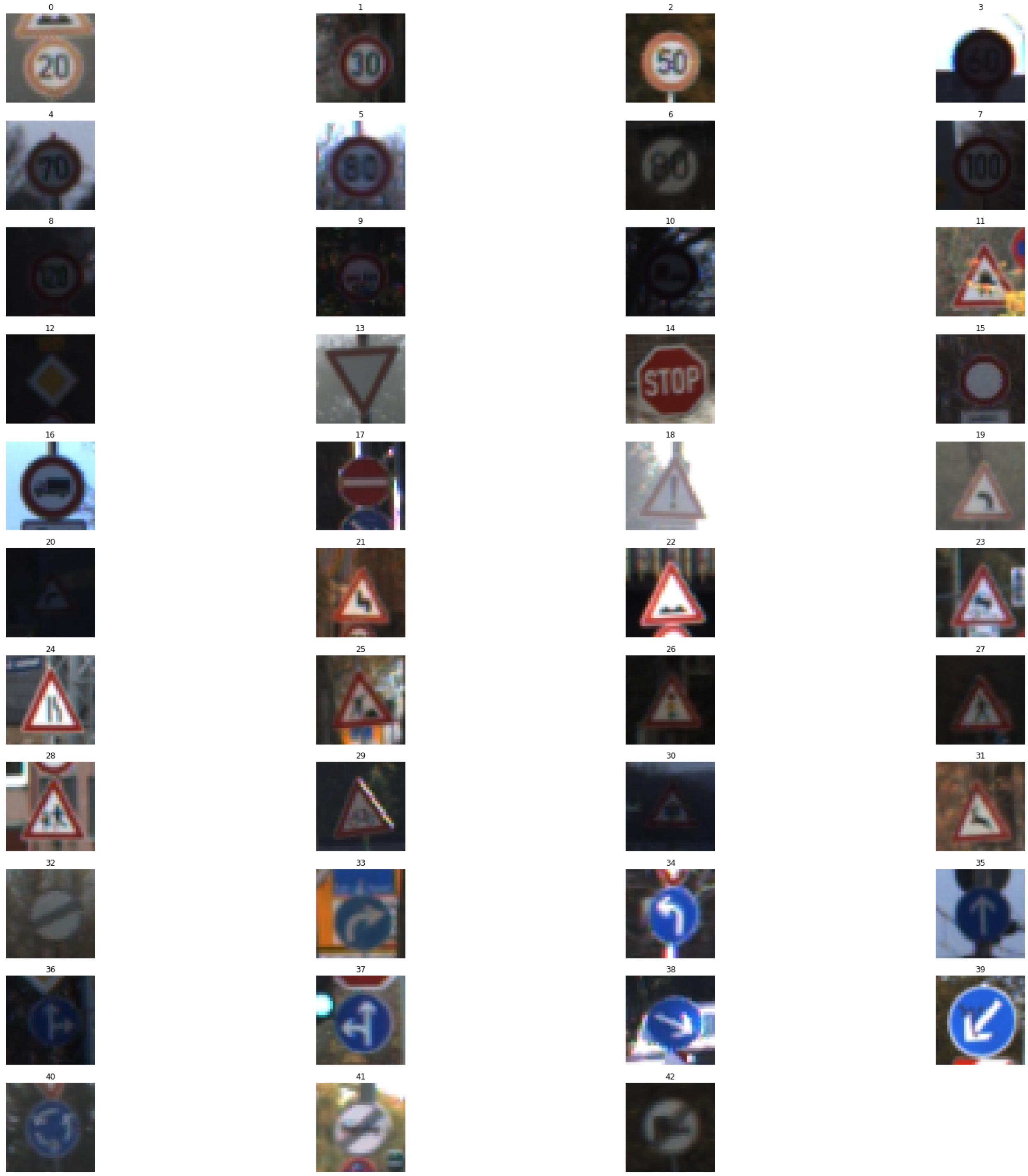
Number of training examples is **34799**;
Number of testing examples is **12630**;
Number of validation examples is **4410**;
Image data shape is **32x32**;
Trere are **43** classes.

