Ain Shams University

Faculty of Engineering

Computer and Systems Engineering Department

Predicting the Driver's Focus of Attention

Project Documentation

Under Supervision of

Dr. Ashraf Salem

Project's Team

Ahmed Tarek Hussien Mohamed
Ahmed Samir Ahmed Emam
Ahmed Sherif Mahmoud Gamal ElDin
Mahmoud AbdElnaby AbdElhafeiz Elkhouly
Abdelrahman Mamdouh Khalil Al-Wali

Year (2019 – 2020)

ABSTRACT

The project goal is to predict the driver's focus of attention.
We aim to estimate what a person would pay attention to while driving, and which part of the scene around the vehicle is more critical for the driving task.
And for that, we propose a computer vision model based on a multi-branch deep architecture that integrates three sources of information: raw video (RGB frames), motion (optical flow), and scene semantics (semantic segmentation).
We also introduce DR(eye)VE, the largest dataset of driving scenes for which eye-tracking annotations are available.
This dataset includes more than 500,000 registered frames, matching ego-centric views (from glasses worn by drivers) and car-centric views (from a roof-mounted camera) and other sensors measurements.
According to the data set analysis, we found that there are a several attention patterns are shared across drivers and can be reproduced to some extent.
Which elements in the scene are likely to capture the driver's attention may benefit several applications in the context of human-vehicle interaction and driver attention analysis.

Project Repo :

Github Repo

Table of Contents

Table of Figures

Chapter 1

Introduction

Problem Definition

According to the J3016 SAE international Standard, which defined the five levels of autonomous driving, cars will provide a fully autonomous journey only at the fifth level. At lower levels of autonomy, computer vision and other sensing systems will still support humans in the driving task.

Human-centric Advanced Driver Assistance Systems (ADAS) have significantly improved safety and comfort in driving (*e.g.* Collision avoidance systems, blind spot control, lane change assistance etc.)

Among ADAS solutions, the most ambitious examples are related to monitoring systems: they parse the attention behavior of the driver together with the road scene to predict potentially unsafe manoeuvres and act on the car in order to avoid them – either by signaling the driver or braking.

However, all these approaches suffer from the complexity of capturing the true driver's attention and rely on a limited set of fixed safety-inspired rules.

Here, they in paper shift the problem from a personal level (*what the driver is looking at*) to a task-driven level (*what most drivers would look at*) introducing a computer vision model able to replicate the human attentional behavior during the driving task.

We achieve this result in two stages:

- First, they conduct a data-driven study on drivers' gaze fixations under different circumstances and scenarios. The study concludes that the semantic of the scene, the speed and bottom-up features all influence the driver's gaze.
- Second, they advocate for the existence of common gaze patterns that are shared among different drivers. They empirically demonstrate the existence of such patterns by developing a deep learning model that can profitably learn to predict where a driver would be looking at in a specific situation.

To this aim they recorded and annotated 555,000 frames (approx.6 hours) of driving sequences in different traffic and weather conditions: the DR(eye)VE dataset.
For every frame they acquired the driver's gaze through an accurate eye tracking device and registered such data to the external view recorded from a roof-mounted camera.
The DR(eye)VE data richness enables us to train an end-to-end deep network that predicts salient regions in car-centric driving videos. The network they propose is based on three branches which estimate attentional maps from
a) Visual information of the scene.
b) Motion cues (in terms of optical flow)
c) semantic segmentation. (Fig. 1)
In contrast to the majority of experiments, which are conducted in controlled laboratory settings or employ sequences of unrelated images, we train our model on data acquired on the field. Final results demonstrate the ability of the network to generalize across different day times, different weather conditions, different landscapes and different drivers.

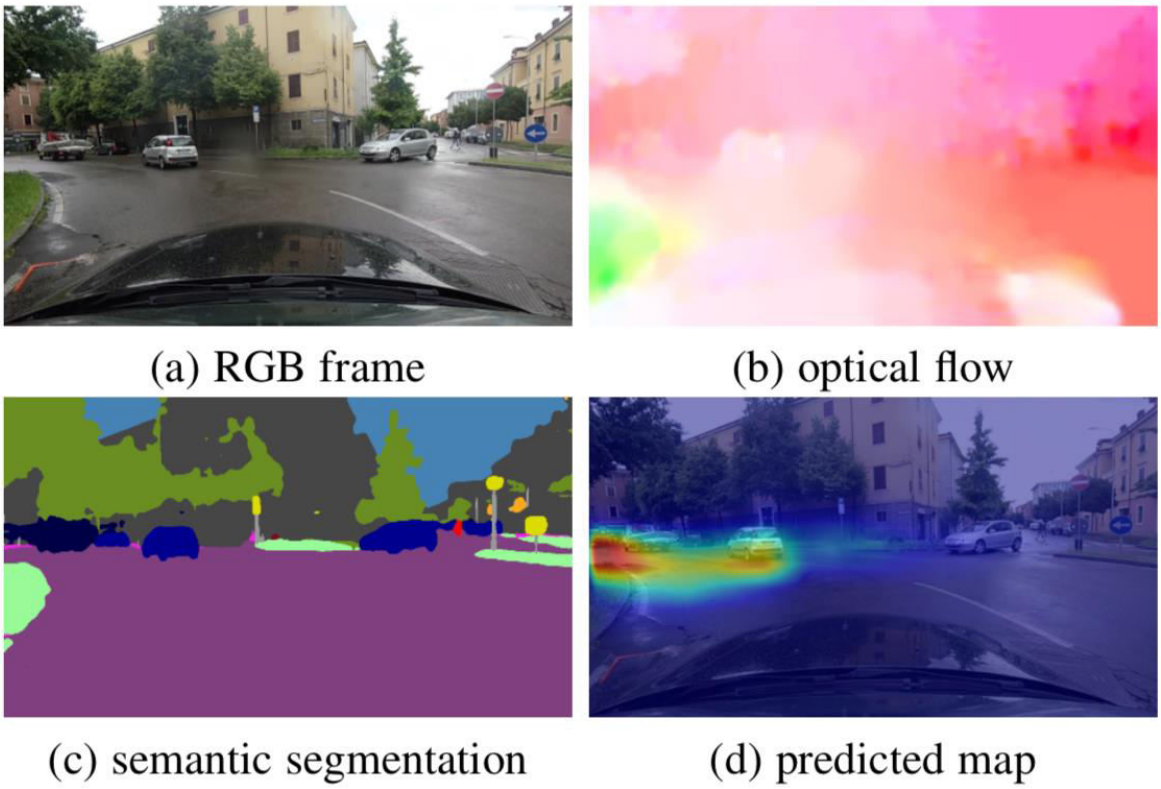


Figure 1: An example of visual attention while driving (d), estimated from our deep model using (a) raw video, (b) optical flow and (c) semantic segmentation

Project impact

Finally, we believe our work can be complementary to the current semantic segmentation and object detection literature, by providing a diverse set of information.

The act of driving combines complex attention mechanisms guided by the driver's past experience, short reactive times and strong contextual constraints. Thus, very little information is needed to drive if guided by a strong focus of attention (FoA) on a limited set of targets: our model aims at predicting them

Chapter 2

Project overview

DR(eye)VE dataset

In this section we present the DR(eye)VE dataset. The protocol adopted for video registration and annotation, the automatic processing of eye-tracker data and the analysis of the driver's behavior in different conditions.

The DR(eye)VE dataset consists of 555,000 frames divided in 74 sequences, each of which is 5 minutes long.

Eight different drivers of varying age from 20 to 40, including 7 men and a woman, took part to the driving experiment that lasted more than two months.

Videos were recorded in different contexts, both in terms of landscape (downtown, countryside, and highway) and traffic condition, ranging from traffic-free to highly cluttered scenarios.

They were recorded in diverse weather conditions (sunny, rainy, and cloudy) and at different hours of the day (both daytime and night).

In (Fig. 2), from left to right: frames from the eye tracking glasses with gaze data, from the roof-mounted camera, temporal aggregated fixation maps and overlays between frames and fixation maps.

Figure 2: Examples taken from a random sequence of DR(eye)VE.

# Videos	# Frames	Drivers	Weather conditions	Lighting	Gaze Info	Metadata	Camera Viewpoint
74	555,000	8	sunny	day	raw fixations	GPS	driver (720p)
			cloudy	evening	gaze map	car speed	car (1080p)
			rainy	night	pupil dilation	car course	

Figure 3: Summary of the DR(eye)VE dataset characteristics.

The Acquisition System

The driver's gaze information was captured using the commercial *SMI ETG 2w* Eye Tracking Glasses (ETG).

ETG capture attention dynamics also in presence of head pose changes, which occur very often during the task of driving.

While a frontal camera acquires the scene at 720p/30fps, users' pupils are tracked at 60Hz.

Gaze information are provided in terms of eye fixations and saccade movements. ETG was manually calibrated before each sequence for every driver. Simultaneously, videos from the car perspective were acquired using the *GARMIN VirbX* camera mounted on the car roof (RMC, Roof-Mounted Camera).

Such sensor captures frames at 1080p/25fps, and includes further information such as GPS data, accelerometer and gyroscope measurements.

Video-gaze registration

The dataset has been processed to move the acquired gaze from the egocentric (ETG) view to the car (RMC) view.

The latter features a much wider field of view (FoV), and can contain fixations that are out of the egocentric view. For instance, this can occur whenever the driver takes a peek at something at the border of this (FoV), but doesn't move his head.

For every sequence, the two videos were manually aligned to cope with the difference in sensors frame rate. Videos were then registered frame-by-frame through a homographic transformation that projects fixation points across views.

More formally, at each time step t the RMC frame I^t and the ETG frame I^t RMC -> ETG are registered by means of a homograph matrix HETG→RMC , computed by matching SIFT descriptors from one view to the other (Fig. 4).

A further RANSAC procedure ensures robustness to outliers. While homographic mapping is theoretically sound only across planar views - which is not the case of outdoor environments - they empirically found that projecting an object from one image to another always recovered the correct position. This makes sense if the distance between the projected object and the camera is far greater than the distance between the object and the projective plane.

Figure 4: Registration between the egocentric and roof-mounted camera views by means of SIFT descriptor matching.

Fixation map computation

The *fixation map* F_t for a frame at time t is built by accumulating projected gaze points in a temporal sliding window of k = 25 frames, centered in t

For each time step t + i in the window, where $i \in \{ -K/2, -K/2 + 1,..., K/2 - 1, K/2\}$, gaze points projections on F_t are estimated through the homograph transformation H_{t+i} that projects points from the image plane at frame t + i, namely p_{t+i}, to the image plane in F_t.

A continuous fixation map is obtained from the projected fixations by centering on each of them a multivariate Gaussian having a diagonal covariance matrix Σ (the spatial variance of each variable is set t σ_s^2 = 200 pixels) taking the *max* value along the time axis:

$$F_t(x,y) = \max_{i \in (-\frac{k}{2},\ldots,\frac{k}{2})} \mathcal{N}((x,y) \mid H^t_{t+i} \cdot p_{t+i}, \Sigma)$$

The Gaussian variance has been computed by averaging the ETG spatial acquisition errors on 20 observers looking at calibration patterns at different distances from 5 to 15 meters.

The described process can be appreciated in (Fig. 5).

Eventually, each map Ft is normalized to sum to 1, so that it can be considered a probability distribution of fixation points.

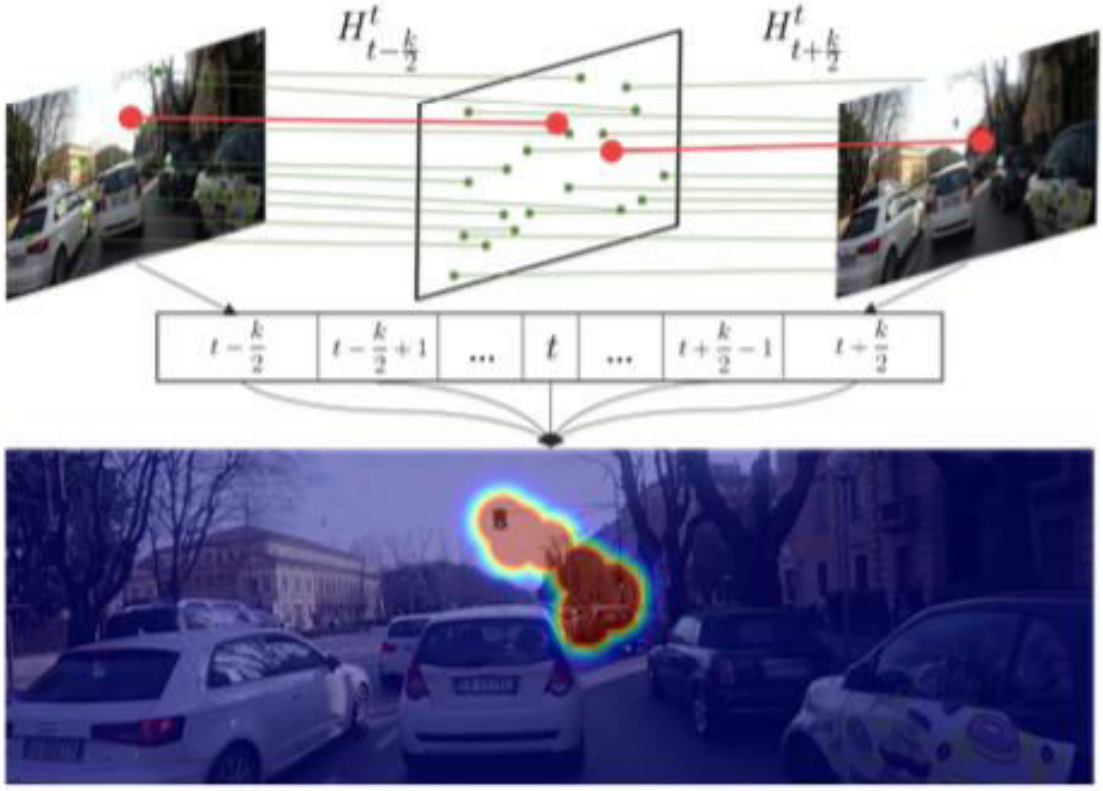


Figure 5: Resulting fixation map from a 1 second integration (25 frames).

Labeling Attention Drifts

Sometimes the drivers focus on something that has no relation to the driving task. And that leads to hard-to-predict events. For that, the dataset has additional information that helps in the dataset selection.

For each video, subsequences whose ground truth poorly correlates with the average ground truth of that sequence are selected. We compute Pearson's Correlation Coefficient (CC) and select subsequences with CC < 0.3. This happens when the attention of the driver focuses far from the vanishing point of the road. As shown in (Fig. 6). Several human selected frames and manually split them into one of these categories (a) acting, (b) inattentive, (c) errors, and (d) subjective events:

a) *Acting* subsequences are particularly interesting as the deviation of driver's attention from the common central pattern denotes an intention linked to task-specific actions (*e.g.* turning, changing lanes, overtaking ...).
b) Inattentive: occurs when the driver focuses his gaze on something unrelated to the driving task (e.g. looking at an advertisement).
c) Errors: happened due to failure in the equipment, in extreme lighting conditions or in the data processing phase (sift matching).
d) Subjective: occurs when the driver focuses on something special for the driver not common among all drivers. a road sign on the side of is so interesting element for someone that has never been on that road before.

For these reasons, subsequences of this kind will have a central role in the evaluation of predictive models in the experiment section.

In (Fig. 6), Examples of the categorization of frames where gaze is far from the mean. Overall, 108 060 frames (~20% of DR(eye)VE) were extended with this type of information.

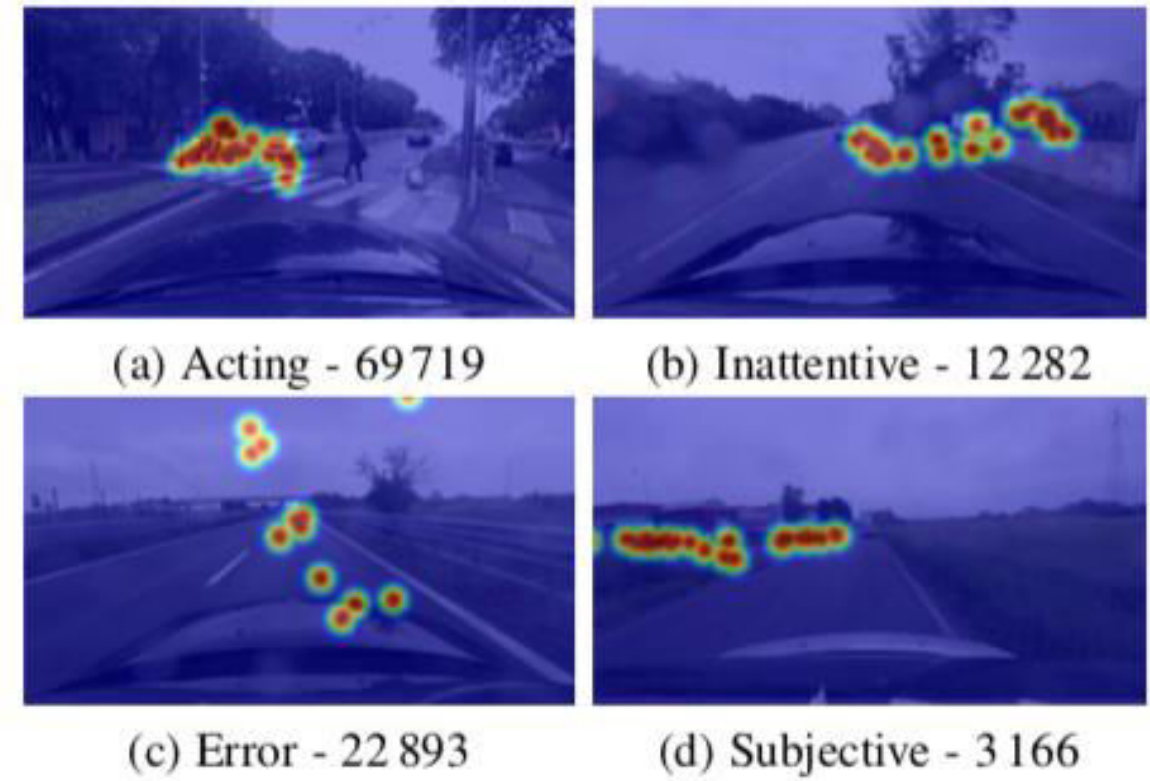


Figure 6: Categories of Driver focuses.

Dataset analysis

When they analyze the dataset they found that the drivers tend to focus on the road vanishing point as shown in (fig. 7). Also disregard the signs on the roadsides, the cars coming from the opposite way and pedestrians on sidewalks.

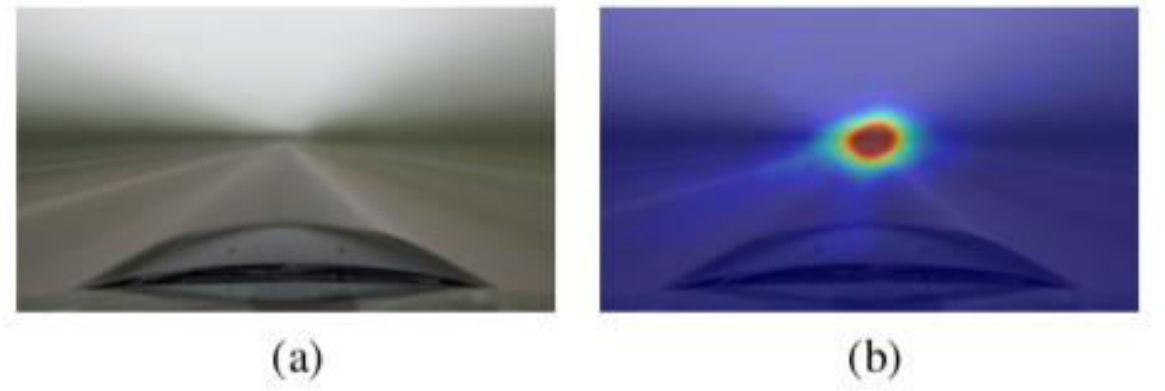


Figure 7: Mean frame (a) and fixation map (b) averaged across the whole sequence 02, highlighting the link between driver's focus and the vanishing point of the road

A driver can achieve a larger area of attention by focusing on the road's vanishing point because of the geometry of the road environment, many of the objects worth of attention are coming from there

The gaze location tends to drift from this central attractor when the context changes in terms of car speed and landscape. Indeed suggests that our brain can compensate spatially or temporally dense information by reducing the visual field size. In particular, as the car travels at higher speed the temporal density of information (the amount of information that the driver needs to elaborate per unit of time) increases as shown in (Fig. 8).

In (Fig. 8), as speed gradually increases, driver's attention converges towards the vanishing point of the road. (a) When the car is approximately stationary, the driver is distracted by many objects in the scene. From (b-e) as the speed increases, the driver's gaze deviates less and less from the vanishing point of the road. To measure this effect quantitatively, a two-dimensional Gaussian is fitted to approximate the mean map for each speed range, and the determinant of the covariance matrix Σ is reported as an indication of its spread (the determinant equals the product of eigenvalues, each of which measures the spread along a different data dimension). The bar plots illustrate the amount of downtown (red), countryside (green) and highway (blue) frames that concurred to generate the average gaze position for a specific speed range. Best viewed on screen.

The dataset shows that the driver's gaze is attracted to specific semantic categories.

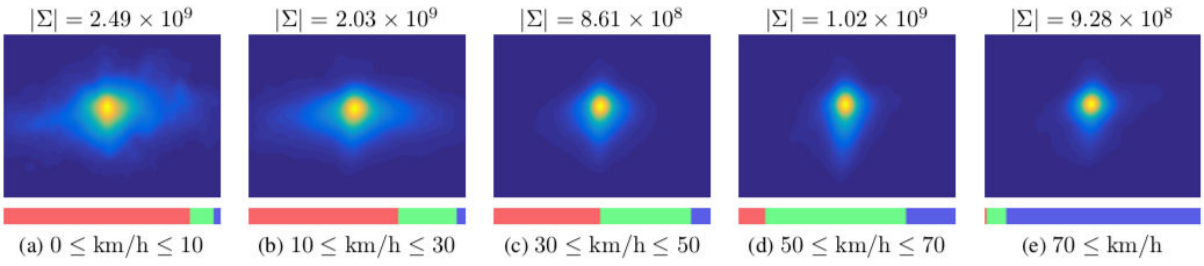


Figure 8: the amount of information that the driver needs to elaborate as speed increases.

