Lane Detection using Convolution Neural Network.

Authors: Dwijay D Shanbhag, Jyothsna Nagaraj Jois, Swamy Honnebagi Mallappa

1. Abstract

Convolutional Neural Network is a form of Artificial Neural Network architecture which is used for tasks like image-driven pattern recognition. The paper discusses the use of a UNET model to detect road lanes.
A dataset which consists of road lanes and their ground truths as their labels were used to train the model and then the model was tested for a set of customized test image, and the performance was measured in terms of Mean Squared error (MSE) loss.

2. Model Architecture and working

2.1 Flowchart:

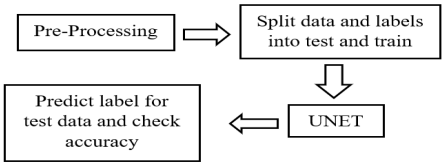

Fig 1: Flowchart

2.1.1 Pre-processing:

The task at hand is to segment the lane, on the path of travel. The dataset provided by UDACITY used for lane segmentation. The data and the label look like the follow:

Fig 2: Image

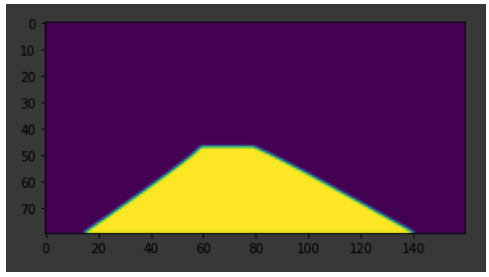

Fig 3: Data label

The images are normalized around zero mean by the following formula:

$$I_N = (I - \text{Min})\frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin}$$

I = Image
Min= Minimum value of Intensity of image
Max= Maximum value of Intensity of image
newMax= Maximum value of Intensity of desired normalized image I_N
newMin = Minimum value of Intensity of desired normalized image I_N
newMax=1, newMin=0

2.2 Model Architecture:

2.2.1 Convolution Layer:

Convolution layer is an important part of a CNN model.
CNN allows for the low-level feature extraction in the hidden layers which combine into high-level features at the higher output levels. This helps in better extraction of features for training a model. The process gets very tedious for large images due to increased parameters, memory occupied, and computation required.
Hence, convolution layers have been used.

Convolution layer preserves the relationship between pixels by learning image feature using small squares of input data. It takes two inputs, image matrix and a filter. Consider an image matrix of dimension (h ∗ w ∗ d), filter of dimension (f_h ∗ f_w ∗ d). The filter moves over the entire image, and each time, the dot product between the filter and slice of the input is computed. The
output of every dot product is a scalar.
The dimensions of the output so obtained is given by

$$((h - f_h + 1)*(w - f_w + 1)*d).$$

In order to have same input and output dimensions, 0 padding can be done.
The output dimension with padding is given by:

$$(\frac{h + 2p - f_h}{s} + 1) * (\frac{w + 2p - f_w}{s} + 1) * (d)$$

- p is the number of padding
- s is the stride with which the filter is slid over the image

2.2.2 Transposed Neural network:

Much similar to pooling, used to reduce the size of image, we use *upsampling* to increase the size of image. In this particular model, we use something called "Deconvolution Layer". *Deconvolution layer* basically works as a transposed convolution layer.
Pooling is very different from up-sampling, which increases size but cannot be learned, whereas transposed convolutional layer introduces a filter which on multiplication with the input increases the size and can also be learned. This model implements transposed convolutional layer to increase the size. Each transposed convolutional layer doubles the image size while reducing the features to half.

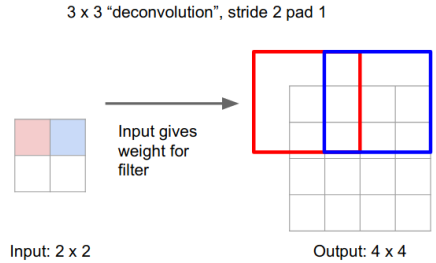

Fig 4: deconvolution

The above figure demonstrates the working of deconvolution. An input image of size 2*2 is updated with specific weights to obtain a 4*4 matrix. The 1*1 red cube is hence upscaled to a size of 2*2.