Classes names and corresponding indices

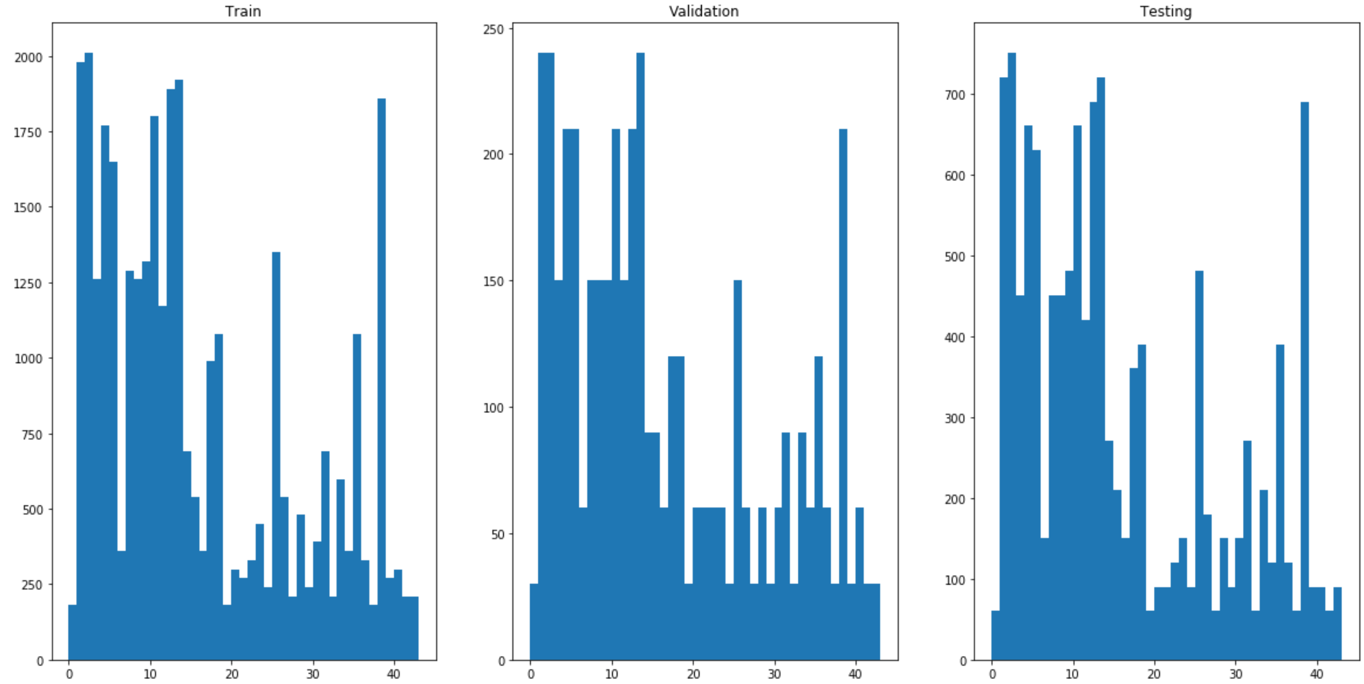
In [34]:
```python
import pandas as pd

print(pd.read_csv('signnames.csv'))
```

```
    ClassId                                           SignName
0         0                               Speed limit (20km/h)
1         1                               Speed limit (30km/h)
2         2                               Speed limit (50km/h)
3         3                               Speed limit (60km/h)
4         4                               Speed limit (70km/h)
5         5                               Speed limit (80km/h)
6         6                        End of speed limit (80km/h)
7         7                              Speed limit (100km/h)
8         8                              Speed limit (120km/h)
9         9                                         No passing
10       10       No passing for vehicles over 3.5 metric tons
11       11              Right-of-way at the next intersection
12       12                                      Priority road
13       13                                              Yield
14       14                                               Stop
15       15                                        No vehicles
16       16           Vehicles over 3.5 metric tons prohibited
17       17                                           No entry
18       18                                    General caution
19       19                        Dangerous curve to the left
20       20                       Dangerous curve to the right
21       21                                       Double curve
22       22                                         Bumpy road
23       23                                      Slippery road
24       24                          Road narrows on the right
25       25                                          Road work
26       26                                    Traffic signals
27       27                                        Pedestrians
28       28                                  Children crossing
29       29                                  Bicycles crossing
30       30                                 Beware of ice/snow
31       31                              Wild animals crossing
32       32                End of all speed and passing limits
33       33                                   Turn right ahead
34       34                                    Turn left ahead
35       35                                         Ahead only
36       36                               Go straight or right
37       37                                Go straight or left
38       38                                         Keep right
39       39                                          Keep left
40       40                               Roundabout mandatory
41       41                                  End of no passing
42       42  End of no passing by vehicles over 3.5 metric ...
```

Classes representatives



Classes distribution in training, validation and testing sets



Step 2: Design and Test a Model Architecture

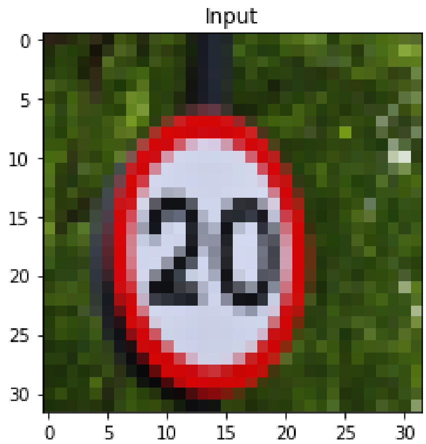
Design and implement a deep learning model that learns to recognize traffic signs. Train and test your model on the [German Traffic Sign Dataset (http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset)](http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset).

The LeNet-5 implementation shown in the [classroom (https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81)](https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/6df7ae49-c61c-4bb2-a23e-6527e69209ec/lessons/601ae704-1035-4287-8b11-e2c2716217ad/concepts/d4aca031-508f-4e0b-b493-e7b706120f81) was taken as a base.

The final validation set accuracy is **0.95**.

Generate fake data

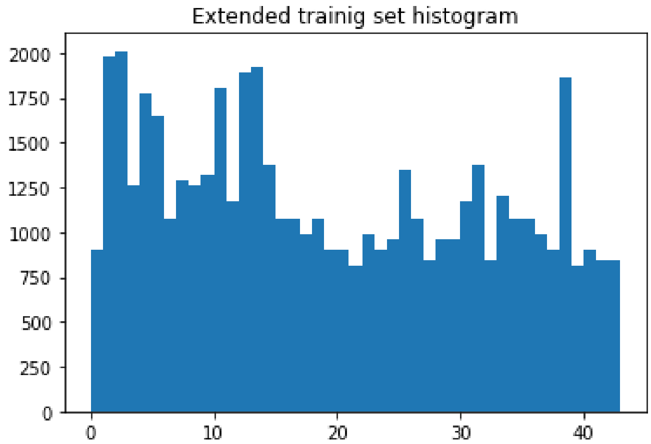
If the amount of class samples is less than mean value. Then for every sample in this class several new samples will be generated to fill the gap between original and mean numbers.
Two ways of transformation were implemented. First, rotate image by random angle in range ±10. Then translate this image.
Second method applyes affine tranformaion by randomly choosing source and destination points in certain range.
In both cases the image brightness can be also changed.