In (Fig. 9), displays the result: for each class, the probability of a pixel to fall within the region of interest is reported for each threshold value. The figure provides insight into which categories represent the real focus of attention and which ones tend to fall inside the attention region just by proximity with the formers. Object classes that exhibit a positive trend, such as road, vehicles, and people, are the real focus of the gaze since the ratio of pixels classified accordingly increases when the observed area shrinks around the fixation point. In a broader sense, the figure suggests that despite while driving our focus is dominated by road and vehicles, we often observe specific objects categories even if they contain little information useful to drive.

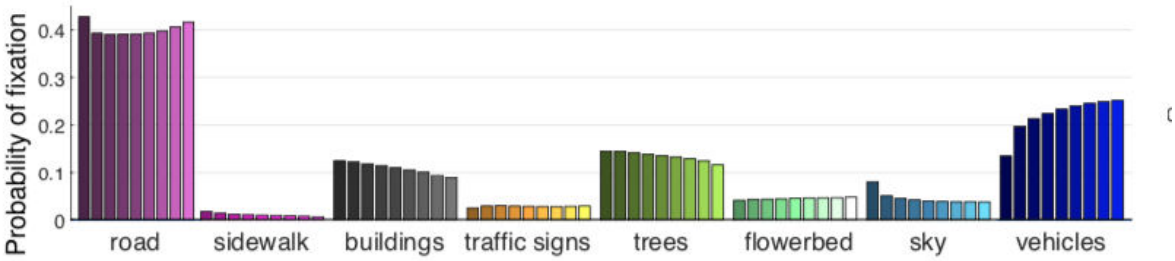


Figure 9: Proportion of semantic categories.

Chapter 3

Background

CNN

A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing images.

The pre-processing required in a ConvNet is much lower as compared to other classification algorithms.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters.

The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights.

The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image.

A Convolutional Neural Network (CNN) has three important building blocks:

1. A **convolutional layer** that extracts features from the image or parts of an image.
2. A **subsampling or pooling layer** that reduces the dimensionality of each feature to focus on the most important elements (typically there are several rounds of convolution and pooling).
3. A **fully connected layer** that takes a flattened form of the features identified in the previous layers and uses them to make a prediction about the image.

The **COARSE** module, the convolutional backbone which provides the rough estimate of the attentional map corresponding to a given clip.

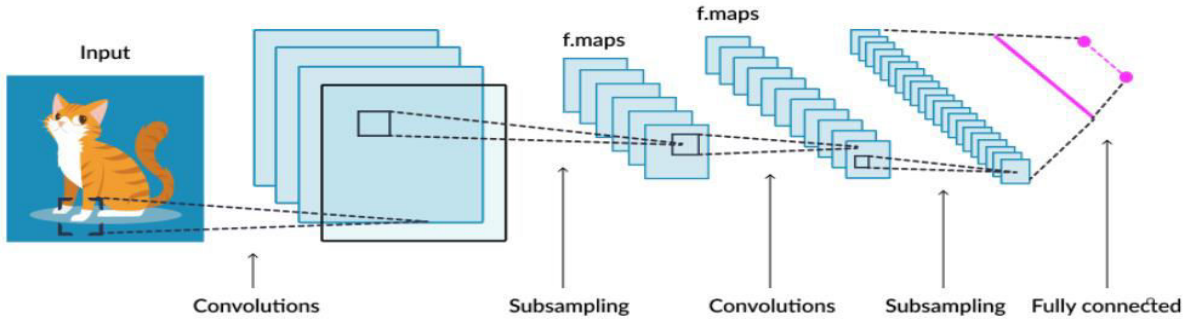


Figure 10 : CNN

The **COARSE** module consists of three important blocks:

1. Convolution Layer:
   Consists of four Conv3D blocks and one Conv2D block
2. Pooling Layer:
   Consists of three pool3D blocks
3. Up-sampling layer:
   Consists of one up2D block

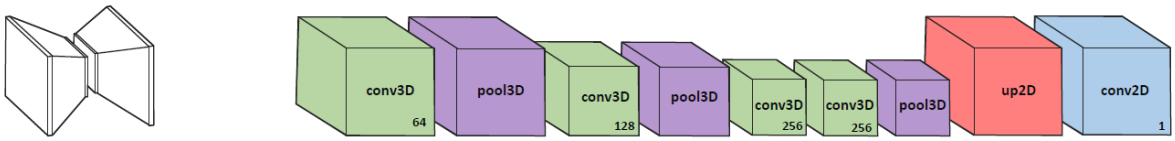


Figure 11: The COARSE module

Convolution Layer

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB).

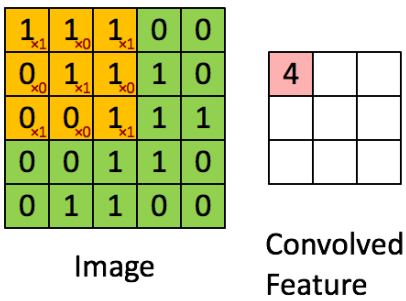


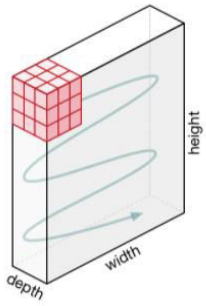
Figure 12: Convolution operation with an image

The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter.

Where Kernel =

1	0	1
0	1	0
1	0	1

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on.



Movement of the Kernel

Figure 13: movement of the kernel

The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

Padding: Pad the picture with zeros (zero-padding) so that it fits

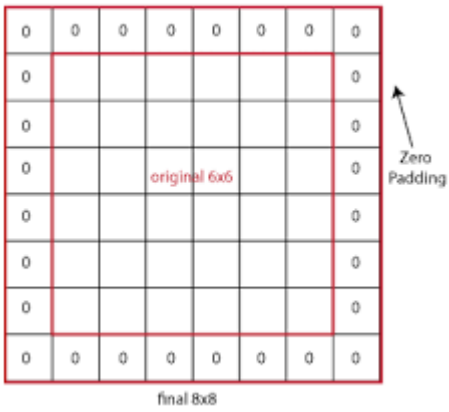


Figure 14: Padding

You can use the following equations to calculate the exact size of the convolution output (width = W , height = H , filter width = Fw , filter height = Fh):

$$\text{output width} = \frac{W - F_w + 2P}{S_w} + 1$$

$$\text{output height} = \frac{H - F_h + 2P}{S_h} + 1$$

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

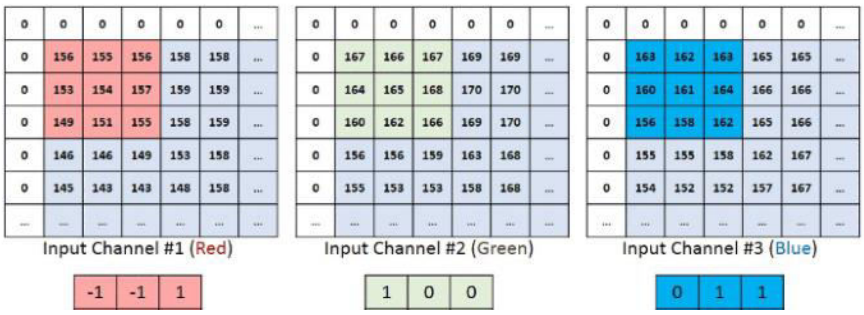
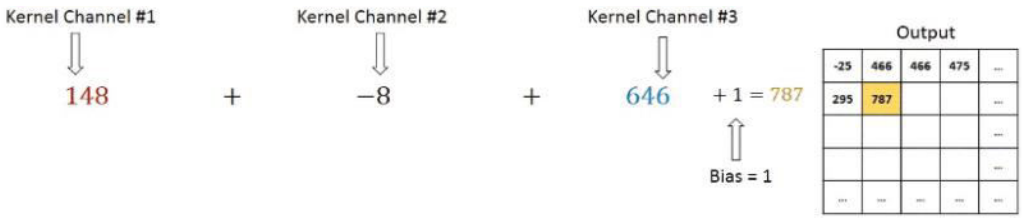


Figure 15: Convolution operation within 3 channels (RGB)



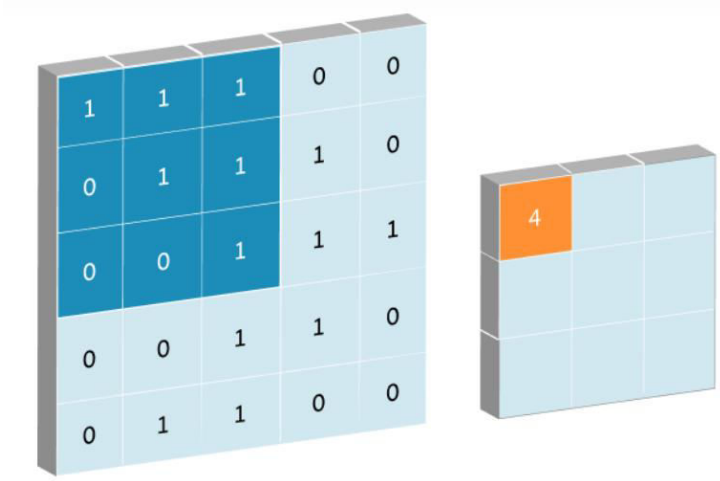
Conv2D

A 2D convolution layer means that the input of the convolution operation is three-dimensional.

This is a bit confusing, as you would expect the input to be two-dimensional. But the "2D" in "2D convolution" refers to the movement of the filter.

For example, a color image which has a value for each pixel across three layers: red, blue, and green. The filter is then run across the image three times, once for each layer.

2D convolution applied on an image will result an image.



Hence, 2D ConvNets lose temporal information of the input signal right after every convolution operation. Only 3D convolution preserves the temporal information of The input signals resulting in an output volume.

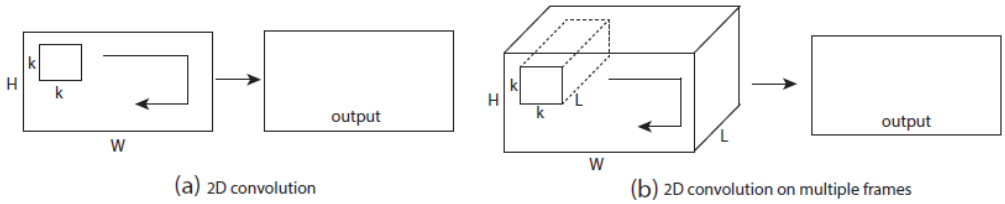


Figure 16: a) Applying 2D convolution on an image results in an image. b) Applying 2D convolution on a video volume (multiple frames as multiple channels) also results in an image.

Conv3D

3D convolution applied on an image will result a video.
In our project we use **C3D** net.

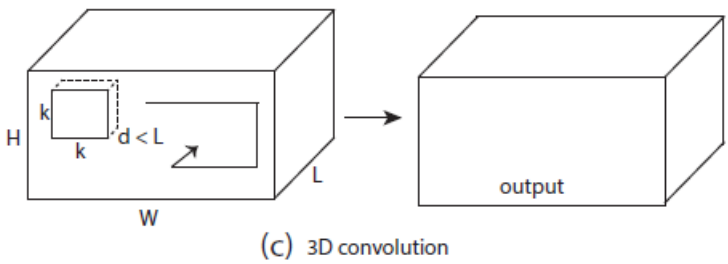


Figure 17: c) Applying 3D convolution on a video volume results in another volume, preserving temporal information of the input signal.

C3D Architecture

All of 3D convolution filters are 3 x 3 x 3 with stride 1 x1 x 1. All 3D pooling layers are 2 x 2 x 2 with stride 2 x 2 x 2 except for pool1 which has kernel size of 1 x 2 x 2 and stride 1 x 2 x 2 with the intention of preserving the temporal information in the early phase.
Each fully connected layer has 4096 output units.

3D ConvNet have 8 convolution layers, 5 pooling layers, followed by two fully connected layers, and a softmax output layer.

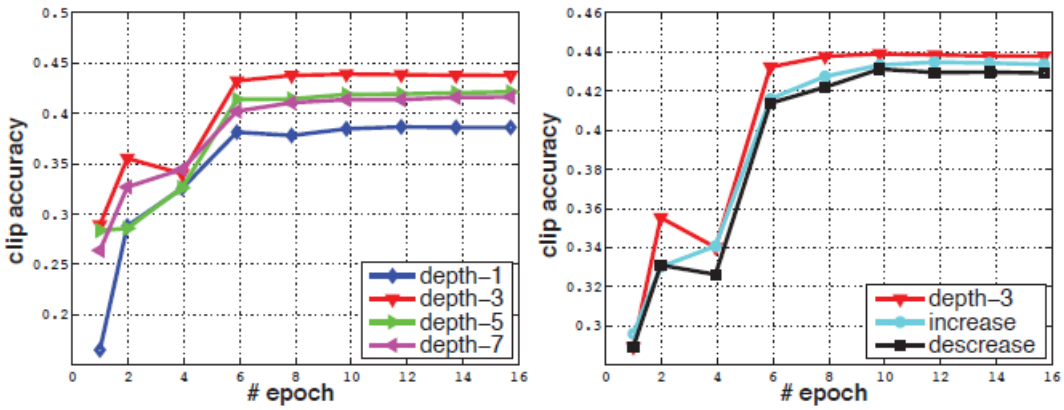


Figure 18: 3D convolution kernel temporal depth search. Action recognition clip accuracy on UCF101 test split-1 of different kernel temporal depth settings. 2D ConvNet performs worst and 3D ConvNet with 3 x 3 x 3 kernels performs best among the experimented nets.

The **COARSE** module is based on the C3D architecture that
encodes video dynamics by applying a 3D convolutional kernel
on the 4D input tensor.
As opposed to 2D convolutions that stride along the width and height dimension of the input tensor, a 3D convolution also strides along time.

Formally, the j-th feature map in the i-th layer at position (x, y) at time t is computed as:

$$v_{i,j}^{x,y,t} = b_{i,j} + \sum_{m} \sum_{p=0}^{P_{i-1}} \sum_{q=0}^{Q_{i-1}} \sum_{r=0}^{R_{i-1}} w_{i,j,m}^{p,q,r} v_{i-1,m}^{x+p,y+q,t+r}$$

where **m** indexes different input feature maps, **w** is the value
at the position **(p,q)** at time **r** of the kernel connected to the **m-th** feature map, and **Pi**, **Qi** and **Ri** are the dimensions of the kernel along width, height and temporal axis respectively; **bi,j** is the bias from **layer i to layer j**.

From **C3D**, only the most general-purpose features are retained by removing the last convolutional layer and the fully connected layers which are strongly linked to the original action recognition task.

Pooling Layer

Like the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction.
Pooling is basically “downscaling” the image obtained from the previous layers. It can be compared to shrinking an image to reduce its pixel density.

There are four types of Pooling:

1- Max pooling
2- Average pooling
3- Global Max Pooling
4- Global Average Pooling

Max Pooling

this is one of the 4 x 4 pixels feature maps from our ConvNet:

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

If we want to down sample it, we can use a pooling operation what is known as “max pooling” (more specifically, this is two-dimensional max pooling). In this pooling operation, a H×W “block” slides over the input data, where H is the height and W the

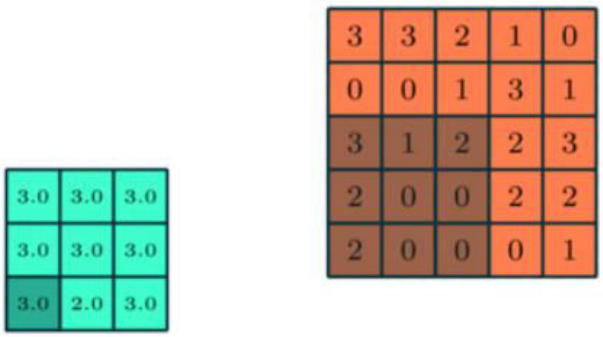


Figure 20: 3x3 pooling over 5x5 convolved feature

width of the block.

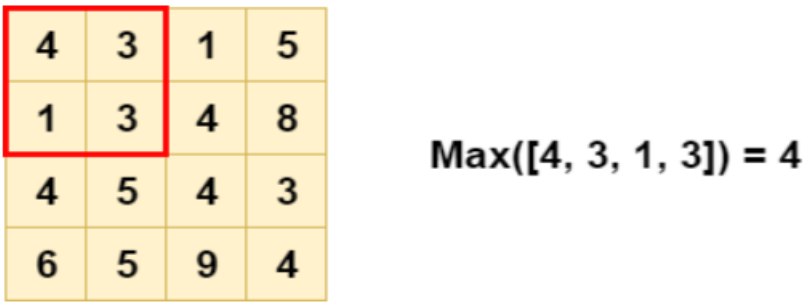


Figure 19: Max pooling over 4x4 matrix

For each block, or “pool”, the operation simply involves computing the max value, like this:

For each pool, we get a down sampled outcome, benefiting the spatial hierarchy we need:

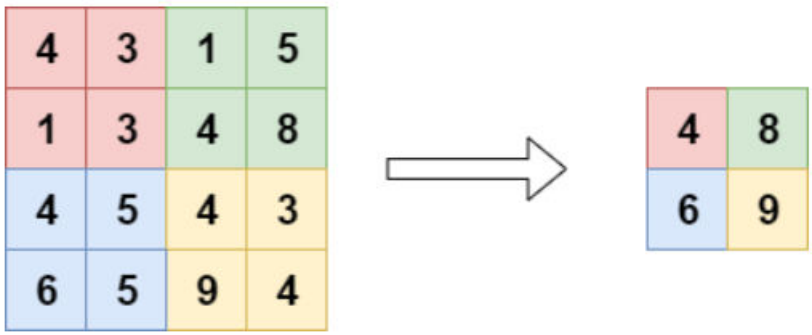


Figure 21: The result of Max pooling

Average Pooling

Another type of pooling layers is the Average Pooling layer. Here, rather than a max value, the avg for each block is computed:

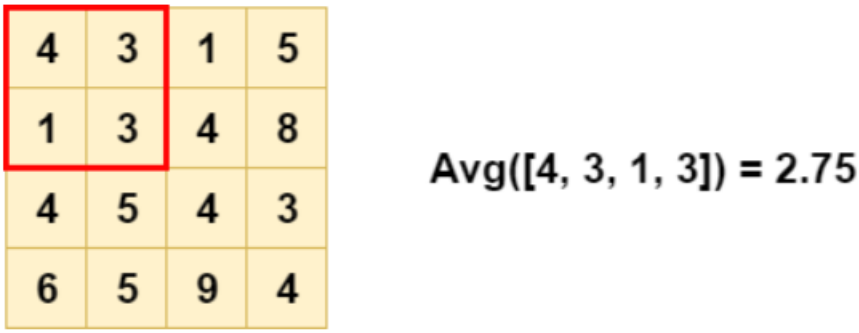


Figure 22: Average Pooling over 4x4 matrix

As you can see, the output is also different from the max pooling and less extreme than the result of max pooling

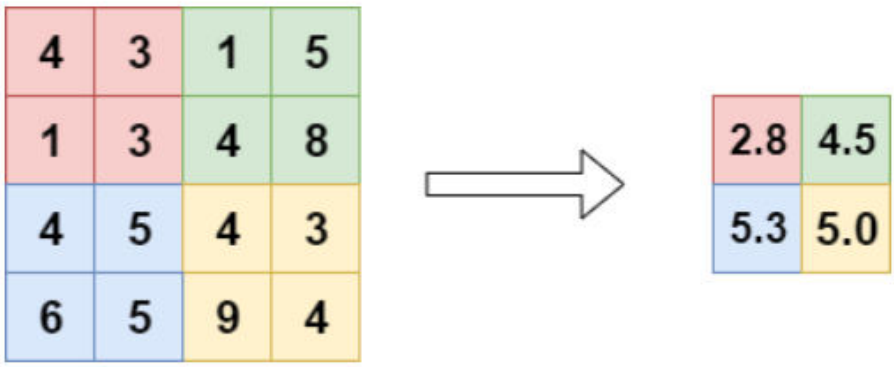


Figure 23: The result of Average Pooling

Max Pooling vs Average Pooling

If the position of objects is not important, Max Pooling seems to be the better choice. If it is, it seems that better results can be achieved with Average Pooling.

Global Max Pooling

Another type of pooling layer is the Global Max Pooling layer. Here, we set the pool size equal to the input size, so that the max of the entire input is computed as the output value

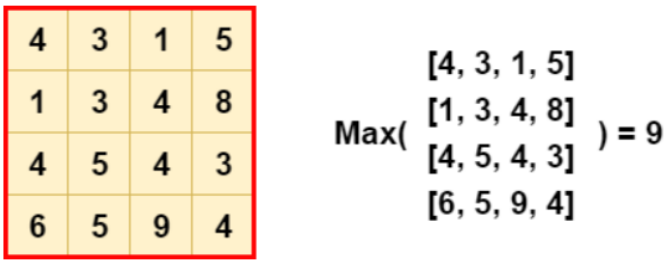


Figure 24 : Global Max Pooling

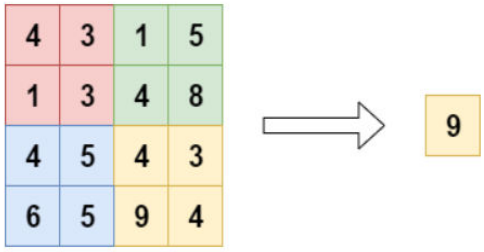


Figure 25: another Visualization on Global Max Pooling

Global pooling layers can be used in a variety of cases. Primarily, it can be used to reduce the dimensionality of the feature maps output by some convolutional layer, to replace Flattening and sometimes even dense layers in your classifier.

What is more, it can also be used for e.g. word spotting.
This is due to the property that it allows detecting noise, and thus.
However, this is also one of the downsides of Global Max Pooling, and like the regular one, we next cover Global Average Pooling.

