Automotive intelligent systems: Tunnel recognition using deep learning techniques

Radu Galan
Computer Science Department
Babeș-Bolyai University
Cluj-Napoca, Romania
radu.galan1@gmail.com

***Abstract*—There is a continuous demand for intelligent software models that enhances the subsystems of autonomous driving. The recognition of specific elements in the environment during driving is specifically important for the awareness of the automobile or the automobile network, but it is also quite challenging. Tunnels and bridges are amongst the most prominent and commonly found pieces of infrastructure on the road so, it is obvious that we should strongly consider their presence or absence in the surroundings during the decision-making process. There are multiple motivations for obtaining this information: headlights correction, ventilation and temperature setting, sensors calibration, windscreen wipers usage reduction, speed adjustment, alerts regarding maximum allowed height, and even strategizing. In the following paper we will focus on different methods and techniques based on convolutional neural networks that can be used to resolve this problem of classification, and we will conclude which one is best for which situation. We will also present the process of developing a complete environment that allows the user to train or validate models, analyze statistics, and visualize the results.**

Keywords—convolutional neural networks, tunnel recognition, advanced driving-assistance systems, transfer learning, active learning, image processing

I. Introduction

We want to predict approaching infrastructure on the road (tunnels and bridges) from a moving point of view located on a vehicle and using sensors present on the vehicle. Some of the motivations for acquiring this information are related to the anticipated calibration of the sensors located on the vehicle prior to entering a tunnel, adjustment of the on-board subsystems (e.g. air ventilation, headlights, screen wipers), or general strategizing for autonomous driving.

Our solution involves an intelligent agent that receives input from the sensors (on-board camera) as grayscale images and will predict whether there is an incoming tunnel (or bridge). The algorithm will run on the server (it is not designed to work real-time on a vehicle) and will consist of a convolutional neural network already trained on a database formed from similar data. We will create an environment where we can easily construct and test different networks to choose the best one for each situation or dataset.

We will first explain the architecture with which we are working and then go right into the technical part with implementation descriptions. Then we will describe the user interface and present our results. In the end we will begin to interpret the findings, draw conclusions, and compare to the state of the art.

II. Scope strategies

The first and most obvious approach regarding the algorithm behind the intelligent agent would be the CNN (Convolutional Neural Network) with binary classification (containing the labels: tunnel and no tunnel). An alternative that might be able to offer powerful insight could be based on a CNN with multi-classification (containing the labels: entrance in a tunnel, inside a tunnel and exit from the tunnel).

A slightly different approach is constructed around the fact that there are strong similarities between tunnels and bridges (especially wide ones). That is where we encounter a lot of false positives, and most certainly we should acknowledge this and try to include images of bridges into the classification. One solution is to augment the learning dataset with more such sequences, another one would be to introduce a new label specifically for bridges.

More important variations that are considered after having a working network are the usage of active learning or ensemble learning. They would most likely improve the performance of the model and require far little effort and resources.

III. Architecture

A. System architecture

One of the essential components for our project is the implementation, and in order to start explaining that we will first have to describe the environment we are working with. As can be seen in Figure III-1 we are using a distributed architecture with a remote machine (the server) and a local machine (the personal computer). The entire environment will function best on similar systems only.

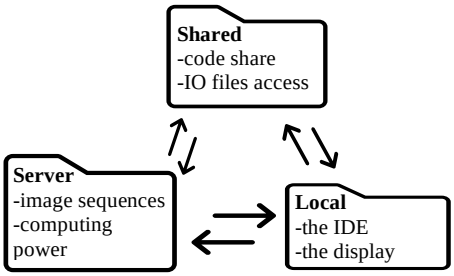

Figure III-1

The server is where we have all the available computing power (used for training the model and for predicting) and all the image sequences (all the data used).

The local environment (the laptop) is where we are going to do all the coding and the visualization (for labelling and also results analysis).

To be able to do all the coding in an IDE (PyCharm) we had to create a shared space between the server and the local system. This space was used to upload the code automatically and run it from the IDE, but also to make all the transfer of files between the two systems easier.

B. *Environment*

First our system is basically split between the local device and the server, so there are two individual setups. Whilst we cannot go into too much detail, the server contains an Anaconda environment (conda 4.5.13) with python 3.6.8, cuda 10.1.105 and cudnn 10.1_v7.6. Locally we are using as IDE the PyCharm 2019.3.5 that can connect to a shared location and directly run scripts on the server using an SSH connection.

