Mask R-CNN in Lane Detection

Facundo Calcagno
EISTI, Universidad de Buenos Aires
calcagnofa@eisti.eu

(Dated: 26 June 2018)

The computer vision field has had a extraordinary growth since the emergence of Convolution Neural Networks (CNN) and its implementation in GPU's. Several object recognition challenges have appeared all around the world, and this technique has overpowered all other techniques in such a way that has monopolized the field and has been taken as the new standard. Our approach is to use this approach and a 2017 implementation called Mask R-CNN to dive into the Lane Recognition Problem and treat it as Instance segmentation. We show top results for it's predictions and run-time performance in different environments. The Code is available at: `http://github.com/fmcalcagno/MASK_Lane_Detection`

Keywords: Deep Learning, Image Segmentation, Object Detection, Self Driving Car

The aim of this paper is to introduce to the newcomers the ideas of Deep Neural Networks started by Yan LeCun and continued by Alex A., NYU, Google and Facebook teams, make a small panorama of the more common types of Neural Networks available and explain in detail a new and very successful architecture called Mask R-CNN that has won recognition all around the world. After this big introduction we will dive into the resolution of the problem of Lane Recognition with images taken from inside cars using CuLanes dataset. We will see how difficult and problematic this type of images can be due to the different and possible geometric issues that diverse landscapes have. Nevertheless, we will show that the technique is applicable in this specific problem and could be improved to be automatized and implemented in a self-driving car.

I. INTRODUCTION

Since the appearing of the first Convolutional Neural Network[1] (CNN) by Y. LeCun in 1998, LeNet, followed by superb improvements by the famous AlexNet in 2012[2] and its followers ZFNET[3], GoogLeNet [4] and VGGNet[5] in 2014, to the most commonly used Deep Neural Network today, the ResNet[6], in 2015, research labs all around the world are diving into solving really complex problems with this new tools and CNN became the gold standard in image classification, even beating the human eye. Section II will cover an explanation of these networks architecture.

Furthermore, one of the problems that is gaining sight these days, is image Segmentation, which means the ability to recognize different types of objects in the same image and their exact position within the image covered in Section III.

The first approach to these problem came in 2013 by a team in Berkeley University with an acronym named R-CNN[7], which stands for Regional Convolutional Neural Networks. After this approach, several other improvements came along, Fast R-CNN[8] and Faster R-CNN[9] which implement in a clever way the same idea.

In 2017, the Facebook AI team launched a new improvement to this idea, and named in Mask R-CNN[10]. The main improvement is the idea of recognizing exactly where the object is with a pixel-to-pixel identification. Moreover, Mask R-CNN has been implemented with success in many different scenarios and it outperforms all existing, single-model entries on every task, including the COCO 2016 challenge winners.

In this paper we provide an implementation of the Mask R-CNN algorithm in a lane detection problem. The idea of a self driving car has been in the air since the arrival of neural networks applied to specific problems. In Section IV, we will explain the dataset, called Culanes[11], which is a large scale challenging dataset for academic research on traffic lane detection, by the Multimedia Laboratory of The Chinese University of Hong Kong, and who our pixel-to-pixel approach is different to ours.

Section V includes the description of the solution cloud architecture and how to manage an scalable GPU training environment.

In Section VI includes results of the Mask R-CNN Lane Detection implementation and some of the issues we have faced.

Section VII includes conclusions of our work and how to continue this line of research.

II. CONVOLUTIONAL NEURAL NETWORKS

Multi-layer networks trained with gradient descent have demonstrated to be able to generalize very complex and high-dimensional data when there is enough information to feed the algorithm. In the case of images, the breakthrough of CNN's didn't come only with the architecture of this algorithm, which will be explained later, but also with a different way of dealing with the data. Traditionally, for each image that was inputed into the algorithm, there was a structural need of looking for the most important kpi's (key performance indicators) that

could explain the image and feed the algorithms. These Feature Selectors where inputed into a multilayer Neural Network and the idea of the algorithm is that using Steepest Descent, the algorithm would be able to generalize and learn the necessary weights for each of it's neurons to optimize a given loss function.

With the development of Convolutional Neural Networks, there was no more need of looking for these *feature selectors* due to the fact that CNN's inputed the full image into it's architecture.

In the following sub-sections there is a description of all the key parts of a CNN architecture. Understanding each part of the network is crucial to sense why CNN's are such good estimators for all type of optimization functions and the existence of a hype into this technology.

A. Convolutional Layer

The **Convolutional Layer** is the key stone in Convolutional Neural Networks. The idea of this layer is to look for specific features in the images by sliding different filters throwout all the image. Every filter has a small dimension (width and high-wise) but extends through the full depth of the input volume. During the forward pass, this filter slides (or convolves) across the width and high of all the input data and computes the dot product between the input and this filter at all positions.

While sliding the filter throw the full images, height and weight, we will generate a 2-dimensional activation map that gives the result the dot product in all the original image positions. Little by little, the CNN will learn filters that activate when they recognize some visual specific characteristic (edges, colors or orientations). Each Convolutional Layer will have a different numbers of filters and generate a 2-dimensional map for each filter.

When the algorithm finishes, these filters will be able to distinguish different types of images by understanding the patters that we are trying to generalize. That is the key of "learning".

Local Connectivity is another key part of CNN's. Connecting all neurons of the previous layer, to all neurons of the following layer has turned to be impractical and very bad for learning. Instead, each neuron will be connected to a small number of layers in the previous layer, and in the first layer, it will connect only to a small region of the input image. The spatial constraint is an hyper-parameter that sets the receipted field of the neuron as we can see in Figure 1.

Three hyper-parameters control the size of the output layer: the **depth**, the **stride** and the **zero-padding**. The **depth** is the number of filters we would like to apply in that layer. The **stride** is a parameter we need to set to know how to slide the filter. If it's set to one, then we will slice the picture 1 pixel at a time. Setting the stride to 2 means to jump 2 pixels at a time when sliding. The bigger the slide parameter is, the smaller the output spacial size will be. Depending on the input size and

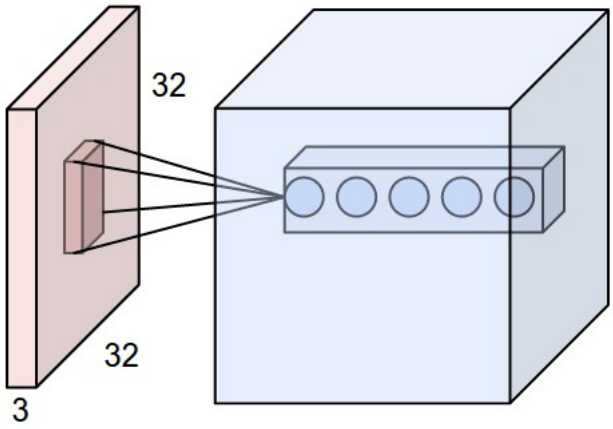

FIG. 1. Convolutional Layer: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input