Global Average Pooling

When applying Global Average Pooling, the pool size is still set to the size of the layer input, but rather than the maximum, the average of the pool is taken:

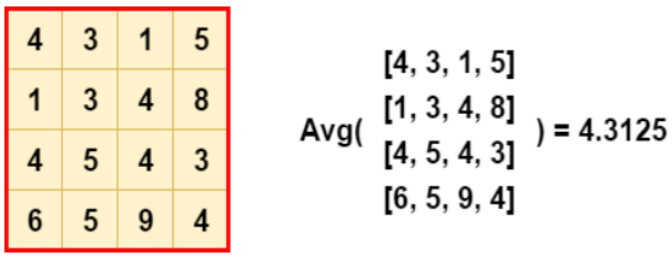


Figure 27: Global Average Pooling

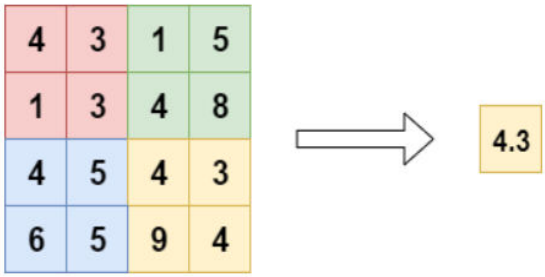


Figure 26: Another visualization on Global Average Pooling

They are often used to replace the fully connected or densely connected layers in a classifier. Instead, the model ends with a convolutional layer that generates as many feature maps as the number of target classes, and applies global average pooling to each to convert each feature map into one value.

We used **3D pooling** and modified its **size** to cover the remaining temporal dimension entirely. This collapse the tensor from 4D to 3D, making the output independent of time.
Eventually, a bilinear up sampling brings the tensor back to the input spatial resolution and a 2D convolution merges all features into one channel.

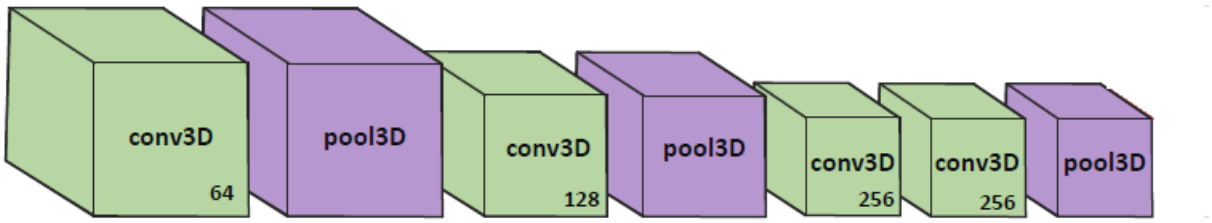


Figure 28: 3D pooling Blocks down-samples the output of Conv3D

Up-Sampling Layer

The goal of **up-sampling** operations is to increase the resolution from low-resolution map data and to **up-sample** original data to a high-resolution map, aiming to make up for the information lost in previous **layers** because of convolution operations.
Also, it is a layer that simply doubles the dimensions of the input.

UpSampling2D

UpSampling2D is just a simple scaling up of the image by using nearest neighbor or bilinear up-sampling, so nothing smart.
Advantage is it is cheap.

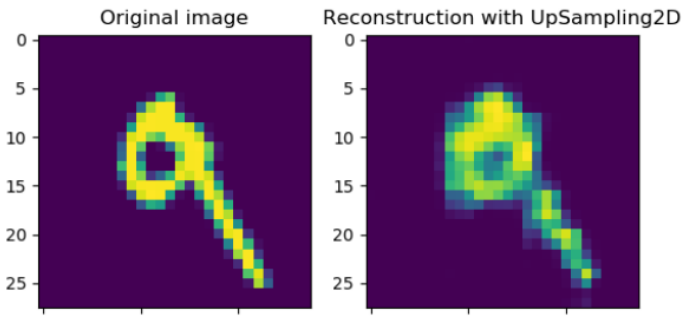


Figure 29: the result of Up-Sampling.

Chapter 3

Multi-branch deep architecture

Design architecture

According to the video analysis like video classification and video segmentation they found that a sequence of frames as input give better results than a single frame because of the time dependencies where we can't decide which part of the scene more critical for the driving task from one frame so that we need to handle the temporal dependencies. There is two approaches for that goal:

- First using a 3D convolution (short-range correlation)
- Second by recurrent architectures like LSTM and GRU (long-range correlation).

Our model follows the first approach, according to the assumption that a small time window (half a second) contains sufficient contextual information for predicting where the driver would focus at that moment.

Indeed, human drivers can take even less time to react to an unexpected stimulus. Our architecture takes a sequence of 16 consecutive frames (≈ 0.65s) as input (called clips) and predicts the fixation map for the last frame of such a clip. According to the dataset analysis; we ended up the following results:

- The drivers' (FoA) shows consistent patterns, and it can be reproduced by a computational model.
- The drivers' focus is affected by objects semantics, e.g. drivers tend to focus on items lying on the road; motion cues, like vehicle speed, are also key factors that influence the driver's focus.

Finally, the model output merges three branches with the same architecture, unshared parameters, and different input domains:

The RGB image, the semantic segmentation, and the optical flow field. We call this architecture a multi-branch model. Following a bottom-up approach.

Optical Flow

Introduction

Estimation of optical flow is one of the key problems in computer vision. Optical flow is the apparent motion of objects or surfaces or pixels between two frames in a sequence. It is relevant in applications such as surveillance and tracking. In the last two decades the quality of optical flow estimation methods has increased dramatically. Starting from the original approaches of Horn and Schunk as well as Lucas and Kanade, significant improvement has been achieved in the estimation of optical flow, with the best published results provided by Brox et al

The reason optical flow is the apparent motion can be shown in figure $.1 as it shows the direction of the motion of the cube from the image on the left to the image on the right but it does not show arrows on the white side of the rotating table as its motion cannot be seen because there is no contrast or texture so there is no indication that the pattern is moving.

Figure 30: Example of optical flow for a picture of a Rubik's cube on a rotating table rotating table.

Sparse optical flow

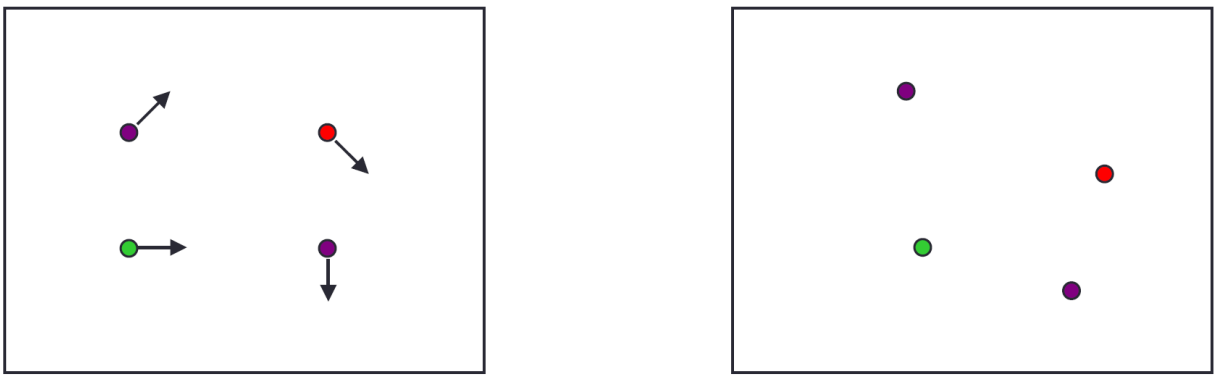
Sparse optical flow selects a feature set of pixels e.g. interesting features such as edges and corners to track its velocity vectors. The extracted features are passed in the optical flow function from frame to frame to ensure that the same points are being tracked

Dense optical flow

Dense optical flow attempts to compute the optical flow vector for every pixel of each frame. While such computation may be slower, it gives a more accurate result

Figure 31: Sparse vs. Dense optical flow.

Problem Definition



$I(x, y, t)$ $I(x, y, t+1)$

Figure 32: two images of the same scene at different times.

We are given two images of the same scene at different times, possibly from different perspectives, and the goal is to estimate a motion vector (u,v) at every point (x , y) such that I1(x , y) and I2(x +u , y +v) correspond. First, we assume that the two images are samples of a function I of both space and time, so that I1(x,y) = I (x , y , t) and I2(x , y) = I (x , y , t +1) for some time t.

Brightness constancy assumption

The key assumption of optical flow is that image intensities of corresponding points are preserved over time; that is,

$$\mathrm{I}\,(\mathrm{x}+\mathrm{u},\mathrm{y}+\mathrm{v},\mathrm{t}+1)=\mathrm{I}\,(\mathrm{x},\mathrm{y},\mathrm{t}) \qquad (1)$$

The brightness constancy assumption is sensitive to noise in the images. It is customary to accommodate for this by pre-blurring the image or equivalently by using weighted windows around each pixel. This Equation actually combines

several assumptions, including that the surfaces in the scene have the same brightness regardless of viewing direction, that the illumination of the scene doesn't change, and that the image formation process is ideal e.g., there is no darkening of the image toward its edges.

If we perform a Taylor expansion of Equation (1) around (x, y) assuming that u and v are small and by neglecting higher order derivatives, we obtain

$$\mathrm{I}\,(\mathrm{x}+\mathrm{u},\mathrm{y}\,+\,\mathrm{v},\mathrm{t}\,+\,1)=\frac{\partial I}{\partial x}u+\frac{\partial I}{\partial y}v+\,\mathrm{I}\,(\mathrm{x},\mathrm{y}\,,\mathrm{t}\,+\,1) \qquad (2)$$

By combining equation (1) and equation (2) we get

$$[\mathrm{I}\,(\mathrm{x},\mathrm{y}\,,\mathrm{t}\,+\,1)-\,\mathrm{I}\,(\mathrm{x},\mathrm{y}\,,\mathrm{t})]+\frac{\partial I}{\partial x}u+\frac{\partial I}{\partial y}v=0$$

Therefore, we get:

$$\frac{\partial I}{\partial x}u+\frac{\partial I}{\partial y}v+\frac{\partial I}{\partial t}=0$$

Another form for this equation is

$$\nabla I\,.\,\begin{bmatrix}u\\v\end{bmatrix}=\frac{-\partial I}{\partial t}$$

Without any other information, this is the only part of the optical flow field that can be obtained. This makes sense because the problem is inherently under constrained: we want to estimate two unknowns (u and v) at each pixel, but only have one equation Therefore, we must make additional assumptions to resolve the remaining degree of freedom at each pixel. So far, the model estimates the displacement of a pixel only locally without taking any interaction between neighboring pixels into account. Therefore, it runs into problems as soon as the gradient vanishes somewhere, or if only the flow in normal direction to the gradient can be estimated (aperture problem).

Aperture problem

Since that at each pixel, we have one equation, two unknowns. This means that only the flow component in the gradient direction can be determined. This can be shown at (Fig. 13) We can see that at the left image without the aperture the bode shown moves down and to the right but when adding the aperture in the right image the same body motion seems that it is an upward motion

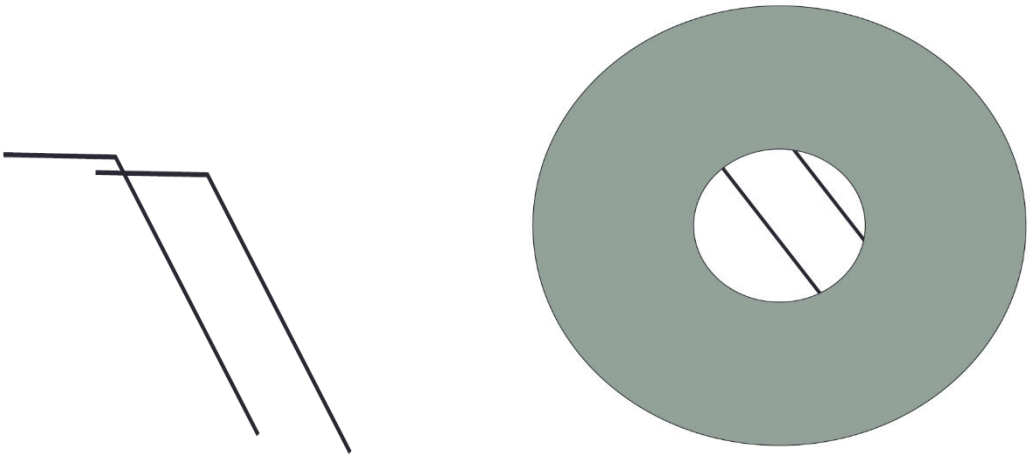


Figure 33: Aperture problem.

Smooth Optical Flow (Horn and Schunck)

The most natural assumption is that the optical flow field varies smoothly across the image; that is, neighboring pixels should have similar flow vectors. Horn and Schunck phrased this constraint by requiring that the gradient magnitude of the flow field, namely the quantity should be very small.

$$(\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial v}{\partial y})^2$$

We here assume that the motion vector (u, v) from one point to the next should change slowly we would like this equation to be zero which means all the motion vectors are constant so the derivatives can be zero this assumes that the whole image was moving in the same direction all other values of (u, v) will cause the quantity to have some value and we want to minimize this.

This leads to an energy function to be minimized over the flow fields u(x,y) and v(x,y)

$$E_{smoothness} = (\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial v}{\partial y})^2$$

By combining the two constrains we get an overall energy function to be minimized over the flow fields u(x, y) and v(x, y)

$$E_{SH} = \sum_{x,y} (\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t})^2 + \gamma((\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial v}{\partial y})^2)$$

$$= E_{data} + \gamma\, E_{smoothness}$$

Where γ is a parameter that specifies the influence of the smoothness term, also known as a **regularization** parameter. The larger the value of γ, the smoother the optical flow field. A large weight on the regularizer (which depends on the domain and range of *I*) is often used to enforce a smooth flow field minimizing a function can be accomplished by solving the Euler-Lagrange equations, which in this case are:

$$\gamma\, \nabla^2 u = \left(\frac{\partial I}{\partial x}\right)^2 u + \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} v + \frac{\partial I}{\partial x}\frac{\partial I}{\partial t}$$

$$\gamma\, \nabla^2 u = \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} u + \left(\frac{\partial I}{\partial y}\right)^2 v + \frac{\partial I}{\partial y}\frac{\partial I}{\partial t}$$

Evaluated simultaneously for all the pixels in the image, the partial derivatives of the spatiotemporal function *I* are approximated using finite differences between the two given images. That is, at pixel *(x, y)*

$$\frac{\partial I}{\partial x} \approx \frac{1}{4}\,(\, I_1\,(x+1, y) - I_1(x, y) + I_1(x+1, y+1) - I_1(x, y+1)$$
$$+ I_2\,(x+1, y) - I_2(x, y) + I_2(x+1, y+1) - I_2(x, y+1))$$

$$\frac{\partial I}{\partial y} \approx \frac{1}{4}\,(\, I_1\,(x, y+1) - I_1(x, y) + I_1(x+1, y+1) - I_1(x+1, y)$$
$$+ I_2\,(x, y+1) - I_2(x, y) + I_2(x+1, y+1) - I_2(x+1, y))$$

$$\frac{\partial I}{\partial t} \approx \frac{1}{4}\,(I_2(x,y) - I_1(x,y) + I_2(x+1,y) - I_1(x+1,y)$$
$$+I_2(x,y+1) - I_1(x,y+1) + I_2(x+1,y+1) - I_1(x+1,y+1))$$

The drawback of the Horn and Schunck method is that the method was mot very clear on how nearby pixels are related to each other except through the smoothness term

The Lucas-Kanade Method

The Lucas-Kanade optical flow algorithm is a simple technique which can provide an estimate of the movement of interesting features in successive images of a scene. We would like to associate a movement vector (u, v) to every such interesting pixel in the scene, obtained by comparing the two consecutive images we pretend that the pixel's neighbors within a window w*w have the same motion vector (u, v)

Since

$$E_{LK}\,(u,v) = \sum_{x,y} w(x,y)(I\,(x\ +u, y\ +v, t\ +1) - I\,(x,y,t))2$$

Therefore, within the window

$$\nabla I\,.\ \begin{bmatrix} u \\ v \end{bmatrix} = \frac{-\partial I}{\partial t}$$

Let this equation be equivalent to A*d = b let's assume that he window is 5*5 So we will get 25 equation for only two unknowns we can solve this by minimizing the least squares error in the problem

$$(A^T A)\ d = \ A^T b$$

So we get

$$\begin{bmatrix} \sum_w (\frac{\partial I}{\partial x}) & \sum_w (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ \sum_w (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & \sum_w (\frac{\partial I}{\partial y}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum_w (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial t}) \\ -\sum_w (\frac{\partial I}{\partial y})(\frac{\partial I}{\partial t}) \end{bmatrix}$$

This function is solvable only if

- $A^T A$ should be invertible
- $A^T A$ should not be too small due to noise
    - eigenvalues λ1 and λ2 of $A^T A$ should not be too small
- $A^T A$ should be well-conditioned
    - λ1/ λ2 should not be too large (λ1 = larger eigenvalue)

The eigenvalues of the matrix on the left-hand side of Equation — called the Harris matrix — used as the basis for detecting good feature locations. Windows where one of the eigenvalues is small correspond to flat or edge like regions that suffer from the aperture problem and for which a reliable optical flow vector can't be estimated. Thus, the only way to use Equation to obtain across the whole image is to make the support of the window large enough to ensure both eigenvalues are far from zero. Using large windows may not do a very good job of estimating flow, large square windows are unlikely to remain square in the presence of camera motion and also then the smoothness assumption will not be valid.

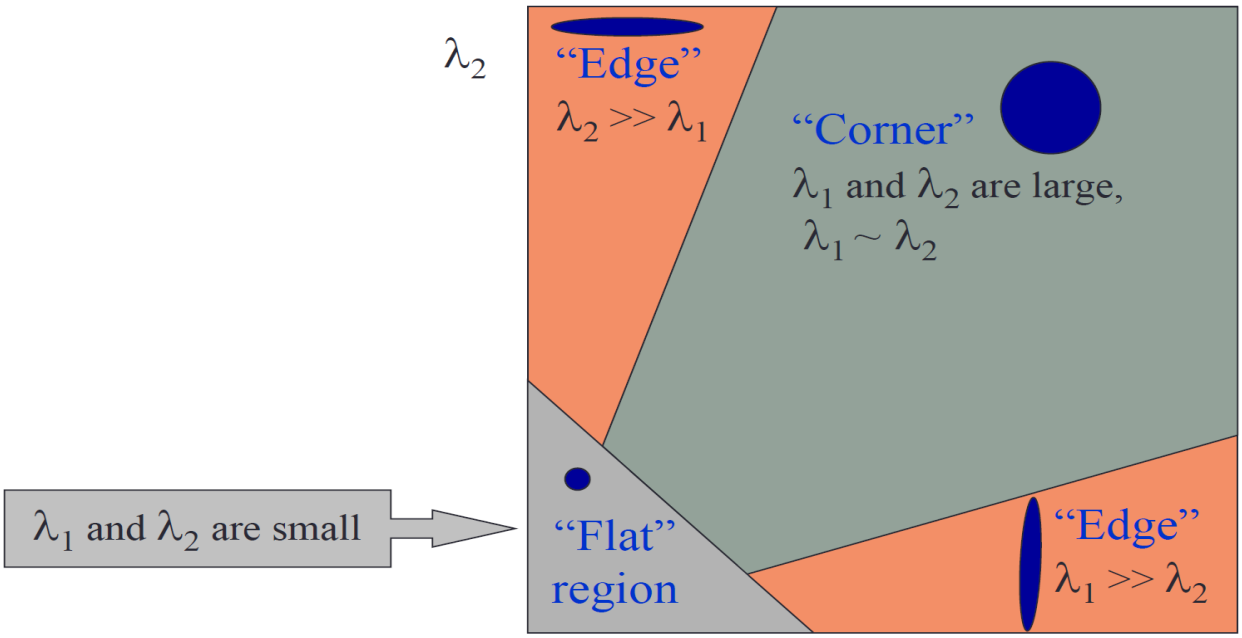


Figure 34: Classification of images according to the Eigen values of the matrix$A^T A$.

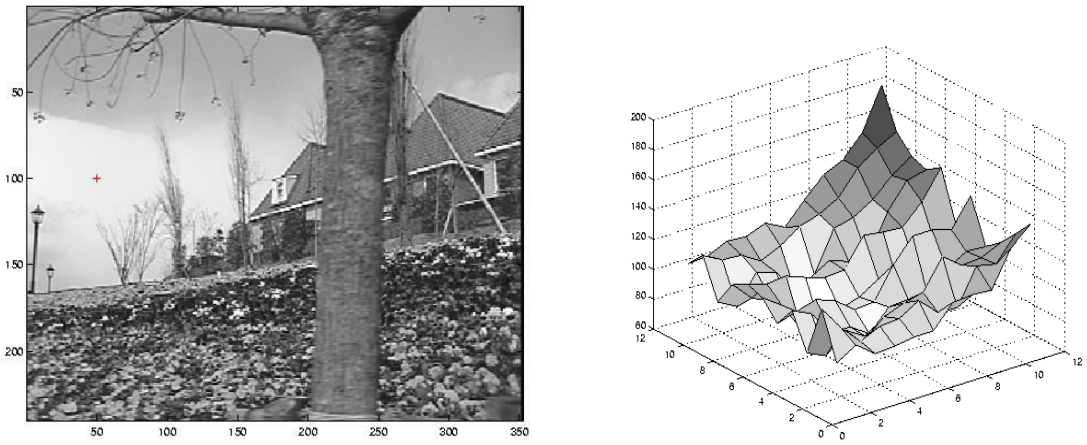


Figure 35 Example of image point with gradients have small magnitude smallλ1, small λ2.

