Project 3: Traffic Sign Classifier

The goal of this project is to implement concepts of Deep Neural Network and Convolutional Neural Network to image classification and classify German Traffic Signs

This is achieved in following steps:

1. Load German Traffic Sign dataset containing training, validation and test sets
2. Visualize the dataset (I did this by displaying one image from each class and plotting a bar chart showing number of images in each class for all three sets)
3. Design, train and test a model architecture
4. Tune hyper parameters to generate a highly accurate model
5. Use this model to make predictions on new images
6. Analyze softmax probabilities of new images
7. Visualize the Neural Network at its convolutional layer

Data Set Summary and Exploration

- Traffic Sign Dataset –
  - The traffic sign data to be used for training model is saved in three pickle files –
    - *train.p*
    - *valid.p*
    - *test.p*
  - Pickled data is a dictionary with 4 values
    - Features - 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels)
    - Labels - 1D array containing the label/class id of the traffic sign
    - Sizes - list containing tuples, (width, height) representing the original width and height the image
    - Coords - list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image.
  - The file 'signnames.csv' contains traffic sign ids for each class and their name mappings
  - Images in the pickle files are all color images resized to 32X32
- Dataset Summary –
  - I used shape function from Numpy library on 'features' and 'labels' of pickle dataset to determine number of examples in all three datasets and along with width, height and channels in each image.
    - Number of training examples = 34799
    - Number of validation examples = 4410
    - Number of testing examples = 12630
    - Image data shape = (32, 32, 3)
  - I determined number of unique traffic signs used in the training dataset using the numpy function 'unique'
    - Number of classes = 43
  - Using readcsv function from pandas library, I mapped traffic sign names of all 43 classes to their IDs
- Data Visualization –
  - For visualization of the dataset I plotted one random image from each class from training set along with its mapped traffic sign name