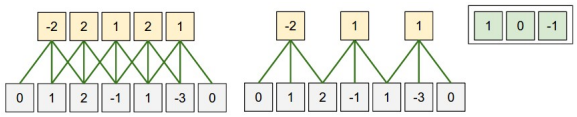

FIG. 2. llustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1. Left: The neuron strided across the input in stride of S = 1, giving output of size (5 - 3 + 2)/1+1 = 5. Right: The neuron uses stride of S = 2, giving output of size (5 - 3 + 2)/2+1 = 3. Notice that stride S = 3 could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since (5 - 3 + 2) = 4 is not divisible by 3. The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

the desired output size, frequently there is a need to pad zeros around the border of the input. This feature allows us to control the spacial size of the output volume. In Figure 2 we can see an example to fully understand these 3 hyper-parameters.

The ability to **share parameters** throwout the network allow us to control the number of parameters. The Alexnet Architecture that won the ImageNet competition in 2012 accepted images of size [227x227x3]. In its first convolutional layer uses neurons with a receptive field size F=11, stride S=4, and no zero-padding P=0. This means that the Convolutional layer had a depth of K=96, the Conv layer output had size [55x55x96]. Each of the 55*55*96 neurons in this volume was connected to a region of size [11x11x3] in the input volume. In this architecture we see that there are 55*55*96 = 290,400

neurons in the first Conv Layer, and each has 11*11*3 = 363 weights and 1 bias. Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone. This is a very high number and that's why parameter sharing was introduced into CNN's.

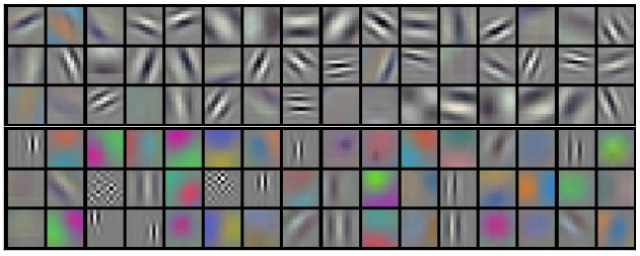

FIG. 3. Filters. Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice.

Using the assumption that only one feature an be useful at each spacial position (x,y), constraining the neurons in each depth slice to use the same weight and bias. With this new schema, the first layer of AlexNet would have only 96 unique set of weights, for a total o 96*11*11*3 = 34,848 unique weights or 34,944 parameters (including the 96 bias), much less that without sharing parameters. In Figure 3 we can see the filters of this first Convolutional layer. As all neurons in a single depth are using the same weight vector, then the forward pass of the convolutional layer is a convolution of the neuron weights with the input volume. We will refer, from now on, to the set of weights as a **filter** or a **kernel** which convolutes with the input.

B. Pooling Layer

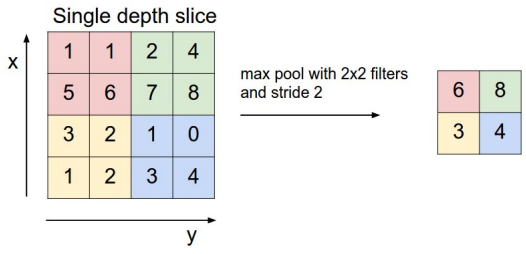

FIG. 4. MaxPool layer with stride 2.

The **pooling layer** is used to down-sample the output of a convolutional layer. Normally, it's used in between of successive convolutional layers. The main idea is to keep the most important features of the previous layer but reducing the dimensionality of the output, helping the neural network to generalize. Several pooling strategies have been used over the years; Max pooling, Average pooling or L2-norm pooling for example. Historically speaking, average pooling was deeply used but over the last years it has been replaced with max pooling due to better performance. An example of Max Pooling is shown in Figure 4.

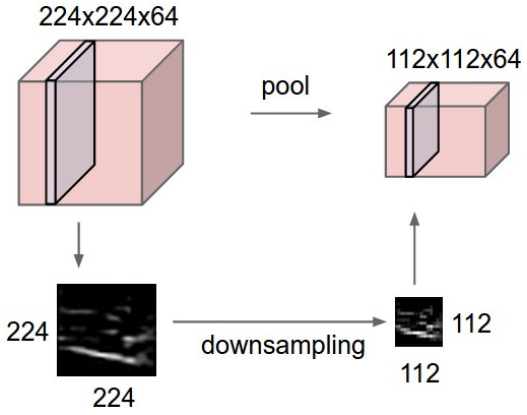

FIG. 5. Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved.

It requires 2 hyper-parameters, their spatial extent F and the stride S. As a result, it returns a volume size of the same depth but with different width and height, depending of these 2 hyper parameters. In Figure 5 we show an example of the behavior of a pooling layer.

C. Fully Connected Layer

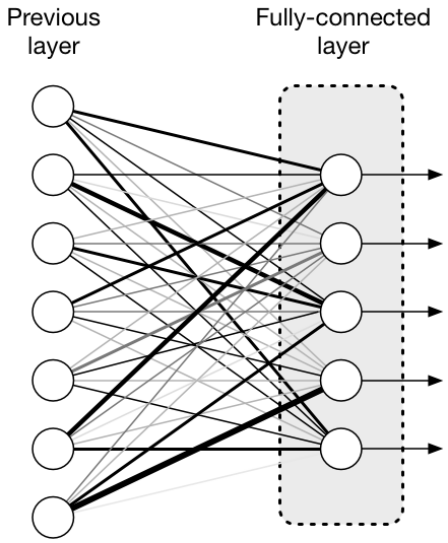

FIG. 6. Fully Connected Layer with 7 inputs and 5 outputs. This architecture has (7*5)+7= 42 parameters

A **fully connected layer** or **FC layer** represents a layer in a Neural Network in which all neurons of that

layer are connected to all neurons of the previous layer. It's the most widely used layer in conventional neural networks. The activations of this layer can be explained as a matrix multiplication of the weights of the FC layer with the outputs of the previous layer followed by an addition of a bias vector. In Figure 6 we can see an example of this layer.

D. Activation Layer

The purpose of an **activation layer** is to introduce a non-linearity to the system which basically has been computing linear operations during it's convolutional layers. Several different non-linear functions where used over the years, including *tanh* and *sigmoid*, but researchers found out that a **ReLU**[12] layer worked better because the network is able to train faster and it also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.