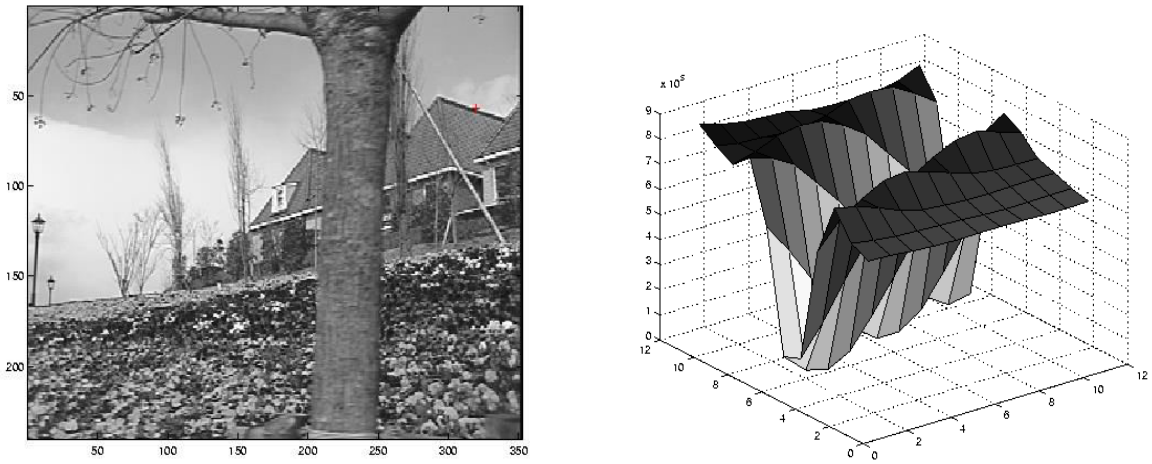


Figure 36: Example of image point with similar and large gradients largeλ1, small λ2.

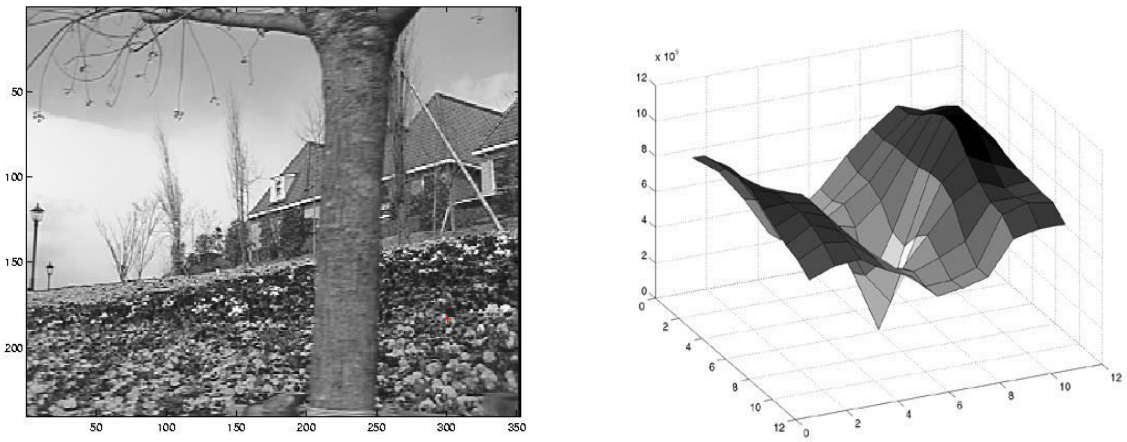


Figure 37: Example of image point gradients are different, large magnitudes large λ1, large λ2.

On the other hand, the Lucas-Kanade algorithm is local, in that the flow vector can be computed at each pixel independently, while the Horn-Schunck algorithm is global, since all the flow vectors depend on each other. This makes the Lucas-Kanade problem computationally easier to solve. We have seen that both local and global differential methods have complementary advantages and shortcomings. Hence it would be interesting to construct a hybrid technique that constitutes the best of two worlds: It should combine the robustness of local methods with the density of global approaches.

Multi-channel and Lucas-Kanade

The same equations are applied for RGB images the difference is that we will have the constraints for every layer. So, if we have a 5*5 window we will have 25 * 3 equations for every pixel.

$$\nabla I[0,1,2] \; . \; \begin{bmatrix} u \\ v \end{bmatrix} = \frac{-\partial I[0,1,2]}{\partial t}$$

Gradient constancy assumption

Original Horn-Schunck data term encapsulates the assumption that pixel intensities don't change over time, which isn't realistic for many real-world scenes, so the gradient constancy assumption was introduced, according to which the gradient of the grey level of objects is constant in consecutive frames. This assumption accommodates for slight changes in the illumination of the scene.

$$\nabla I\ (x\ + u, y\ + v, t\ + 1) = \nabla I\ (x, y, t)$$

Where ∇ is the spatial gradient. This allows some local variation to illumination changes; consequently a modified data term reflecting both brightness and gradient constancy will be:

$$E_{data}(u,v) = \sum_{x,y} (I2(x\ + u, y\ + v) - I1(x,y)\ )^2 + \gamma(\|\ \nabla I2(x\ + u, y\ + v) - \nabla I1(x,y)\ \|)_2^2$$

Where γ weights the contribution of the gradient terms.

Combining Lucas-Kanade and Horn-Schunck

Combining these two methods will give a technique that will give us the best of each a technique we will use the data error part from the Lucas-Kanade and the smoothness part from Horn-Schunck

$$E_{data\ LK} = \sum_{w} (I2(x\ + u, y\ + v) - I1(x, y)\ )^2 + \gamma(\|\ \nabla I2(x\ + u, y\ + v) - \nabla I1(x, y)\ \|)_2^2$$

$$E_{smoothness\ SH} = ((\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial v}{\partial y})^2)$$

Therefore

$$E = E_{data\ LK} + \beta\ E_{smoothness\ SH}$$

Robust Cost Functions

The previous technique penalized deviation from the brightness constancy and smoothness assumptions using quadratic functions in the respective terms. However, it's well known that quadratic functions are extremely sensitive to noise; for example, an incorrect estimate in one of the gradient computations can have a disproportionate effect on the optical flow field, pulling the solution far away from the correct answer.

The idea is to replace the quadratic function in the data term with the function

$$E = \sum_{w} \Psi(I2(x\ + u, y\ + v) - I1(x, y)\ )^2 + \gamma\ \Psi\ (\|\ \nabla I2(x\ + u, y\ + v) - \nabla I1(x, y)\ \|)_2^2 + \Phi((\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial v}{\partial y})^2)$$

Where we assume $\Psi\ and\ \Phi$ are robust functions if the values of the inputs of the robust function are expected to contain outliers then we want to choose $\Psi\ and\ \Phi$ to reduce the weight of these outliers.

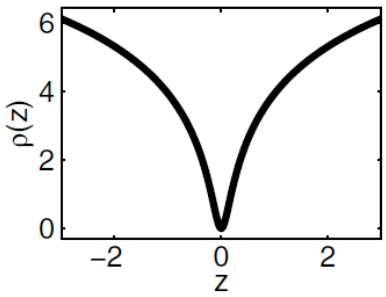
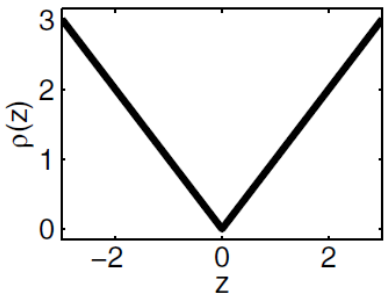
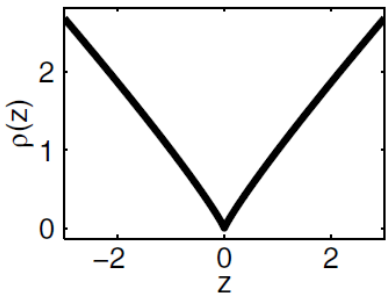
Name	Definition	$\rho(z)$
Lorentzian	$\rho(z) = \log\left(1 + \frac{1}{2}\left(\frac{z}{\beta}\right)^2\right)$	
Charbonnier	$\rho(z) = \sqrt{z^2 + \varepsilon^2}$	
Generalized Charbonnier	$\rho(z) = (z^2 + \varepsilon^2)^\beta$	

Figure 38: Examples of Robust Functions.

Image pyramid

Sometimes, we need to work with images of different resolution of the same image. For example, while searching for something in an image, like face, we are not sure at what size the object will be present in the image. In that case, we will need to create a set of images with different resolution and search for object in all the images. These set of images with different resolution are called Image Pyramids in which a signal or an image is subject to repeated smoothing and subsampling

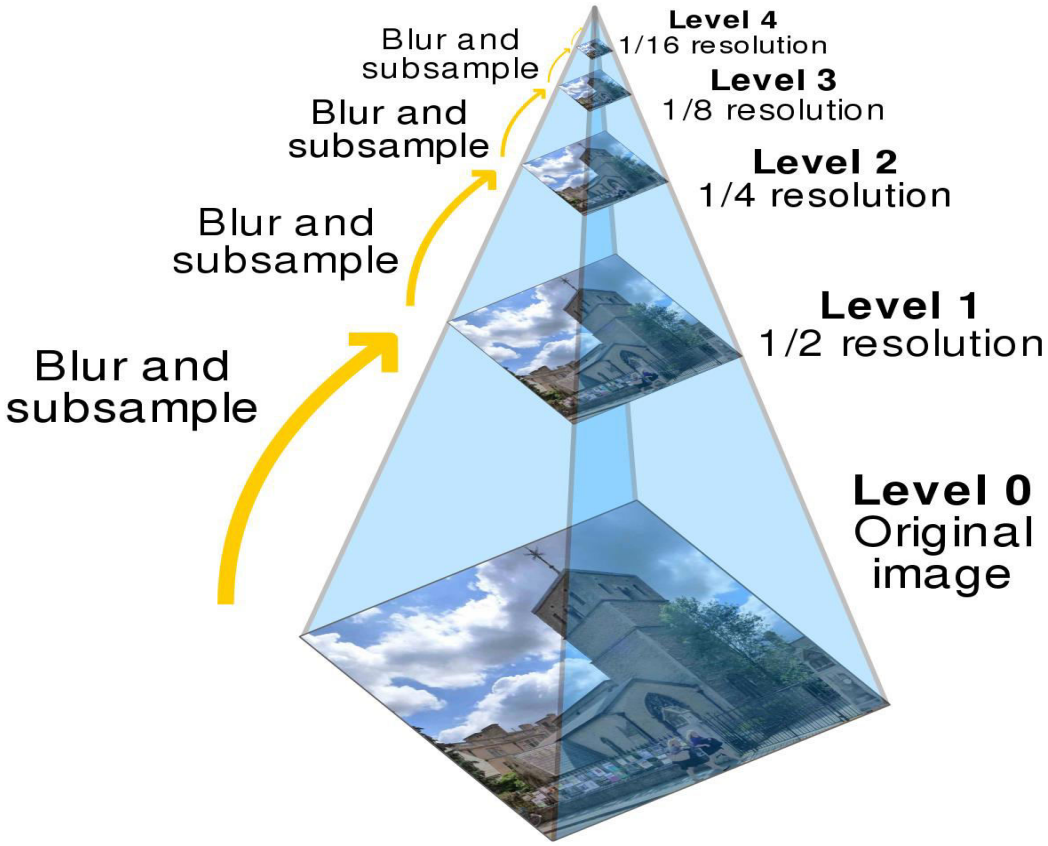


Figure 39: Image pyramid

Coarse to Fine Algorithm

The multi-scale coarse-to-fine approach is used by most modern algorithms for optical flow estimation, in order to support large motion and for improved accuracy. This approach relies on estimating the flow in an image pyramid, where the apex is the original image at a coarse scale, and the levels beneath it are warped representations of the images based on the flow estimated at the preceding scale. The most important benefit of this algorithm is that it solves the limitation that (u,v) must be small to make the tailor expansion valid as if the motion vector at the finest image is large it will be very small at the coarsest image so there will be no problem calculating the optical flow at the coarsest images and from it we can continue the algorithm iteratively to compute the optical flow at the finest Image.

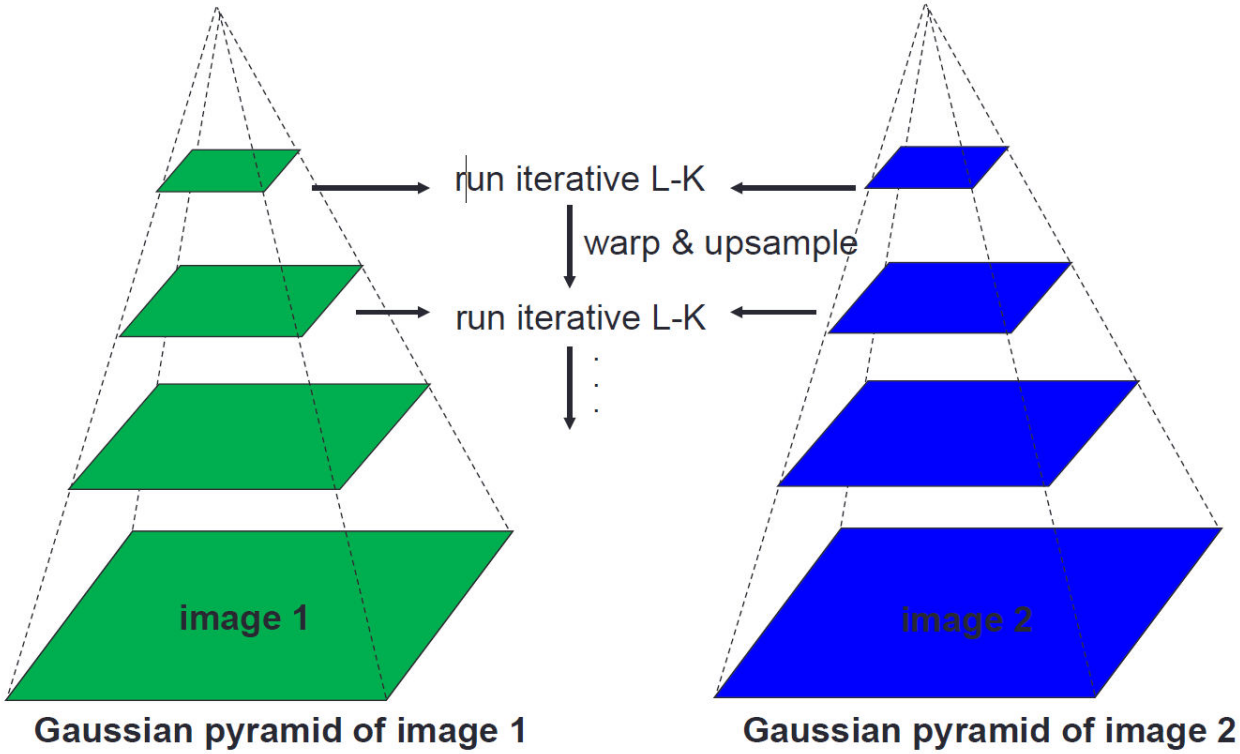


Figure 40: Coarse to fine algorithm.

The coarsest level should be chosen so that the expected (u, v) is on the order of a pixel at that level. Now we consider the next higher level of the pyramid. We can't simply multiply the estimated (u, v) at each pixel by two and use these as initial guesses for the flow field at the next level, since the Taylor series expansion at this higher level won't be valid for vectors with these magnitudes. Instead, we warp the second image plane toward the first using the estimated flow, so that the flow between the first image and the warped second image is again expected to be nearly zero. We continue this process as we go up the pyramid, computing the final optical flow vectors as the sum of several small increments at each level.

To summarize the algorithm

1. Create the Gaussian pyramids for I1 and I2, indexed from 0 to N, where N is the coarsest level of the pyramid. Initialize the working level n to N and let ˜I1 and ˜I2 be the images at the coarsest level of the pyramid, i.e., ˜I1 = $I1^N$, ˜I2 = $I2^N$.

2. Estimate the optical flow field $(u, v)^n$ between ˜I1 and ˜I2 at level n.

3. Construct the optical flow field $(u, v)^{warp}$ as $\sum_{k=n}^{N} 2^{k-n+1} (u, v)^k$ that is, the sum of all the incremental motion fields so far, at the scale of level n−1. Since this generates motion vector estimates only for even x and y, use bilinear interpolation to fill in motion vectors for the whole image.

4. Warp the pyramid image for I2 at level n −1 using $(u, v)^{warp}$ to create a new image ˜I2. That is, ˜I2(x, y) = $I2^{n-1}(x + u^{warp}, y + v^{warp})$. Use bilinear interpolation to sample pixels from $I2^{n-1}$. Let ˜I1 = $I1^{n-1}$.

5. Let n = n−1.

6. If n ≥ 0, go to Step 2. Otherwise, the final optical flow field is (u, v) = $\sum_{k=0}^{N} 2^k (u, v)^k$

Semantic segmentation

It's considered a dense classification process at which we aim to classify each pixel in our label (i.e. give each pixel the corresponding class label). This process clusters the image generally on pixel-level and assigns one of the predefined classes to each pixel. This is in stark contrast to classification, where a single label is assigned to the entire picture.

Figure 41: Semantic Segmentation Example.

Past and Modern Implementations

Before 2000, we used several methods in digital image processing: threshold segmentation, region segmentation, edge segmentation, texture features, clustering and so on.

From 2000 to 2010, there are four main methods: graph theory, clustering, classification and combination of clustering and classification.

One of the classical implementations of semantic segmentation is gray level

Segmentation which is the simplest form of semantic segmentation; it involves assigning hard-coded rules or properties a region must satisfy for it to be assigned a particular label. The rules can be framed in terms of the pixel's properties such as its gray level intensity.
One such method that uses this technique is the Split and Merge algorithm. This algorithm recursively splits an image into sub-regions until a label can be assigned, and then combines adjacent sub-regions with the same label by merging them.
The problem with this method is that rules must be hard-coded. Moreover, it is extremely difficult to represent complex classes such as humans with just gray level information. Hence, feature extraction and optimization techniques are needed to properly learn the representations required for such complex classes.

Semantic segmentation has seen considerable progress due to the employment of deep learning architectures, especially convolutional neural networks (CNN).
This progress has resulted in a much better quality of real-world applications, such as autonomous driving, medical diagnosis, and aerial image segmentation.
These applications tend to rely on real-time processing with high-resolution inputs, which is the main obstacle of most modern semantic segmentation networks. For autonomous driving –which is our interest; we need to equip cars with the necessary perception to understand their environment so that self-driving (driverless) cars can safely integrate into our existing roads.

Different Approaches for Semantic Segmentation

There are a variety of techniques used to get the semantic segmentation of an input image; they vary based on computational cost and/or performance.

There's some sort of trade-off between performance and computational cost, that if you can get high performance then you are probably sacrificing the computational cost, and if you care a lot about the cost, then you may sacrifice the performance which is critical in many applications that require real time response just like autonomous driving cars, robots, or augmented reality kits.

First approach we will discuss is sliding window: that we choose a suitable kernel size, and scan the whole image using this kernel, such that we classify each region underneath the kernel; by the end of such a process we have a full understanding of each region in the image (i.e. each region with it corresponding label).

One of the drawbacks of this technique is that it doesn't reuse shared features among overlapping patches which is

inefficient.

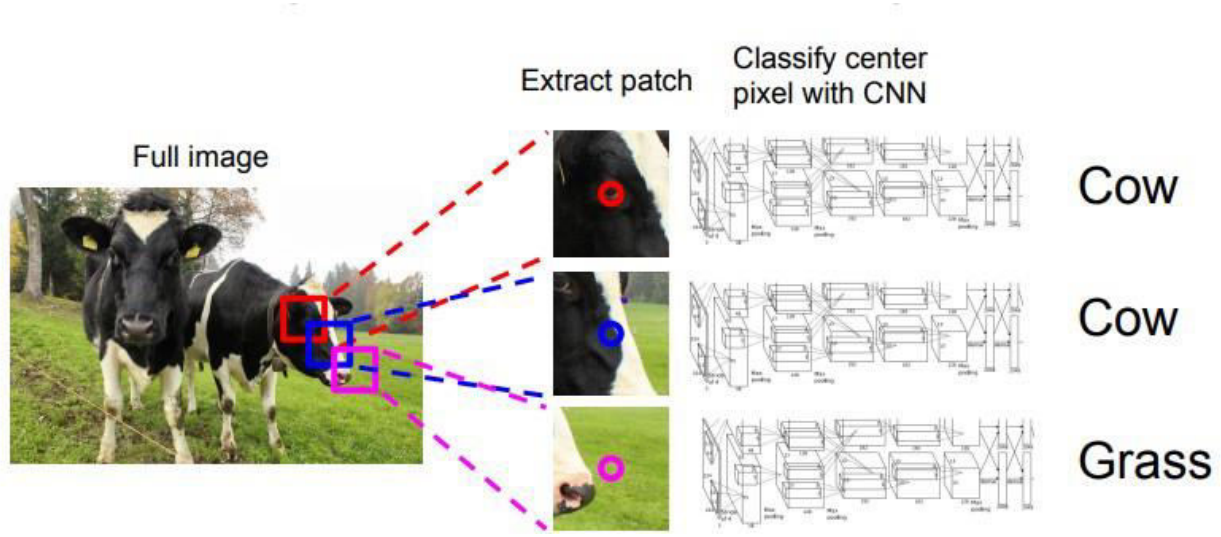


Figure 42: Semantic Segmentation using Sliding Window Approach.

Second approach is using just convolutional layers with same padding to preserve the original dimensions of the original image.

This requires a huge number of loss computations; such that we need loss-function for each pixel in the original image so that we know if it matches the ground truth label for such a pixel.

The acquisition of such a training data is very expensive; we need annotation for each pixel in each image in our training data.

One of the drawbacks of this technique is that preserving image dimensions throughout entire network is computationally expensive.

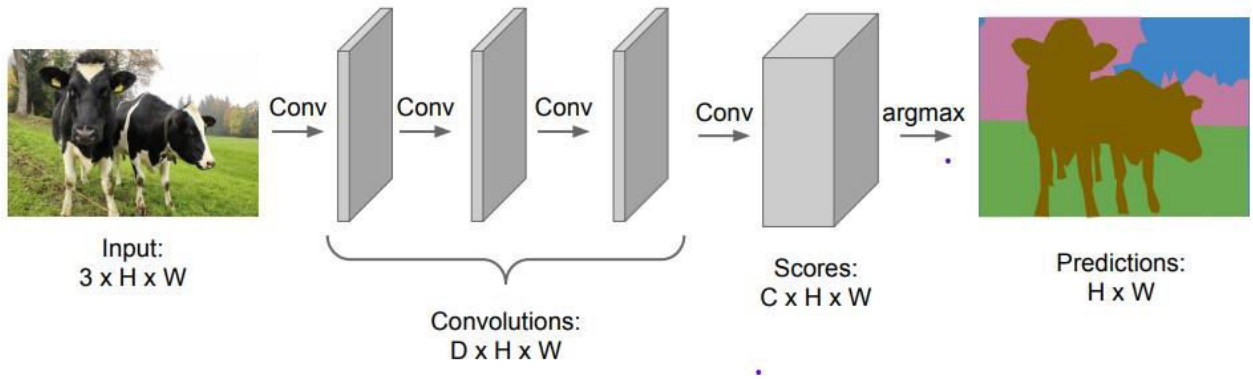


Figure 43: Architecture of Semantic Segmentation Using FCN.

