SDCND - Project 2

Akshay Khole
akshay@reversehack.in

Dataset summary:

The training and validation set are both normalized. The training set is also augmented by adding images that are pre-processed. In the pre-processing steps, I have created a image copy that is an equalised histogram of the original image. This creates an image that is brighter and well contrasted than original image.

This way, the training set has original images of not so good quality and color adjusted images that are more clear. This will help the model learn both kinds of images. The dataset size effectively doubles at this point.

Finally we are also normalizing all the images of training and validation set.

Exploratory Visualization:

I have defined a print_image function I am using at several points in code to inspect images and what effects are different pre-processing techniques having on the images.

The output is printed here again:

./p2-test-images/1.jpg

Prediction:
End of all speed and passing limits

Top 5 Softmax Probabilities are:
TopKV2(values=array(**[0.02672832, 0.02638547, 0.02612905, 0.02590962, 0.02583173]**, dtype=float32), indices=array([32, 20, 26, 24, 31], dtype=int32))

./p2-test-images/2.jpg

Prediction:
Yield

Top 5 Softmax Probabilities are:
TopKV2(values=array(**[0.03067694, 0.0303033 , 0.03014619, 0.02993043, 0.02789136]**, dtype=float32), indices=array([13, 21, 9, 33, 22], dtype=int32))

./p2-test-images/3.jpg

Prediction:
Speed limit (50km/h)

Top 5 Softmax Probabilities are:
TopKV2(values=array(**[0.03793728, 0.03666491, 0.03582141, 0.03535096, 0.03477382]**, dtype=float32), indices=array([2, 34, 11, 5, 26], dtype=int32))

./p2-test-images/4.jpg

Prediction:
Go straight or left

Top 5 Softmax Probabilities are:
TopKV2(values=array(**[0.03025315, 0.02702978, 0.02655631, 0.02651535, 0.02607049]**, dtype=float32), indices=array([37, 2, 12, 24, 42], dtype=int32))

./p2-test-images/5.jpg

Prediction:
Vehicles over 3.5 metric tons prohibited

Top 5 Softmax Probabilities are:
TopKV2(values=array(**[0.02696596, 0.02628897, 0.02623263, 0.025883 , 0.02575901]**, dtype=float32), indices=array([16, 22, 3, 37, 23], dtype=int32))

As we can see the model isn't very certain about its prediction. The reason could be that the training set isn't training enough to include images of all variations. Also, it could be possible that the images I downloaded from the Internet are not suiting the how the model is trained against out data set. This results in a performance of 0% on new images. I am still trying to find ways to increase this %. Slack and mentors are helping a lot :) Hopefully I get to a solution soon and don't get too wrapped up with P3!

Preprocessing techniques used and why these techniques were chosen:

When I started visualizing the images from the training set, all the images seemed to be either very unclear or dark. It was hard for me to even recognize the images. While my testing set (images from Internet) were screenshots of very clear images in very ideal conditions.

But since my model was trained on dark and unclear images, it is expected that the model would not be able to recognize clear images. Hence I tried normalization. But even after normalization, on visualization, I found out that the normalized images from training and test data were looking very different because of perhaps the intensity.

My next thought was augmenting the dataset by pre-processing images and training the model on the unclear, dark images and brighter images too. While searching the Internet for ways to do image transformations, I came across Histogram Equalization (https://www.packtpub.com/mapt/book/application-development/9781785283932/2/ch02lvl1sec26/enhancing-the-contrast-in-an-image) Using this preprocessing technique to enhance the image and augment the dataset.

Model Type, Architecture and Layers:

I am using the LeNet architecture. I chose this model because we learnt a lot about that model in the preceding lessons and it fetched me satisfactory results in the first few runs itself. It is also a great starting point for classifying images.