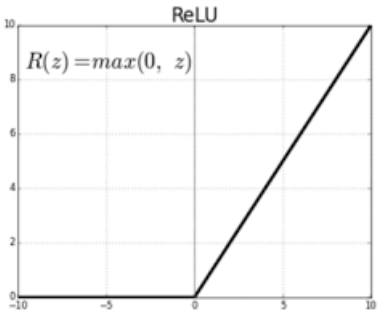

FIG. 7. ReLU Layer.

The **ReLU** layer (Rectified Linear Unit), Figure 7, applies the function f(x) = max(0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the convolutional layer.

E. Softmax Layer

When dealing with a classification problem, the main idea is to be able to classify an input into a number of classes. For example in the MNIST Dataset using by LeCun for LeNet, the objective was to classify an image of a handwritten number into 10 classes, numbers from 0 to 9. The idea is to be able to merge all the outputs of previous layers and be able to give a score to each output class, choosing the class with better score. This is the main idea behind the softmax layer.

The Softmax layer is usually the last layer of a Neural Network. It uses the softmax function defined as:

$$softmax(a) = \frac{exp(a_i)}{\sum_j exp(a_i)}$$

As we can see the result of this function will be always 1, because it will be the sum of probabilities of the input being in one of the output classes. You could see the softmax layer as a competition of the units that participate in it. At the extreme, it becomes a form of **winner-take-all** game, an the winner unit will get the classification.

F. Dropout

Overfitting is a very big problem when training deep convolutional neural networks. Dropout[13] is a technique that addresses this problematic preventing units to co-adapt. It provides an approach to combine exponentially many different neural network architectures with a very big efficiency.

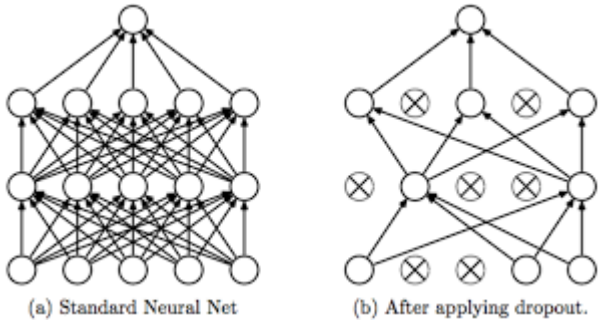

FIG. 8. Dropout

The main idea of this architecture is to randomly drop units in a neural network at training time. This means that, with a temporal constraint, some units won't receive any input and won't output any signal as shown in Figure 8. The selection of which units to dropout is random with a fixed probability. That probability must be the same throwout all the training.

G. Data Augmentation

Data Augmentation is yet another technique used when training convolutional neural networks to reduce over-fitting[14]. It allows the algorithm to work with a bigger number of training samples than the existing ones by making little variations in the sample without changing the nature of the input. The main procedure is called *data warping*, which augments the input data by applying geometric or color augmentations, such as reflecting, cropping, translating, changing the palette of the image or enhancing certain parts of the image, as we can see in Figure 9.

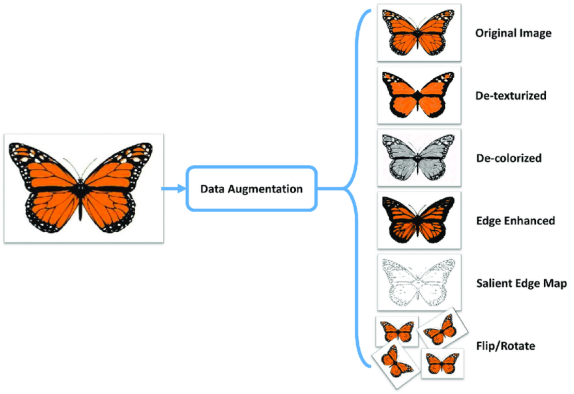

FIG. 9. Data Augmentation[15]

H. Batch Normalization

When training Deep Neural Networks we often come across to a problem called *internal covariate shift*. *internal covariate shift* is the fact that after certain time training slows down by requiring lower training rates and careful parameter initialization. This is needed because the distribution of each layer's input changes during training when the parameters of the previous layer change.

Batch Normalization[16] is a method that aims to correct this issue by using normalization as a part of the model architecture and carrying out normalization for each training mini-batch. It can be used as well as a model normalizer, and takes out the need of using **dropout**.

Implementing **Batch Normalization** means including a BatchNorm layer just after fully connected layers or convolutional layers and before non-linearities. In traditional deep neural networks, big learning rates can result in gradients that may explode or vanish, or it might reach a local minima. Batch Normalization also aims to attack this issue by normalization all activations all over the network and preventing little changes in the parameters to widen into changes that would lead to sub-optimal results.

I. Optimization Techniques

Diverse optimization techniques where implemented to optimize deep neural networks since stochastic gradient descent (SGD)[17] by Robbins and Monro, 1951. The most remarkable algorithms are: Adagrad[18], Adadelta[19], Adam[20] and Adam with restarts[21].

Adagrad[18] dynamically incorporates knowledge of the geometry of the data seen in previous iterations to perform more revealing gradient-based learning. The main concept is to give to very frequent features low learning rates and to infrequent features high learning rates.

Adadelta[19] is an extension to Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size **w**.

Adaptive Moment Estimation (**Adam**[20]) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t.

Adam with restarts[21] is a technique that solves the issue of weights not decaying multiplicatively but by an additive constant factor. It works by decoupling weight decay and the optimization steps taken with respect to the loss function.

J. Layer Patterns

In Convolutional Layers Architectures is very common to see stacks of a Few **Conv-ReLU** Layers, followed with **Pool** layers, and repeating this pattern until we can reduce substantially the size of the image. After several repetitions of this pattern, it's common to attach some **FC** layer to transition into the number of categories we are trying to generalize the input pictures. This can be achieved by attaching several FC layers with continuing shrinking size to the last FC layer that will contain the number of classes we want the algorithm to output.

K. Famous CNN's

As wee seen before, a CNN is a special kind of multilayer neural networks, that are designed to recognize visual patterns directly from the pixels with no feature extractors and minimal processing techniques. In the last years, diverse layer configurations have been implemented to be better in recognizing objects and patterns. In the following lines, we describe the LeNet[1], by Yan LeCun in 1998, the first CNN, to the very famous and state of the art ResNet[6], which is one of the standards in Object Recognition in this days.

1. LeNet

LeNet was developed by Yan LeCun in 1989 to recognize handwritten numbers. In this section we will discus on a specific network called LeNet-5[1]. This architecture comprises 7 layers, without taking into account the input, 60K 32x32 pixels images which are normalized to accelerate learning.

The network has 7 layers,:C1, C3 and C5 are convolutional layers; S2, S4 are sub-sampling layers; F6 is a fully-connected layer; and the final layer is a Radial Basis Function (RBF) with 10 outputs (1 for each possible digit) as seen in Figure 10.