For deep convolutional networks, earlier layers tend to learn low-level concepts while later layers develop more high-level (and specialized) feature mappings. In order to maintain expressiveness, we typically need to increase the number of feature maps (channels) as we get deeper in the network.

This didn't necessarily pose a problem for the task of image classification, because for that task we only care about *what* the image contains (and not where it is located). Thus, we could alleviate computational burden by periodically down-sampling our feature maps through pooling or strided convolutions (i.e. compressing the spatial resolution) without concern. However, for image segmentation, we would like our model to produce a full-resolution semantic prediction. That's why this technique is very expensive.

Third technique is using encoder and decoder networks. Such that an encoder Takes an input image and generates a high-dimensional feature vector also it aggregates features at multiple levels. An encoder network is just like any classification network without the fully connected layer like VGG, ResNet, or Inception.

The building blocks for the CNNs used for such a task are:

- Convolution
- Transpose convolution (Up-sampling)
- Down-sampling

Convolution:

Let $F : \mathbb{Z}^2 \to \mathbb{R}$ be a discrete function. Let $\Omega_r = [-r, r]^2 \cap \mathbb{Z}^2$ and let $k : \Omega_r \to \mathbb{R}$ be a discrete filter of size $(2r + 1)^2$. The discrete convolution operator $*$ can be defined as

$$(F * k)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} F(\mathbf{s})\, k(\mathbf{t}). \tag{1}$$

We now generalize this operator. Let l be a dilation factor and let $*_l$ be defined as

$$(F *_l k)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=\mathbf{p}} F(\mathbf{s})\, k(\mathbf{t}). \tag{2}$$

We will refer to $*_l$ as a dilated convolution or an l-dilated convolution. The familiar discrete convolution $*$ is simply the 1-dilated convolution.

So Equation (1) illustrate normal convolution between input and kernel with dilation = 1.

But, Equation (2) generalize the formula with the dilation value considered in the equation.

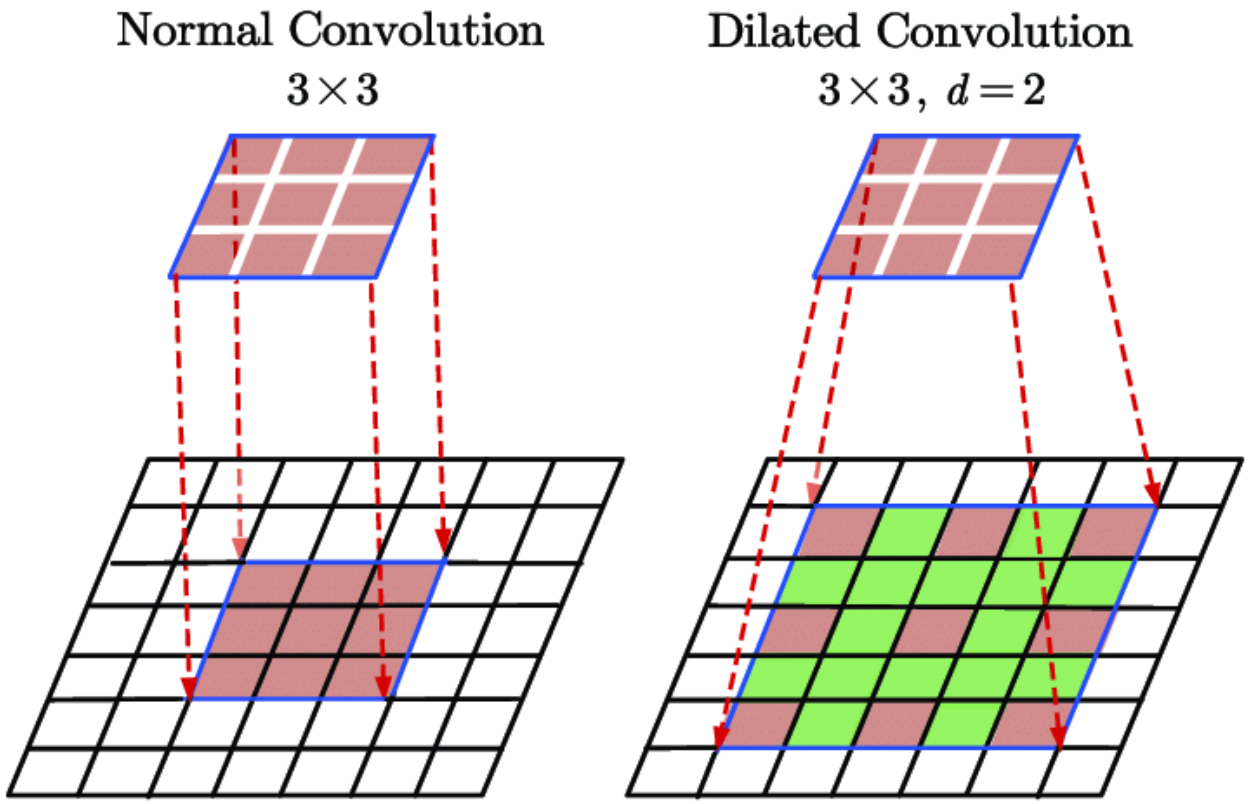


Figure 44: Normal vs. Dilated Convolution.

Normal Convolution: A convolutional layer takes a certain number of input channels (I) and calculates a specific number of output channels (O). The needed parameters for such a layer can be calculated by I*O*K, where K equals the number of values in the kernel.

Dilated convolution: Dilation rate defines a spacing between the values in a kernel. A 3x3 kernel with a dilation rate of 2 will have the same field of view as a 5x5 kernel, while only using 9 parameters. Imagine taking a 5x5 kernel and deleting every second column and row. This delivers a wider field of view at the same computational cost.

Dilated convolutions are particularly popular in the field of real-time segmentation as:

It give global view of an image, but with less parameters.

Broader view, it captures more contextual information of the input.

The detection of fine details by processing inputs in higher resolutions.

It's computationally efficient.

So We use them if we need a wide field of view and cannot afford multiple convolutions or larger kernels.

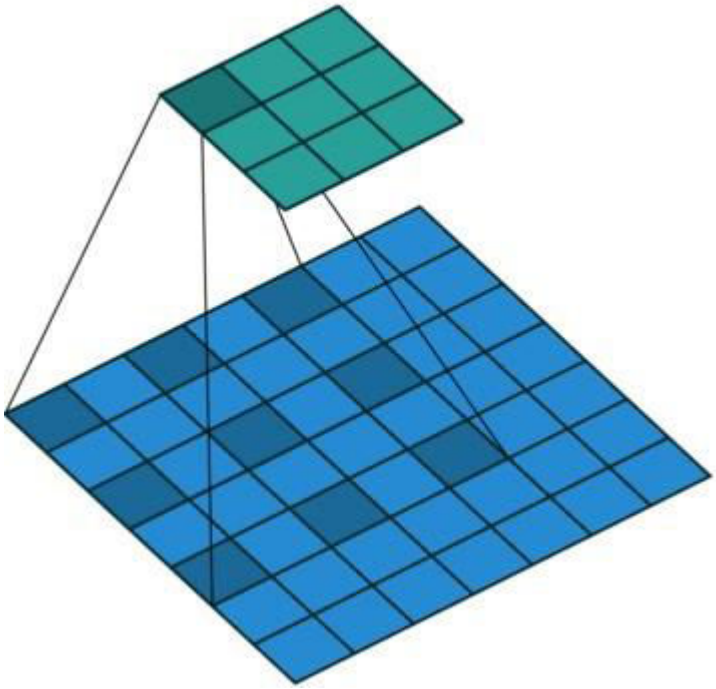


Figure 45: Dilated Convolution.

Transposed convolution: It carries out the ordinary convolution but reverts its spatial transformation.

For example if we have a convolution layer like this:

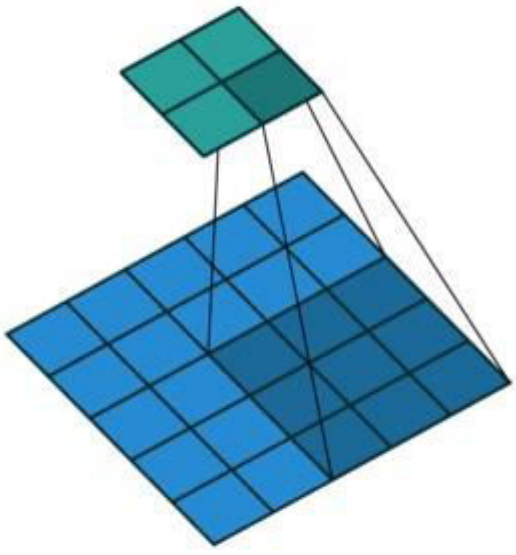


Figure 46: Stage 1 of Transposed Convolution.

Then the transpose convolution layer would be something like this:

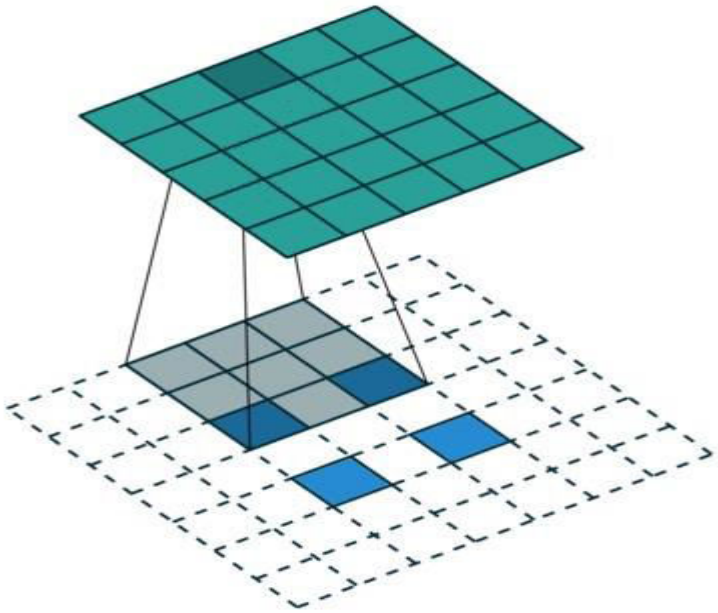


Figure 47: Stage 2 of Transposed Convolution.

This is how we restore the original resolution of the image which is crucial in case of encoder-decoder architecture used for semantic segmentation.

This way; we can combine the up-scaling (i.e. transpose convolution) of an image with a convolution, instead of doing two separate processes.

Down-sampling:

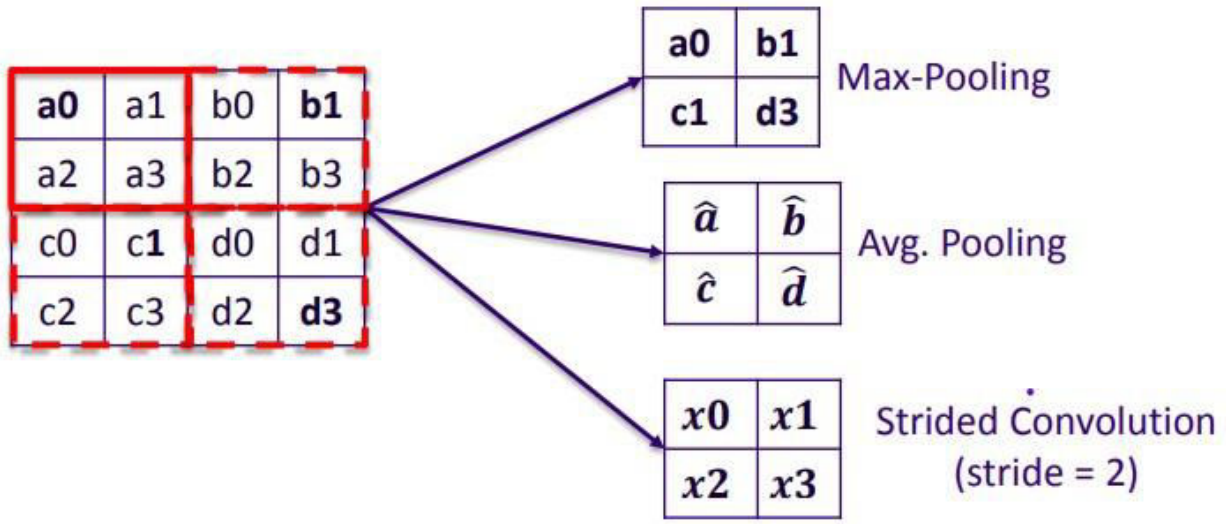


Figure 48: Different Types of Pooling.

This step at which we reduce the resolution of the image, that's why we need to use up-sampling after it to restore the original resolution.

Most of the approaches in the literature are not efficient for real-time applications as they employed large backbone networks such as GoogleNet, Xcepetion, or ResNet, or large CNN architectures for both the encoder or decoder sides. This has resulted in having a large number of parameters to be tuned and a large floating point operations (FLOPS), even though they are efficient from accuracy perspective.

Many approaches have been proposed to deal with this problem, e.g., ERFNet employed a residual connection and depth-wise separable convolution to increase receptive field to achieve high accuracy with a reasonable performance.

Alternatively, ESPNet proposed an efficient module called efficient spatial pyramid (ESP), which uses point wise convolution and spatial pyramid of dilated convolution. ESPnet along with Enet provide a lightweight architectures but with a degradation in accuracy.

RTSeg provided a decoupled encoder-decoder architecture which allows plug any encoder (i.e., VGG16, MobileNet, ShuffleNet, ResNet18) or decoder (i.e., UNet, Dilation, SkipNet) architectures independently. They have found out that using SkipNet architecture along with MobileNet and ShuffleNet provided the best tradeoff between accuracy and performance.

Till now, we have discussed what semantic segmentation means, past & modern implementations for it, different approaches of implementation with the drawbacks of each approach, and finally the building blocks of CNNs including different types of convolutions with the suitable application for each type.

The network we use in our model is different a little bit from the mentioned networks as it’s a lightweight network as we try to combine the performance with the accuracy in our model, knowing that the performance is much more important in our use case because our model is intended to be used in an autonomous car at which the performance is crucial beside the accuracy.

We use Light-Weight RefineNet for Real-Time Semantic Segmentation; it has a promising performance beside its accuracy.
The proposed approach is built upon RefineNet, a powerful semantic segmentation architecture that can be easily used with any backbone network, such as ResNet, DenseNet, NASNet, or any other. This architecture belongs to the family of the 'encoder-decoder' segmentation networks, where the input image is first progressively down-sampled, and later progressively up-sampled in order to recover the original input size.

This approach has been chosen as it shows the best performance among the 'encoder-decoder' approaches, and does not operate over large feature maps in the last layers as methods exploiting a’ trous convolution do. The processing of large feature maps significantly hinders real-time performance of such architectures as DeepLab and PSPNet due to the increase in the number of floating point operations.

One of the drawbacks of the 'encoder-decoder' approaches is that they tend to have a larger number of parameters due to the decoder part that recovers the high resolution output –as mentioned above-. In this work, the real-time performance problem is tackled by specifically concentrating on the decoder and empirically showing that both the number of parameters and floating point operations can be drastically reduced without a significant drop in accuracy.

The real-time performance is solved in two steps:

1. Outline important engineering improvements that save the number of parameters by more than 50% in the original RefineNet.
2. Specify the redundancy of residual blocks in the architecture and show that the performance stays the same with those blocks being removed.

This approach has passed through extensive experiments on three competitive benchmarks for semantic segmentation - PASCAL VOC, NYUDv2 and PASCAL Person-Part, and with five different backbone networks - ResNet-50, ResNet-101, ResNet-152, along with recently released NASNet-Mobile, and MobileNet-v2.

The first three backbones are used for the direct comparison between the proposed approach and the original RefineNet, while the last two are used to showcase that this method is independent to the backbone compression. This model what we use in our training and testing but first we will explain the semantic segmentation that used by paper of the project and then explain what we use.

Multi-Scale Context Aggregation by Dilated Convolutions (Paper Semantic)

State-of-the-art models for semantic segmentation are based on adaptations of convolutional networks that had originally been designed for image classification. However, dense prediction problems such as semantic segmentation are structurally different from image classification.

In this work, they develop a new convolutional network module that is specifically designed for dense prediction. The presented module uses dilated convolutions to systematically aggregate multi-scale contextual information without losing resolution. The architecture is based on the fact that dilated convolutions support exponential expansion of the receptive field without loss of resolution or coverage. They show

That the presented context module increases the accuracy of state-of-the-art semantic segmentation systems.

To illustrate What Multi-scale contextual information means let's consider we have network with 10 convolution layers, and we take output from every second layers, resize them to the same size, and concatenate. Now we have pie of information from different layers that we can use for prediction.

Why follow this approach instead of getting information from last layer? As with each layer we get more and more global features, so small features in each layer are missed. And if we take only small features then we will lose global context. Here is a good example that describe main problem of context in CNN (Fig 18).

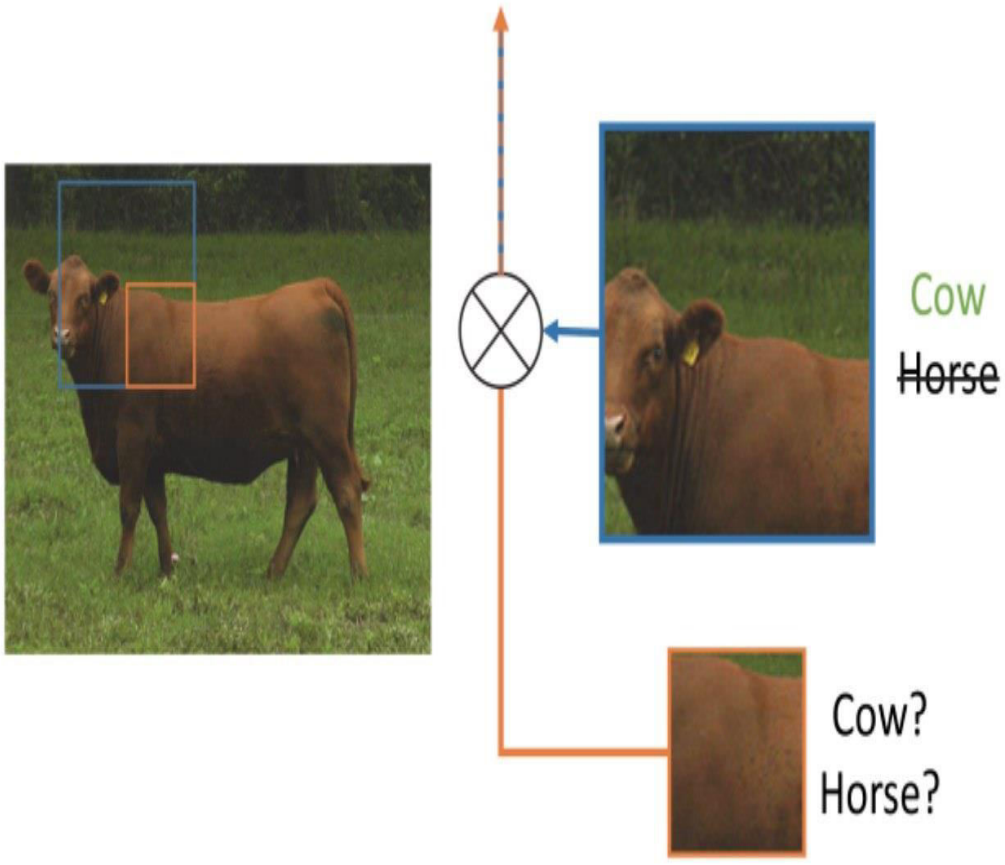


Figure 49: Global vs. Small features.

Modern image classification networks integrate multi-scale contextual information via successive pooling and subsampling layers that reduce resolution until a global prediction is obtained.

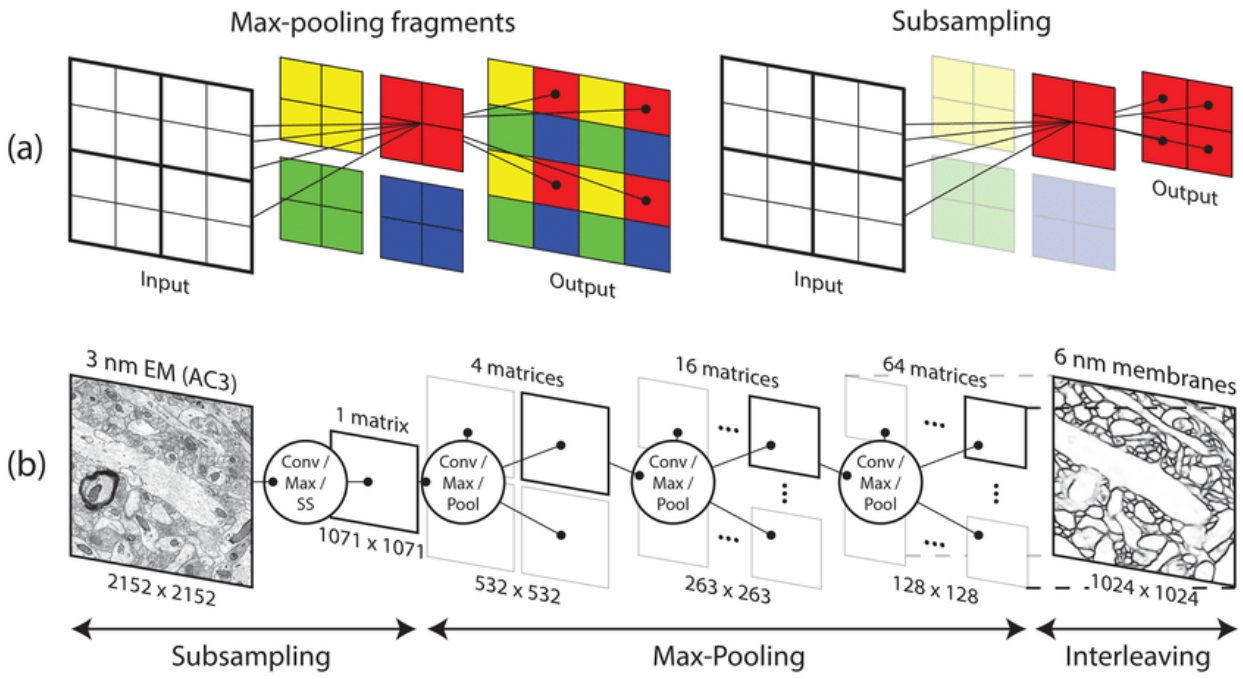


Figure 50: Pooling and Subsampling

Recent work has studied two approaches to deal with the conflicting demands of multi-scale reasoning and full-resolution dense prediction. First approach involves repeated up-convolutions that aim to recover lost resolution while carrying over the global perspective from down-sampled layers. Second approach involves providing multiple rescaled versions of the image as input to the network and combining the predictions obtained for these multiple inputs.

