Information Systems and Machine Learning Lab.

Stiftung Universität Hildesheim

Marienburger Platz 22, 31141 Hildesheim

Prof. Dr. Dr. Lars Schmidt-Thieme

Mr. Mohsan Jameel

Supervisor

AUTONOMOUS MACHINES

Student Research Project – 2017/18

Abdul Rehman Liaqat (271336), liaqat@uni-hildesheim.de

Amir Javed (271307), javeda@uni-hildesheim.de

Manish Mishra (271340), mishra@uni-hildesheim.de

Raghavendran Tata (271441), tatara@uni-hildesheim.de

Zafar Mahmood (271325) mahmood@uni-hildesheim.de

Abstract

In modern times there has been great advancement in the field of deep learning because of increased computational performance of the systems. This has led to huge number of research in making fully autonomous machines where human expertise was irreplaceable. One such area is self-driving cars. Traditionally object detection based robotic control approaches have been used for making driving decisions. This approach computes a high dimensional world representation, which might include redundant information for relatively low dimensional tasks of speed and steering angle in self driving cars. This leads to unnecessary complex systems.

Our current focus is End-to-End learning for steering angle prediction in self driving cars. Unlike object detection approach, end-to-end learning directly maps the input images to a driving action of steering angle and speed. In this approach the system automatically learns the internal representation of the necessary features required for making driving decision without explicit feature engineering. This type of systems optimizes the internal processes automatically and hence leads to a simpler, powerful and efficient system. More specifically we explore three different types of end-to-end learning approaches for steering angle prediction – CNN based single task architecture, CNN based multitask architecture and CNN+LSTM based architecture. In Multitask learning approach we learn related tasks of steering angle and speed simultaneously, belonging to the same domain. The idea is that the related tasks will help each other in getting better internal representation of the features. In CNN+LSTM based approach, we try to learn the internal feature using CNN but we try to exploit the time relatedness of these feature by using recurrent layers (LSTM). We have done transfer learning as well as model tuning on already trained models to get the better convergence for our models.

Along with this we also worked on other supporting modules like Object Detection, Object Distance Estimation, Lane Detection and Traffic Sign Detection that are trained on different publicly available datasets.

Table of Contents

Abbreviations:

RNN	*Recurrent Neural Network*
CNN	*Convolutional Neural Network*
LSTM	*Long Short-Term Memory*
MTL	*Multitask Learning*
σ	*Standard Deviation*
μ	*Mean*
I	*Image*
K	*Kernel*
λ	*Multitask Loss Coefficient*
h	*Hidden Layer Activation*

1. Introduction

Human lives have been revolving around transportation and machines since the very beginning. With the passage of time, more and more tasks are transferred from a human's control to machine's control. Lately, only those tasks are performed by humans which are either very difficult to be controlled by a machine or very expensive if done so. We are trying to focus on these difficult tasks. Such machine, also called **Autonomous Machine,** must be able to perform all tasks without any human intervention. The basic ingredient of this autonomy is moving from one point to another point. In the process, all the obstacles and possible circumstances are handled by machine on its own. Now if we think about it, there are two following ways to achieve this task:

1. Provide a decoded sensory input, instructions of almost all possible circumstances against that input and corrective action as a result. This method is known as Robotics approach.

2. Let the machine learn and experience on its own. The more data is provided the better it will become. This approach is called End-to-End Approach

We strived to build crucial part of both options with more focus on the later one.

1.1 Objective

Autonomous Machine has mobility as basic component which is why building a self-driving car control was major objective. Continuing from the two ways described above, each was further divided into sub-problems and solved individually. Thus:

1. Following the robotics approach, sub problems which we worked at are:

   a. Lane Detection

   b. Object Detection

   c. Distance and Collision Detection

   d. Steering Angle Prediction

2. End-to-End learning approach was used to predict steering angle but with different variants. Main variants are:

   a. CNN based architecture (NVIDIA [1])

   b. CNN based multitask architecture

   c. CNN and LSTM combined architecture

Here, one thing to be noticed is that End-to-End learning is a comprehensive approach in which raw data (images) is fed at the input, traversed through the network for prediction and then prediction is compared with the original reading. Error in the prediction is back-propagated to correctify the network. During this process no feature extraction is performed on the input. Thus, in End-to-End learning the network automatically learns which features areas or pixels of image are to be focused thus resulting in more compact and integrated system. Such system can also perform well then robotics approach. This is why, we exploited End-to-End learning mostly.

Figure 1 displays general workflow for training of the End-to-End learning architecture. Raw images or frames of video are fed from cameras with some random noise. These are passed through the architecture and a steering-angle is predicted using the current parameter values of network. This value is compared with original value of steering angle and at the end error in form of corrective action is back-propagated. This back-propagation will change the parameter values of the network such that by inputting same frame, network will predict almost same value as original steering angle value (or value decided by a human driver).

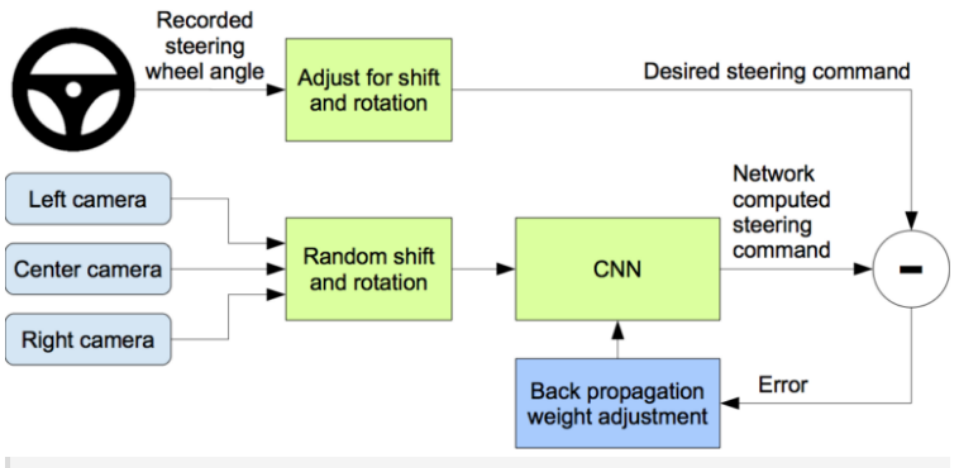

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Error
Back propagation weight adjustment

Figure 1: Training Workflow of the End-to-End learning architecture

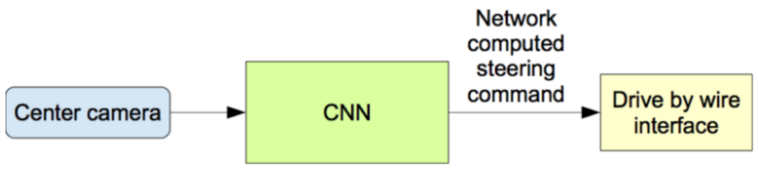


Figure 2: displays the prediction or End-to-End learning architecture during production. Frames from video cameras will be fed to the previously trained architecture and it will predict steering angle which will be used immediately

1.2 Motivation

End-to-End learning can also be described as how humans learn something. Initially it is mostly exploration but with experience (or with more generation of data) learning is achieved. During this process, senses decode necessary input and brain learns the patterns which are crucial to remember. This was also one of the reason our main research was focused on End-to-End learning.

1.2.1 Why End-to-End Learning for Self-Driving Cars?

Main actions of a human driver can be broken down as:

1. Watching the road through windscreen, estimate or predict a steering angle and steer the steering wheel accordingly

2. At times, more than one variable is actuated at the same time for example throttle, de-accelerating or other such action

3. Lastly, change in the view with time is also considered beforehand and actuated variables (steering-angle, throttle etc.) are modified accordingly

The very first action is the basis of End-to-End learning research and also what the based model follows. Steering-angle is predicted through frames of a video. One image is fed and steering-angle is predicted against it. Obviously, this is not enough but still it gives a reasonable accuracy as described in the experiments section.

1.2.2 Why Multitask End-to-End Learning?

As described in the second action of a human driver, it was hypothesized that by actuating multiple variables at the same time, one variables' value affects others. Which is why this hypothesis was translated into a multitask End-to-End learning problem. Both steering-angle and throttle are optimized at the same time with half of the network having common parameters. It was assumed that each objective will benefit from other's training and there will be inclusive transfer learning between them.

1.2.3 Why Recurrent Neural Network for End-to-End Learning

Finally change in view is also very critical information in the driving process. To cater that information Recurrent neural network with Convolutional Neural Network was used.