2.2.3 Network Architecture

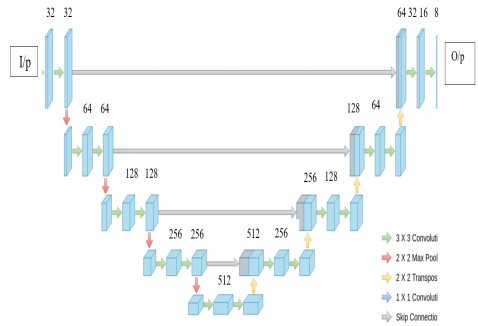

Fig 5: Network Architecture

The network architecture is symmetric, having an Encoder that extracts spatial features from the image, and a Decoder that constructs the segmentation map from the encoded features. The Encoder follows the typical formation of a convolutional network. It involves a sequence of two 3 ×3 convolution operations, which is followed by a max pooling operation with a pooling size of 2 × 2 and stride of 2. This sequence is

repeated four times, and after each downsampling the number of filters in the convolutional layers are doubled. Finally, a progression of two 3 × 3 convolution operations connects the Encoder to the Decoder. On the contrary, the Decoder first up-samples the feature map using a 2×2 transposed convolution operation, reducing the feature channels by half. Then, again a sequence of two 3 × 3 convolution operations is performed. Similar to the Encoder, this succession of up-sampling and two convolution operations is repeated four times, halving the number of filters in each stage. Finally, a 1×1 convolution operation is performed to generate the final segmentation map. All convolutional layers in this architecture except for the final one uses the ReLU (Rectified Linear Unit) activation function.

2.2.4 Loss Function:

The loss function used = Mean squared error (MSE) defined as the following

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

N= No of datapoints

Y_i = value returned by the model

$\hat{Y}_i$ = actual value

It was observed that MSE cannot differentiate between the classes very well. The model trained with MSE as loss function was able to localize the lane, but it failed to work accurately along the borders of the lane. In order to improve the performance, the loss function must be updated.

A better alternative to the MSE would be the dice loss/coefficient defined by,

$$DSC = \frac{2TP}{2TP + FP + FN}$$

T P (True positive) are the number of pixels which correctly match the annotated ground truth

F P (False positive) are the number of pixels which are erroneously segmented by the DNN

F N (False negative) are the number of pixels which should have been segmented but were not segmented.

2.2.5 Initialization and Activation Function

In initial models the filter weights were initialized with a standard deviation of 0.05 which worked pretty fine.

The papers defined ways to initialize the weights to make the model reach its global minima faster. This initialization just defines weight with a certain standard division and around a certain mean, both of which depend upon number of input elements and the activation function. After implementing initialization, the training was faster and also the global minima was achieved in less number of epochs. less number of epochs means less computation power and time during training.

RELU function is applied after each convolution layer. It is given by:

R(z) = max(0, z)

It is used in almost all deep learning models as it avoids and rectifies the problem of vanishing gradient.

3. Results and Conclusions

While visualizing the training and validation error, it was seen that the MSE error tends to reduce as the model gets trained. We trained the model for 10 epochs and are sure that increasing the number of epochs would increase the optimization. We also used the inbuilt cross-validation function by using train_test_split and shuffle functions from tensorflow to improve the model's performance like we've seen in the take home exams.

The following is the graph that depicts how the MSE loss reduces with the increase in the number of epochs.

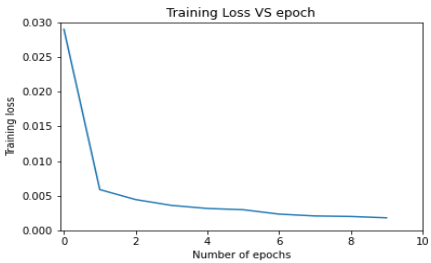

Fig 6: Plot of training loss

Validation loss is the measure of loss when the test data is fit into the model. The validation loss also decreases which depicts that our model is testing data with reasonable accuracy.