In this work, they develop a convolutional network module that aggregates multi-scale contextual information without losing resolution or analyzing rescaled images. It is a rectangular prism of convolutional layers, with no pooling or subsampling. The module is based on dilated convolutions, which support exponential expansion of the receptive field without loss of resolution or coverage.

The experiments proved that plugging this module into existing semantic segmentation architectures reliably increases their accuracy.

So following figures illustrates how dilated convolutions support exponential expansion of the receptive field without loss of resolution or coverage.

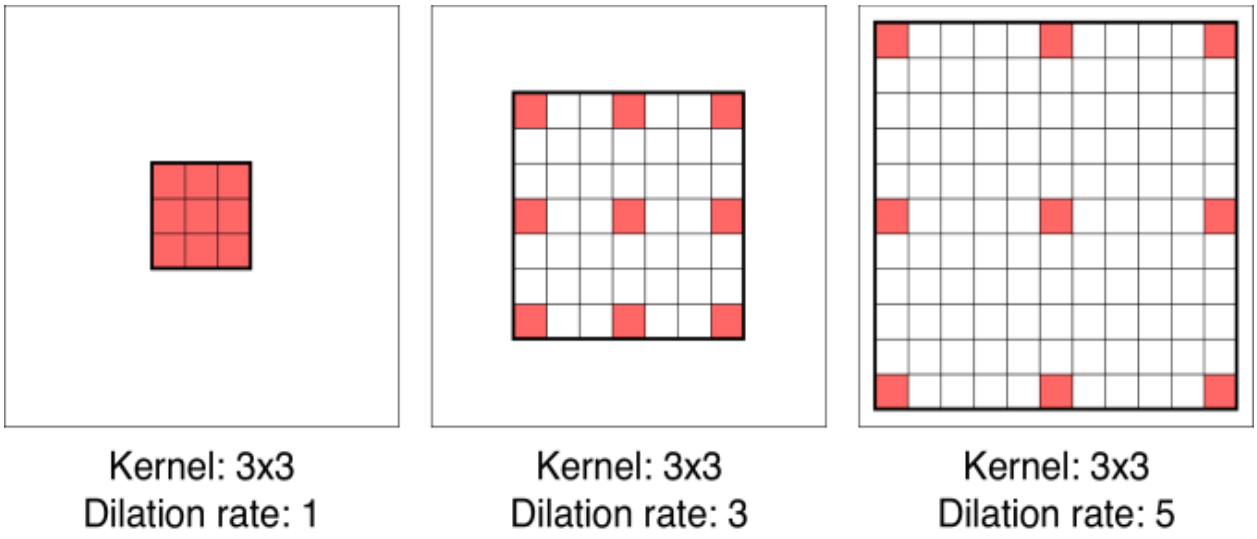


Figure 51: Dilated Kernels

Let $F_0, F_1, \ldots, F_{n-1} : Z^2 \rightarrow R$ be discrete functions and let $k_0, k_1, \ldots, k_{n-2} : \Omega_1 \rightarrow R$ be discrete 3×3 filters. Consider applying the filters with exponentially increasing dilation:

$$F_{i+1} = F_i *_{2^i} k_i \qquad \text{for } i = 0, 1, \ldots, n-2.$$

Define the receptive field of an element p in F_{i+1} as the set of elements in F_0 that modify the value of $F_{i+1}(p)$. Let the size of the receptive field of p in F_{i+1} be the number of these elements. It is easy to see that the size of the receptive field of each element in F_{i+1} is $(2^{i+2} - 1) \times (2^{i+2} - 1)$. The receptive field is a square of exponentially increasing size. This is illustrated in (Fig. 21).

In (Fig. 21), Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1

by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

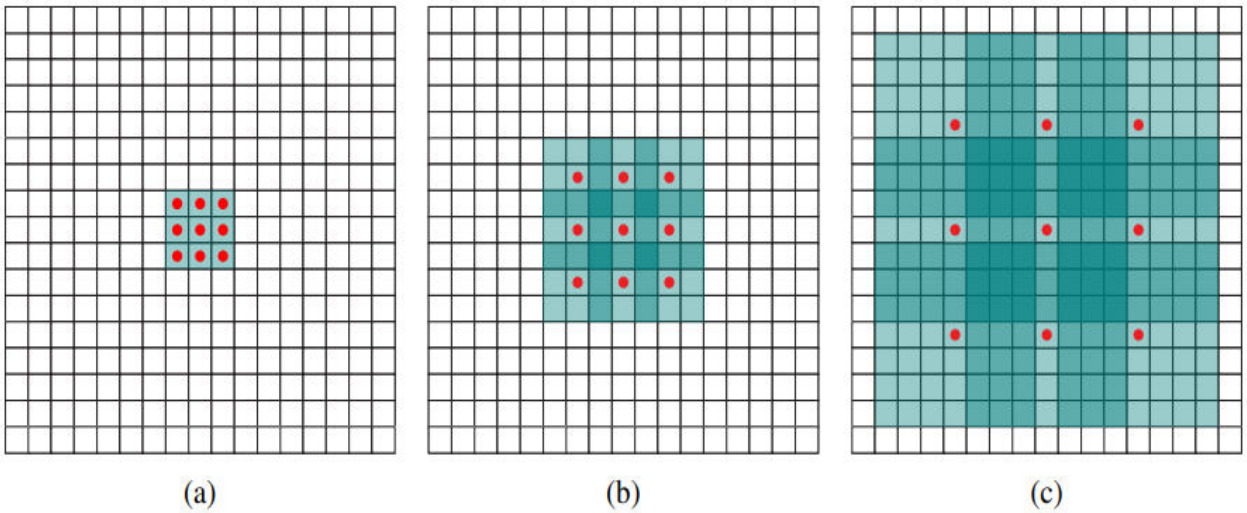


Figure 52: Receptive Field of Different Dilation rate

Multi-Scale Context Aggregation: The context module is designed to increase the performance of dense prediction architectures by aggregating multi-scale contextual information. The module takes C feature maps as input and produces C feature maps as output. The input and output have the same form, thus the module can be plugged into existing dense prediction architectures.

The basic context module has 7 layers that apply 3×3 convolutions with different dilation factors. The dilations are 1, 1, 2, 4, 8, 16, and 1. Each of these convolutions is followed by a point-wise truncation max(·, 0). A final layer performs 1×1×C convolutions and produces the output of the module. Frontend of this module provides feature maps at 64×64 resolution as input to the context network so they stop the exponential expansion of the receptive field after layer 6. They found that random initialization schemes were not effective for the context module. So they found an alternative initialization with clear semantics to be much more effective:

$$k^b(t, a) = 1_{[t=0]}1_{[a=b]},$$

Where a is the index of the input feature map and b is the index of the output map. This initialization sets all filters such that each layer simply passes the input directly to the next. There was a concern that this modification put network in a mode that backpropagation cannot significantly improve the default behavior, but experiments indicate that backpropagation reliably harvests the contextual information provided by the network to increase the accuracy of the processed maps.

Layer	1	2	3	4	5	6	7	8
Convolution	3×3	3×3	3×3	3×3	3×3	3×3	3×3	1×1
Dilation	1	1	2	4	8	16	1	1
Truncation	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Receptive field	3×3	5×5	9×9	17×17	33×33	65×65	67×67	67×67
Output channels								
Basic	C	C	C	C	C	C	C	C
Large	$2C$	$2C$	$4C$	$8C$	$16C$	$32C$	$32C$	C

Figure 53: CONTEXT Network Architecture

Front End: They implement and train a front-end prediction module that takes a color image as input and produces C = 21 feature maps as output. They use the VGG-16 network but remove the last two pooling and striding layers. All subsequent layers were dilated by a factor of 2 for each pooling layer that was ablated. Thus convolutions in the final layers are dilated by a factor of 4. The front-end module takes padded images as input and produces feature maps at resolution 64×64. They use reflection padding so buffer zone is filled by reflecting the image about each edge.

In training they used VOC 2012 training set and raining was performed by stochastic gradient descent (SGD) with mini-batch size 14, learning rate 10^{-3}, and momentum 0.9. The network was trained for 60K iterations.

Conclusion: In the end, if we look for model to produce high-resolution output and high accuracy they believe that high-resolution operation throughout the network is both feasible and desirable. Their work shows that the dilated convolution operator is particularly suited to dense prediction due to its ability to expand the receptive field without losing resolution or coverage. Also they utilize dilated convolutions to design a new network structure that reliably increases accuracy when plugged into existing semantic segmentation systems.

But in our case we concerned with more performance over less accuracy as this system must run in real time so we decide to replace this algorithm with lightweight real time semantic segmentation to fasten the system more and more so we you use

Light-Weight RefineNet for Real-Time Semantic Segmentation
Vladimir Nekrasov, Chunhua Shen, Ian Reid
In BMVC 2018

Light-Weight RefineNet for Real-Time Semantic Segmentation

Most techniques for semantic segmentation either suffer from a large number of parameters, a large number of floating point operations, or both. These issues constitute a significant drawback of exploiting such models in applications requiring real-time processing like autonomous driving which is our interest in our project which is intended to be used in a self-driving car.

To overcome these problems, there have been several task-specific approaches where the authors adapt PSPNet to deal with multiple image scales progressively. They attain the speed of 30 FPS on 1024×2048 images, and 67% mean IOU on the validation set of CityScapes, but it is not clear whether this approach would still acquire solid performance on other datasets with low-resolution inputs.

Also; there are real-time segmentation networks that have been following the encoder-decoder paradigm, but have not been able to acquire decent performance

levels. Concretely, SegNet achieved 40 FPS on inputs of size 360×480 with only 57.0% mean IOU on CityScapes, while ENet were able to perform inference of 20 FPS on inputs of size 1920×1080 with 58.3% mean IOU on CityScapes.

So; it's clear that most of the proposed networks either achieve high performance with low accuracy, or high accuracy with low performance, meaning that they can't achieve high performance and accuracy together.

Before diving into our network (lightweight RefineNet) we should explain the original network (RefineNet) briefly so that we can mark the differences between the two networks knowing that the lightweight RefineNet is built upon RefineNet as mentioned above.

The RefineNet architecture belongs to the family of the encoder-decoder approaches. As the encoder backbone, it can re-use most popular classification networks; in particular, the original approach used residual networks. The approach does not modify the classification network in any way except for removing last classification layers.

In the decoder part, RefineNet depends on two types of additional abstractions: residual convolutional unit (RCU) and chained residual pooling (CRP).The first one is a simplification of the original residual block without batch normalization, while the second one is a sequence of multiple convolutional and pooling layers, also arranged in a residual manner. All of these blocks use 3 × 3 convolutions and 5 × 5 pooling with appropriate padding so that the spatial size would stay the same like the input image.

The decoding process starts with the propagation of the last output from the classifier (with the lowest resolution) through two RCU blocks followed by four pooling blocks

of CRP and another three RCU blocks before being fed into a fusion block along with the second to last feature map.

Inside the fusion block, each path is convolved with 3×3 convolution and up-sampled to the largest resolution among the paths.

Two paths are then summed up, and analogously further propagated through several RCU, CRP and fusion blocks until the desired resolution is reached. The final convolutional layer produces the score map.

A significant practical benefit of the RefineNet architecture is that it can be mixed with any backbone network as long as the backbone has several subsampling operations inside (which is the case for most SOTA classifiers). Thus, there are no specific changes needed to be made in order to apply the RefineNet architecture using any other model.

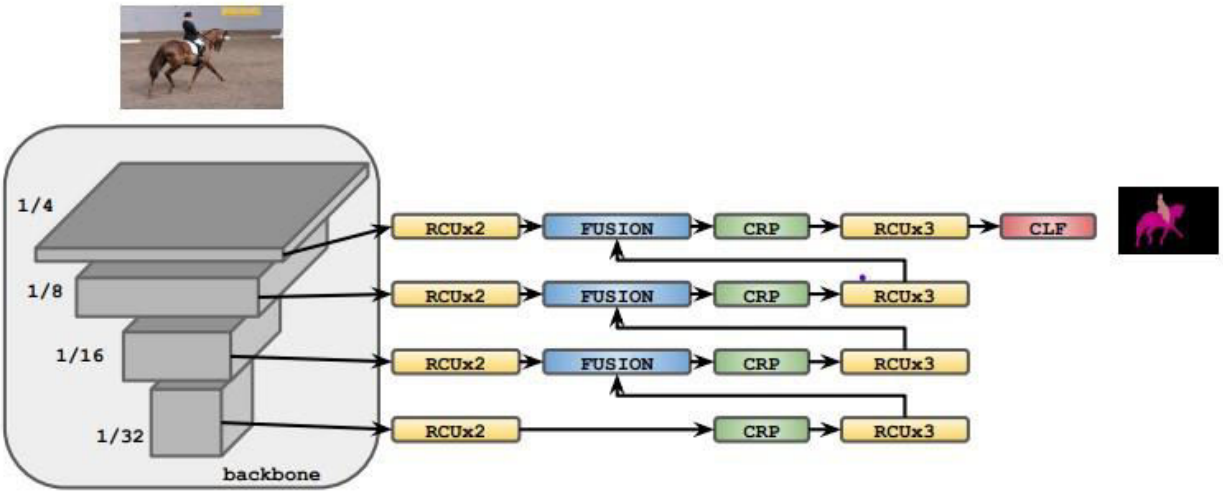


Figure 54: RefineNet Architecture.

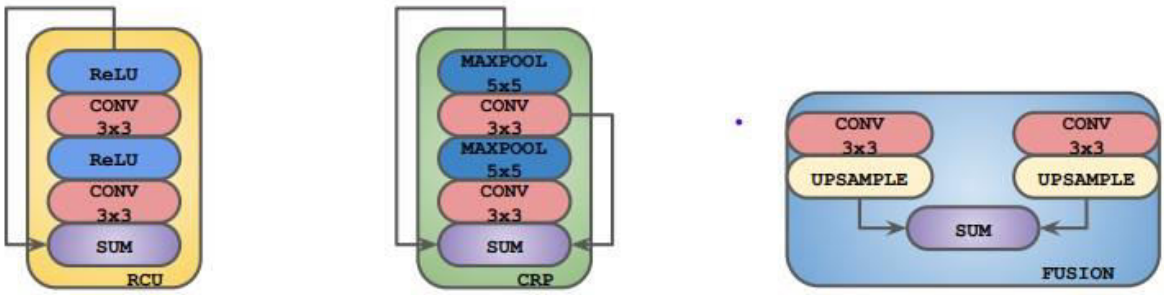


Figure 55: RCU, CRP, and FUSION Blocks in RefineNet from Left to Right Respectively.

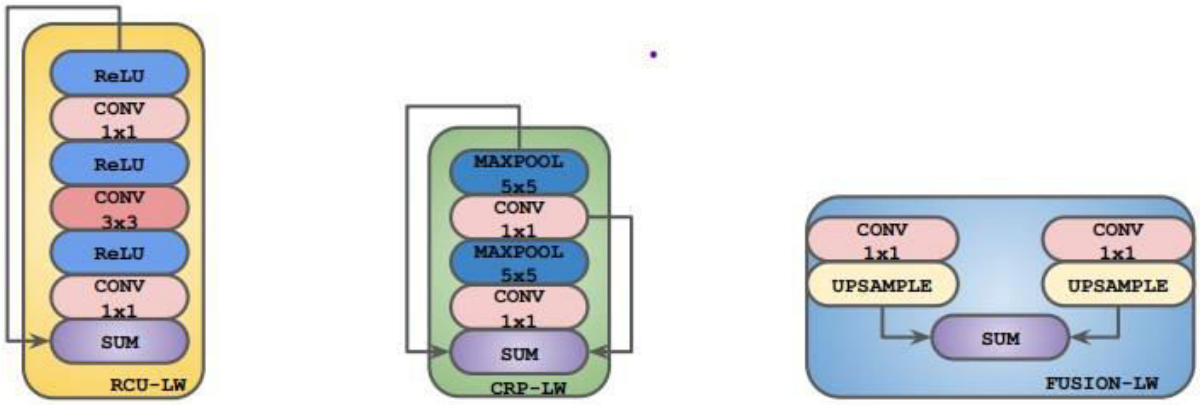


Figure 56: RCU LW, CRP LW, and FUSION LW Blocks in RefineNet from Left to Right Respectively.

So, this is the architecture of the RefineNet and its main blocks, and we can notice that the most expensive parts in terms of both the number of parameters and the number of floating point operations of the original RefineNet come from the usage of 3×3 convolutions. So; the lightweight RefineNet focuses on replacing them with simpler counterparts without performance drops.

Convolutions with larger kernel sizes aim to increase the receptive field size (and the global context coverage), while 1 × 1 convolutions are merely transforming per-pixel features from one space to another locally. This approach argues that in the case of RefineNet we do not require expensive 3×3 convolutions at all, and we are able to show empirically that replacing them with 1×1 convolutions does not hurt performance.

Furthermore, it evaluates the empirical receptive field size for both cases, and does not find any significant difference.
In particular, this approach replaces 3 × 3 convolutions within CRP and fusion blocks with 1 × 1 counterpart. This way, we reduce the number of parameters by more than two times, and the number of FLOPs by more than three times.

Actually, dropping 3×3 convolutions should significantly harm the receptive field size of the original architecture. But, it has been noticed that the new architecture does not experience this due to the skip-design structure of RefineNet, where low-level features are being summed up with the high-level ones, and keeping CRP blocks that are responsible for gathering contextual information.

The lightweight architecture outlined above has been trained using PASCAL VOC and it was able to achieve close to the original network performance, and, furthermore, it has been observed that removing RCU blocks did not lead to any drop in accuracy, and, in fact, the weights in RCU blocks almost completely saturated.

To confirm that this only occurs in the light weight case, RCU blocks have been dropped from the original RefineNet architecture, and it has experienced more than 5% performance drop on the same dataset. This happens due to the fact that RCU blocks being redundant in the 1 × 1 convolution regime, as the only important goal of increasing the contextual coverage is essentially performed by pooling layers inside CRP. The final architecture does not contain any RCU blocks and only relies on CRP blocks with 1×1 convolutions and 5×5 max-pooling inside, which makes our method extremely fast and light-weight.

Till now we have discussed the architecture of the RefineNet and the modifications made to it in the lightweight RefineNet to decrease the number of parameters and

floating point operations of the original RefineNet architecture, while keeping the performance levels the same.

A lot of experiments have been made to the proposed network to prove that it is able to achieve similar performance levels with the original RefineNet while significantly reducing the number of parameters and drastically increasing the speed of a forward pass; and to highlight the possibility of applying our method using other architectures. To achieve such a goal; three segmentation datasets have been used, namely, NYUDv2, PASCAL VOC and PASCAL Person-Part dataset, and five classification networks, i.e., ResNet-50, ResNet-101, ResNet-152, NASNet-Mobile and MobileNet-v2 (only for Pascal VOC), all of which have been pre-trained on ImageNet.

Model	val mIoU,%	test mIoU,%	FLOPS,B
DeepLab-v2-ResNet-101-CRF [8]	77.7	79.7	-
RefineNet-101 [46]	-	82.4	263
RefineNet-152 [46]	-	**83.4**	283
RefineNet-LW-50 (ours)	78.5	81.1[1]	**33**
RefineNet-LW-101 (ours)	80.3	82.0[2]	52
RefineNet-LW-152 (ours)	82.1	82.7[3]	71
MobileNet-v1-DeepLab-v3 [60]	75.3	-	14.2
MobileNet-v2-DeepLab-v3 [60]	75.7	-	**5.8**
RefineNet-LW-MobileNet-v2 (ours)	**76.2**	**79.2**[4]	**9.3**
RefineNet-LW-NASNet-Mobile (ours)	**77.4**	**79.3**[5]	11.4

Figure 57: Quantitative Results on PASCAL VOC. Mean IOU and the Number of FLOPs on 512×512 inputs are reported, where possible.

The results show that the proposed network –lightweight RefineNet- outperforms both MobileNet-v1+DeepLab-v3 and MobileNet-v2+DeepLab-v3 approaches, which are closely related to the proposed network.