Our base case model is based on a non-sequential approach using CNN architecture; it takes one image at a time and predicts the steering angle just based on that image. It doesn't take into account the sequence of input frames presented before that particular image input. However, in self driving car images are captured sequentially with time and hence are related. The non-sequential learning approaches might fail to predict correctly in cases where the current image is not captured clearly because of surrounding light. The non-sequential approaches might also lose on accuracy because they are not utilizing the sequence of information available. In real life we also, while driving, use the sequence of visual input to make the driving decision. Similar concept can also be used in End-to-End learning network by using recurrent layers which has a memory component that keeps track of necessary information in sequential images. The recurrent networks combine the present input and the recent past input to respond to new data.

2. State of the Art

Nvidia Net [1] was state of the art for End-to-End learning approach.

3. End-to-End Learning

In all of our End-to-End architectures following general operations and components were used:

1. Input normalization: Main input is in the form of frames of videos. Each frame consists of 3 channels (Red, Green, Blue converted to Y, U and V channel). Furthermore, each image is composed of pixels. Each pixel value ranges from 0 to 255 which is converted to 0 to 1 range. This operation is called normalization and is performed by subtracting mean pixel value from all pixels and then dividing them by the standard deviation. In other words:

$$z = \frac{x-\mu}{\sigma}$$

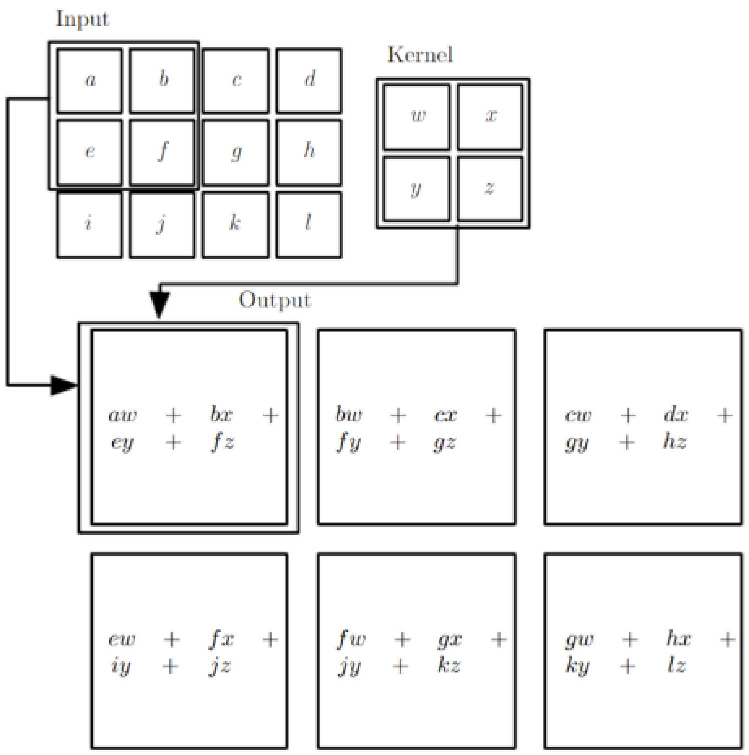
Where μ is mean and σ is standard deviation

2. Convolution Block: Convolution block is one of the main building block of state-of-the art machine learning. This is used to extract important pattern from image. For a two dimensional the convolution operation would be:

$$S(i,j) = (I \times K)(i,j) = \sum_n \sum_m I(i+m, j+n)K(m,n)$$

*Where **I** is an image and **K** is a Kernel or Filter. A kernel or Filter is a matrix which is used to perform convolution on image.*

Graphically, a convolution operation can be shown as following:



Thus, a convolution block consists of many kernels and each kernel perform convolution operation on input image individually. Values of a kernel matrix are learned during the training process.

3. Flattening: Usually, at the end of a convolution operations, the resultant matrices or tenors are flattened in a 1d vector by concatenating each row after another. This operation is called flattening.

4. Fully Connected Layers: A fully connected layer perform high level reasoning by each neuron of fully connected layer having connection to all neurons of previous layer. Each connection is weighted and the weight value is learned during training process. After applying weight values, all are summed and passed through a non-linearity.

Further specialized blocks and components are explained at their usage level.

3.1 End-to-End Learning with NVIDIA CNN Net

This is the first case, where an Image passes through the architecture and predicts a steering angle. The architecture of NVIDIA CNN Net [1] is shown in *figure 3*. The constituent elements of the architecture are:

- Raw input with three channels with size 480x640 (height x width)
- Each channel is normalized
- Three convolution blocks with filter size of 5x5 having 24,36 and 48 number of kernels respectively
- Two convolution blocks with filter size of 3x3 filter having 64 number of kernels each
- Output of final convolution block in previous step is flattened
- Afterwards there are three fully connected layers with 100,50 and 10 neurons each respectively
- Output of last neuron is prediction of steering angle

During this process, *Elu activation* is used. *Elu activation* can be defined as:

The loss function used is:

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

Where y is corresponding steering angle

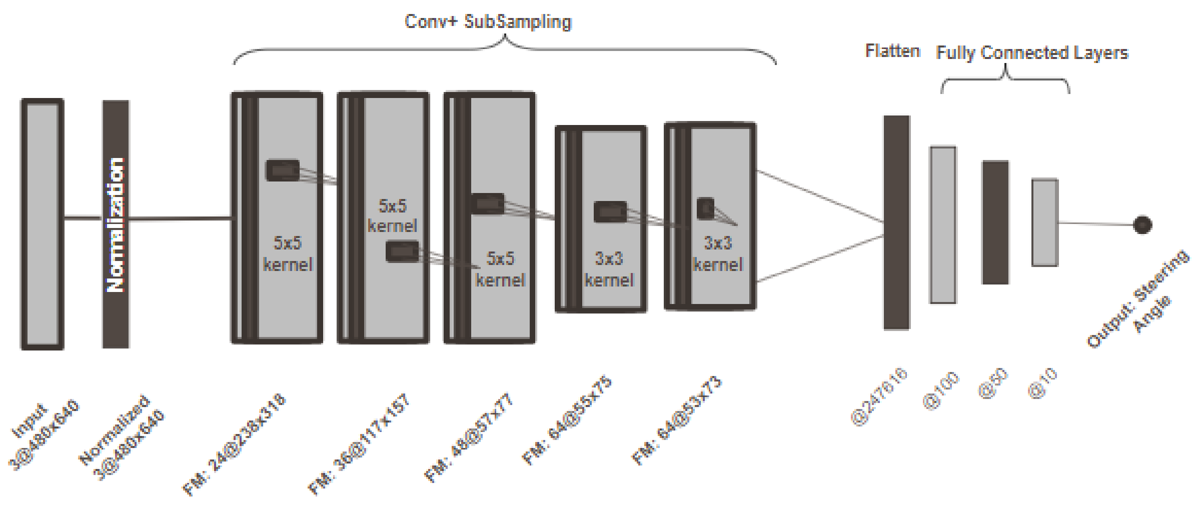
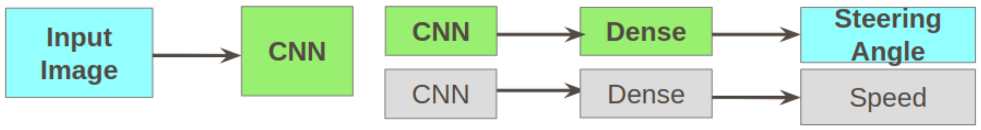


Figure 3: Architecture for End-to-End Learning: Nvidia CNN Net End-to-End Multitask Learning

As shown in *figure 3*, End-to-End multitask learning has similar architecture as NVIDIA CNN net. The only difference is two identical streams optimizing difference objective in parallel. This design helps us in transferring learning from previously trained NVIDIA CNN. Training of End-to-End multitask learning was performed as following:

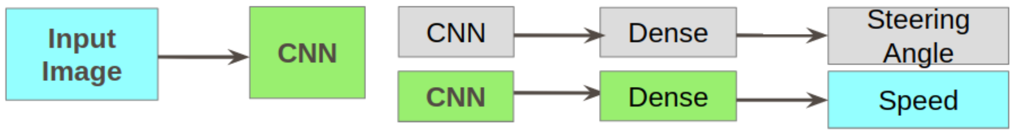
1. Used weights from the previous NVIDIA CNN Net as initialization point of common blocks and steering-angle stream. Here steering-angle stream means the blocks which are not common and used for the prediction of steering angle



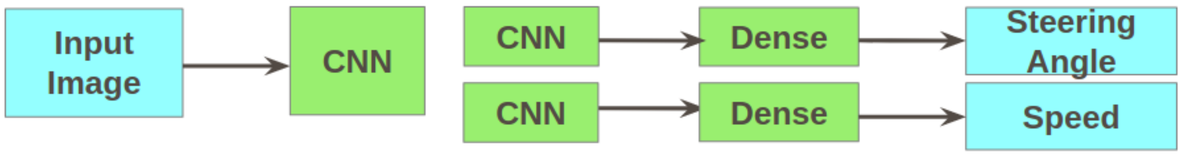
Grayed out network part was frozen.