IV. Implementation

A. *Data*

One of the most important components that is needed for our algorithm to perform properly is the data. Originally it was stored on the server as sequences of images, and individual images needed extraction from them. We exported every third frame in a grayscale format. The images were copied on the local memory for visualization and for the labelling process.

The main filter used in selecting the sequences was the presence of a tunnel in it and the compatibility with our system. Out of the couple of thousands of sequences we extracted about 500 sequences with tunnels in it, and another 500 sequence containing bridges. That sums up to about 140'000 images from tunnel sequences and over 300'000 images from bridges sequences.

After the images were present locally, we had to manually label each image and of course that due to the time constraints we were not able to label all of them. During several labelling sessions (about 3 different batches) we summed up to 50'000 manually labelled images. The labels used were: Tunnel/Bridge/None/Both/Delete. To improve the labelling process 'None' and 'Both' were added to be able to use all labels for binary, multi-class, and ternary classification alike. Another labelled dataset has been created for the positive classes of multi-class classification with the labels: None/Entry/Inside/Exit.

In order to achieve a correct validation and testing of the model, the images must not be present in two different files, and moreover no sequence of images should be split between two different files. We do this to avoid a common evaluation error that happens when we test the model on data too similar to the data it has also been trained on. Even though the images are not exactly the same, the similarity between one image and another in the same sequence (3 or more frames apart) could be considerably higher than two images in different sequences.

After separating the images into 3 different files we had to consider the unbalance of the classes. We could use weights for each class whilst training the model or we could simply remove some images. Although we only used sequences where tunnels appeared, there were plenty of negative frames (before the tunnel showed and after exiting the tunnel, same for bridges), in fact we ended up with more of the latter (~45% tunnel images and ~55% no tunnel images, ~10% bridge images and ~90% no bridge images). We will remove negative images until we achieve a balanced dataset.

Another important matter to consider is the order in which the images are given to the model, because it is known that if a neural network learns all the positive entries and then all the negative ones it will have bigger variance and will probably overfit. To avoid this, we will shuffle each dataset file individually.

B. *Image preprocessing*

We use filters/effects on images before giving them to the network. We have experimented with multiple filters, but on the architecture that we applied the best results came in when we used no effects. This most probably happened because the models were already trained on very large datasets that consisted of a very wide range of images (preprocessing could not be used so easily). We experimented with random filters (not randomly chosen, but random ratios/values/intensities): brightness, contrast, resizing, rotations, and even Lagrange filter, but most of them ended up not contributing much in optimizing the model, because all tunnels usually look the same, they don't rotate and they come in different sizes by default (as the car approaches the entry).

Figure IV-1

In the end we used cropping to remove some of the imperfections caused by processing and visual defects on the sides of the images caused by turning the image into a flat wide format (as can be seen in Figure IV-1). All the images had to be cropped to the exact same size since the neural network must know beforehand the expected format.

We applied a normalization operation on every image using the maximum pixel value, mean value, and standard deviation to limit the variation in input (especially helpful for very sunny days where glare appears or at night).

C. *Application architecture*

The work has been separated in three big projects, each of them responsible for a specific phase.

First, we have the preprocessing project that contains a series of scripts that were run every time we added new data to the project (this project runs on the local environment only). After having available the complete list of 140'000 images (more exactly server paths) that are tunnel related we will have to go through the following steps:

- Copy the image files on the local memory and convert paths to local address
- Use the interface to manually label the images and generate a complete dataset with the paths of the remote address

- Create 3 different files (train, validate and test data) from independent continuous sequences using 60/30/10 percentage (60% train data, 30% validate data, 10% test data) and shuffle each file

The second project can be used to create the trained models. The database must be transferred to the server because this project runs exclusively remote (all the code has been written in a shared location). We are using transfer learning to import the first layers that detect the most generalized features, and then add a couple of dense fully connected layers that will make sense of the features and classify them correctly. There is one algorithm here that can be run with different parameters (arguments for the compilation of the neural network, for the data generator, and for the training of the network), and it is composed of a couple of main components:

- Data generator (defines the preprocessing operations, how the data is read and processed in batches) – overridden class
- Model construction (initialization of the model, loading the weights, adding the new layers, fitting the data, saving the model)
- Model testing (it predicts the test data and then generates metrics for comparison)