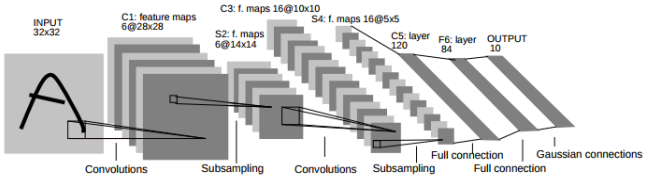

FIG. 10. Architecture of LeNet-5 for digit recognition. Each plane represents a feature map.

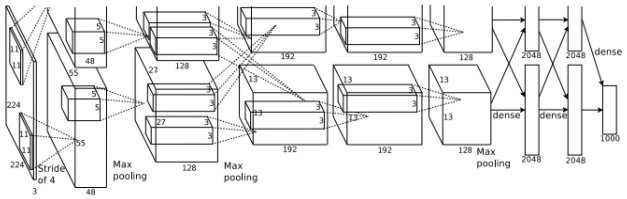

FIG. 11. AlexNet

The first layer, C1, is a convolutional layer with 6 features maps. Every unit of those feature maps is connected to a 5x5 region in the input image. This resolves into 6 different [28x28] feature maps.

The second layer, S2, is a sub-sampling layer of size [14x14]. The 6 feature maps remain but with lower dimension representation. Each unit is connected to a [2x2] region of the previous layer, which sums the 2 positions, multiplies by a trainable coefficient, adds a bias and passes through a sigmoid function.

The third layer, C3, is a convolutional layer with 16 feature maps. Each unit is connected to a [5x5] neighborhood of he previous layer. Feature maps of the previous layer are combined, in groups of all possible combinations of 3 to 5, to generate 16 feature maps in the output of C3. This idea was implemented to crack the symmetry in the network and connect different parts of the image.

The forth layer, S4, is another sub-sampling layer with 16 feature maps of size [5x5]. We repeat the process made in S2 connecting each unit to a [2x2] region of its input.

The fifth layer, C5, is the last convolutional layer which contains 120 feature maps. Now, as the input is a [5x5] image, each unit is connected to a 1x1 area from the input.

The sixth layer, F6, is a fully-connected layer with 84 units and fully connected to C5.

The last layer, is composed of Euclidean Radial Basis Function units (RBF), one for each class, with 84 inputs each.

LeNet-5 was a pioneering network that started the idea of CNN's and was applied by several banks to recognize hand-written numbers on checks digitized in [32x32] pixel grey-scale input images.

2. AlexNet

This network has a similar architecture to the previously described LeCun-5 but deeper, with a higher number of filters per layer, adding the idea of stacked convolutional layers, ReLU activations, dropout, max pooling, data augmentation and Stochastic Gradient Descent (SGD) with momentum.

AlexNet[2] is a deep CNN with 11 layers, distributed in 5 stacked convolutional layers, 3 max pooling layers and 3 stacked fully connected layers as you can see in Figure 11.

It's important to clarify that the architecture is divided into 2 different parts to be able to train in 2 different GPU's.

Is the first architecture to implement the Relu Layer as non-linearity helping combat the saturating gradients and training the algorithm in less time. In addition, it also implements overlapping pooling whereas all previous architectures used non-overlapping pooling to reduce dimensionality.

The first 4 layers are a repetition of a pattern. Two convolutional layers (one with 96 filters of size [11x11] and stride 4, the other with 256 filters of size [5x5], stride 1 and padding 2) with a Max-Pooling layer right after (both [3x3] filter with stride 2). After these 4 layers, 3 stacked convolutional layers with 384, 384 and 256 filters respectively (all three with size [33], stride 1, padding 1). Followed by a Max-Pooling layer (with size [3x3] and stride 2). Ending with 3 stacked fully connected layers with 4096, 4096 and 1000 neurons each respectively.

The output of the network was set to 1000 class labels because it uses the previously mentioned ImageNet dataset.

In order to reduce over-fitting data-augmentation and dropout were implemented in AlexNet.

Applying data-augmentation was done in two different ways. Firstly, using horizontal reflections and generating translations of the original images. Secondly, was altering the intensities in the RGB channels in the training set by performing a PCA in the training set and adding multiples of the found principal components.

Dropout was implemented with a p=0.5 probability for each neuron on training time. In test time all neurons are used.

AlexNet was the winner of the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

3. ZFNet

Following AlexNet ILSVRC 2012 victory, there where several CNN's developed for the ILSVRC 2013 event. The winner of that competition was ZFNet[3] developed by a team in NYU, achieving a 11.2% error rate. The ZFNet is very similar to AlexNet and we can defined as a fined tuned version of the previous winner of the com-

petition.

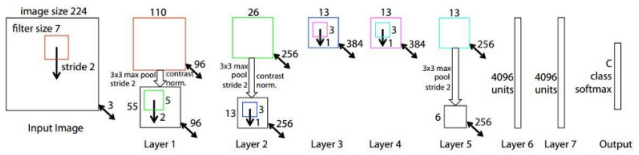

FIG. 12. ZFNet Architecture

ZFNet trained in only 1.3 million images (in comparison to the 15 million AlexNet used), and changed the first Convolutional layer filters from [11x11] in AlexNet to [7x7] pixels with stride 2 as we can see in Figure 12. The justification to this change is that a smaller filter size can retain much more detail from the original image. Another change is the increased number of filters used in this network in comparison with AlexNet.

4. GoogLeNet

Moving forward in the chronology of winners of ILSVRC, the 2014 winner was a deep convolutional neural networked called Inception or it's incarnation called GoogLeNet which achieved a top-5 error rate of 6.67%.

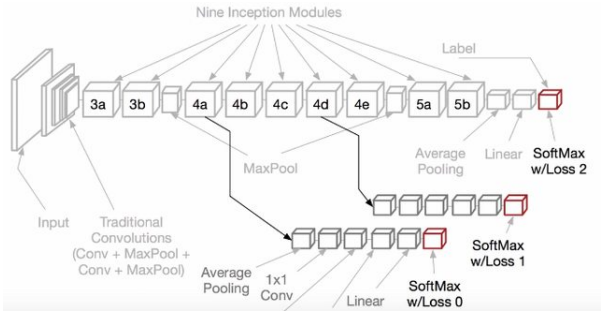

FIG. 13. GoogLeNet Architecture

GoogLeNet is a 22 layer deep network that implemented a new idea, the development of an **Inception Module** than can be repeated in different parts of the Network as you can see in Figure 13. This module is based on several very small convolutions in order to achieve parameter reduction, lowering the total number of parameters of the network to 4 Million.