2. Blocks in the speed-stream are initialized randomly

3. Freeze the steering-angle stream and train for 10 epochs. Note that during this process, weight values of only common layers and speed-stream are being updated



4. Finally, activate all layers and train them simultaneously



Loss function used in this step was:

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(y_i^{T1} - y_i^{T1})^2 + \lambda \times (\frac{1}{n}\sum_{j=1}^{n}(y_j^{T2} - \hat{y}_j^{T2})^2)$$

where T1 is steering-angle and T2 is speed. Value of lambda is 0.5.

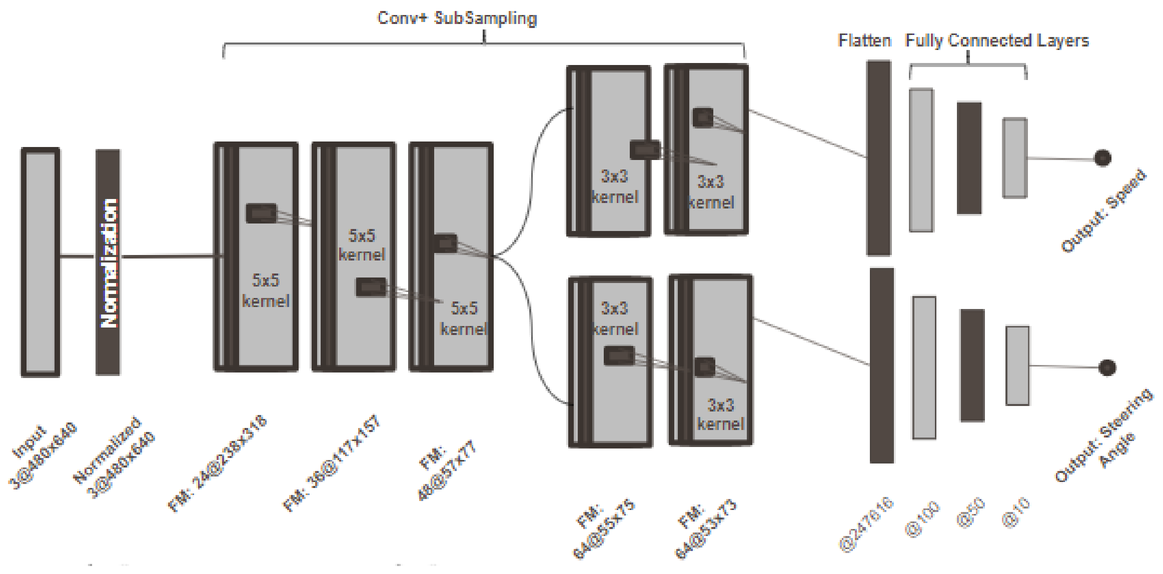


Figure 4: *Architecture for multitask End-to-End learning*

3.2 End-to-End Learning with CNN + LSTM

In this section we will first see what a Recurrent Neural Network (RNN) is and how it works. Then we will see a special type of RNN units called Long Short-Term Memory Units (LSTM) and its advantage over normal RNN unit. Then we will see our CNN + LSTM architecture.

3.2.1 Recurrent Neural Network (RNN)

There are multiple tasks in everyday life which depend on a sequence of information and get completely disrupted when their sequence is disturbed. For instance, in language modeling - the sequence of words defines their meaning, in time series data – where time defines the occurrence of events, the data of a genome sequence- where every sequence has a different meaning. There are multiple such cases wherein the sequence of information determines the event itself. If we are trying to use such data for any reasonable output, we need a network which has access to some prior knowledge about the data to completely understand it. Recurrent neural networks thus come into play.

Recurrent neural networks or RNNs are a family of neural networks for processing sequential data. The decision a recurrent net reached at time step t-1 affects the decision it will reach one moment later at time step t. So recurrent networks have two sources of input, the present and the recent past, which combine to determine how they respond to new data, much as we do in life. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps. Here is what a typical RNN looks like:

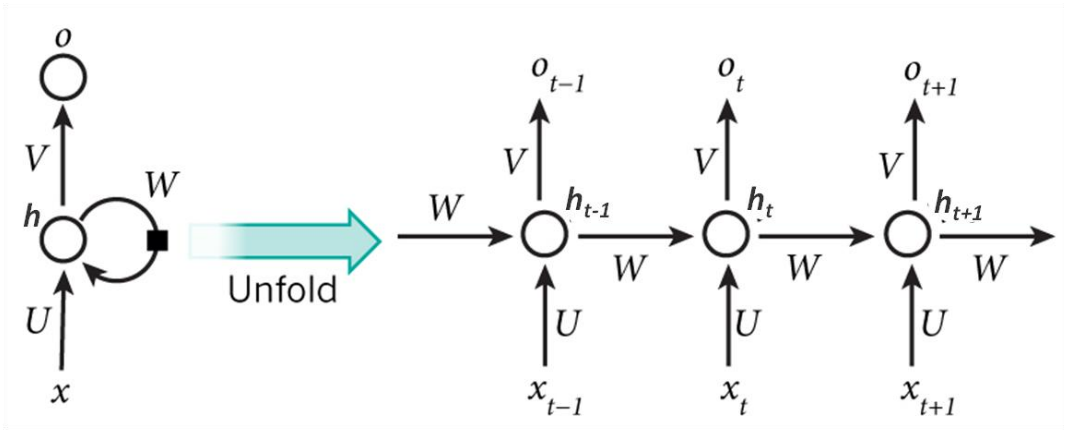


Figure 5: A recurrent neural network and the unfolding in time of the back propagation

The above diagram shows a RNN being unfolded into a full network. This recurrent network just processes the information from the input ***x*** by incorporating it into the state ***h*** that is passed forward through time. In the left circuit diagram, the black square indicates a delay of a single time step. In the right the same network seen as an unfolded computational graph, where each node is now associated with one particular time instance. Here U, V and W are the weights for different steps.

The activation at current step (h_t) depends upon the current input (x_t) and the previous activation (h_{t-1}):

$$h_t = f(h_{t-1}, x_t; U, W)$$

And the output at any step (o_t) depends on the current activation:

$$o_t = f(h_t; V)$$

3.2.2 Long Short-Term Memory (LSTM) Units

The Long Short-Term Memory (LSTM) network is a very special type of RNN, capable of learning long-term dependencies. The vanilla RNN units face vanishing gradients or exploding gradients problem. Because of the long sequence of inputs, the gradients might vanish or explode down the sequence, this makes training the RNN network difficult. This is where the gated RNN units like LSTM are useful. The clever idea of introducing self-loops to produce paths where the gradient can flow for long durations is a core contribution of the LSTM units.

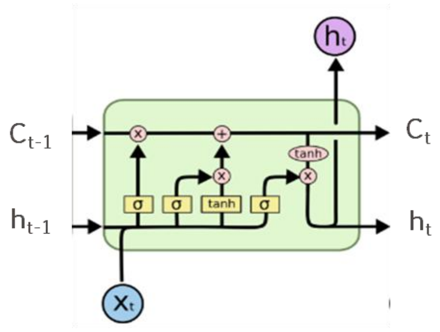


Figure 6: The representation of inside of a LSTM unit

In the picture above, h_{t-1}, h_t and C_{t-1}, C_t represent the Short and the Long Terms respectively. Each LSTM cell corresponds to a timestep and receives both terms from the previous timestep/cell in addition to the corresponding input x_t. Inside each cell there are four "*gates*", whose function is to determine which part of the Long and Short Memories to delete, which to store and which to use on the output. As in the RNN case, cells are timewise connected, and backprop on LSTM is similar to the RNN one. The governing equations for different gates inside a LSTM unit are:

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C)$$

$$C_t = f_t + C_{t-1} + i_t \times \tilde{C}_t$$

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

3.2.3 CNN + LSTM Architecture

Input with spatial structure, like images, cannot be modeled easily with the standard Vanilla LSTM. In our case, we have images as the input data, so we need to first extract features from the images and then pass on those features as a sequence to LSTM layer for steering angle prediction. This is achieved by the "CNN LSTM" architecture. Here the CNN layers are to extract features from the images, followed by LSTM layer for sequence prediction. The different layers of the "CNN LSTM" architecture are shown below.