The third and final project is the user interface and it runs on the local machine, but also accesses the server remotely using the SSH protocol. The local project is structured simply by the Module-View-Controller architecture, but has two controllers: one runs the operations that do not need serious computations locally, and the other one implements a communication interface with the server in order to run the CNN. The prediction (or training) algorithm will work on another thread and this will allow the user to access the rest of the components at the same time. It will open an SSH connection with the server and begin sending the commands needed in order to predict the results (train the model):

- *module load conda/4.5.13 cuda/10.1.105 cudnn/10.1_v7.6* – it will load the packages
- *source activate condaEnvironment* – it will open the environment (already created)
- *taskset –cpu-list 12 python predict.py -model_name VGG16.2 –frame_folder data/testData4.csv –result_file /results/resultFile4.csv* – it will start a task on a specified CPU with the command "python..." (the GPU is also specified, but in the python code), command that will run a given script with the given parameters

This last command is rather interesting because as simple as it is, it actually contains the final product of the entire training project. What it will actually do is use the given model (it will load the weights and configure it with the already tested parameters that worked best for us) and dataset (which will be preprocessed as presented in subchapter A. Data) to generate the best predictions and store them in the given location.

A training algorithm has been designed in a very similar manner to the one previously explained. Both those algorithms have been used in creating a new class that works as an interface for the computational operations. With a beautiful abstraction we managed to reduce the interface to a few rudimentary methods: train, predict, validate, splitByThreshold, augmentDataBase. All these methods facilitate very easily the implementation of the Active Learning and Ensemble Learning strategies.

V. User Interface

Although the user interface was not a priority, it was used for two purposes: the labelling process and the visualization process. The designated user for the first one is the developer only, with the sole purpose of preparing the data. The latter could be used by anyone with access to the server because it communicates through SSH during the predicting functionality.

As we can see in the Figure 4.6 there are a lot of functionalities available. Although it began only as a simple way of visualizing the results, it slowly turned into a feauture-rich desktop application that can do everything one might need after training the model. Some of the key features are:

- Select a model by classifier and view details about it (Figure IV-2)
- Select an image database, view details about it (a wide range of information regarding data exploration), or visualize its particularities (graphical distributions for some of the proprieties) - (Figure IV-2)

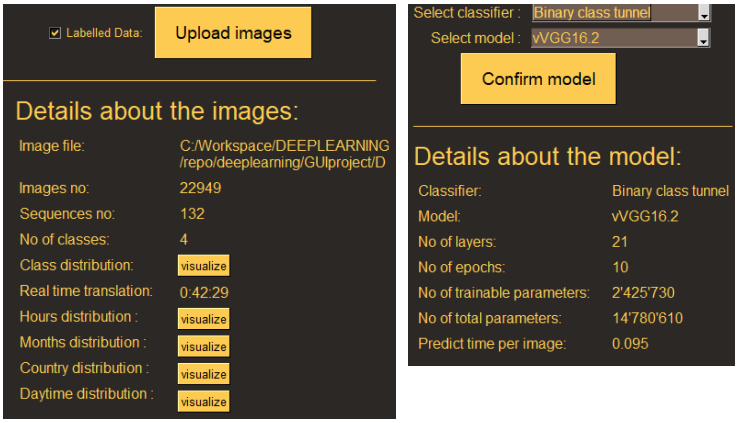

Figure IV-2 – fragment of the interface containing information regarding the selected model, respectively the dataset

- Predict or load results (using a model and a database), analyze metrics (statistics of the results, confusion matrix and other metrics), or visualize them directly (generate an animation with all the given frames and the prediction) – (Figure V-1 and Figure IV-3)

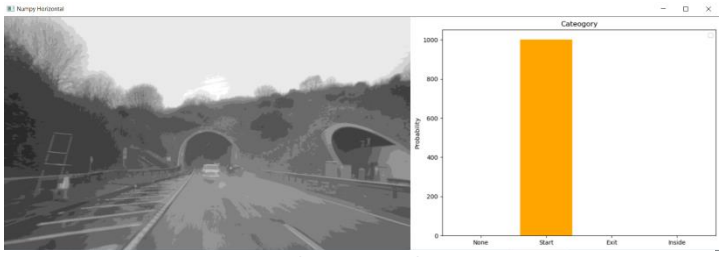

Figure IV-3- animation of the results from the multi-class model

- Apply active learning (with a given model and at least 3 datasets – for first training, unlabeled predictions, and validation) for one epoch or more