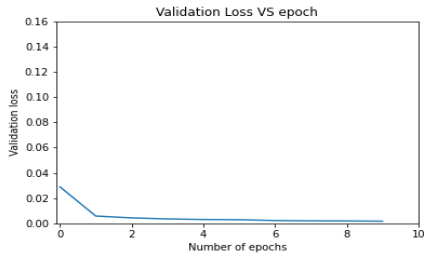

Fig 7: Plot depicting the reduce in validation loss

The model was tested for an image it had never encountered before:

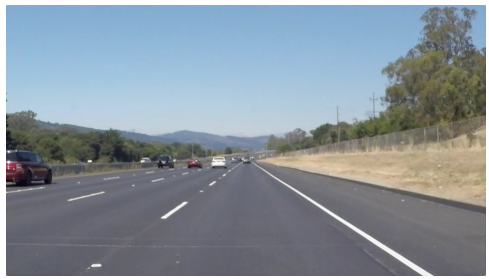

Fig 8: Custom Test image

The model predicted the following lane:

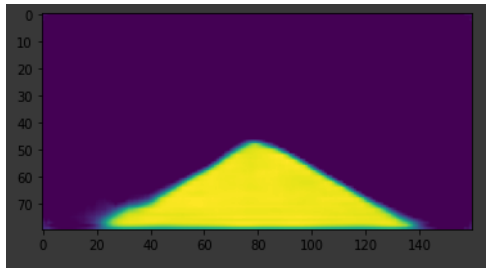

Fig 9: Predicted output

It is clearly seen that the model can depict the lane the car is on pretty accurately.

4. Future Work:

We noticed that the images in the dataset that was used for this project entirely consisted of images of lanes that are very clearly differentiable.

Hence, when we tested on an image with an odd depiction, the following problems were encountered:

Fig 10: Image very different from the training set used

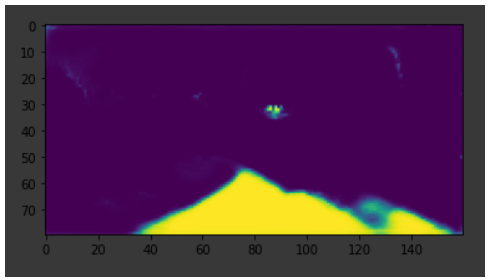

Fig 11: Predicted output from the model

It can be seen that, although the model predicts the lane accurately, it loses track of it as and when the solid white dotted lines reduce and considers the big solid arrow as the lane boundary. This would be a big problem in real world applications.
Hence, we plan on increasing the validity of the model by improvising our own dataset with very adverse images in all conditions.
We also plan on extending the model to predict all available lanes on the road and not just the one we're on.

5. References

1. Olaf Ronneberger, Philipp Fischer, Thomas Brox : U-Net: Convolutional Networks for Biomedical Image Segmentation.(2015), arXiv:1505.04597

2. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semanticsegmentation (2014), arXiv:1411.4038

3. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding(2014), arXiv:1408.5093

4. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fastfeature embedding. InProceedings of the ACM International Conference onMultimedia, pages 675–67

5. Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive deconvolutionalnetworks for mid and high level feature learning. InComputer Vision (ICCV),2011 IEEE International Conference on, pages 2018–2025. IEEE.

6. Carole H. Sudre1,2, Wenqi Li1, Tom Vercauteren1, Sebastien Ourselin1,2, M. Jorge Cardoso : Generalised Dice overlap as a deep learning lossfunction for highly unbalanced segmentations (2017), arXiv: 1707.03237

Dataset link:

https://drive.google.com/drive/folders/1nxKTEmP2ltz58aGurO5-lQ5tCVQY6san?usp=sharing