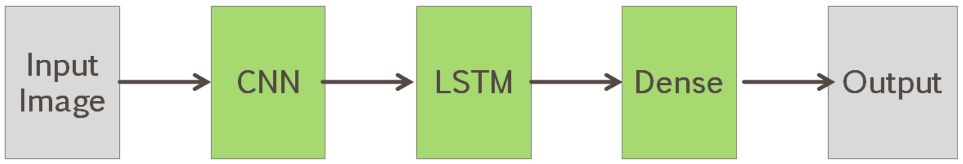


Figure 7: "CNN LSTM" model flow

In our architecture we keep the CNN layers same as that in NVIDIA CNN layers, so that we can compare the performance of the two. We also use transfer learning to train our model. We took the pre-trained weights of the NVIDIA CNN model, uploaded those weights in the CNN layers of our architecture, we froze those layers and trained the rest of the layers (LSTM and dense layers).

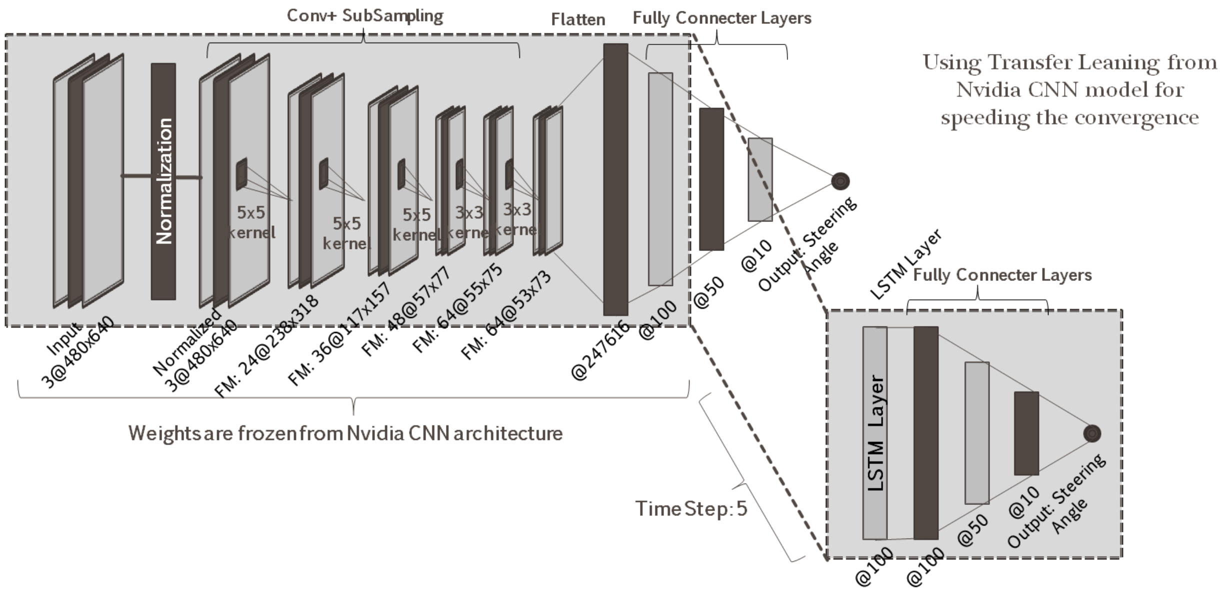


Figure 8: NVIDIA CNN + LSTM Architecture with Transfer Learning

The complete architecture is shown above. The top left block shows the NVIDIA CNN architecture. We have taken the layers until the first fully connected layers containing 100 hidden units. The flattened layer has 247,616 hidden units; hence we could directly feed this to LSTM layer because of the computational challenge. Hence, we took the first fully connected layer as well so as to get a concise feature representation of our image input. Weights of these layers are same as trained weight in NVDIA CNN model, and we keep the weights frozen during the training of our model. These layers are then fed to a LSTM layer containing 100 LSTM units; followed fully connected layers contained 100, 50 and 10 hidden units. At the end we have output as the steering angle. Only the weights of the last 4 layers are trained during training, weights of rest of the layers are kept same. We have used a time step of 5 that means the LSTM layer gets extracted features of the last 5 images to predict the steering current angle. We also experimented with higher timesteps but because of the computational resource constrains, we kept it very simple at 5 time steps.

The loss function used is:

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

Where y is corresponding steering angle

4. Experiments and Results

4.1 Datasets

End-to-End Learning:

There were two different type of datasets used.

1. Udacity dataset [7]: This is a real life driving dataset with videos frames from three different cameras and corresponding variable values of timestamp, steering angle, torque, speed, latitude, longitude, and altitude. Each image frame of size 640x480. Training data contains 33.8k images and test set contains 5650

2. Unity3d Simulator [8]: An open source simulator with car driving in simulated environment was used to generate dataset. Dataset of one hour of driving on two different tracks each were collected. Later these datasets were used train the network. The trained network ultimately, drove car in the simulator autonomously

Other modules:

Module Name	***Dataset***
Traffic Sign Classification [9]	We have used German Traffic Sign Dataset (GTSRB). This dataset is used as a benchmark for achieving traffic sign classification task. This dataset is composed of 39209 training images, 12630 testing images
MS COCO 2014[10]	MS-COCO large scale designed dataset for multiple purpose for academic and research areas. This dataset is composed of 80 object categories, contain approx. 330,000 images for training and testing.

4.2 Experimental Setup

Experiments were performed either locally or in a remote server with 4 XeonPhi coprocessors with 128 GB RAM, 40 cores and 11.5 TB of diskspace.

Due to the limited computing resources following decisions were made:

- Video frames only from the central camera were used
- Each model was evaluated on 50 epochs with early stopping
- Models were trained using Keras with TensorFlow backend or TensorFlow only

4.3 Model Convergence and Accuracy

4.3.1 Model Convergence

Nvidia CNN Single Task Model (Base Case): As shown in the image below the mean square error (MSE) is converging continuously and we get the minimum error at 43rd epoch.

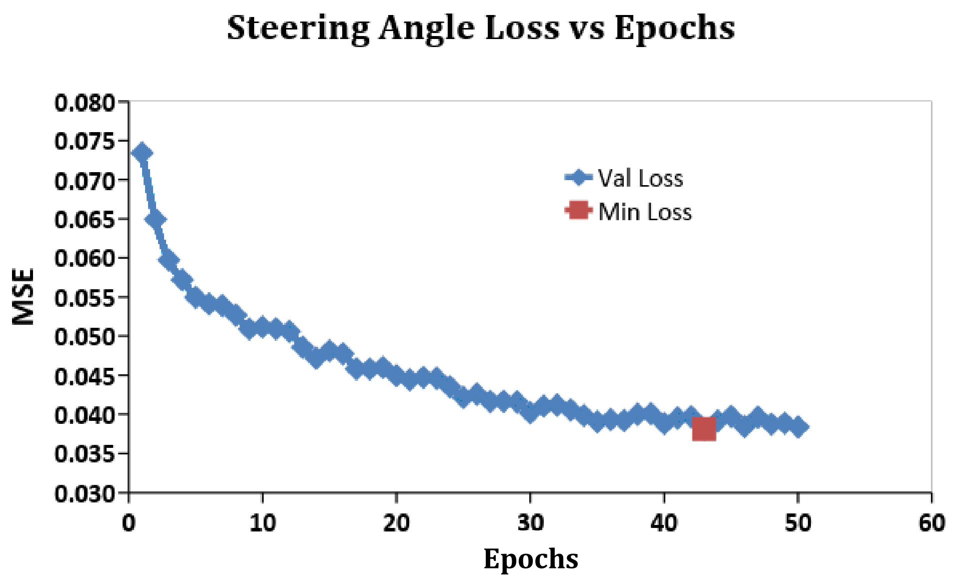


Figure 9: Loss Convergence for Nvidia CNN Single Task Model

Nvidia CNN Multitask Model: The graph below shows the overall loss (Steering angle MSE + Speed MSE) with epochs. After around 20 epochs the validation loss becomes stagnant while the training loss is still decreasing. We achieved the minimum validation loss at 24th epoch.