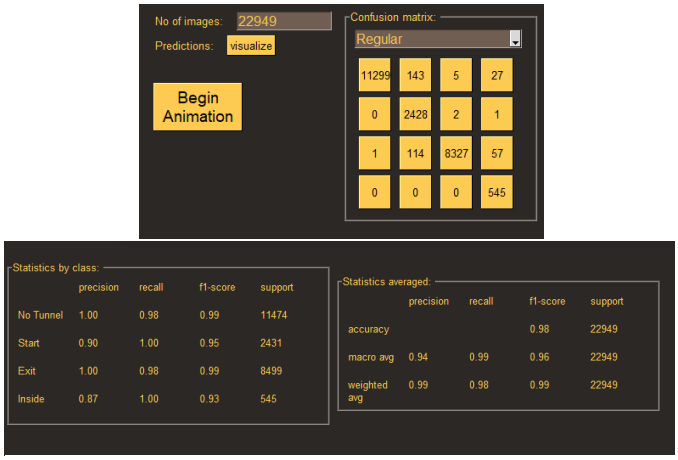

Figure V-1– fragment of the interface containing metrics and statistics on the results

The interface is largely separated into 3 components: the classifier-model, the image database, the prediction. The first component uses an already known list of all the models that are currently trained and upon selection it simply populates all the details fields. The second component receives from the user an input file path, it reads it completely and then it calculates a couple of information and generate the necessary data for all the graphic plots available by category/property. It uses not only the given path, but also a complete list of all the sequences ever extracted that contains essential details. The last component can be populated by two means; either predict new results, this requires a model and a database already selected, or load a previously generated result file. Both methods will load at the end details about the results: confusion matrix, metrics of the evaluation (which will make sense only on labelled data), and it will unlock the visualization methods.

The active learning functionality will be activated by confirming a model stored in the category "Active learning". The user can then select the initial database and press the button "Begin training". The user will need to provide again input: the path to the unlabeled database (used as augmentation), and the path to the validation database (used for comparing models). The server will be contacted through an SSH connection and it will respond by sending results files and statistics that are going to be displayed in real time on the interface.

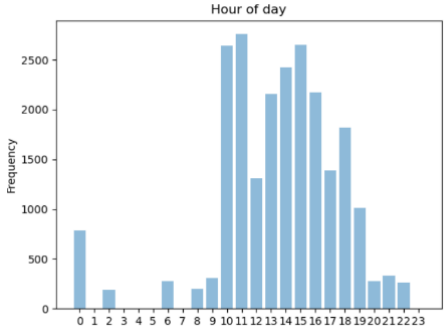

Figure V-2 – example of a graphical distribution plotting

VI. Model selection

Since we intend to analyze multiple methods and strategies and only have a limited amount of resources (computational power and time), we find ourselves forced to choose only one architecture with which to continue.

A. *Models comparison*

To find the architecture that best fits our data we will need to experiment with multiple ones. For a wide and accessible variety, we are going to choose the following public and efficient pretrained architectures: InceptionV3 (although it has 42 layers and over 23 million parameters, it uses useful operations e.g. factorizing convolution, batch normalization, grid size smoothing and regularizations) [1] ,VGG16 (with only 16 layers and over 100 million parameters it uses convolutional layers with ReLu, max pooling and fully connected layer) [2] , ResNet50 (it introduced the concept of skipping connections or residual information, with 50 layers and about 23 million parameters) [3], DenseNet121(contains 121 layers and about 7 million parameters with dense blocks and pooling operations) [4]. All four models are very interesting, and each one has many particularities that could generate interesting results.

After experimenting with every model on different parameters setups (custom chosen depending on the architecture of each one) we have come up with the results from Table 2 and Table 3. The first table contains comparative information about each model and the way in which it has been trained. All the models have been trained using the same algorithm and the same dataset, but different parameters setups. We only displayed the metrics for the best version of each model architecture. The second table contains metrics obtained after the training during the testing phase. For this we used one dataset of almost 1500 images from diverse sequences.

B. *Interpreting the model comparison*

Right after the training phase we could see that ResNet50 did not complete its 10 epochs, but instead stopped early due to lack of improvement. A little overfit appeared for every model due to the reduced size of the image dataset relative to the number of parameters. All of the models surpassed the 90% mark and VGG16 even 99%.

The conclusion is rather obvious, that VGG16 outperformed all other three architectures for our problem and the given dataset, but the same comparison was also done on the multi-classification for tunnels problem with a bigger dataset (over 40k) and the results turned out similar (VGG16 – 91% , ResNet – 69%, DenseNet 81%). We will attempt further improvement on the next strategies and techniques using the VGG16 architecture.