The idea behind the Inception module is to leave the decision on which type of convolution you need to the model and implement different types of convolutions in parallel. In Figure 14 we can see how 4 different convolutions are done in parallel, specifically [1x1], [3x3] and [5x5] convolutions with a 33 max pooling.

The architecture has 3 softmax layers to reduce the vanishing gradient problem during training. The main reason for that decision was to add classifiers to the intermediate layers a well, such as the final loss to be a combination of the intermediate loss with a discount weight

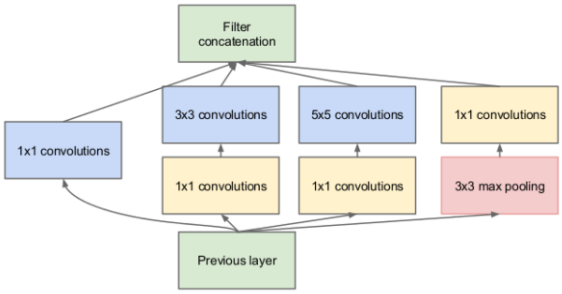

FIG. 14. Inception module architecture

and the final loss. These losses are discarded at inference time.

Furthermore, GoogLeNet removes fully connected layers, and replaces them with average pooling at the top of the convolutional network, reducing a large number of parameters that weren't necessary. In addition, it uses batch normalization, data augmentation with image distortion and RMSprop[18], a gradient descent method with adaptive learning by an exponentially decaying average of squared gradients, similar to Adadelta, explained in Section **??**.

5. VGGNet

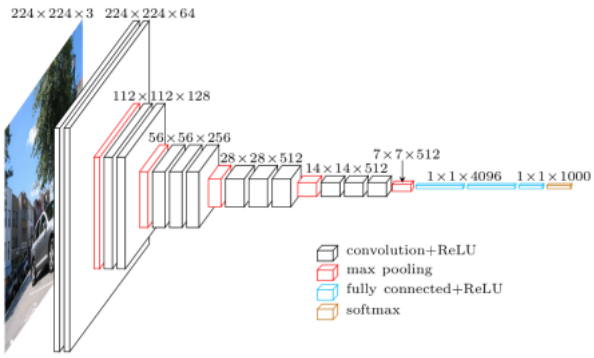

FIG. 15. VGGNet Architecture

VGGNet[5] is a deep convolutional neural network developed and trained by Oxford's renowned Visual Geometry Group (VGG), is the runner-up of the ILSVRC 2014 event, just behind GoogLeNet, with a 7.1% error in the validation set.

The network takes the simplicity of LeNet and AlexNet stacking [3x3] convolutional layers with ReLU with[2X2] max pooling layers on top of that stack. In the final layers in includes 3 fully connected layers and a softmax layer as the output as the Figure 15 shows.

The most important thing to learn about this model is that it exposes that the depth of the network is key for achieving good performances.

6. ResNet

As a last CNN architecture we would like to show the architecture of the ILSVRC 2015 winner, the **ResNet**[6]. With a top-5 3.57% error rate in ImageNet it most brilliant idea was to solve a common problem in training very-deep neural networks, *accuracy saturation*.

Accuracy saturation occurs when deeper networks start converging. The deeper the neural network is, the more saturated the accuracy gets and then degrades rapidly. This means that the network could be replaced by another network with less layers, leaving the last layers as just identity functions.

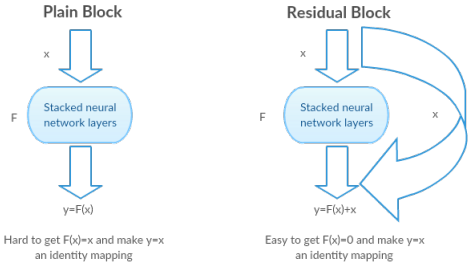

FIG. 16. Identity mapping in Residual blocks

The main idea developed by Kaiming He et al. in the Residual Network, or ResNet, is instead of learning a direct mapping of x -> y with a function H(x), we now represent H(x) = F(x) + x, where F(x) represents the stacked non-linear layers and x the identity function (passing the input into the output). In Figure 16 we can see the difference between a normal block and a residual block.

Furthermore, ResNet uses [3x3] filters mostly and Pooling layers use Stride 2. The architecture doesn't have fully connected layers at the end. It features skip connections as detailed before with it's residual blocks and a heavy use of batch normalization.

ResNet has implementations with 18, 34, 50, 101 and 152 layers and till 2017 was the state of the art in CNN models. In Figure 18 we can see a representation of ResNet34.

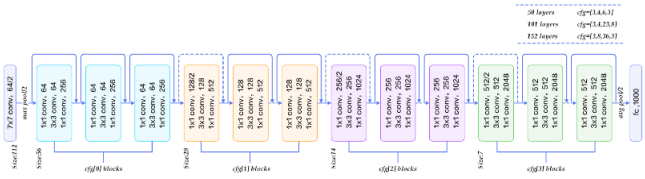

FIG. 17. ResNet34

III. IMAGE SEGMENTATION

The previous chapter shows how deep neural networks are very good at classifying images with single objects. But real life problems are usually more complex and require to identify diverse objects in the same image, this is called **image segmentation**. Diverse architectures of CNN's where used to attack this issue, and in the following section we will address them.

A. R-CNN

The first approach to Image Segmentation using CNNs was from a small team at UC Berkeley, led by Professor Jitendra Malik. The found tat the problem could be solved using results from AlexNet but adding additional layers to that architecture.

The team developed an algorithm that given an image could correctly identify where the main objets where in the image by encapsulating them in bounding boxes. They called the algorithm **R-CNN**[7], for Regional CNN.

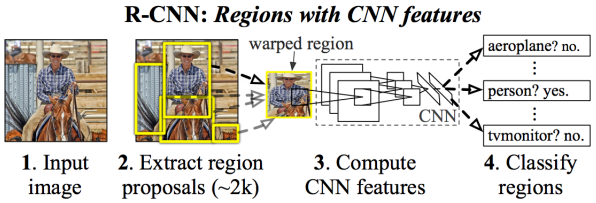

FIG. 18. RNN

R-CNN proposes a number of boxes in the image and checks if any of them corresponds to an object. The algorithm creates these bounding boxes, or regional proposals, by applying a process called *Selective Search*. This process, shown in Figure **??** inspects the image through windows of different sizes, and for each box it tries to group pixels by color, intensity or texture with the goal of identifying objects.

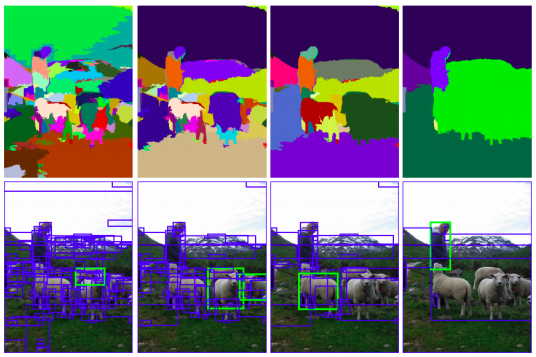

FIG. 19. Selective Search example.