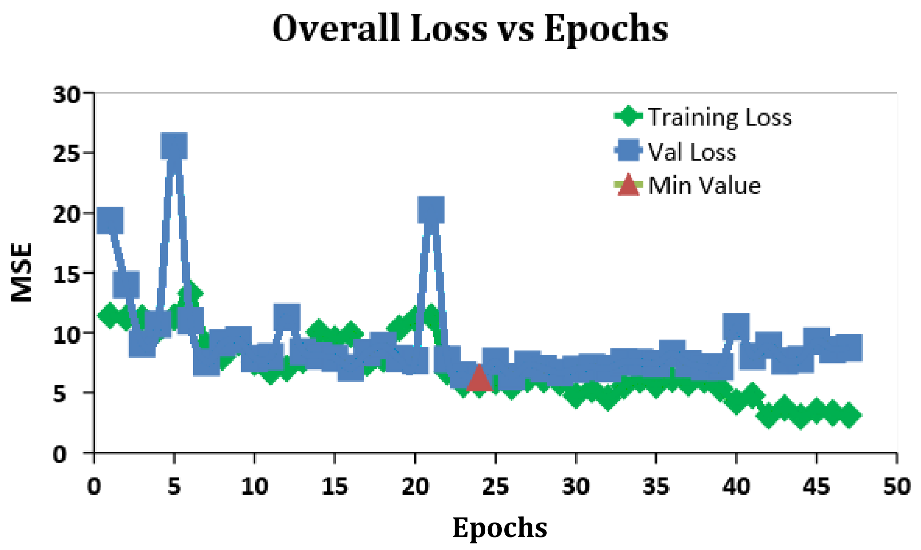


Figure 10: Overall Loss Convergence for Nvidia CNN Multitask Model

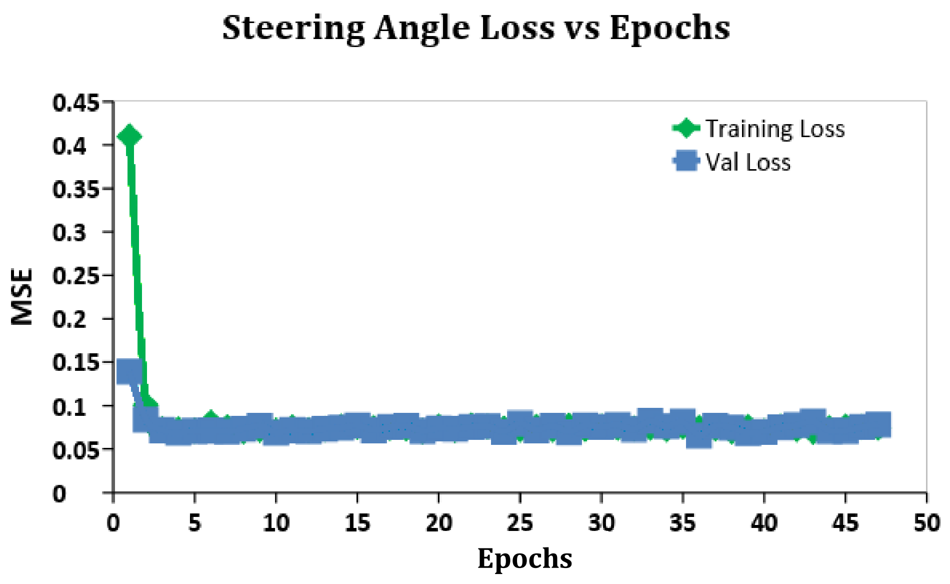


Figure 11: Steering Angle Loss Convergence for Nvidia CNN Multitask Model

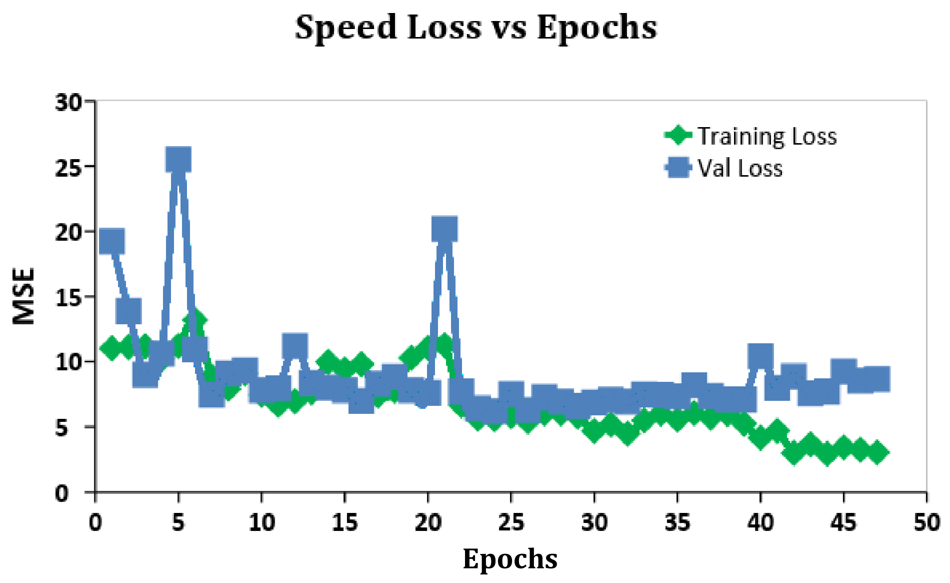


Figure 12: Speed Loss Convergence for Nvidia CNN Multitask Model

In two separate graphs above we have also shown the separate losses for the individual tasks (Steering angle and Speed). From the graph we can observe that although the loss for the steering angle is decreasing continuously and converges very quickly, the loss for speed is little wavy and speed validation loss gets stagnant after sometime even though the training loss in decreasing. This graph shows that the model for the multitask needs some improvement with respect to regularization, more specifically the task of speed needs better regularization.

Nvidia CNN + LSTM Model: The graph below shows the convergence for CNN +LSTM model. We can observe that its converging smoothly and we achieved the minimum validation MSE at 24th epoch.

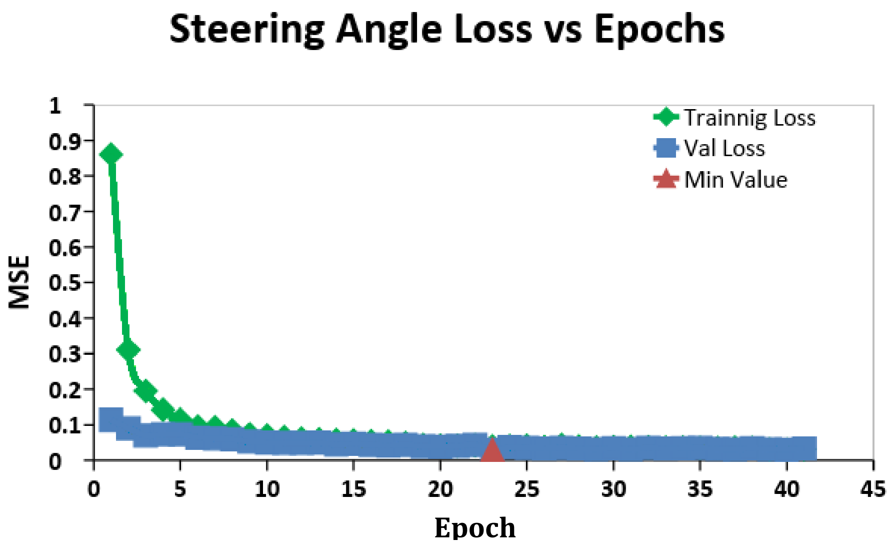


Figure 13: Loss Convergence for Nvidia CNN+ LSTM Model

4.3.2 Accuracy

The table below shows the MSE and RMSE on Test Data for various models. The base case RMSE for NVIDIA CNN single task model is 0.18. The RMSE for NVIDIA CNN + LSTM model comes out to 0.165 (~8% improvement over base case RMSE). This clearly shows that using more than one input images improves the predictability for steering angle prediction which proves our initial hypothesis. However, we didn't achieve improvement in NVIDIA CNN Multitask model. The reason for this could be that this model need better tuning and better regularization as we also observed in the convergence graph for this model.

Model	MSE	RMSE
NVIDIA CNN Single Task (Base case)	0.0324	0.18
NVIDIA CNN Multitask	0.0428	0.207
NVIDIA CNN+LSTM	0.0275	0.165

Table 1: Accuracy comparison for different models

5. Other Modules

5.1 Object Detection using YOLO

For the purpose of object detection there are many algorithms that exist in market considering OHEM and R-CNN have pretty high accuracy rate when it comes to multiple object detection on given images but it does come with the drawback that lack's high-speed process for detection and classification which is one of the core part for our present research project. As always when we are driving car in the street and have to make a split-second decision based on visually perceived information, considering these facts YOLO fits the best because of its high frame processing rate and good accuracy performance. Below in the section it includes the architectural and implementation details with a precise comparison of previous existing solutions. In this project YOLO is reimplemented based on its paper.