Table 2 – statistics and information for the training phase

MODEL	Data Size (train/valid)	Param. Trainable	Param. Total	Train Time (avg epoch)	Epochs	Train Acc	Valid Acc
InceptionV3	10600/5300	262'530	22'065'314	940 sec	10	93%	85%
ResNet50	10600/5300	820'310	23'408'022	990 sec	5	95%	49%
DenseNet121	10600/5300	133'506	7'168'962	940 sec	10	95%	92%
VGG16	10600/5300	2'425'730	14'780'610	995 sec	10	99.55%	99.13%

Table 3 – details and metrics for the performance on the first dataset

MODEL	Test Size (p/n)	Class Distr. (p/n)	Positive Precision	Negative Precision	Pos. F1-sc	Neg. F1-sc	Weighted Average Precision	False Pos.	False Neg.
InceptionV3	717/771	48/52	93%	75%	77%	84%	84%	37	244
ResNet50	-	-	-	-	-	-	-	-	-
DenseNet121	717/771	48/52	97%	88%	91%	92%	92%	19	105
VGG16	717/771	48/52	100%	99%	100%	100%	100%	1	4

VII. Strategies results

In the following subchapters we are going to shortly explain each strategy, and if an implementation was attempted, we will explain how it performed and under what circumstances, otherwise we will justify why we decided not to implement a model.

A. *Supervised tunnel binary classification*

This model takes in data labelled as follows: 1 if a tunnel is in the image, 0 otherwise. It seeks to identify tunnel entrances, tunnel insides, and tunnel exits under one positive label. This does not simplify the learning process because if the models learn the same features, then theoretically the only differences between this and the multi-classification of tunnels should be in the last fully connected layer.

Although there are clear signs of overfitting on the training data it is rather interesting than on the testing data (particularly selected to be a little harder to recognize) it performed with over 5 percent better than in the validation stage (an average of 94.5) as can be seen in Table 4 and Table 5 (*VGG16.2*).

B. *Supervised tunnel binary classification improvements*

We added 2500 images that are known to contain bridges only to the training and validation data set with the intention of correcting the false positives mentioned before. This is *VGG16.4* (also found in Table 4 and Table 5) and even though we expected a decrease in the number of false positives, the exact opposite happened, and the number of false positive increased. On a closer look (using the visualization methods) we observed something interesting: overall the network did predict slightly better, but relatively light oscillation in the probability caused no statistical improvement.

C. *Supervised tunnel multiclass classification*

This model uses 4 distinct labels: 0 for no tunnel, 1 for tunnel entrance, 2 for inside the tunnel, 3 for tunnel exit. We created the same database of about 40'000 images labelled accordingly and trained *VGG16.1*. Although we added numbers to Table 4 and Table 5, they do not really emphasize the improvement. In Figure 4-10 we have a better view of what really changes, and by visualization (on this test data and also on other non labelled data) we can gain an even better understanding. Although statistically only the false positives number are getting better (smaller) this actually gives us a really powerful inside view. We can conclude even only using this limited testing set and visualization that actually the biggest issue is not with the model, but with the labelling data process. We are going to discuss this thoroughly later. The predictions are rather relative and hard to statistically verify, but by visualizing the data we can obviously quantify the accuracy of the model.

```
Time with next is : 433.0094573497772
Confusion Matrix
[[1684  189   14    9]
 [ 120  276    1    6]
 [   5    1   75    9]
 [   5   53   16 1329]]
Classification Report
              precision    recall  f1-score   support

   No Tunnel       0.93      0.89      0.91      1896
       Start       0.53      0.68      0.60       403
        Exit       0.71      0.83      0.77        90
      Inside       0.98      0.95      0.96      1403

   micro avg       0.89      0.89      0.89      3792
   macro avg       0.79      0.84      0.81      3792
weighted avg       0.90      0.89      0.89      3792
```

Figure VII-1

D. *Untested strategies*

We did not attempt to implement the following possible strategies: supervised bridge binary classification, supervised bridge-multiclass classification, supervised bridge-tunnel multiclass classification, supervised bridge-tunnel ternary classification, because all of them require a significant augmentation of the data base (labelling bridges).