Input
20

For following classes were generated additional samples:

- For class 0 generate 4 new images based on each image. Total 720.
- For class 6 generate 2 new images based on each image. Total 720.
- For class 14 generate 1 new images based on each image. Total 690.
- For class 15 generate 1 new images based on each image. Total 540.
- For class 16 generate 2 new images based on each image. Total 720.
- For class 19 generate 4 new images based on each image. Total 720.
- For class 20 generate 2 new images based on each image. Total 600.
- For class 21 generate 2 new images based on each image. Total 540.
- For class 22 generate 2 new images based on each image. Total 660.
- For class 23 generate 1 new images based on each image. Total 450.
- For class 24 generate 3 new images based on each image. Total 720.
- For class 26 generate 1 new images based on each image. Total 540.
- For class 27 generate 3 new images based on each image. Total 630.
- For class 28 generate 1 new images based on each image. Total 480.
- For class 29 generate 3 new images based on each image. Total 720.
- For class 30 generate 2 new images based on each image. Total 780.
- For class 31 generate 1 new images based on each image. Total 690.
- For class 32 generate 3 new images based on each image. Total 630.
- For class 33 generate 1 new images based on each image. Total 599.
- For class 34 generate 2 new images based on each image. Total 720.
- For class 36 generate 2 new images based on each image. Total 660.
- For class 37 generate 4 new images based on each image. Total 720.
- For class 39 generate 2 new images based on each image. Total 540.
- For class 40 generate 2 new images based on each image. Total 600.
- For class 41 generate 3 new images based on each image. Total 630.
- For class 42 generate 3 new images based on each image. Total 630.

Classes distribution in augmented training set

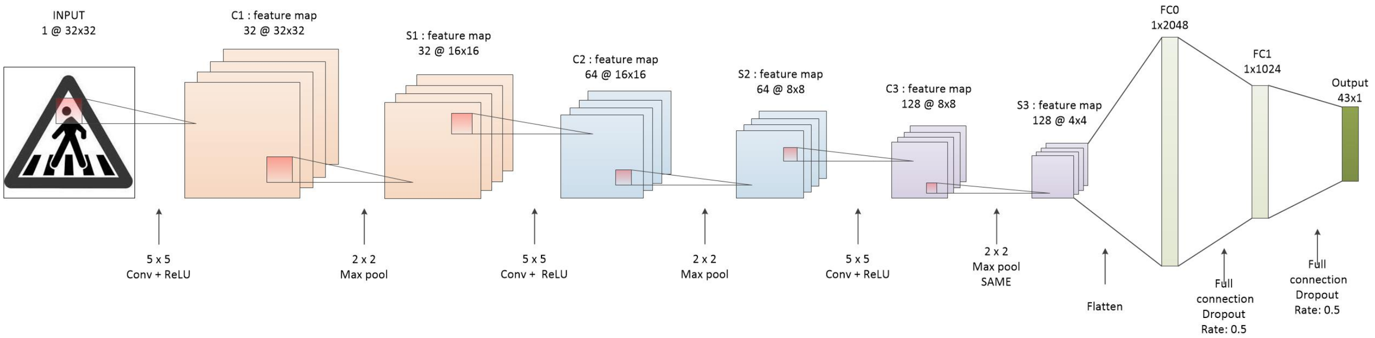


Pre-process the Data Set

The image data are normalized so that the data has mean zero and equal variance. For image data, `(pixel - 128)/ 128` is a quick way to approximately normalize the data.

Then data were converted to grayscale. The color information can be ignored in this task due to the reason there are no two traffic signs which differ only by color.

Model Architecture



The model architecture is similar to **Lenet**. It has three convolutional layers and three fully connected layers.

Layer	Description
Input	32x32x1 Grayscale image
Convolution 5x5	1x1 stride, 'SAME' padding, output 32x32x32, Leaky ReLU
Max pooling	window size 2, output 16x16x32
Convolution 5x5	1x1 stride, 'SAME' padding, output 16x16x64, Leaky ReLU
Max pooling	window size 2, output 8x8x64
Convolution 5x5	1x1 stride, 'SAME' padding, output 8x8x128, Leaky ReLU
Max pooling	window size 2, output 4x4x128
Fully connected	Input 2048, output 1024, Leaky ReLU
Dropout	Drop rate 0.5
Fully connected	Input 1024, output 340, Leaky ReLU
Dropout	Drop rate 0.5
Fully connected	Input 340, output 43
Softmax	