Yolo simply works in an end to end single structural network for the detection and recognition of objects in images with high frames per-second. It was inspired for the facilitation of autonomous driving considering high speed vehicles needs fast end-to-end network that just look once at the image and detects all on-sight objects. To come up with this kind of solution author have to come up with new parameterization. So, consider a single image for detection, where the computation starts by imagining an overlying grid, which would be responsible for detecting the number of available objects separated by bounded boxes with their confidence values (that formally represents the probability that box contains an object). Additionally, the lower the confidence value, lower would be the chance that it contains an object and to get above a certain threshold to be recognized. Now after this we would have map of all the objects and boxes ranked by their confidence values. At this point we have identified the objects but actually we don't know what those objects are. Next thing is merging those bounded-boxes with their class probabilities, similar to segmentation mapping. So, take the classical example of a dog and bicycle (*figure 14*) where these objects were joined together in a grid using conditional probabilities. Now if we multiply this conditional probability with confidence values we get all merged or single object bounded boxes. Lot of the them are with very low confidence values for detection in any class category but remaining those who get selected, simply threshold the prediction with non-max suppression (NMS) to get rid of duplicates. This parameterization fixes the output size for detection so now basically it's a tensor that this network is going to predict that contain grid cells and each grid cells is going to predict some class probabilities and bounded boxes. Now in a scenario of pascal VOC the dimension of tensor would be 7x7x30 that generally give approx. 1500 outputs which is pretty low in comparison with bigger data-set's (*figure 15*).

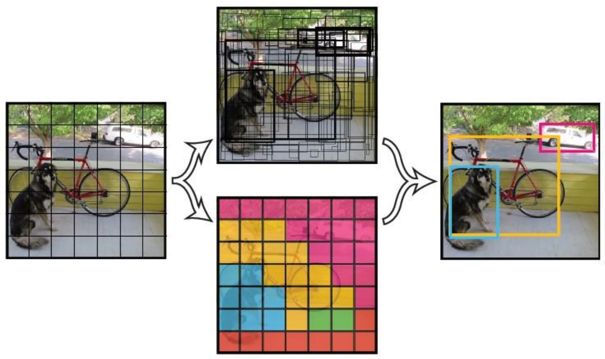


Figure 14: YOLO image classification workflow

Interestingly, according to the author this model also learns the objects that relatively co-occur together with their bound-box size and location. For that, training will be done on full images with the ground truth labels using standard loss function [3], first each matched label is compared with the grid cell vice-versa during testing the center bounded-box in the grid is going to be responsible for predicting the class label, after that same box proposal is expanded by the most overlapped boxes with the comparison of ground truth label. When it is matched author increase the confidence for the detected class and decrease for other boxes that certainly have no object.

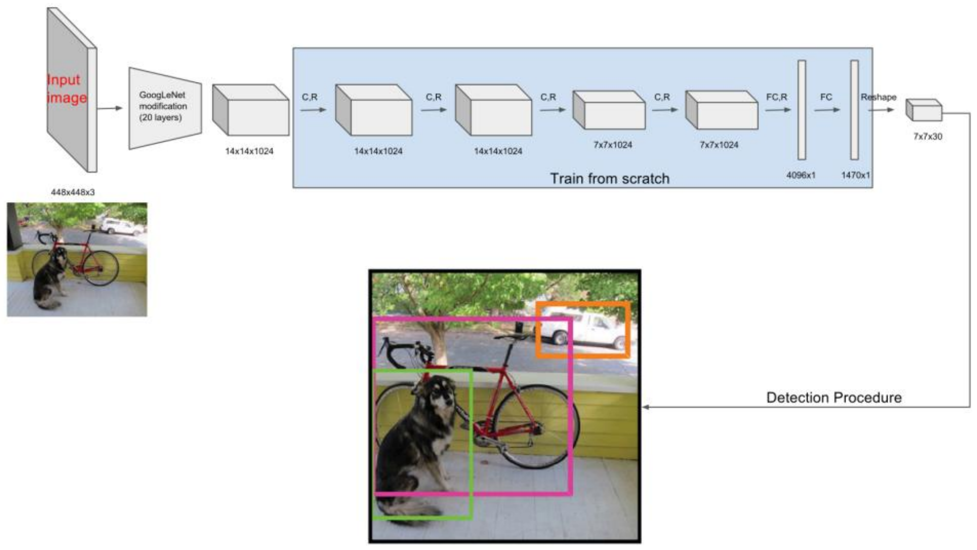


Figure 15: YOLO Prediction Flow

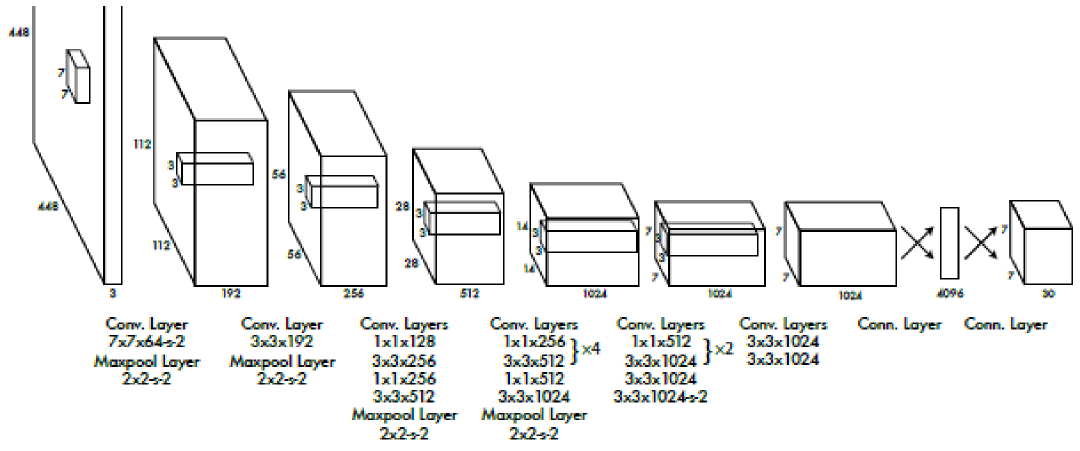

448
448
7
7
3
112
112
3
3
192
56
56
3
3
256
28
28
3
3
512
14
14
3
3
1024
7
7
3
3
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 16: YOLO Architecture

YOLO Standard Performance Measure

As YOLO is one pass through neural network, which evaluate all detections with same speed as classification pipeline. This network is based on 20 convolution layers followed by average polling layers and a fully connected layer (*figure 16*) . Similar to previous training steps, this one is also trained on VOC data-set first in comparison with the fastest version of R-CNN, in-terms of results it produces even less background errors and gave a high increase in performance. It is also compared with other models too which previously gave state of the art performance (mAP), and YOLO shows by outperforming them all.

Now for the understanding, YOLO makes fewer background mistakes which eventually gave boost in performance. For detail comparison with R-CNN [11], author compares each predicted bounded box with YOLO which was implemented using different models, after these results we came to the conclusion yet R-CNN performs the best (71.8 % mAP) but with combined YOLO shows the performance boost with the gain of 3.2% mAP [3].

So, far these are the academic data-sets which make an ease for learning and experimentation based on selective scenarios but in wild these use-cases diverge to much bigger domain. To cover this scenario author has also tested YOLO on Picasso and People-Art data-set (*figure 18*). Furthermore, he also tested with web-cam during the conference presentation (CVPR 2016) by maintaining the real-time detection speed regardless of accuracy.

This network that was used in our use-case was initially trained on MSCOCO dataset. The weights of this pre-trained network are used for the initialization of our network and then trained on MSCOCO 2014 (train, Val) datasets to gain quick convergence below 10 epochs because of early stopping (*figure 17*). Our work here is done using Keras backend with TensorFlow.