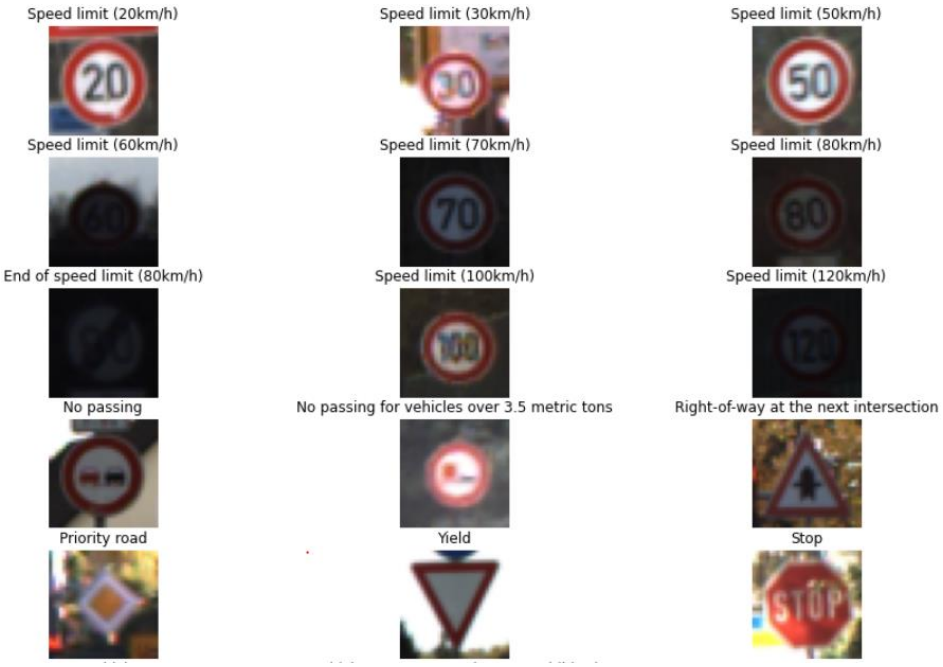

Figure 1: Example of data visualization

- I also plotted a bar chart of showing number of images per class in each set

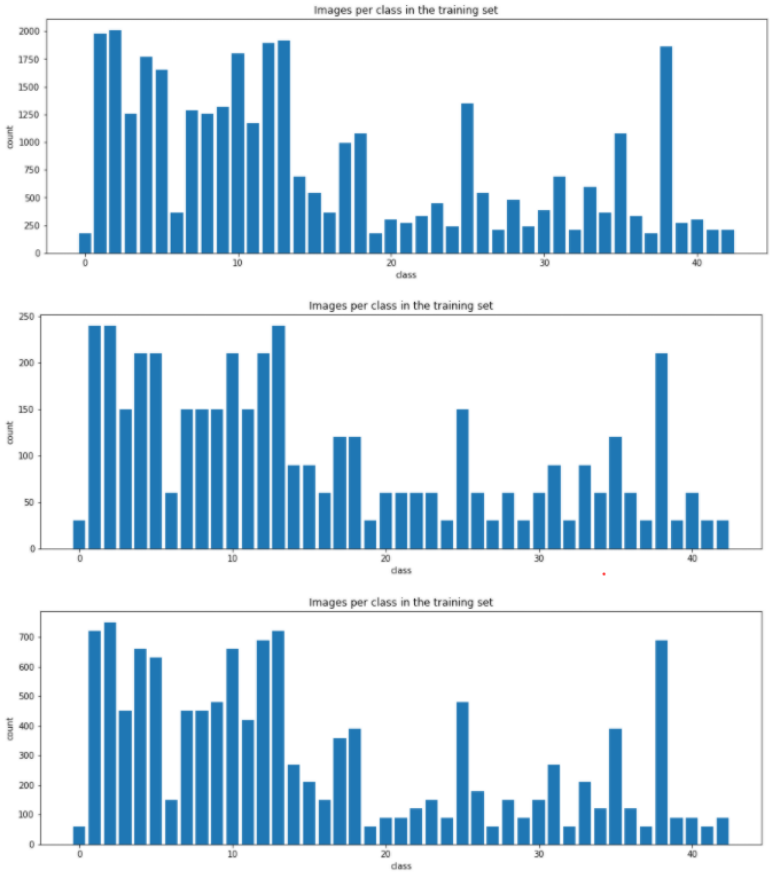

Figure 2: number of images per class

Design and Test a Model Architecture

Training set is used to design and tune a deep learning model to accurately recognize traffic signs from validation and test set as well as traffic signs belonging to one of the 43 defined classes but which are taken from the web and are not in either of the three sets.

- Preprocess the dataset –
  - Grayscale – used to convert three channel data into one channel. This is useful to reduce training time
  - Normalization – Data is normalized to (-1,1) so that it has approximately zero mean and equal variance. This is done to compress data distribution and simplify training of the dataset using single learning rate
  - Shuffling – It's important to include this step to avoid the model training to get influenced by sequence of the images in the dataset
- Model Architecture –
  I used LeNet architecture described in the classroom with some modifications as described below.