In [7]:
```
def TrafficSignsNet(features, dropout_rate, is_training):
    # Arguments used for tf.truncated_normal, randomly defines variables for the weights
 and biases for each layer
    mu = 0
    sigma = 0.1
    alpha = 0.01 # Leaky ReLU parameter
    padding = 'SAME'

    ####################################################################################
#########
    # Layer 1: Convolutional. Input = 32x32x1. Output = 32x32x32.
    conv1_K = 32
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, conv1_K), mean = mu, stddev
= sigma))
    conv1_b = tf.Variable(tf.zeros(conv1_K))
    conv1   = tf.nn.conv2d(features, conv1_W, strides=[1, 1, 1, 1], padding=padding) + co
nv1_b

    # Activation. Leaky ReLU
    conv1 = tf.nn.leaky_relu(conv1, alpha)


    # Max Pooling. Input = 32x32x32. Output = 16x16x32.
    conv1 = max_pool(conv1, 2, name='conv1', padding=padding)

    ####################################################################################
#########
    # Layer 2: Convolutional. Input = 16x16x32 Output = 16x16x64.
    conv2_K = 64
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, conv1_K, conv2_K), mean = mu,
stddev = sigma))
    conv2_b = tf.Variable(tf.zeros(conv2_K))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding=padding) + conv2
_b

    # Activation. Leaky ReLU
    conv2 = tf.nn.leaky_relu(conv2, alpha)


    # Max Pooling. Input = 16x16x64. Output = 8x8x64.
    conv2 = max_pool(conv2, 2, name='conv2', padding=padding)

    ####################################################################################
#########
    # Layer 3: Convolutional. Input = 8x8x64 Output = 8x8x128.
    conv3_K = 128
    conv3_W = tf.Variable(tf.truncated_normal(shape=(5, 5, conv2_K, conv3_K), mean = mu,
stddev = sigma))
    conv3_b = tf.Variable(tf.zeros(conv3_K))
    conv3   = tf.nn.conv2d(conv2, conv3_W, strides=[1, 1, 1, 1], padding=padding) + conv3
_b

    # Activation. Leaky ReLU
    conv3 = tf.nn.leaky_relu(conv3, alpha)

    # Max Pooling. Input = 8x8x128. Output = 4x4x128.
    conv3 = max_pool(conv3, 2, name='conv3', padding=padding)

    ####################################################################################
#########
    # Flatten. Input = 4x4x128. Output = 2048.
    fc0    = flatten(conv3)
    fc0_out = fc0.get_shape().as_list()[1]

    ####################################################################################
#########
    # Layer 4: Fully Connected. Input = 2048. Output = 1024.
    fc1_out = 1024
    fc1_W = tf.Variable(tf.truncated_normal(shape=(fc0_out, fc1_out), mean = mu, stddev =
```

```
sigma))
    fc1_b = tf.Variable(tf.zeros(fc1_out))

    fc1 = tf.add(tf.matmul(fc0, fc1_W), fc1_b, name='fc1')

    # Activation.
    fc1    = tf.nn.leaky_relu(fc1, alpha)
    # Dropout
    fc1 = tf.layers.dropout(inputs=fc1, rate=dropout_rate, training=is_training)

    # Layer 5: Fully Connected. Input = 1024. Output = 340.
    fc2_out = 340
    fc2_W = tf.Variable(tf.truncated_normal(shape=(fc1_out, fc2_out), mean = mu, stddev =
sigma))
    fc2_b = tf.Variable(tf.zeros(fc2_out))

    fc2 = tf.add(tf.matmul(fc1, fc2_W), fc2_b, name='fc2')

    # Activation.
    fc2    = tf.nn.leaky_relu(fc2, alpha)
    # Dropout
    fc2 = tf.layers.dropout(inputs=fc2, rate=dropout_rate, training=is_training)

    ####################################################################################
#########
    # Layer 6: Fully Connected. Input = 340. Output = 43.
    fc3_out = 43
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(fc2_out, fc3_out), mean = mu, stddev
= sigma))
    fc3_b  = tf.Variable(tf.zeros(fc3_out))

    logits = tf.add(tf.matmul(fc2, fc3_W), fc3_b, name='fc3')

    return logits
```

`x` is a placeholder for a batch of input images.
`y` is a placeholder for a batch of output labels.
`drop_rate` is a dropout rate, between 0 and 1. E.g. "rate=0.1" would drop out 10% of input units.
`is_training` indicates whether to return the output in training mode (apply dropout) or in inference mode (return the input untouched).

`learn_rate` is one of the most important hyperparameters. Decides 'how rough' the weights are updated during training. This parameter is set to **0.001**.

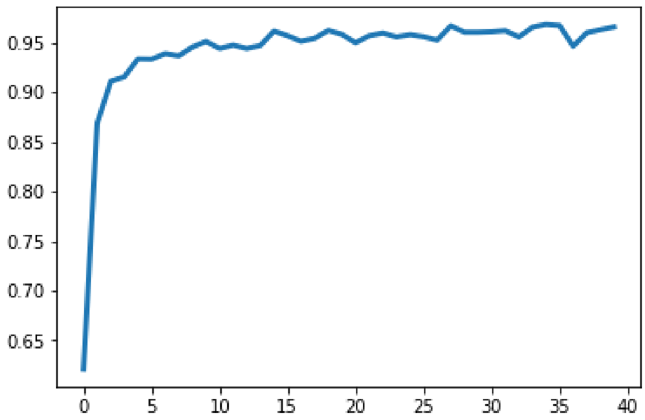
Softmax cross entropy between logits and labels is computed by using *tf.nn.softmax_cross_entropy_with_logits_v2* function. The result is provided as an input to *tf.reduce_mean* function. It is used to compute the total cost.
As an oprimizer the **Adam optimization algorithm** is used. It is an extension to stochastic gradient descent and one of the most used optimization algorithms.