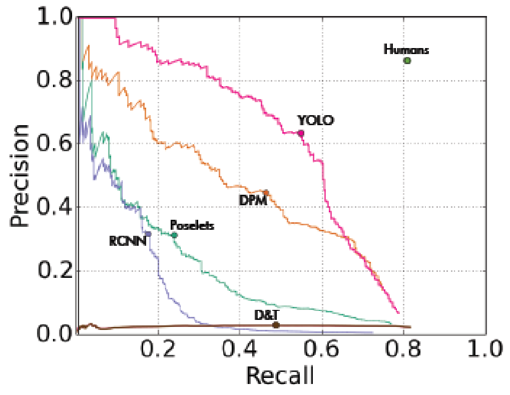


Figure 17: Precision vs Recall

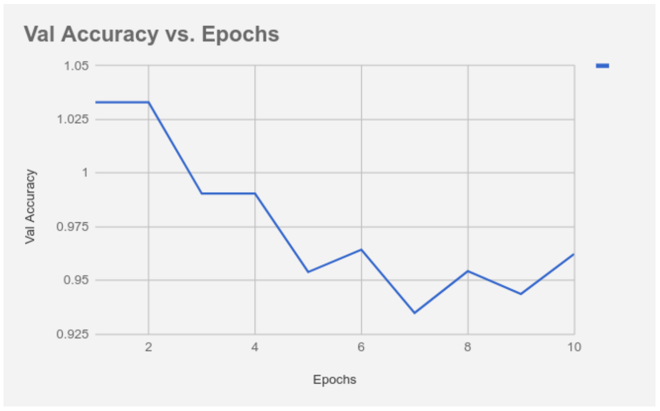


Figure 18: Validation Accuracy

5.2 Lane Detection

In lane Detection, following steps were followed:

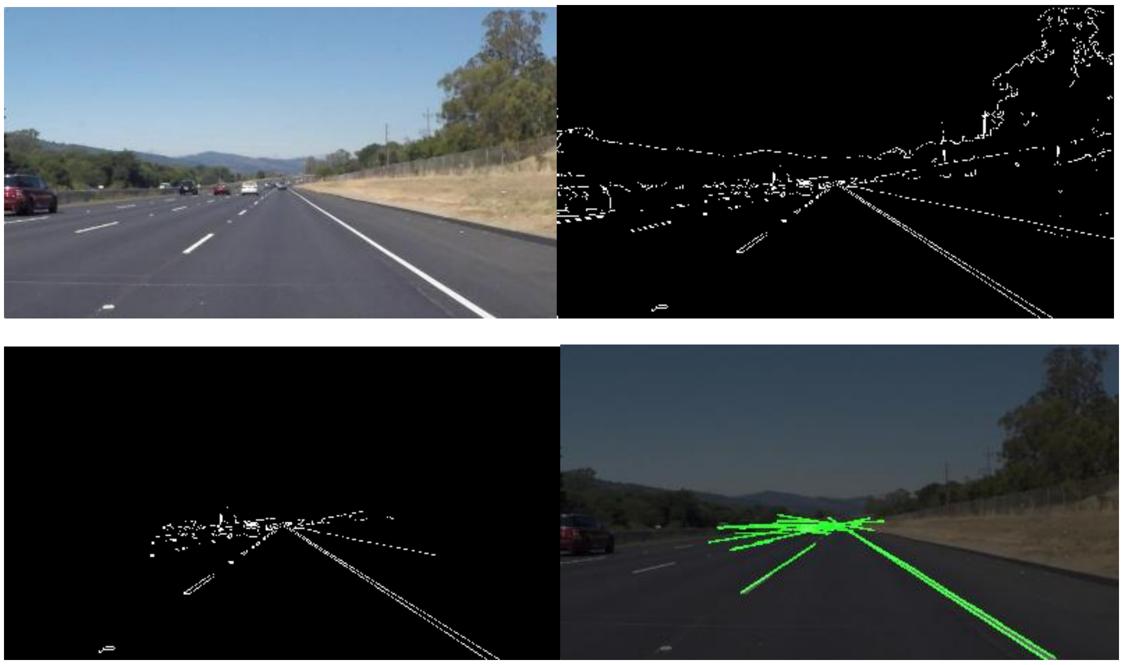


Figure 19: Top left is original image. Top right after edge detection. Bottom left image cropping and color thresholding. Bottom right is the original image with lane detection.

- Edge detection: Edge detection with Sobel filter
- Image cropping and color thresholding: Cropping with trapezoid shape and color thresholding of yellow and white color
- Extrapolation of straight lines: Extrapolation of straight lines with Hough linear transform

5.3 Traffic Sign Classification

In this module we will see how we built a Traffic Sign Classifier. We have used pickled data of German Traffic Sign Dataset. The dataset description is discussed in previous sections for your reference. The traditional ConvNet architecture to the task of traffic sign classification is tweaked a but by feeding 1st stage features in addition to 2nd stage features to the classifier. (Architecture with two subsampling layers)

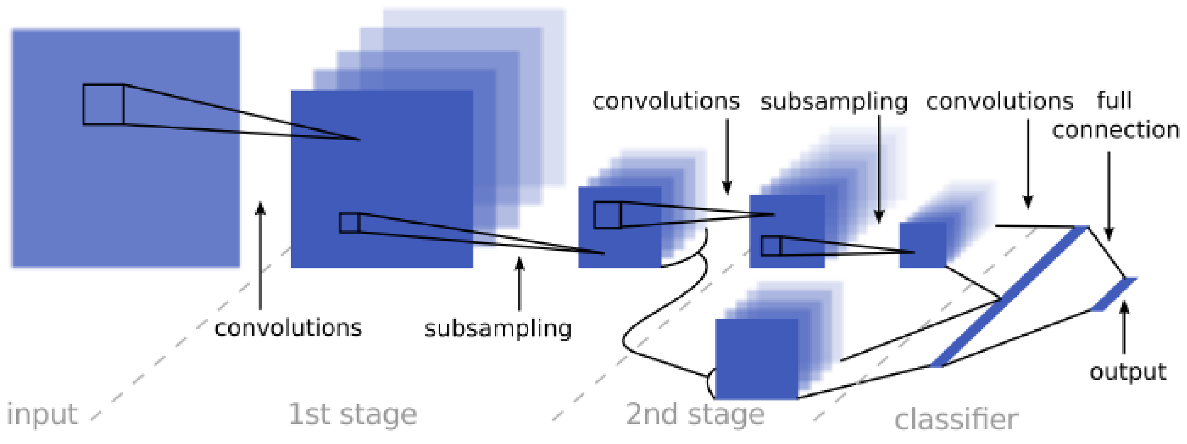


Figure 20: The input is processed in a feedforward manner through two stages of convolutions and subsampling, and finally classified with a linear classifier. The output of the 1st stage is also fed directly to the classifier as higher-resolution features. [5]

Architecture:

As this module is supportive module, we followed the architecture from our base paper [5]. As the figure shows the architecture consists of two subsampling stages before the output layer. The advantage over traditional ConvNet's is that branches 1st-stage outputs are more subsampled and yields to higher accuracy.

- "local" motifs with more precise details are extracted in 1st stages of features
- The "global" and invariant shapes and structures are extracted in 2nd stages of features
- No. of features at each stage: 108-108, 108-200, 38- 64, 50-100, 72-128, 22-38 (the left and right numbers are the number of features at the first and second stages respectively)
- Classifier architecture: single layer (fully connected) classifier or 2-layer (fully connected) classifier with the following number of hidden units: 10, 20, 50, 100, 200, 400
- Different learning rates and regularization values

My inference: After running through series of epochs, model's accuracy in predicting the signs was around 91%.

5.4 Object Distance Estimation and Collision Warning

The objective of this module was to identify objects (especially cars, pedestrians, bikes etc.) in a video feed of a car and then estimate the distances of those objects from our car while driving. Also, if any of the objects are coming within the critical distance from our car, then we want to show a warning message.

For this task we used TensorFlow Object Detection API and OpenCV. Using OpenCV and python PIL libraries we can grab a video stream and convert it to images and corresponding matrix. Our objective is to get a decent number of frames per second from the video stream. Then we use TensorFlow Object Detection API to detect the objects in those images. TensorFlow API predicts different objects with their corresponding boundary boxes. Now in order to estimate the distance of those boundary boxes we use the number of pixels occupied by the width of the object. For our current purpose we only considered the cars and human objects. The width of the object in the image in terms of the pixels can be configured to estimate the approximate distance of the object. Our module was able to estimate the object distance and generate warnings with reasonable accuracy.