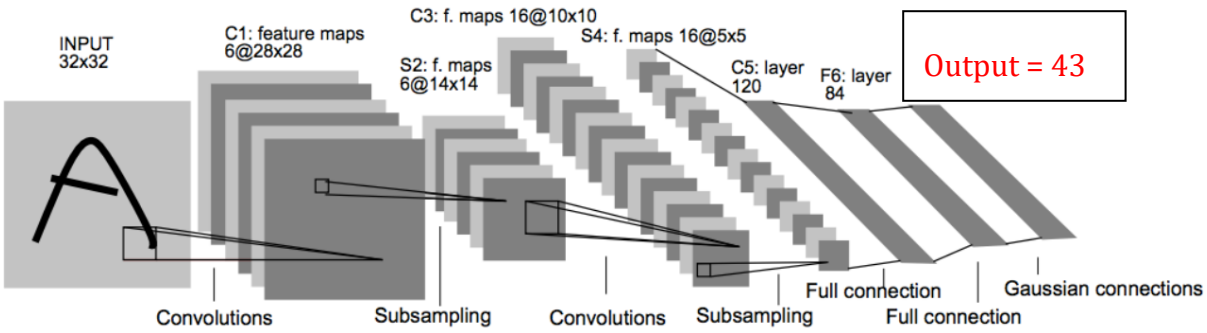

Figure 3: Original LeNet Model Architecture

  - My LeNet model architecture is based on the architecture LeNet lab model from classroom (As shown in the above figure) and it includes following layers
    1. **Layer 1**: Convolutional. Input = 32x32x1. Output = 28x28x6, Filter = 5x5, Stride = 1x1
    2. Activation – RELU
    3. Pooling, Input = 28x28x6. Output = 14x14x6, kernel = 2x2, Stride = 2x2
    4. **Layer 2**: Convolutional. Input = 14x14x6 Output = 10x10x16, Filter = 5x5, Stride = 1x1
    5. Activation – RELU
    6. Pooling. Input = 10x10x16. Output = 5x5x16, kernel = 2x2, Stride = 2x2
    7. **Layer 3**: Convolutional. Input = 5x5x16 Output = 1x1x400, Filter = 5x5, Stride = 1x1
    8. Activation – RELU
    9. Flatten. Input = 1x1x400. Output = 400.
    10. Dropout
    11. **Layer 4**: Fully Connected. Input = 400. Output = 120 (sigma = 0.1)
    12. Activation – RELU
    13. **Layer 5**: Fully Connected. Input = 120. Output = 84 (sigma = 0.1)
    14. Activation – RELU
    15. **Layer 6**: Fully Connected. Input = 84. Output = 43 (sigma = 0.1)

- output of this layer is modified to 43 to match number of classes of the traffic signs

- This LeNet model architecture follows a classic structure of Convolutional Neural Network consisting of convolutional layers followed by activation and pooling layers, followed by fully connected layers
    1. Convolutional Layer – First three layers of LeNet model are convolutional layers activated by RELU (to make the model nonlinear) and followed by max pooling (to reduce input size while focusing on most important features of the image by retaining only maximum values of each filtered area)
        - After three sets of convolutional layers, the output size is modified to 1X1X400
        - Formulae used to define output size of each convolutional layer – $out_{height} = \frac{in_{height} - filter_{height} + 1}{stride}$, $out_{width} = \frac{in_{width} - filter_{width} + 1}{stride}$
        - Formulae used to define output size after max pooling – $out_{height} = \frac{in_{height} - filter_{height}}{stride} + 1$, $out_{width} = \frac{in_{width} - filter_{width}}{stride} + 1$
    2. Flatten – output of the third convolutional layer is flattened from 1x1x400 to a vector of length 400.
    3. Dropout – Partial data output from layer three is dropped randomly so that network learns to train the model redundantly is independent of any particular set of data within training set. For training set I have kept the dropout rate to 80% and during testing I keep it at 0% so that all units are utilized to maximize power of the model. This is adjusted by the parameter keep_prob.
    4. Fully connected layers – output of the dropout is then passed through 3 fully connected layers with standard deviation set at 0.1
    5. Final Output layer – Final output layer has a size of 43, equal to number of unique classes of German traffic signs. This are logits, output of the LeNet architecture.
- Hyper-parameters for training of the model –
    1. Optimizer – Adam Optimizer from LeNet lab in the classroom.
    2. Batch size – 128
    3. Epochs - 40
    4. Learning rate – 0.001
    5. Mu – 0
    6. Sigma – 0.1
    7. Dropout keep probability = 0.2
- Approach for hyper parameter tuning to train model
    1. I used Adam Optimizer from LeNet lab in the classroom. It is an optimization algorithm for stochastic gradient descent for training deep learning models.
    2. I started with same hyper parameter settings as used in the LeNet lab from classroom (Batch size – 128, Epochs – 40, Learning rate – 0.001)
    3. As can be seen in the figure below, with only 10 epochs, although the training time was very low, the model does not achieve a very high accuracy.
    4. So as a next step, I tried combinations of decreasing learning rate, increasing number of epochs and increasing batch size to reach current combination of hyperparameters, which gave me a training accuracy of 0.998 and a validation accuracy of 0.962.
    5. Learning rate tells tensor flow how quickly to update network's weights. Epoch is the number of times our training data is passed through network. More number of epoch is beneficial for higher accuracy but the model takes longer to train. Batch size decides the number of training images to run through the network at a time. Larger batch size results in faster model training but it is limited by the processor limit.

6. I kept mu and sigma at their default values
7. Based on high accuracy of training and validation models and validation accuracy reaching a constant value in the following figure. I concluded that the model fit is sufficiently accurate.