Table 4 - details and metrics for the performance on the first dataset

MODEL	Test Size (p/n)	Class Distr. (p/n)	Positive Precision	Negative Precision	Pos. F1-sc	Neg. F1-sc	Weighted Average Precision	False Pos.	False Neg.
VGG16.2	1898/1894	50/50	92%	98%	95%	94%	95%	170	38
VGG16.4	1898/1894	50/50	89%	98%	93%	93%	93%	230	36
VGG16.4	1896/403/90/1403	50/10/ 3/37	87%	93%	88%	91%	90%	200	228

MODEL	Data Size (train/valid)	Param. Trainable	Param. Total	Train Time (avg epoch)	Epochs	Train Acc	Valid Acc
VGG16.2	23000/11500	2'425'730	14'780'610	2570 sec	10	99.7%	89.12%
VGG16.4	24500/12000	2'425'730	14'780'610	2650 sec	10	99%	93%
VGG16.1	23000/11500	2'425'730	14'780'610	2300 sec	10	99.3%	91.9%

Table 5- statistics and information for the training phase

E. Comparison of chosen models

A significant improvement has been made based on the fact that most of the false positives were happening under three conditions: extreme lighting (a stroboscopic effect or glare that can be extremely difficult to ignore or learn), extreme darkness (the lights on the side of the road would sometimes appear as if they are lights on the ceiling of the tunnel, but this can be challenging to identify even for a human), and the presence (and quantity) of images containing bridges in the datasets (due to their similarity).

In the first table (Table 4) we have metrics from the testing phase, in the second table (Table 5) we have information about the training process, validation process, and general details about the model. It should be considered that for VGG16.1 the database was the same but partitioned differently because the sequences are randomly split between the final files (training, validation, and testing database).

Another important factor is computational time and our models manage to predict an image in 0.1 seconds on average (VGG16.2: 0.095 seconds, VGG16.1: 0.11 seconds, VGG16.4:0.098 seconds). The predicting algorithm has been tested on a GeForce GTX 1080 Ti.

F. Active learning strategy

The implementation was complicated due to the restrictions and limitations posed by the SSH connectivity to the server, but by creating a simplified interface we integrated all the necessary methods. We automated the process of active learning, we initialized a class with a model name and three input files (the first training database, the unlabeled database for the augmentation and a validation database for comparing). The model is trained on the first database and then it makes predictions on the second one. The predictions with a high certainty will be added directly to the training set, and the other will be labelled by a user and then added to the training set. The model will be retrained on the extended data set and then the process can be repeated if another unlabeled database is given as input. After each training phase the model will be tested on the validation data set. Since all the models will be tested on the same data and for all of them the data is new, we can guarantee the fairness of the results.

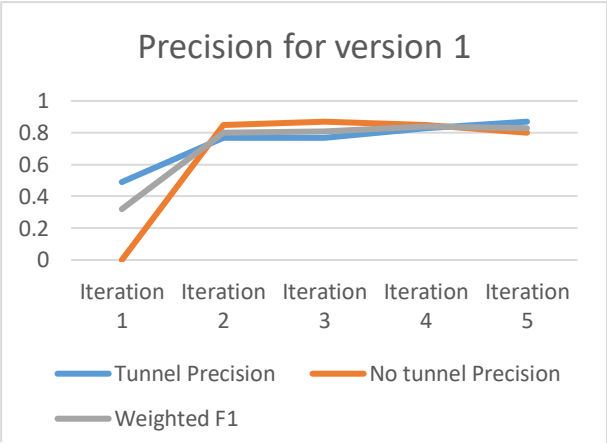

Figure VII-2

The models could not be trained for too much time because there were time limitations to the SSH connection and this complicated the process, but we managed to completely train 2 versions. The first version went through 5 iterations using datasets with an average size of 100 images augmented per iteration and a validation dataset of 456 images. That means the last iterations trained on a little over 600 images. The second version trained for 6 iterations with over 900 images on average per iterations (a total of about 6000 images) and was tested on a validation dataset of 5746.