Train, Validate and Test the Model

A validation set is used to assess how well the model is performing.

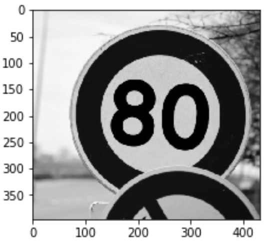
Epoch	Validation Accuracy	---	Epoch	Validation Accuracy
1	0.621	---	21	0.950
2	0.869	---	22	0.957
3	0.911	---	23	0.959
4	0.915	---	24	0.956
5	0.933	---	25	0.958
6	0.933	---	26	0.956
7	0.939	---	27	0.952
8	0.937	---	28	0.967
9	0.945	---	29	0.960
10	0.951	---	30	0.960
11	0.944	---	31	0.961
12	0.947	---	32	0.962
13	0.944	---	33	0.956
14	0.947	---	34	0.965
15	0.961	---	35	0.968
16	0.957	---	36	0.967
17	0.951	---	37	0.946
18	0.954	---	38	0.960
19	0.962	---	39	0.963
20	0.958	---	40	0.966

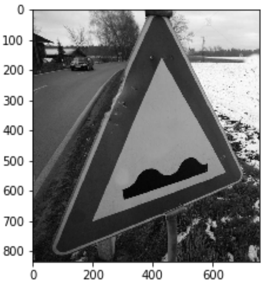
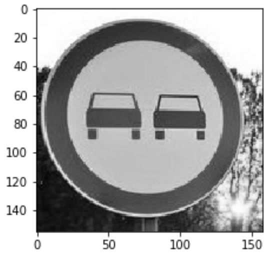
Validation accuracy plot through epochs



Step 3: Test a Model on New Images

5 german traffic signs images were downloaded. Images were normalized and converted to grayscale. Sign Type was correcty predicted for every image.



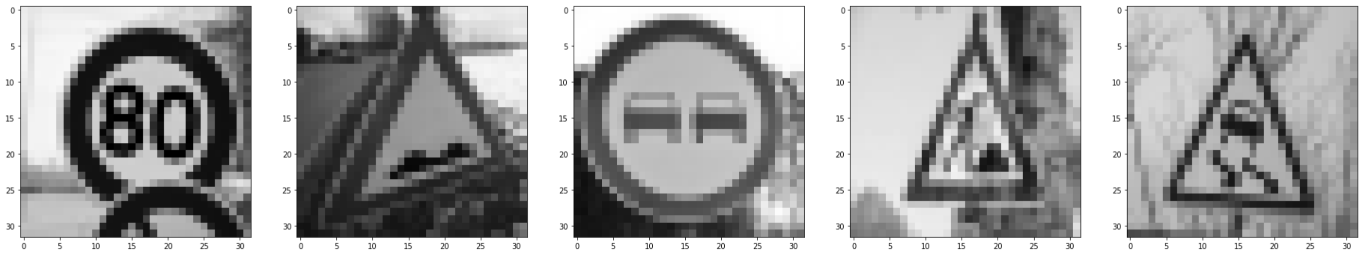
Predict the Sign Type for Each Image

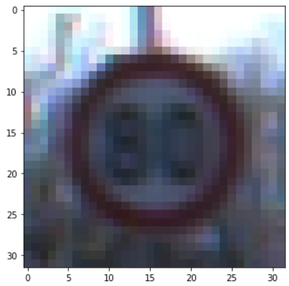
traffic_signs_net accurace on testing test set is **0.950**

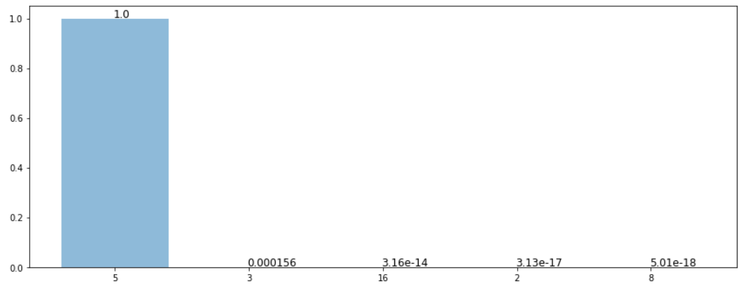
Analyze Performance

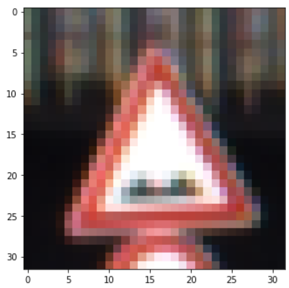
Output Top 5 Softmax Probabilities For Each Image Found on the Web

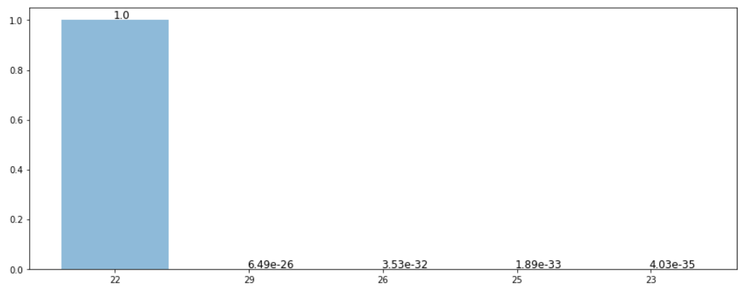
For each of the new images, the model's softmax probabilities to show the **certainty** of the model's predictions were printed out.