When the Selective Search process is finished and all proposals are created, the algorithm uses the box as the

input of a tweaked version of AlexNet to look for a class to classify the object. On the last layer, **R-CNN** includes a Support Vector Machine (SVM) that has the task to decide if whether the box has an object or not, and in the positive case which one. The full process is explained in Figure 19 and a result of the algorithm can be seen in Figure 20.

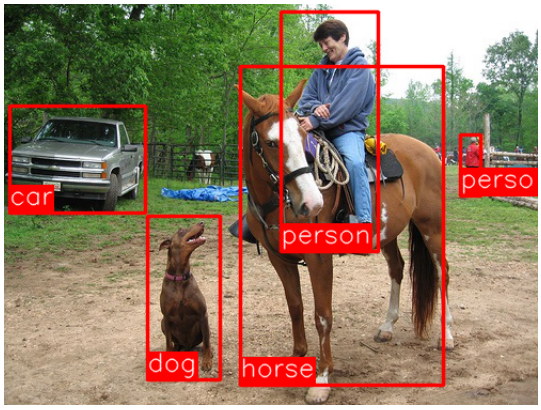

FIG. 20. R-CNN object detection

B. Fast-CNN

R-CNN has a very good performance. Nevertheless, it is really slow because it make a forward pass for every single regional proposal and due to having to train three different models independently. These two disadvantages make R-CNN very difficult to train.

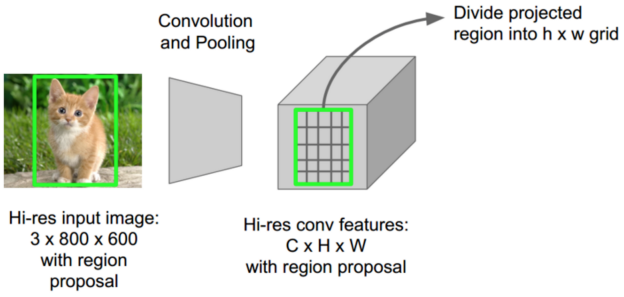

FIG. 21. In RoI Pool, in a single forward pass all the features are extracted for each region of interest

Fast R-CNN appeared to solve these issues. It implements a technique called *RoIPool* (Region of Interest Pooling), shown in Figure 21 that runs the CNN only once per image instead of repeating the process per region proposal. The RoI Pooling Layer uses max pooling to convert the features inside any valid bounding box into a feature map with a fixed height and weight (hyperparameters).

The second advantage of Fast R-CNN is to train the CNN, the classifier and the bounding boxing regressor as a single network. Figure 22 shows how the full process works.

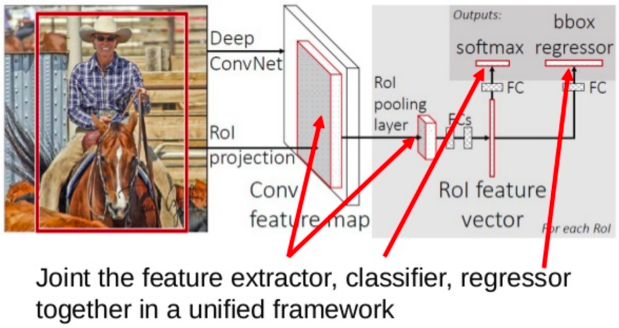

FIG. 22. Fast R-CNN Framework

C. Faster R-CNN

Both R-CNN and Fast R-CNN have a big bottleneck, this is the region proposer. In both architectures this step is very costly in terms of memory and time. **Faster R-CNN**[9] arrives to attend this problematic by generating proposals reusing the same CNN results used for the forward pass of the CNN, as we can see in Figure 23. In this case one only one CNN is used to generate the region proposals and the classification, reducing drastically the time needed to achieve the first task.

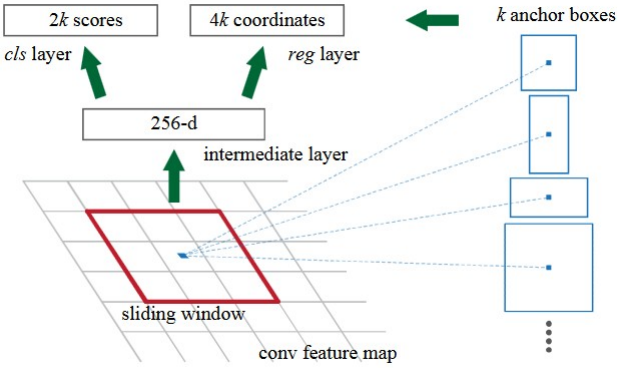

FIG. 23. Region Proposal Network implemented in Faster R-CNN, the RPN slides a window over the features of the CNN. At each window location, the network outputs a score and a bounding box per anchor

Moreover, Faster R-CNN includes a Region Proposal Network, shown in Figure 23 on top of the previously mentioned CNN, which is a fully connected layer. In short, it works passing sliding windows over the CNN feature map and outputting, for each window, k potential bounding boxes and scores that represent how good the predictions are.

D. Mask R-CNN

The previous models, R-CNN, Fast R-CNN and Faster R-CNN have the ability to detect objects instances in an image and wrap them into individual bounding boxes. The problem that Instance Segmentation tries to tackle is to detect individual object instances but it also wants to have a mask o each of the object instances.

Mask R-CNN[10] is a framework that extends Faster R-CNN adding a third branch that outputs the object mask in parallel with the classification and the bounding boxes regression as shown in Figure 24.

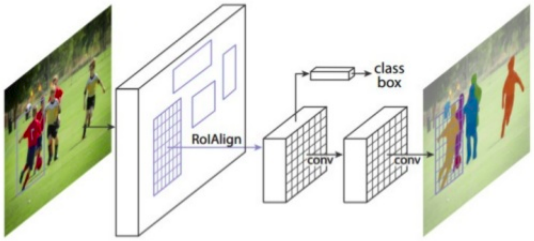

FIG. 24. Mask R-CNN Architecture

Moreover, Mask R-CNN adopts the same two-stages procedure, with the same Region Proposal Network as Faster R-CNN. In the second stage, it includes another step, in parallel t output a binary mask for each Region of Interest. When training, it uses a multi task-loss that includes the classification loss, the bounding-box loss and a mask loss, which is the average binary cross-entropy loss per pixel sigmoid. This definition of the mask loss allows the network to generate masks for all classes without competition between them. Each Roi gets a m x m mask prediction using a fully convolutional network.