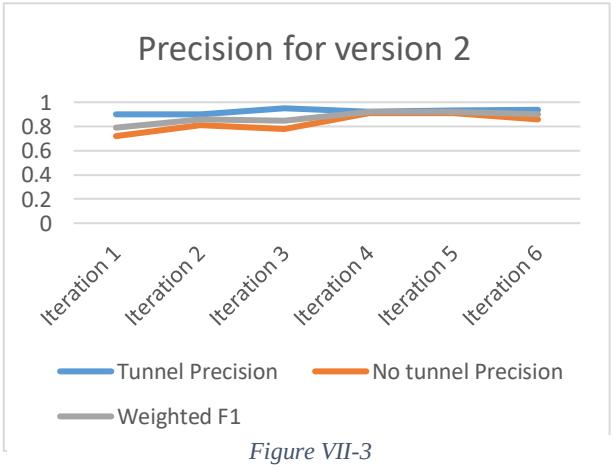

Figure VII-3

G. Active learning strategy interpretation

As we can see in Figure VII-2 and Figure VII-3 the results are quite positive for such a little dataset. Using the same validation data set (only reduced for version 1) as the original algorithm: supervised tunnel binary classification, we see that we obtain very similar results and with a very different setup and limited resources. Every individual model is trained the same as the original algorithm only it takes far less data (600/40'000, respectively 6000/40'000), trains for way less time (5 and 15 minutes compared to one hour each epoch) and for fewer epochs (maximum 5 compared to 10). The most important fact is that they needed way less labelling and human intervention (they required 225 labelled images out of 500 total images, respectively 1150/6000). We have some huge advantages here, very little effort for pretty good accuracies, but the models could have been trained for more time and with more data. This model might be lacking the robustness, consistency of the original one and it certainly has bigger variations between iteration, but with more resources it will most likely surpass its predecessor.

VIII. Conclusions

We made continuous improvements with each new strategy implemented, even if we cannot properly quantify the improvement with anything other than visualization techniques. It must be considered that the ultimate task in this problem is to recognize tunnels on the street with as great an accuracy and as few false positives as possible. The false positives remained the same, but the situation in which they appeared changed. Instead of predicting false positives while going under a bridge or during the night, they were predicted more in the time interval right before a tunnel entrance or exit was labelled. This is now a labelling issue and our model should recognize most tunnels (from a sequence instead an individual image) out there and nothing more.

It has been observed that the original binary tunnel classification model although having a great accuracy, it lacked versatility. It had certain cases in which it consistently predicted badly (nighttime with bad illumination, daytime with direct sunlight), but it managed to recognize every individual tunnel sooner or later. The two improvements implemented for this method seemed to be solving the problem, although at the cost of accuracy (which we should not forget that is very strongly related to the human labelling standards).

With the second classification it was proved that using multi-class for each of the tunnel section the areas in which the model failed (at least statistically) were the entrance and exit. That confirmed our concerns that human labelling was the problem, and also confirmed our hopes that the model was in did predicting very well overall, but not exactly at the same pace with the human.

The last method was a strategy implemented on the untrained model from supervised binary tunnel classification using active learning. This attempt proved to improve immensely the information usage from the given data, the model evolved at a much quicker rate and with very little resources comparative to its predecessor. Moreover, it seemed that training in only a fraction of the original time it came close to the accuracy of the original model.

IX. Limitations

Even though there are a great number of improvements compared to any other published prediction technology of this kind, there are of course a couple of limitations. The two main considerable limitations are related to the manner in which the labelling is done, and to the time constraints.

Let us explain in more detail the problem related to labelling. This situation forces us to set up some standards that need to be respected when labelling, standards regarding the distance until a tunnel is considered visible. Since this is entirely contextual we decided to label as positive any image that contains a clear tunnel, approximately centered, and that covers a certain portion of the image (width of tunnel is at least 10% to 15% out of the total width of the image). The same concept is applied for the exit from a tunnel. This standard is quite relative and open to considerable human errors. The only disadvantage is the fluctuation/oscillation that appears when going from one label to another (right at that time when the entrance/exit is going from 10% of the image to 20%), due to the fact that it didn't learn a very rigorous boundary between the labels.

The second problem is quite simple and is related to the time needed to predict one image. On average the algorithm predicts the class of an image in about 0.095 second, that means it can go up to 10 frames per second. It must be considered that this computation took place on a very powerful GPU and it could not run efficiently in real-time on a car unless the car has special feature-specific equipment. It is rather obvious that if we want to achieve greater precision and accuracy, then we must sacrifice computation time, so this might not be the best solution for any system.

X. Comparison

Out of the two main works that attempt to solve the same problem as us, none of them uses artificial intelligent. Both of them rely strictly on image processing, which gives them an advantage related to the prediction time. An interesting approach and with a better accuracy over a large array of data has the Road Tunnel Entrance Recognition System [5], but the statistical better one regarding accuracy and data base size (not to forget the huge difference in runtime) is the Fast Vision-Based Road Tunnel Detection [6]. Both of them use considerable smaller datasets (given by the number of sequences and variety in the environment).