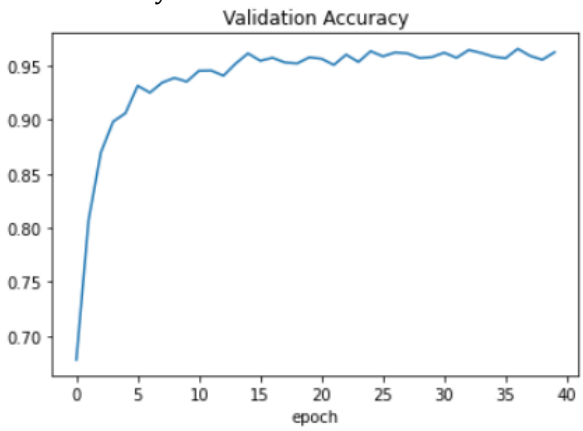

Figure 4: Validation Accuracy with epoch

- Training pipeline
  1. First I set up the tensorflow placeholder variables – x to hold input batch of image dimension 32x32x1and y to store labels. This labels are integers which are then one hot encoded
  2. Next, I passed the input data x to from the LeNet function as descried above to calculate logits
  3. These logits are then compared with one hot encoded label values using tf.nn.softmax_cross_entropy_with_logits() functions and calculate cross entropy to measure difference between logits and ground truth labels.
  4. Tf.reduce() mean function is used to average this cross entropy
  5. This is passed through Adam Optimizer to minimize the loss function similar to Stochastic gradient descent
  6. Finally, I used the minimize() function on the optimizer to backpropogate the training network and minimize training loss
- Evaluation Pipeline
  1. Once model is trained, it is passed through this pipeline to evaluate its accuracy.
  2. This model measures whether a given prediction is correct by comparing logits with one hot encoded y labels
  3. Then we calculate model's overall accuracy by taking a mean of individual prediction accuracy
  4. This is done by dividing dataset into batches
  5. At the time of model training, batches are passed through network for every epoch and the end of every epoch we calculate validation accuracy.
  6. Once the model is completely trained, it is saved to be used later on test dataset
- **Final Accuracy**
  1. **Training set – 0.999**
  2. **Validation set – 0.968**
  3. **Test set – 0.953**

Test a Model on New Images

- Five German signs taken off the internet to test new images
  source: https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

Figure 5: traffic signs from internet

Figure 6: test images after preprocessing

- These images serve as best test images for checking the accuracy of trained model because
    - Their original size is not 32x32x1, so they have to undergo resizing. Also, sizes largely vary from the biggest images being 143x155x3 and smallest being 29x27x3. So the resizing is done in both ways to make model size 32x32x1. This could test one more aspect of the model
    - I have tried to choose images with carrying background brightness to see if the model is trained for different lightness than original dataset
- The model was successfully able to predict all five images which gives an accuracy of 100%, higher than test set accuracy of 95.3%
- Top five softmax probabilities of the model for each test images were as below. Notice that first probability in each case is very close to 100% leading to an accurate prediction

Probability	Prediction label	Traffic sign name
9.9999988e-01	1	Speed limit (30km/h)
1.7601657e-07	2	Speed limit (50km/h)
4.6868327e-09	5	Speed limit (80km/h)
2.5606860e-11	0	Speed limit (20km/h)
5.4909831e-12	7	Speed limit (100km/h)

Figure 7: Traffic sign - 30kph speed limit

Probability	Prediction label	Traffic sign name
1.0000000e+00	2	Speed limit (50km/h)
4.8395728e-18	3	Speed limit (60km/h)
9.8670845e-21	5	Speed limit (80km/h)
6.9551602e-22	31	Wild animals crossing
1.5974899e-22	39	Keep left

Figure 8: 50kph speed limit

Probability	Prediction label	Traffic sign name
1.0000000e+00	14	Stop
4.1907317e-10	38	Keep right
4.6378033e-11	17	No entry
1.0799183e-11	34	Turn left ahead
4.8160330e-12	13	Yield

Figure 9: STOP

Probability	Prediction label	Traffic sign name
1.0000000e+00	33	Turn right ahead
8.8253189e-13	26	Traffic signals
3.3572160e-13	25	Road work
1.3613806e-13	14	Stop
2.0771596e-14	35	Ahead only

Figure 10: Right turn ahead

Probability	Prediction label	Traffic sign name
1.0000000e+00	13	Yield
2.6456933e-18	12	Priority road
2.4757849e-18	35	Ahead only
5.6914271e-21	39	Keep left
2.9874559e-21	25	Road work

Figure 11: Yield

(Optional) Visualizing the Neural Network

Following figure shows an output of the first convolutional layer of a fully trained model in for first test image of – speed limit 30kph. From the feature maps, it looks like images boundaries and contrast is highlighted more than other aspects of the image, to distinguish the image. This is more apparent in the first two layers.