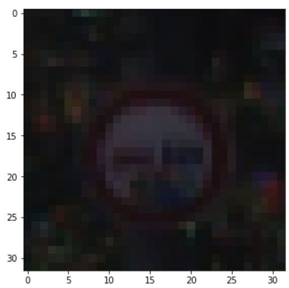


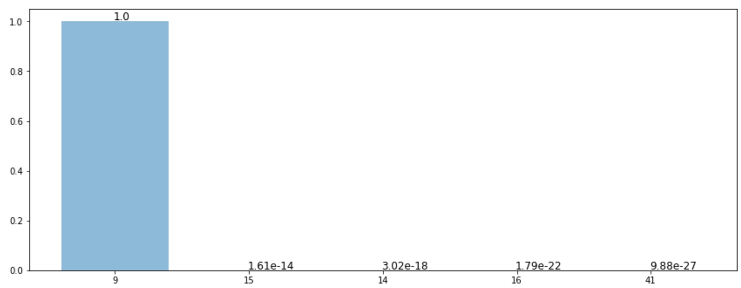


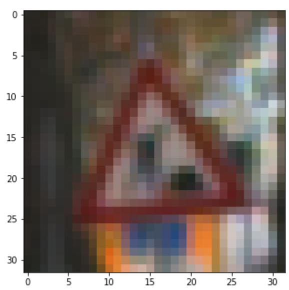

1.0
0.000156
3.16e-14
3.13e-17
5.01e-18
5
3
16
2
8

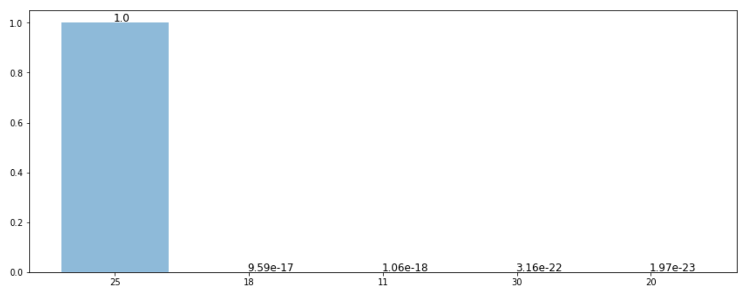


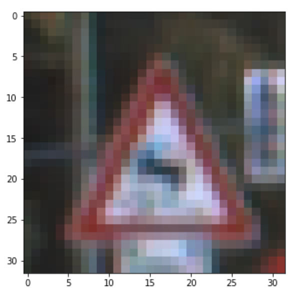

1.0
6.49e-26
3.53e-32
1.89e-33
4.03e-35
22
29
26
25
23

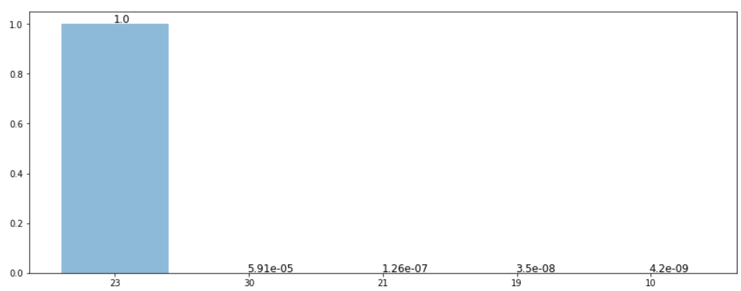



1.0
1.61e-14
3.02e-18
1.79e-22
9.88e-27
9
15
14
16
41




1.0
9.59e-17
1.06e-18
3.16e-22
1.97e-23
25
18
11
30
20




1.0
5.91e-05
1.26e-07
3.5e-08
4.2e-09
23
30
21
19
10

All signs were classified correctly.

Idx	Description	Probability	---	Idx	Description	Probability	---	Idx	Description	Probability
5	**Speed limit (80km/h)**	9.9984431e-01	---	22	**Bumpy road**	1.0000000e+00	---	9	**No passing**	1.0000000e+00
3	Speed limit (60km/h)	1.5569913e-04	---	29	Bicycles crossing	6.4850363e-26	---	15	No vehicles	9.8773351e-27
16	Stop	3.1646842e-14	---	26	Traffic signals	3.5302702e-32	---	14	Stop	3.0215690e-18
2	Speed limit (50km/h)	3.1258351e-17	---	25	Road work	1.8902869e-33	---	16	Vehicles over 3.5 metric tons prohibited	1.7943873e-22
8	Speed limit (120km/h)	5.0107557e-18	---	23	Slippery road	4.0317657e-35	---	41	End of no passing	4.0317657e-35

Idx	Description	Probability	---	Idx	Description	Probability
25	**Road work**	1.0000000e+00	---	23	**Slippery road**	9.9994075e-01
18	General caution	9.5928652e-17	---	30	Beware of ice/snow	5.9071997e-05
11	Right-of-way at the next intersection	1.0563458e-18	---	21	Double curve	1.2561594e-07
30	Beware of ice/snow	3.1563438e-22	---	19	Dangerous curve to the left	3.5005552e-08
20	Dangerous curve to the right	1.9739832e-23	---	10	No passing for vehicles over 3.5 metric tons	4.1987342e-09

Reflection

The main parts of the task were:

- random image generator for training database extension;
- design and implement the network architecture;
- tune hyperparameters.