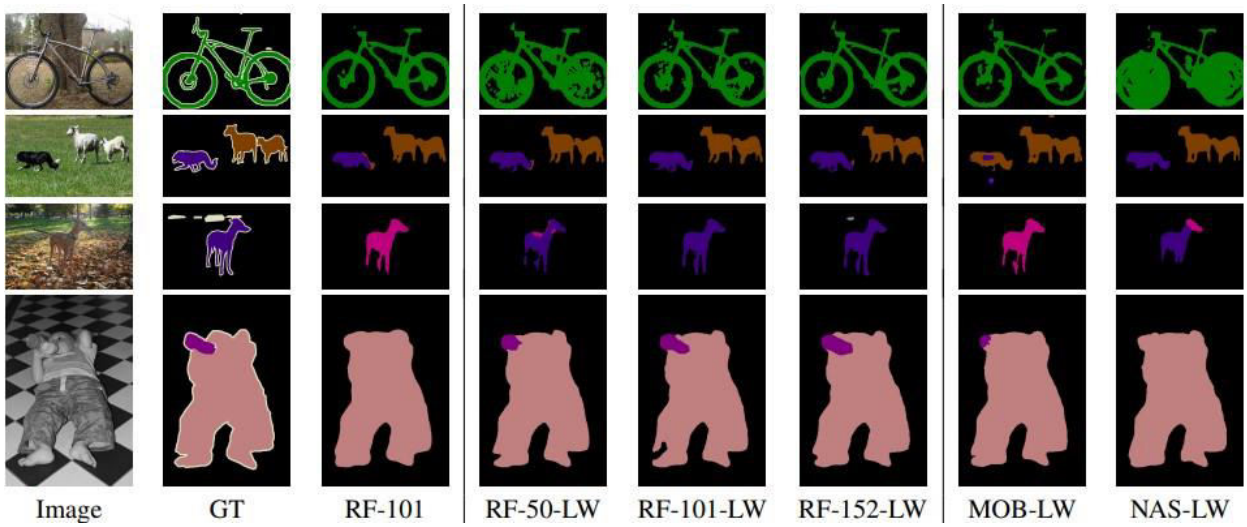


Figure 58: Visual results on validation set of PASCAL VOC with residual models (RF), MobileNet-v2 (MOB) and NASNet-Mobile (NAS). The original RefineNet-101 (RF-101) is The Implemented One in Our Project.

To sum-up; this approach –lightweight RefineNet- tackled the problem of rethinking an existing semantic segmentation architecture into the one suitable for real-time performance, while keeping the performance levels mostly the same.
It achieved that by proposing simple modifications to the existing network and highlighting which building blocks were redundant for the final result.

This method can be applied along with any classification network for any dataset and can further benefit from using light-weight backbone networks, and other compression approaches. Quantitatively, it was able to closely match the performance of the original network while significantly surpassing its runtime and even acquiring 55 FPS on 512×512 inputs (from initial 20 FPS). Besides that, it demonstrates that having convolutions with large kernel sizes can be unnecessary in the decoder part of segmentation networks.

Single FoA branch

Structure

The branch consists of two streams the cropped stream and the resized RGB stream followed by a coarse module the cropped stream end with that but the resized RGB followed by a refined network (2D convolution) and at the test time, only the output of the resized stream will be considered.

We do that to prevent the model from stalling the learning at in early training stages. The coarse module in both streams has the same weights. The convolutional part which provides the rough estimate of the attentional map corresponding to the clip. As shown in (Fig. 59)

The 3D convolution extracts the time dependencies through 4D input (clip) .in addition to the width and the height dimensions in the 2D convolution. The 3D convolution also strides along time. The j-th feature map in the i-th layer at time t at position (x, y) is computed as:

$$v_{i,j}^{x,y,t} = b_{i,j} + \sum_{m} \sum_{p=0}^{P_{i-1}} \sum_{q=0}^{Q_{i-1}} \sum_{r=0}^{R_{i-1}} w_{i,j,m}^{p,q,r} v_{i-1,m}^{x+p,y+q,t+r}$$

Where

- $W_{p,q,r,m}$ is the value at the position (p,q) at time r of the kernel connected to the m-th feature map.
- P_i, Q_i and R_i are the kernel dimensions along width, height and time axis respectively.
- $b_{i,j}$ is the bias from layer i to layer j.

Coarse module structure

In (Fig. 28), The COARSE module is made of an encoder based on C3D network followed by a bilinear up-sampling (bringing representations back to the resolution of the input image) and a final 2D convolution. During feature extraction, the temporal axis is lost due to 3D pooling. All convolutional layers are preceded by zero paddings in order keep borders, and all kernels have size 3 along all dimensions. Pooling layers have size and stride of (1, 2, 2, 4) and (2, 2, 2, 1) along temporal and spatial dimensions respectively. All activations are ReLUs.

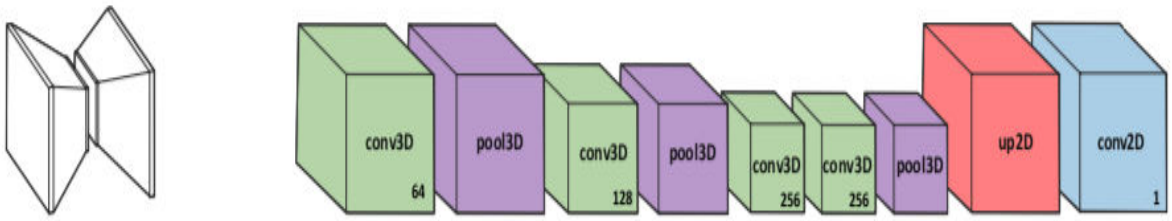


Figure 59: The COARSE module.

The Refined Network Structure

The refined network consists of a stack of convolutional layers (refinement module) with the aim of refining the prediction.

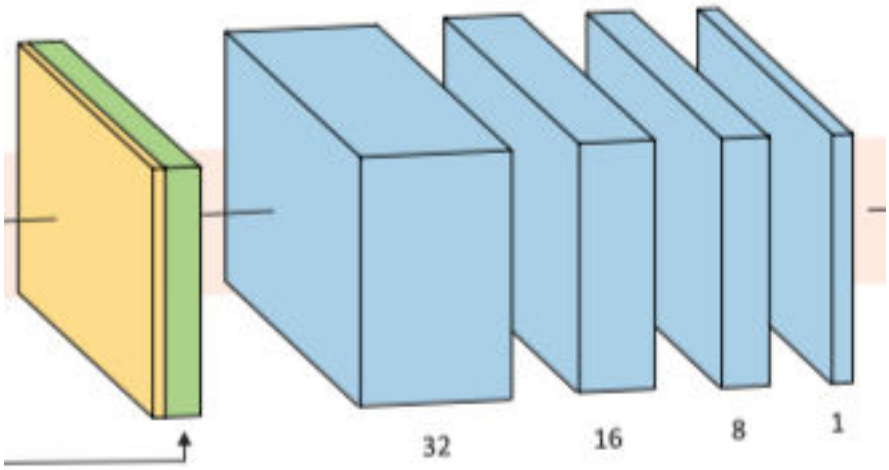


Figure 60: Refined Network Structure.

Training phase

The single (FoA) architecture is shown in (Fig. 61).

At training time the cropped stream feeds the coarse module with random crops, to prevent the model to learn a center bias where it learns the current focus of attention given visual cues rather than prior spatial location.

The training process described in [65], employs a 128 × 128 image resize, and then a 112 × 112 random crop. Despite this, the small difference within the two resolutions restricts the variance of gaze position in the ground truth fixation maps and is not sufficient to prevent the attraction towards the center of the image.

For this, training images are resized to 256 × 256 before being cropped to 112 × 112. This cropped design generates an image that covers less than a quarter of the original image so guaranteeing a sufficient variation in prediction ends. As the cropped image becomes smaller, the ratio of pixels in the ground truth masked by gaze increases, leading the model to learn larger maps.

In opposition, the resized stream feeds the same COARSE model with the same images, this time resized to 112 × 112 and not cropped.

The coarse result that comes from the COARSE model is then concatenated with the final frame of the input clip, the frame corresponding to the final prediction. Finally, the concatenated tensor goes through the REFINE module to obtain the higher resolution prediction of the (FoA).

The two-stream training process for a single branch is described in Algorithm 1.

In (Fig. 61), A single FoA branch of our prediction structure. The COARSE module is applied to both a cropped and a resized version of the input image, which is a video clip of 16 consecutive frames. The cropped input is used during training to augment the data and the variety of ground truth fixation maps. The prediction of the resized input is accumulated with the last frame of the video clip and fed to a stack of convolutional layers (refinement module) to refine the prediction. Training is performed end-to-end, and weights between COARSE modules are shared. At test time, only the refined predictions are used. Note that the complete model is composed of three of these branches (see Fig. 63), each of which predicting visual attention for different inputs (namely image, optical flow, and semantic segmentation). All activations in the refinement module are LeakyReLU with $\alpha = 10-3$, except for the last single-channel convolution that features ReLUs. Crop and resize streams are highlighted by light blue and orange arrows respectively.

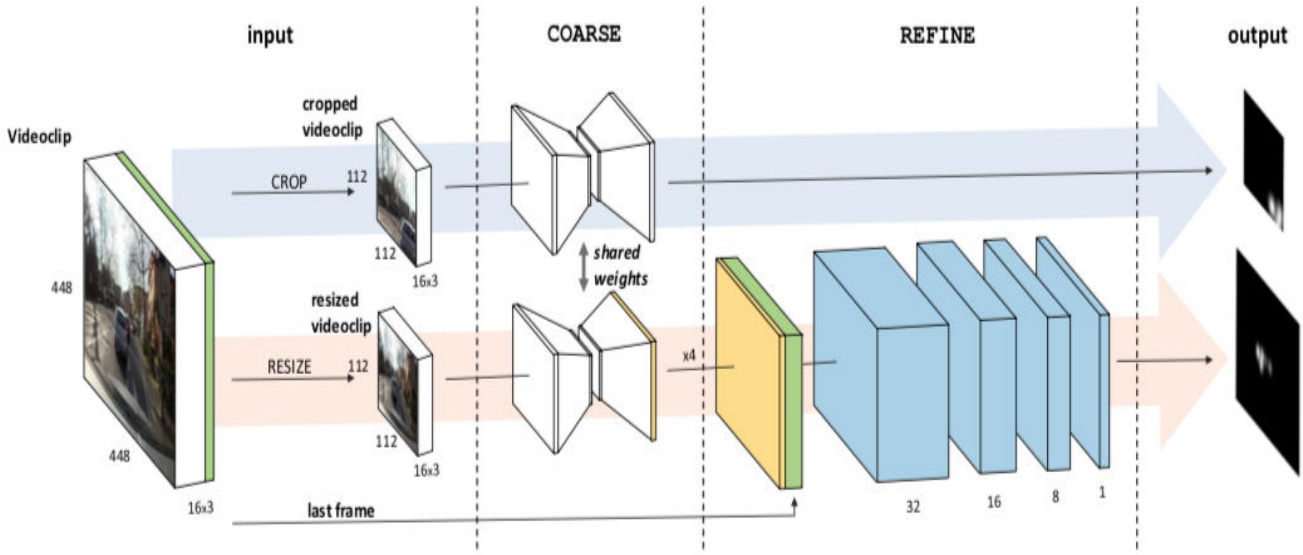


Figure 61: A single FoA branch of our prediction structure.

Algorithm 1 TRAINING. The model is trained in two steps: first each branch is trained separately through iterations detailed in **procedure** SINGLE_BRANCH_TRAINING_ITERATION, then the three branches are fine-tuned altogether as shown by **procedure** MULTI_BRANCH_FINE-TUNING_ITERATION. For clarity, we omit from notation: i) the subscript b denoting the current domain in all X, x and $\hat{y}$ variables in the single branch iteration and ii) the normalization of the sum of the outputs from each branch in line 13.

1: **procedure A:** SINGLE_BRANCH_TRAINING_ITERATION
input: domain data $X = \{x_1, x_2, \ldots, x_{16}\}$, true attentional map y of last frame x_{16} of videoclip X
output: branch loss $\mathcal{L}_b$ computed on input sample (X, y)
2: $X_{res} \leftarrow \texttt{resize}(X, (112, 112))$
3: $X_{crop}, y_{crop} \leftarrow \texttt{get\_crop}((X, y), (112, 112))$
4: $\hat{y}_{crop} \leftarrow \texttt{COARSE}(X_{crop})$ # get coarse prediction on uncentered crop
5: $\hat{y} \leftarrow \texttt{REFINE}(\texttt{stack}(x_{16}, \texttt{upsample}(\texttt{COARSE}(X_{res}))))$ # get refined prediction over whole image
6: $\mathcal{L}_b(X, Y) \leftarrow D_{KL}(y_{crop} \| \hat{y}_{crop}) + D_{KL}(y \| \hat{y})$ # compute branch loss as in Eq. 4

7: **procedure B:** MULTI_BRANCH_FINE-TUNING_ITERATION
input: data $X = \{x_1, x_2, \ldots, x_{16}\}$ for all domains, true attentional map y of last frame x_{16} of videoclip X
output: overall loss $\mathcal{L}$ computed on input sample (X, y)
8: $X_{res} \leftarrow \texttt{resize}(X, (112, 112))$
9: $X_{crop}, y_{crop} \leftarrow \texttt{get\_crop}((X, y), (112, 112))$
10: **for** branch $b \in \{\text{RGB}, \text{flow}, \text{seg}\}$ **do**
11: $\hat{y}_{b_{crop}} \leftarrow \texttt{COARSE}(X_{b_{crop}})$ # as in line 4 of the above procedure
12: $\hat{y}_b \leftarrow \texttt{REFINE}(\texttt{stack}(x_{b_{16}}, \texttt{upsample}(\texttt{COARSE}(X_{b_{res}}))))$ # as in line 5 of the above procedure
13: $\mathcal{L}(X, Y) \leftarrow D_{KL}(y_{crop} \| \sum_b \hat{y}_{b_{crop}}) + D_{KL}(y \| \sum_b \hat{y}_b)$ # compute overall loss as in Eq. 5

Figure 62: Algorithm 1. Training process.

Training objective

The cost function can be minimized in terms of Kullback-Leibler:

$$D_{KL}(Y \| \hat{Y}) = \sum_i Y(i) \log \left(\epsilon + \frac{Y(i)}{\epsilon + \hat{Y}(i)} \right)$$

Where Y is the ground truth, Yˆ is the branch output,

ε is a small constant that guarantees numerical stability. As each single FoA branch computes an error on both the cropped image and the resized image, the branch loss function defines as:

$$\mathcal{L}_b(\mathcal{X}_b, \mathcal{Y}) = \sum_m \Big(D_{KL}(\phi(Y^m) \| \mathcal{C}(\phi(X_b^m))) + D_{KL}(Y^m \| \mathcal{R}(\mathcal{C}(\psi(X_b^m)), X_b^m))) \Big)$$

Where C indicate to the COARSE , R REFINE modules , (X_b^m, Y^m) ∈ Xb × Y is the m-th training example in the b-th domain (RGB, optical flow, semantic segmentation), and ϕ and ψ indicate the crop and the resize functions respectively.

Multi-branch model

In (Fig. 63), the multi-branch model is consists of three branches (RGB, Optical flow, Semantic segmentation) every branch gets information from a different domain and contributes to the final prediction equally.

Each branch exploits information from a different domains and contributes to the final prediction equally.

The first branch acts in the RGB field and processes raw visual data about the scene rgb.

The second branch works on the motion for the optical flow image flow.

Finally, the last branch takes semantic segmentation as input seg. For the last branch, the number of input channels depends on the specific algorithm used to extract the results

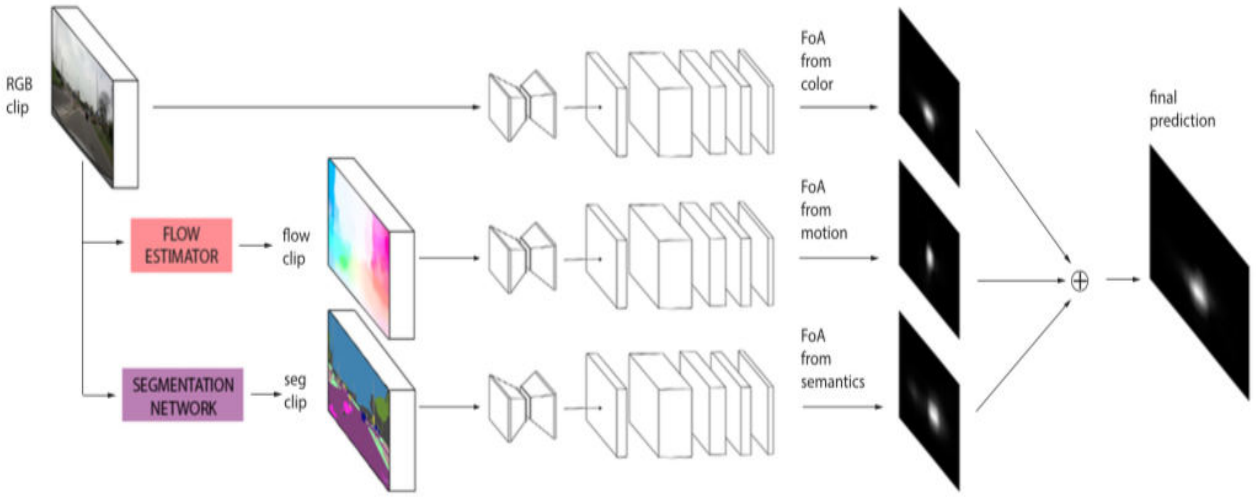


Figure 63: The multi-branch model is composed of three different branches, each of which has its own set of parameters, and their predictions are summed to obtain the final map

The three independent branches generate (FoA) maps that are summed and normalized to generate the final result.

For larger batch size, we train each branch independently according to next eqn. Then, the entire multi-branch model that merges the three branches is fine-tuned with the following loss:

$$\mathcal{L}(\mathcal{X},\mathcal{Y}) = \sum_m \Big(D_{KL}(\phi(Y^m) \| \sum_b \mathcal{C}(\phi(X_b^m))) + D_{KL}(Y^m \| \sum_b \mathcal{R}(\mathcal{C}(\psi(X_b^m)), X_b^m))) \Big).$$

The algorithm describing the complete inference over the multi-branch model in detailed:

Algorithm 2 INFERENCE. At test time, the data extracted from the resized videoclip is input to the three branches and their output is summed and normalized to obtain the final FoA prediction.

input: data $X = \{x_1, x_2, \ldots, x_{16}\}$ for all domains
output: predicted FoA map $\hat{y}$
1: $X_{\text{res}} \leftarrow \texttt{resize}(X, (112, 112))$
2: **for** branch $b \in \{\text{RGB}, \text{flow}, \text{seg}\}$ **do**
3: $\quad \hat{y}_b \leftarrow \texttt{REFINE}(\texttt{stack}(x_{b_{16}}, \texttt{upsample}(\texttt{COARSE}(X_{b_{\text{res}}}))))$
4: $\hat{y} \leftarrow \sum_b \hat{y}_b / \sum_i \sum_b \hat{y}_b(i)$

Figure 64: Algorithm 2. *Complete inference over the multi-branch model.*

Chapter 5

Experiments

Here we compute the multi-branch model performance.

First, we comparing our model against some other models and methods.

For the evaluation phase, we use Pear- son's Correlation Coefficient (CC) and (DKL) measures. Further, we compute the information gain (IG) to measure the quality of output P with respect to a ground truth Y as:

$$IG(P, Y, B) = \frac{1}{N} \sum_i Y_i [(\log_2(\epsilon + P_i) - \log_2(\epsilon + B_i)]$$

Where B is the bias computed as the average training fixation map and ε guarantees numerical stability.

Moreover, we study how different branches affect the final output and how their mutual impact changes in different scenarios.

Implementation details

At the training time for an individual branch, we randomly mirror the clips for the data augmentation. We operate Adam optimizer with parameters as in [32]. Except for the learning rate that would become 10^{-4}. Finally, the batch size was 32 and each branch was trained until convergence.

The dataset is split into train, validation, and test set as follows: sequences 1-40 are for training, sequences 41-74 for testing. The 500 frames in the middle of each training sequence form the validation set.

Furthermore, the complete multi-branch architecture was fine-tuned using the same cropping and data augmentation procedures to minimize the cost function in Eq.5. The batch size will be 4 because of GPU memory and learning rate value will be 10^{-1}.

Model analysis

The dataset has been taken under differing landscapes, weather conditions, and time of day, as shown in (Fig. 65) show that the prediction in highways is better than downtown, where the focus can move towards more distractors at downtown.

The model looks more reliable in evening situations, rather than morning or night, because of the better lighting conditions and lack of shadows, in rainy conditions that the human gaze is easier to model, possibly because of the higher level of awareness demanded to the driver and his consequent inability to focus away from the vanishing point.

To confirm the latter intuition, we measured the performance of the BM baseline (*i.e.* the average training fixation map), grouped for weather conditions. As expected, the DKL value in rainy weather (1.5) is significantly lower than the ones for cloudy (1.6) and sunny weather (1.7), because when rainy the driver is more focused on the road.

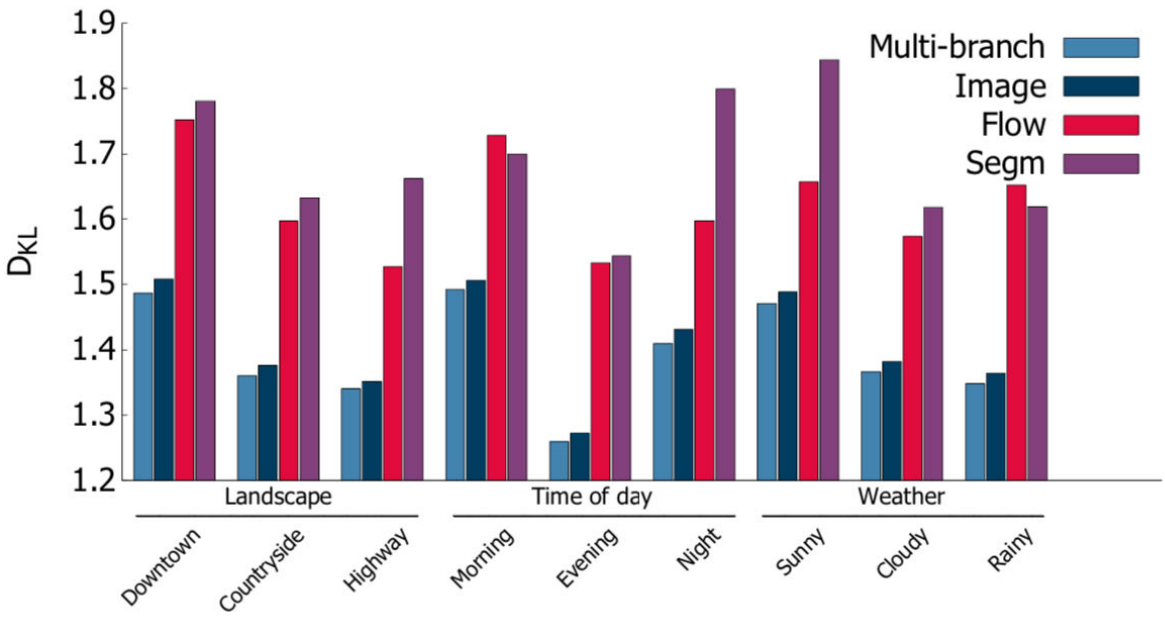


Figure 65: DKL of the different branches in several conditions.