Model Architecture description:

1. Image is fed into the network
2. Convolution is performed
3. Pass it through an Activation layer. We use RELU as an Activation function
4. Next up is a Max Pooling layer. This uses a stride of (2, 2) with “VALID” padding
5. Again pass it through Convolution
6. Pass it through RELU Activation layer
7. Pass it through a Max Pooling Layer. Again a stride of (2, 2) is used with “VALID” padding
8. We flatten the 5x5x16 input to a single dimension 400 size vector
9. We feed this into a Fully connected layer. We get an output of 120 size vector
10. We then pass it through our RELU activation function
11. We pass it to the 2nd fully connected layer to get an output of 84 size vector
12. We pass it through RELU activation
13. Finally we narrow our 84 size vector to 10 size vector which is out output vector via our last fully connected layer.
14. Return the output also referred to as logits (these are probabilities for each class of output predicted by our model)
15. We then feed it to our softmax function and fetch the highest probable index as our predicted traffic sign.

LeNet Architecture:

Hyper Params - Mu - 0, Sigma = 0.1 (Commonly used start values)

Type of Layer (Sequentially listed) (12 Layers)	Input Size	Output Size
Convolution (5x5) Input depth 3, output depth 6	32x32x3	28x28x6
RELU Activation	28x28x6	28x28x6
Max Pooling (2x2) kernel with 2x2 stride	28x28x6	14x14x6
Convolution (5x5) Input depth 6, output depth 16	14x14x6	10x10x16

RELU Activation	10x10x16	10x10x16
Max Pooling (2x2) kernel with 2x2 stride	10x10x16	5x5x16
Flatten	5x5x16 = 400	400
Fully Connected	400	120
RELU Activation	120	120
Fully Connected	120	84
RELU Activation	84	84
Fully Connected Output	84	10

Convolutional Layer:

We use a convolutional layer because it works well for extraction features from input. The convolution transforms input image into another image that has certain features detected (semantic representation), This way we 'squeeze out information'. The initial convolutional layers are good at detecting features like lines, deeper layers are useful for detecting shapes, more complex shapes, higher level features and so on. We use "Valid" padding because we need to squeeze out information and "Valid" padding helps us with shrinking image with 'denser' information.

We use a 5x5 kernel as it is reasonably accurate for training and fast enough and keeps the pipeline lean.

Activation Function:

We use Rectified Linear Unit (RELU) as activation function as it is simplest, has simple derivatives and help us add nonlinearity to the network. This is essential for classification.

Training Pipeline:

Batch size - 128
Epochs - 10
Learning Rate - 0.001

Training steps and explanations:

1. Initialize input images with normalized input set and shuffle input images
2. Split data into sizes indicated by batch size and feed it to the training pipeline “Epochs” times.
3. We run the LeNet architecture over the input images
4. LeNet returns an array of 10 probabilities.
5. We use cross entropy to train the data against labels and calculate cross entropy. Cross entropy help us measure how confident the model is about it’s prediction. We then average the mean of cross entropies.
6. We use the Adam Optimizer that uses the Adam Algorithm to minimize the loss from cross entropy. Adam is more sophisticated than stochastic gradient descent. Simple gradient descent is difficult to scale.
7. We then try to minimize our losses where backpropagation is used for adjusting our weights and coming to a better trained model.

I used the Learning Rate as 0.001 since it is the recommended default learning rate that is a good starting point for training in general. And I suppose 0.001 is also small enough so that the network takes its
own time to train.

Comparison of performance on test and captured images:

After running the test data on the model, I am seeing consistently low accuracy of 0.025 or 2.5% which is very low. Also the, softmax prediction on Captured images is also in the same range of 2.5% to 3% accuracy.

This may be happening because our model has overfit our training data. The solution to this is augmenting the dataset with more images, adding a dropout layer. Certain preprocessing I think we can do is, rotating, slight blur and brightness variations on the training data by augmenting the training data. Adding a dropout layer may help us build a model that is not overfitting. Augmenting the dataset will mean that the model is training on images of all kinds, not just ‘ideal’ images or the input images.