For designed network following hyperparameters had to be set:

- learning rate : 0.001
- drop rate : 0.5
- batch size : 128
- epochs : 40

Also:

- type of padding : 'SAME'
- stride : 1
- convolution filter size : 5x5
- size of intermediate fully connected layers: 1024, 340
- coefficient for leaky ReLU : 0.01
- pooling strategy : max pooling

Initial weights and bias were initialized by tf.truncated_normal function. This is the recommended initializer for neural network weights and filters.
Leaky ReLU activation function was chosen in order to prevent dead ReLU and initialize ReLU neurons with slightly positive biases.

Deep neural network learning process requires a lot of tuning hyperparameters. Due to high level of freedom it's almost impossible to manully try all combinations. In further research it would be interesting to try Hyperopt functionality, although it requires a lot of computational power.

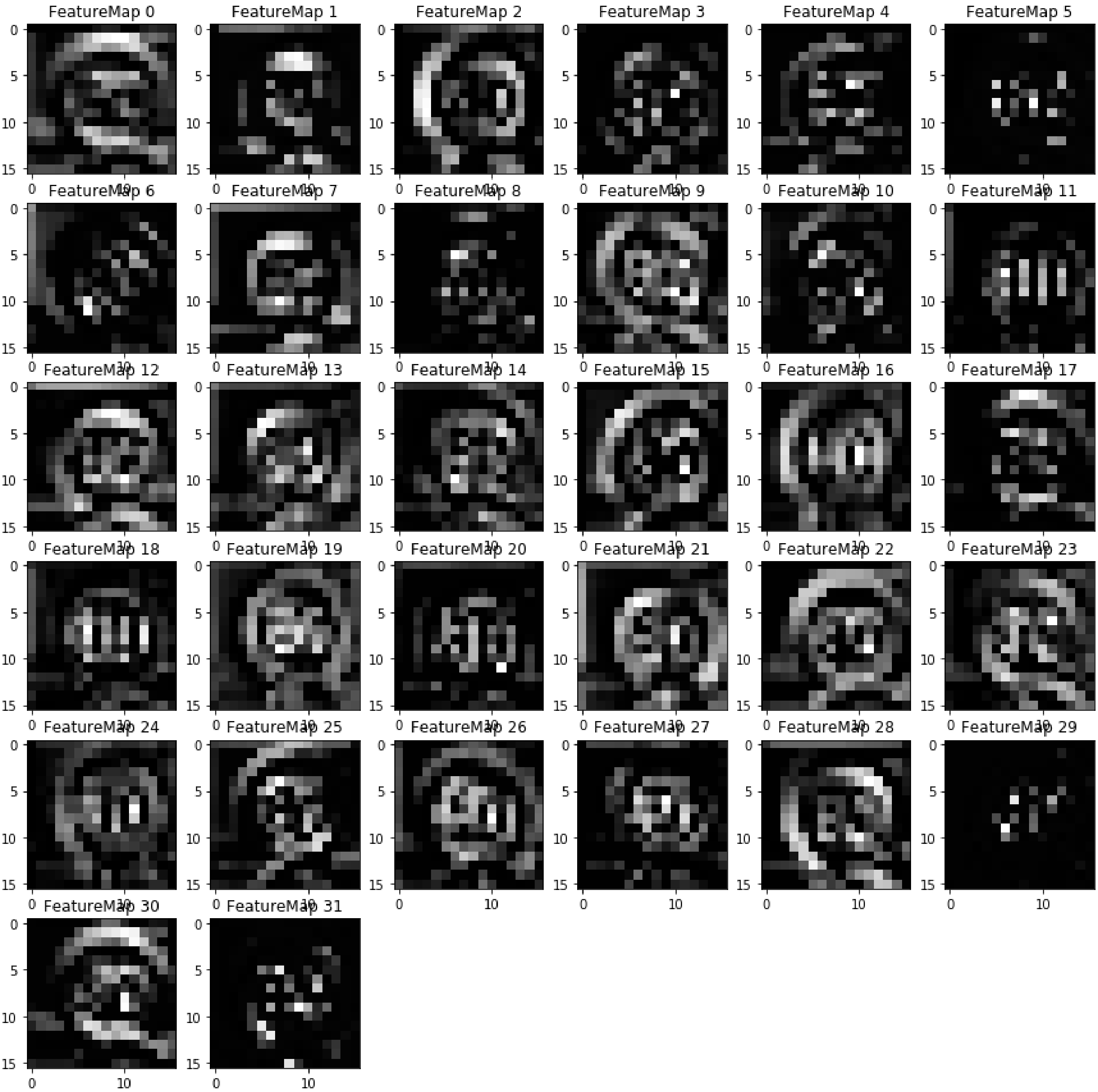
Also an interesting observation: Many articles recommend to replace dropout by batch normalization. Or add batch normalization between every convolutional layer before or after pooling. In this case it gave worse results: 0.3 after 5th epoch. Drop out gave us 0.933 after 5th epoch.