One similar method used high quality image data provided by moving satellites fed into a neural network to detect roads, bridges, and tunnels. The idea is based on the fact that two road segments identified and split by a relatively small space may contain a bridge or a tunnel. [7]

The second mentioned research with similar objective was a paper that uses Radar Signals to identify iron tunnel taking advantage of reflection and diffraction of radar sensors. [8].

Acknowledgment *(Heading 5)*

The research regarding "Chapter 3: Literature overview" has been done by the author alone. It only acknowledges work posted online publicly under any form and before the date of 04 – April – 2020. It cannot be guaranteed that other similar work (in progress or under certain confidentiality restriction) do not exist, furthermore I suspect their existence due to the highly competitive industry of autonomous driving.

XI. References

[1] V. V. S. I. J. S. Z. W. Christian Szegedy, "Rethinking the Inception Architecture for Computer Vision," Cornell University, 2015.

[2] A. Z. Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Cornell University, 2015.

[3] X. Z. S. R. J. S. Kaiming He, "Deep Residual Learning for Image Recognition," Cornell University , 2015.

[4] Z. L. L. v. d. M. K. Q. W. Gao Huang, "Densely Connected Convolutional Networks," Cornell University, 2018.

[5] N. Christoffer, Road Tunnel Entrance - master thesis, Oslo: University of Oslo, 2009.

[6] B. M., B. A., B. G. and M. L., "Fast Vision-Based Road Tunnel Detection.," in *Image Analysis and Processing*, Berlin, Heidelberg, Springer, 2011.

[7] N. Ghasemloo, M. R. Mobasheri, A. M. Zare and M. M. Eftekhari., "Road and Tunnel Extraction from SPOT Satellite Images Using Neural Networks," *Journal of Geographic Information System*, p. 6, 2012-2013.

[8] S. Lee, B.-h. Lee, J.-E. Lee, H. Sim and S.-C. Kim, "Iron Tunnel Recognition Using Statistical Characteristics of Received Signals in Automotive Radar Systems," in *2018 19th International Radar Symposium (IRS)*, Bonn, Germany, 2018.

[9] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev and N. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," *Mathematics and Computers in Simulation*, p. 232–243, 2020.

[10] T. Takeshi, Tunnel detecting device for vehicle and light control device for vehicle, USA: United States Patent Service, 2005-2006.

[11] V. Sze, Y.-H. Chen, T.-J. Yang, J. S. Emer and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *IEEE 105*, 2017.

[12] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, pp. 714-735, 1997.

[13] K. Patel, "Overfitting vs Underfitting in Neural Network and Comparison of Error rate with Complexity Graph," 14 september 2019. [Online]. Available: https://towardsdatascience.com/overfitting-vs-underfitting-ddc80c2fc00d.

[14] P. a. F. F. H. NVIDIA and Vingelmann, *NVIDIA and Vingelmann, Péter and Fitzek, Frank H.P.*, https://developer.nvidia.com/cuda-toolkit, 2020.

[15] D. Jamil, "Getting into Convolution Neural Networks," 3 May 2020. [Online]. Available: https://medium.com/@D3nii/convolution-neural-network-9320fa32fe9c.

[16] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering*, vol. 9, pp. 90-95, 2007.

[17] R. O. Duda, P. E. Hart and D. G. Stork, Pattern Classification - Second edition, New York: WILEY-INTERSCIENCE, 2001.

[18] S. Davies, "Learning in spiking neural networks," %0 Conference Proceedings, 2013.

[19] F. Chollet, *keras*, {https://github.com/fchollet/keras, 2015.

[20] C. C., S. H.C. and S. W., "Tunnel Entrance Recognition for video-based Driver," *Technical University Munich*, 2006.

[21] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[22] M. a. B. P. a. C. J. a. C. Z. a. D. A. a. D. J. a. D. M. a. G. S. a. I. G. a. I. M. a. o. Abadi, "Tensorflow: A system for large-scale machine learning," in *12th Symposium on Operating Systems Design and Implementation*, 2016.

[23] S. Yeung, "Introduction to computer vision," 2015. [Online]. Available: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html.

[24] B. Settles, "Active Learning Literature Survey," University of Wisconsin–Madison, 2010.

[25] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach (3rd ed.), Upper Saddle River, New Jersey, 2009.

[26] S. Raschka, Python Machine Learning, Packt Publishing Ltd., 2015.

[27] G. V. M. Eduardo A.B. da Silva, "Digital Image Processing," *The Electrical Engineering Handbook*, 2005.