6. Conclusion and Future Work

- End-to-End learning results show that the model is able to process images and creates the necessary internal features itself for driving action prediction
- We tried 3 different neural network architectures for end-to-end learning to exploit the relationships between related tasks (Steering Angle and Speed) in CNN multitask model and sequential input data using CNN-LSTM model
- The results show promising improvements from the base case model (Nvidia CNN)
- RMSE for CNN-LSTM model decreases to 0.165 from 0.18 in Nvidia CNN model (~8% improvement in RMSE)
- We didn’t observe improvement in the NVIDIA CNN Multitask model. We observed the reason for this could be suboptimal tuning of the parameters and regularization. For future work this model can be tuned better and better regularized especially for speed prediction task
- Also, for future work, the CNN-LSTM model can also be explored for multitask learning to exploit all the available information and for better generalization
- Other supporting modules can be merged with the results of end to end learning to design intelligent driving control system

References:

[1] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).

[2] Chen, C., Seff, A., Kornhauser, A.L., Xiao, J.: Deep Driving: Learning affordance for direct perception in autonomous driving. In: ICCV (2015).

[3] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[4] Caruana, Rich. "Multitask learning. "Machine Learning 28 (1997): 41-75.

*[5] Pierre Sermanet and Yann LeCun– "Traffic Sign Recognition with Multi-scale CNN" "*http://ieeexplore.ieee.org/document/6033589/*"*

[6] Deep Learning by Goodfellow et. al.

[7] Udacity Dataset. https://github.com/udacity/self-driving-car/tree/master/datasets

[8] Unity3d Simulator. https://github.com/udacity/self-driving-car-sim

[9] http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

[10] http://cocodataset.org/#download

[11] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

Appendix:

Following three other end-to-end architecture were also tested with not so good performance.

Nvidia CNN multitask Net:

Nvidia CNN multitask Net was original Nvidia CNN net with compound objective function to be optimized as following:

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(y_i^{T1} - y_i^{T1})^2 + \lambda \times \left(\frac{1}{n}\sum_{j=1}^{n}(y_j^{T2} - \hat{y}_j^{T2})^2\right)$$

The RMSE value from the architecture of the order of 0.29 for steering angle. One apparent reason of poor performance was network capacity as two objective metrics were to be optimized hence next variation was the modification of the current to increase the capacity of the net.

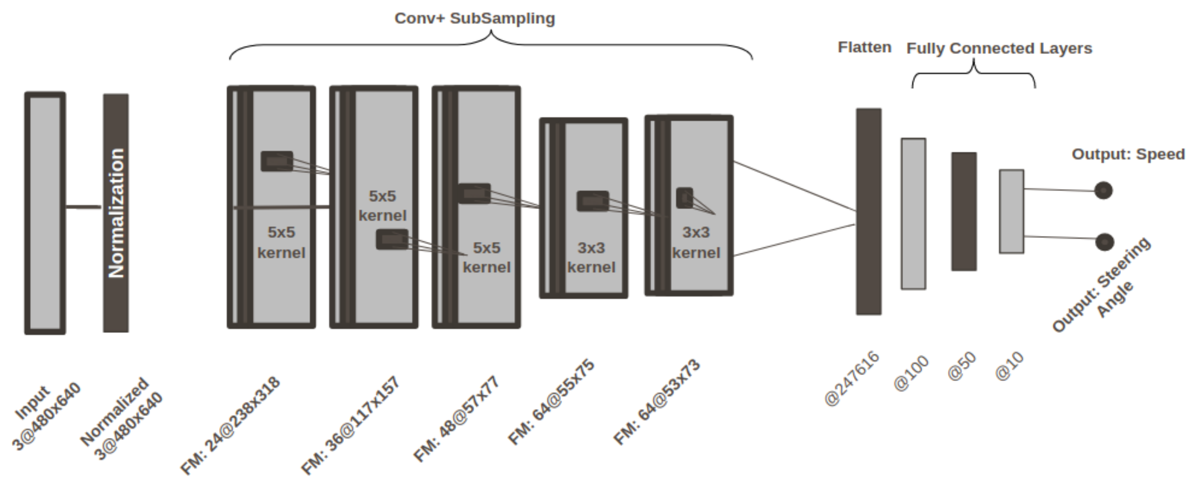


Figure 21: Model Architecture Nvidia CNN Multitask

Nvidia-modified CNN multitask Net:

In the second variation, same objective function was used. Original CNN was modified to cater for the increased complexity required to optimize two objective variables at the same time. Modification was increase in the number of filters in the first three CNN layers and one additional CNN layer with 3x3 filters. Also, two additional fully connected layers were added each having 256 and 128 neurons. RMSE from the net was of the order of 0.258 on steering angle which was better than the previous one but still far from the state-of-the-art.

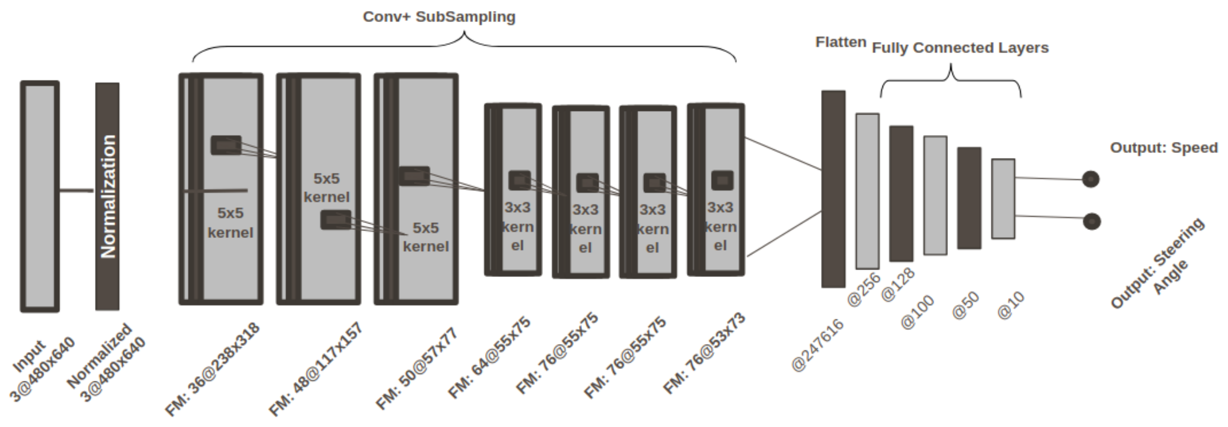


Figure 22: Model Architecture Nvidia CNN Multitask

ResNet50 Transfer learning:

In the final variation, only steering angle was optimized with ResNet50. This was to test the hypothesis that will high capacity net provide better result. Although there were serious computational limitations causing us to use pre-trained weights on ImageNet dataset. Thus, ResNet50 pre-trained weights were frozen with additional fully connected layers at the end. It was still too much for the memory of machine and only 10 epochs were performed with RMSE of 0.267.

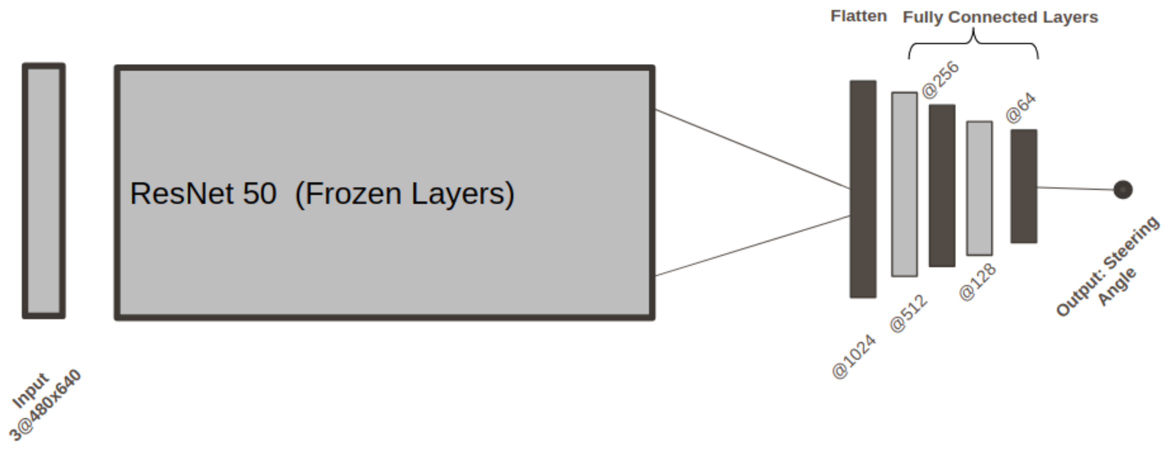


Figure 23: Model Architecture ResNet50

Codes:

All the codes for End-to-End learning and supporting modules are accessible from following public GitHub repository in well documented structure:

https://github.com/Abdul-Rehman-Liaqat/AutonomousMachine