FIG. 25. Mask R-CNN Results

The idea of a pixel-per-pixel mask, needs precision, and RoIPool (the algorithm used in Faster R-CNN) needs to be updates because of its discrete granularity. The network uses a RoIAlign layer, which removes this discrete granularity and uses bi-linear interpolation to compute the exact values of the input features.

Mask R-CNN outperforms all previous results on the COCO instance segmentation task, and some examples can be seen in Figure 25.

IV. CULANES DATASET

The CuLanes[11] dataset is a large scaling challenging dataset on traffic lane detection. Including pictures from six different vehicles from the streets of Beijing, it comprises of 133,235 frames in different traffic situations. These images were unsorted and have a resolution of 1640 x 590 pixels.

The authors of Culanes divided the dataset in three parts, 88880 images for the training set, 9675 for the test set and 34680 for the validation set. Every frame has been manually annotated with the traffic lanes using cubic splines. As well as in the original paper[11], in his project we center consideration in detecting four lane markings, as their are usually the most important in self driving cars. In Figure 26 we can see some examples of the dataset and how it has been annotated pixel by pixel.

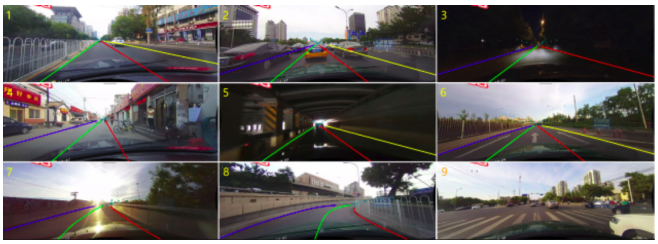

FIG. 26. CuLanes Dataset examples in 9 different traffic situations: Normal, Crowded, Night, No line, Shadow, Arrow, Dazzle light, Curve and Crossroad

The dataset contains two pair of images. The first pair contains the original images, and the second pair contains the mask of the lanes contained in the image, the *mask image*. Each *mask image* contains a number for every pixel in the original image. This number or *mask pixel* represents the existence of a lane in that pixel. This number has 5 different values. When the mask pixel has a zero (0) this means than that pixel has no lane on it. When the pixel has a number between 1 and 4 that means that there is a lane in that pixel and the number indicates which lane that pixel represents.

Every mask Image may have from 0 detected lanes to 4 as a maximum. This means that lanes are numerated from left to right. If the image has 4 lanes, then lanes will be numerated from 1 to 4. If the image has less that 4 lanes they will be numerated depending in which part of the image are located. Lanes 1 and 2 are always located in the left part of the image and lanes, thus, lanes 3 and 4 are in the right part of the image.

A. Data Import

The Mask R-CNN model we are using in this project has a restriction; images must have a square architecture (equal width and height). In order to work with the Culanes dataset certain changes had to be done and images were shrank to a normalized size of 256 pixels by 256 pixels. This size was chosen due to hardware restrictions and to be able to train the model in a single GPU machine. If in the future we could retrain the model in a Multi-GPU architecture we might increase this size to achieve a better performance.

Lane existence information was translated into a tensor of size [n x 4], where n is the dataset size. This information was later merged with the mask information.

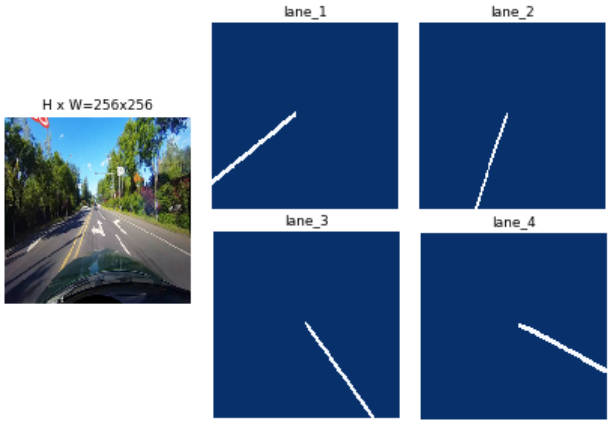

FIG. 27. Individual Lane Mask Generation process final images. In this images we can see how the four lanes are divided into 4 individual images with 1 lane per image.

In order to work with the image mask we had to transform them into tensors (multidimensional vectors). After applying the same re sizing than to original images, for each lane image (as seen in Figure 27, n images where created, one for every lane in the picture, using the previously mentioned lane information tensor. After the creation of the n images, each new image was transformed into a 256 x 256 binary vector (pixels are 1 if the lane n passes trough that pixel, 0 otherwise). Then, for each image, n [256 x 256] vectors (n could be as big as 4) were created, and attached all together. This process output a list of i tensors, i is the number of images, with each tensor of size n x 256 x 256 pixels, with n representing the number of lanes in that image. We will call this list *mask list*. In this way, the algorithm could recognize each lane in an independent way.

To sum up, the model took as input, the re-sized 256x256 pixel images and the re-sized mask tensors, called mask list, and trained the CNN to obtain an architecture that will detect in a non-trained image the location of it's lanes as seen in Figure 28.

FIG. 28. Final Boxes inference. In this image we can see the output of the model, before removing the bounding boxes and resizing the image.

V. ARCHITECTURE

The Mask R-CNN architecture needs as input a collection of different size bounding boxes and the number of bonding boxes to generate per image. In our implementation, we decided to include 50 bounding boxes of sizes [8x8, 16x16, 32x32, 64x64, 128x128] per image, taking into account that lanes will never occupy more than half of the image.

To implement this model, a Google Cloud Virtual Machine was requested. This machine had 8 CPUs with 52 GB or RAM located in a us-west1-b server. A Hard Disk of 130 GB was required to locate all the images and the frameworks required to run the model. In addition, in order to speed up the training a GPU was added to the installation. In this case we used a NVIDEA Tesla K80 with 24 GB of GDDR5 memory and 4992 NVIDIA CUDA cores.

A. Training

When training Deep Convolutions Neural Networks, a common strategy is to start the training with random weights. This strategy, while giving a lot of flexibility to the solution usually takes longer than using the final weights of that CNN that won an international competition. In this way we save time and we can take advantage of previous knowledge that the CNN obtained. Another useful idea is to fix the weights of an already trained neural network and train the weights of the last layers, where the more complex recognition is made. This is made because usually, the first layers detect layers and borders, and the last layers go to the more complex details. Using this information saves time and insures a level of quality of the architecture and therefore a better approximation to the global minimum.

Thus, for our implementation we used as starting weights the ones obtained by a ResNet trained with ImageNet dataset and fixed all layers that where not part

of the Region Proposal Network, the classifier (the fully connected layers of the network) and the mask head. After 8 Epocs- the validation loss was below 0.1 and we decided to stop the training.