Step 4 (Optional): Visualize the Neural Network's State with Test Images

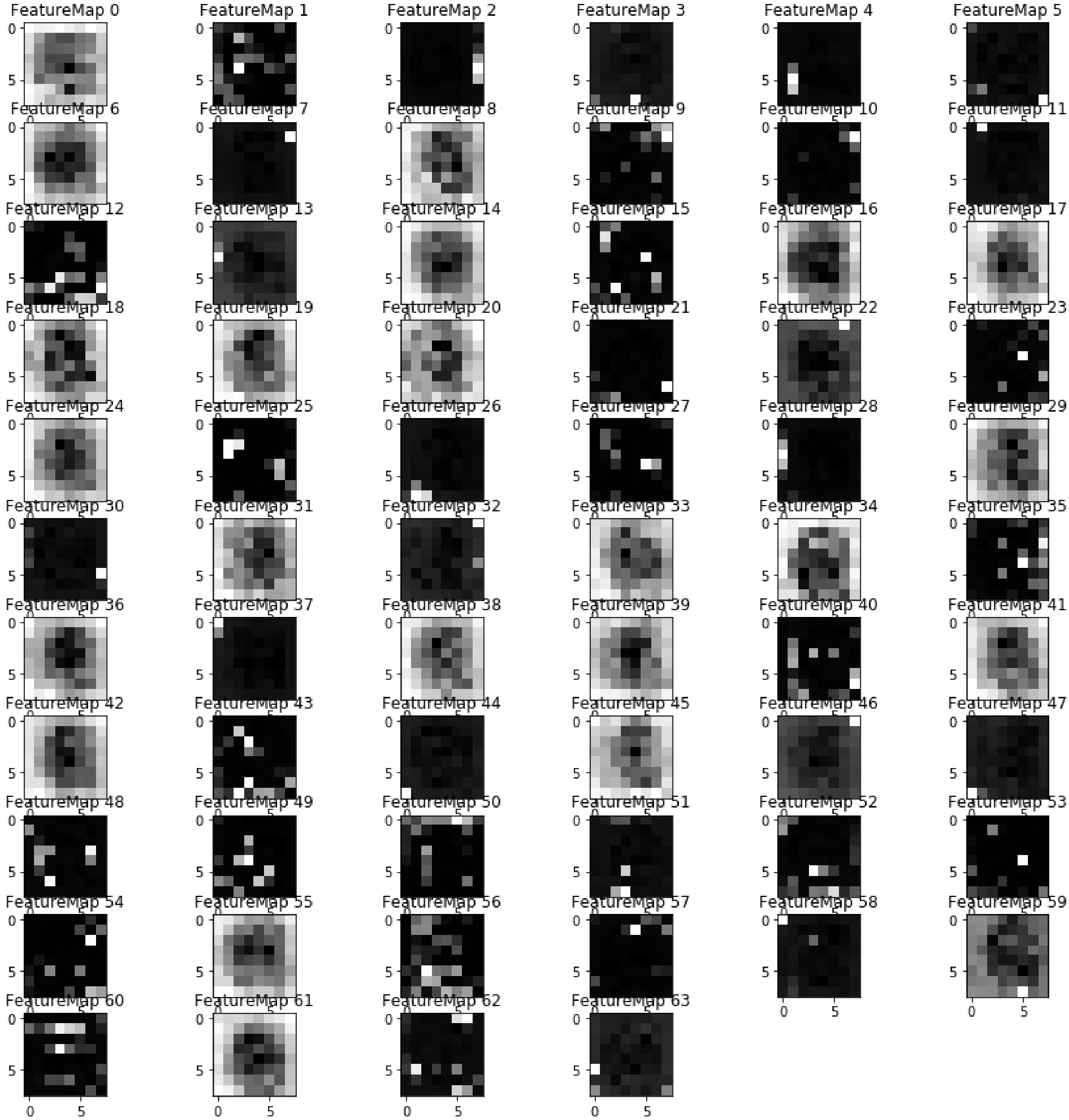
Feature maps for first and second convolutional layers are plotted below.

From these plotted feature maps, it's possible to see what characteristics of an image the network finds interesting. For a sign, the inner network feature maps react with high activation to the sign's boundary outline or to the contrast in the sign's painted symbol.

1st convolutional layer



2nd convolutional layer


FeatureMap 0
FeatureMap 1
FeatureMap 2
FeatureMap 3
FeatureMap 4
FeatureMap 5
FeatureMap 6
FeatureMap 7
FeatureMap 8
FeatureMap 9
FeatureMap 10
FeatureMap 11
FeatureMap 12
FeatureMap 13
FeatureMap 14
FeatureMap 15
FeatureMap 16
FeatureMap 17
FeatureMap 18
FeatureMap 19
FeatureMap 20
FeatureMap 21
FeatureMap 22
FeatureMap 23
FeatureMap 24
FeatureMap 25
FeatureMap 26
FeatureMap 27
FeatureMap 28
FeatureMap 29
FeatureMap 30
FeatureMap 31
FeatureMap 32
FeatureMap 33
FeatureMap 34
FeatureMap 35
FeatureMap 36
FeatureMap 37
FeatureMap 38
FeatureMap 39
FeatureMap 40
FeatureMap 41
FeatureMap 42
FeatureMap 43
FeatureMap 44
FeatureMap 45
FeatureMap 46
FeatureMap 47
FeatureMap 48
FeatureMap 49
FeatureMap 50
FeatureMap 51
FeatureMap 52
FeatureMap 53
FeatureMap 54
FeatureMap 55
FeatureMap 56
FeatureMap 57
FeatureMap 58
FeatureMap 59
FeatureMap 60
FeatureMap 61
FeatureMap 62
FeatureMap 63