Moreover, we tried training the network without any layer restrictions after 8 epocs with restrictions, and we could see that the validation set accuracy when drastically down, so we didn't follow that path. It would be interesting to free the layers one by one, one epoc at a time and have a closer look in the validation loss and accuracy.

VI. RESULTS

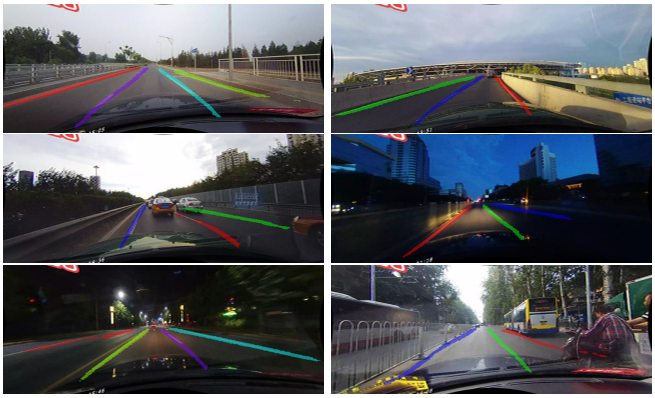

FIG. 29. Results of Culanes applying Mask R-CNN

As the creators of Mask R-CNN explain, Mask R-CNN adopts a two stage procedure, a first stage called Region Proposal Network (RPN) that proposes candidate object bonding boxes, and a second stage, in parallel to predicting the class and the box offset, which outputs the binary mask for each Region of Interest.

Therefore, during training the model has to work with different losses functions in a coordinate way. The loss function is the general loss function of the full model which includes all partial loss functions. Furthermore, this multi-task loss function combines the losses of classification (Lcls), localization (Lbox) and segmentation mask (Lmask). Where Lcls is the log loss function over the classes, as we can easily translate a multi-class classification into a binary classification by predicting a sample being a target object versus not. Lbox measures the difference between the predicted box and the bounding box of a lane. While the Lmask is defined as the average binary cross-entropy loss, only including k-th mask if the region is associated with the ground truth class k.

Moreover, The Lcls function can be divided into two parts; the loss in the first step (RPN) and the loss in the second step (Mask). The Lbox can be divided as well in two step as Lmask but Lmask is only applied in the second step. After 7 epocs the training loss when from 0.7242 to 0.3642, nevertheless the validation loss improved not so much, from 0.7366 to 0.7139.

After several epocs and when the loss function started to enter a plain, we decided to stop the training and start the inference. Figure 29 shows six images taken from the test set and with the algorithm applied. We can see how all the images has it's lanes enlightened in another colour to detonate where the lanes are. The bounding boxes are not shown to give more importance to the pixel to pixel recognition. The algorithm has very good performance in clear ambiances, in day and at night. Moreover, for lanes 2 and 3, the closest lanes to the car, the algorithm maximizes its performance. In addition, when the lanes are not very clear and it's very difficult, even for a human eye, to detect where the lanes are, the algorithm avoids making false predictions and gives a blank mask, which from our point of view is a hit.

Nevertheless, when the image has obstructions, like cars or trucks just in front of the camera, the algorithm has dissimilar results, with a very good approximation of where the lane is in cases where a part of the lane is out of sight and a wrong approach in certain others when the lane is totally covered by another object.

In terms of speed, each recognition in inference mode is made in a fraction of a second and could be improved to work in real time.

VII. CONCLUSIONS

In order to succeed in the construction of a Convolutional Neural Network (CNN) we needed to understand a big number of concepts to comprehend the hype this technology is having.

We started explaining the key characteristics of Neural Networks and how CNN uses local connectivity to improve results of recognition using the full information hidden inside a picture and not only features extracted from the data. We explained the most used layers in CNN's and how its usage affects the results. In addition several techniques to avoid over-fitting where explained, such as Data Augmentation Batch Normalization and Dropout. We also made a small introduction in optimization techniques, explaining some state of the art techniques such as Adam and Adam with restarts. Finally we gave a short explanation of the most famous and challenge winners CNN's explaining its architecture and the reasons of their success.

Furthermore, a chapter including information about Image Segmentation was included. In this chapter we covered some techniques and architectures that tackle this problem. Finally we reached the Mask R-CNN algorithm and we explained with detail how it worked and the reason of our choice based on the existence of a Mask.

After a exhaustive introduction into Deep Learning and image Segmentation we described the Culanes dataset, a large scaling dataset on traffic lane detection made in the streets of Beijing, and we showed how the importation and transformation of this dataset was made

to be useful for our project. Furthermore, a small explanation of the architecture used in Google Cloud to mount the dataset and the Mask R-CNN model was made.

In the Results chapter we showed how we modified the hyper-parameters of the model to guide the *adam* algorithm into a better minimum, and in a final objective, to achieve a better performance in detecting lanes. In addition, we showed the training and validation loss throughout the process and some images with the application of the model in non-trained observations.

Tu sum up, this project makes a good understanding of the Deep Learning problem and its application in images tackling a very difficult and problematic challenge as detecting lanes in pictures taken from inside a car using a state-of -the-art architecture called Mask R-CNN to arrive to impressive results.

For future work it would be interesting to work with Data Augmentation in the input images, unfreezing all layers from the neural network using a detailed planning, work with original images without the need of re-sizing and generating a framework that could execute the model in real time.

VIII. REFERENCES

[1]Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[2]A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[3]M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," vol. abs/1311.2901, 2013.

[4]C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[5]K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[6]K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016.

[7]R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation supplementary material."

[8]R. B. Girshick, "Fast R-CNN," 2015.

[9]S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.

[10]K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask r-cnn," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988, 2017.

[11]X. Pan, "Spatial as deep: Spatial cnn for traffic scene understanding," in *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

[12]G. E. Hinton, "Rectified linear units improve restricted boltzmann machines vinod nair,"

[13]N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[14]L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," vol. abs/1712.04621, 2017.

[15]J. Ahmad, K. Muhammad, and S. Baik, "Data augmentation-assisted deep learning of hand-drawn partially colored sketches for visual search," in *PLOS ONE*, vol. 12, p. e0183838, 08 2017.

[16]S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," pp. 448–456, 2015.

[17]H. Robbins and S. Monro, "A stochastic approximation method," vol. 22, pp. 400–407, 1951.

[18]J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," vol. 12, pp. 2121–2159, 2011.

[19]M. D. Zeiler, "Adadelta: An adaptive learning rate method," vol. abs/1212.5701, 2012.

[20]D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization.," vol. abs/1412.6980, 2014.

[21]I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," 2018.