Model results

Here we compare out model result with some other models (RMDN and MLNet)

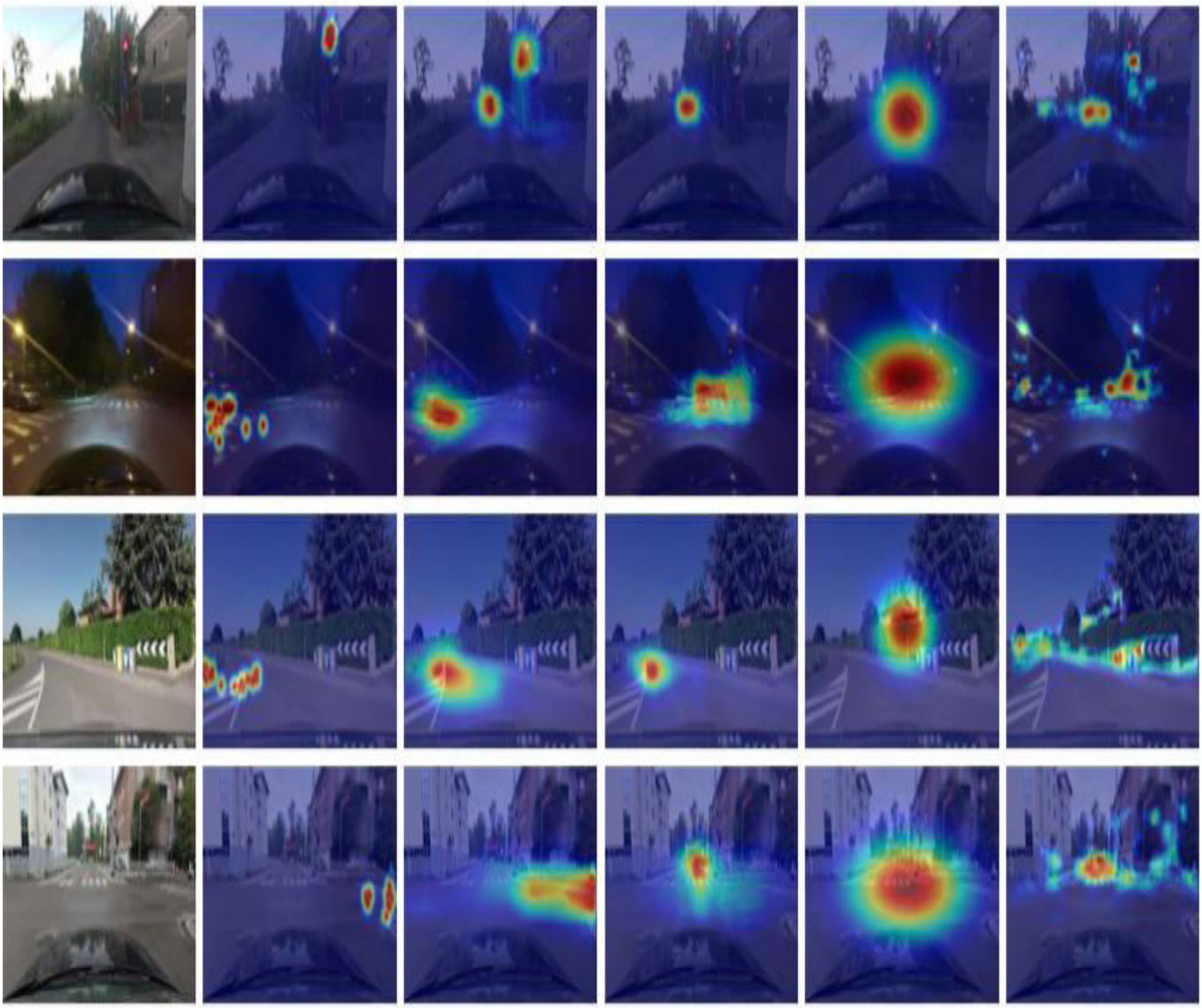


Figure 66: Results of fixation maps. From left to right: input clip, ground truth map, our prediction, prediction of the previous version of the model, prediction of RMDN and prediction of MLNet.

Chapter 6

Technologies Used In Training/Testing

Cloud Computing

"The cloud" refers to servers that are accessed remotely over the Internet, and the software, databases, applications that run on those servers. Cloud servers are located in data centers all over the world. By using cloud computing, users and companies don't have to manage physical servers themselves or run software applications on their own machines.

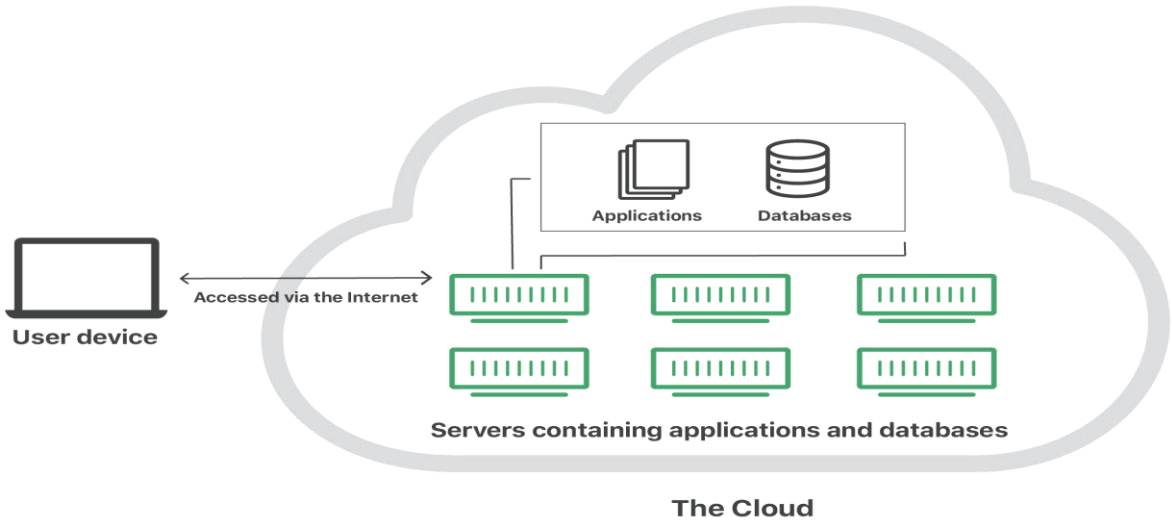


Figure 67: Basic Architecture of Cloud

Cloud computing is basically the on-demand availability of computer system resources. Clouds may be limited to a single organization (private clouds), or be available to many organizations (public cloud) or hybrid. Cloud computing relies on sharing of resources to achieve coherence and economies of scale.

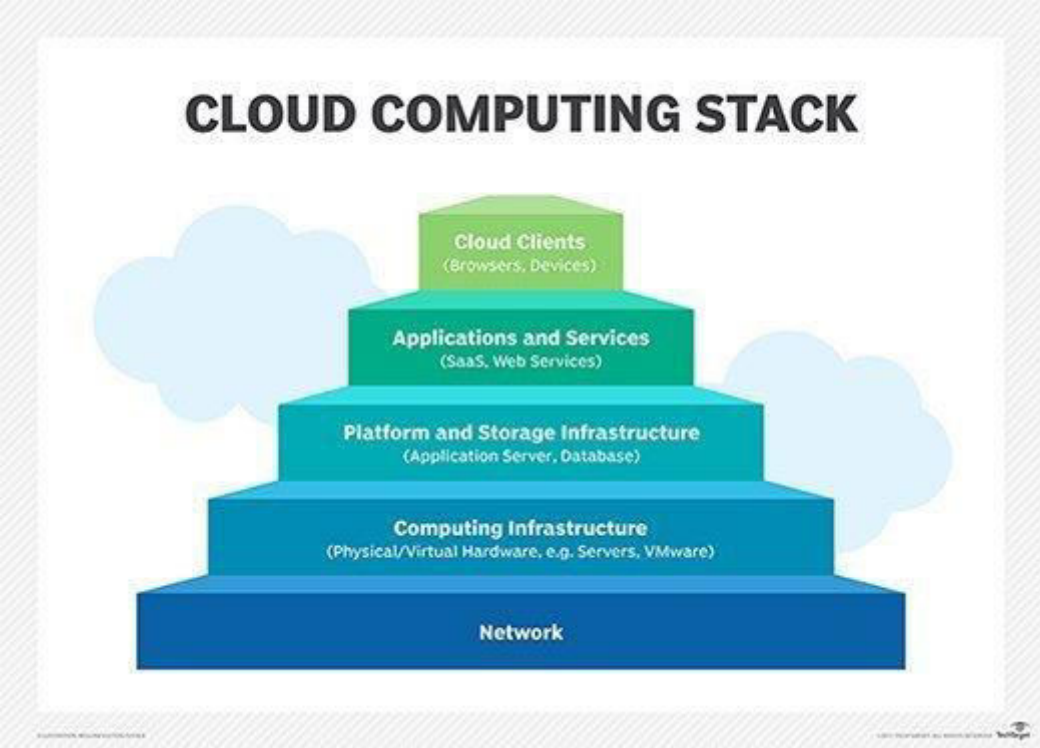


Figure 68: Cloud Computing Stack

So generally cloud provide to us a very big pool of resources and services that can be rented remotely and accessed via internet and pay as you go pricing.

It give companies many advantages as cost savings, no concern about maintenance, flexibility, disaster recovery, mobility, fast performance, high availability, scalability.

Google Colaboratory

Colaboratory is a Google research project created to help spread machine learning education and research. It's a Jupyter notebook environment that requires no setup to use and runs entirely in the cloud.

Most importantly, the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in the notebook.

So firstly, we try to train the model on colab as it’s free and we don’t have GPU but we face some issues:

- Model was written by Keras library with Theano backend but we figure out that colab GPU doesn't support Theano backend so we convert all the model code to Tensorflow backend that was supported.
- Storage issue as colab connected to google drive but free users have only 15gb storage but our dataset was 20gb+ so we buy bigger quota.
- GPU usage limitations as colab only allow free users to continuously use GPU for only 8 hours and suspend user from usage rest of the day so it was waste of time as model need to be trained at least for 20 continuous working days.

So a result of this issues we decide to change the training platform.

Amazon Web Services (AWS)

Amazon Web Services (AWS) is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis.

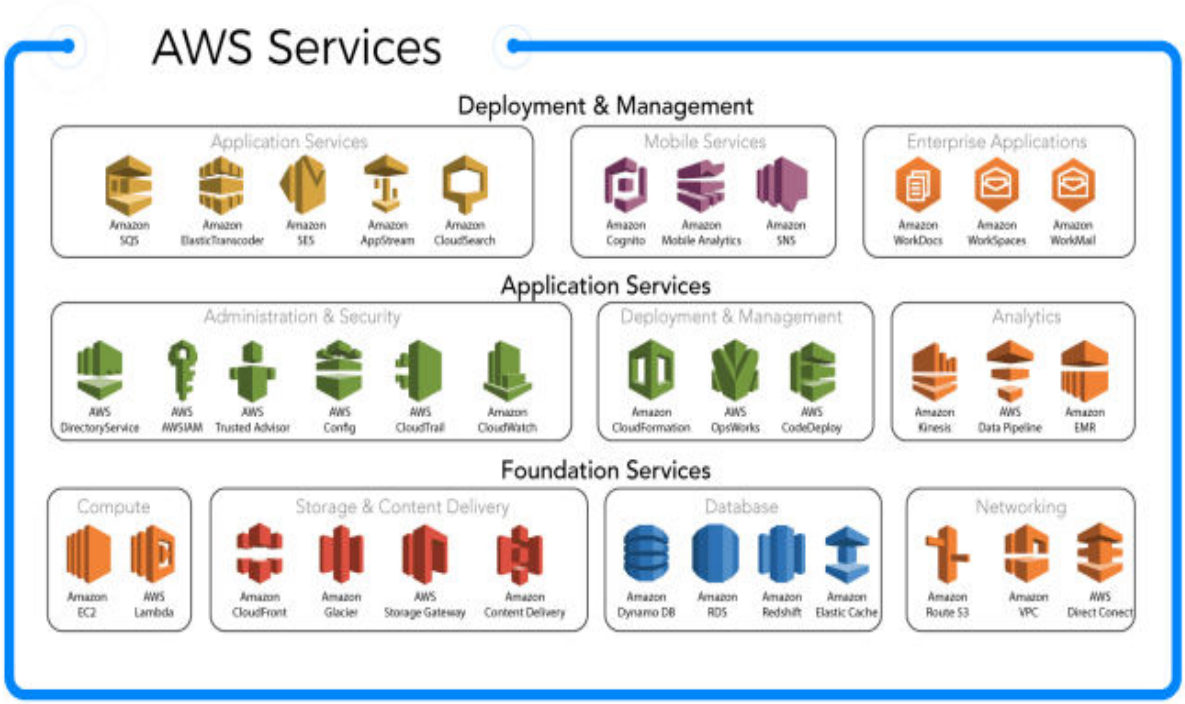


Figure 69 AWS Services

We get interact with only two services:

- Infrastructure as a service (IAAS), AWS Elastic Compute Cloud (EC2) by using education pack that amazon provide we can take 30$ for free for students, But it has many limitations as we can't rent machine with GPU only CPUs and for individual it's pricing is very high.
- Software as a Service (SAAS), we use Amazon SageMaker it's pretty similar to colab it provides notebooks with CPUs and GPUs but education pack again has some limitations on GPU usage and for individual it's pricing is very high.

So a result of this issues we decide to change the training platform.

Google Cloud Platform (GCP)

Google Cloud Platform (GCP), offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search, Gmail and YouTube. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning.

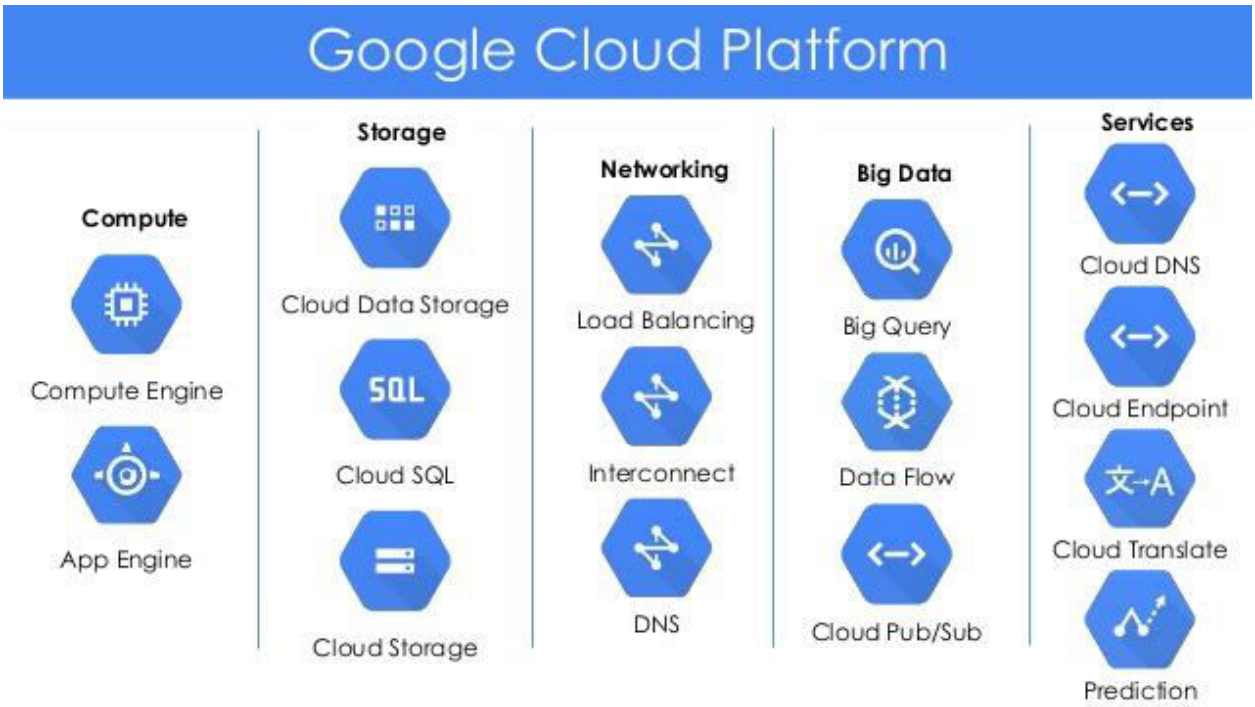


Figure 70: GCP Services

Finally, We settled down on google cloud platform as all issues we faced previously are solved as google cloud offers 300$ free credit on sign up, so money problem we faced on amazon web services solved.

We only interact with one service on google cloud, it's the Infrastructure as a service (IaaS) Cloud Compute it provide us with desired system specification so we rent an machine with following criteria:

- 24 vCPUs
- NVIDIA Tesla V100
- 70GB Ram
- 500GB SSD persistent disk
- Ubuntu 18.04 LTS operating system

That costs 2.35$ hourly.

We can access machine remotely via SSH and control it via terminal, but for sake of easiness we setup a GUI on the machine and access it via it's public ip and port (5901) using application called VNC Viewer.

Results

Example 1

Figure 71 Actual Frame

Figure 72 Ground Truth

Figure 73 Optical Flow Branch Output

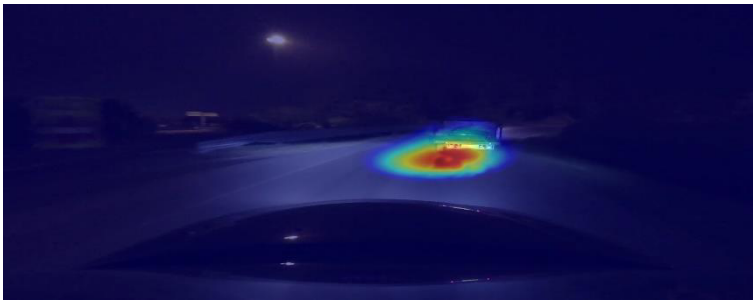


Figure 74 Segmentation Branch Output

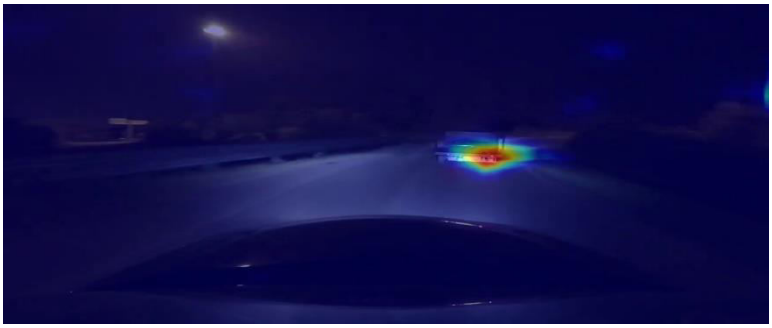


Figure 75 Final Output

Example 2

Figure 76 Actual Frame

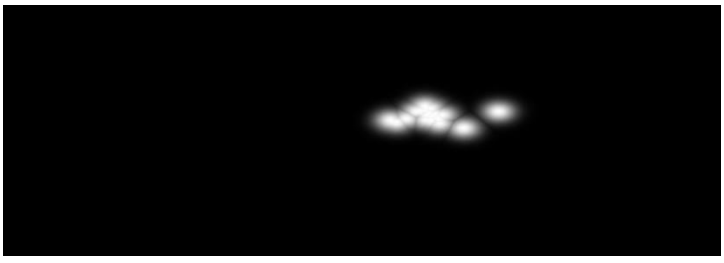


Figure 77 Ground Truth

Figure 78 Optical Flow Branch Output

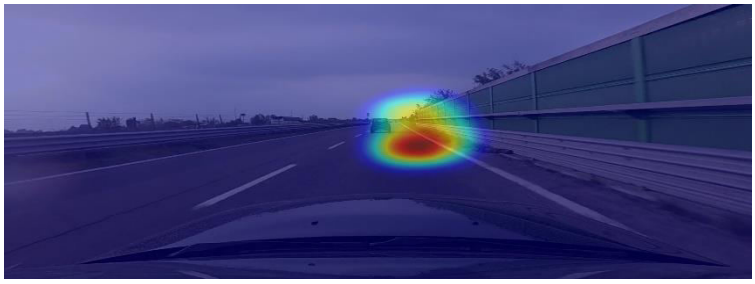


Figure 79 Segmentation Branch Output

Figure 80 Final Output

Example 3

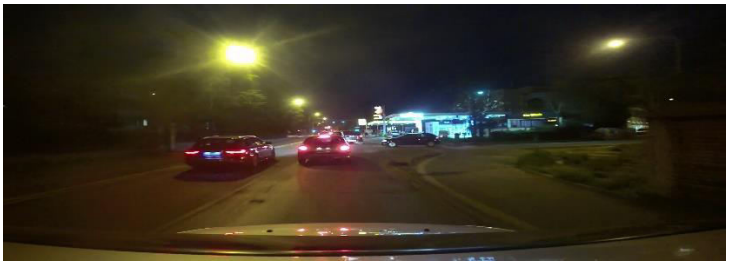


Figure 81 Actual Frame

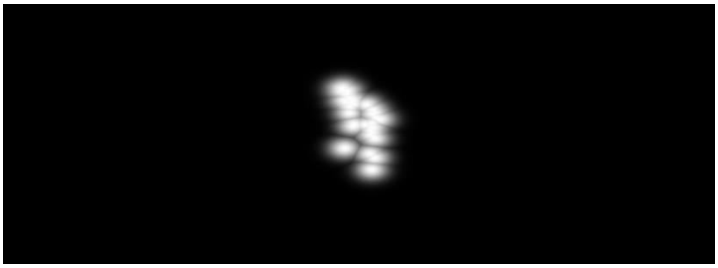


Figure 82 Ground Truth

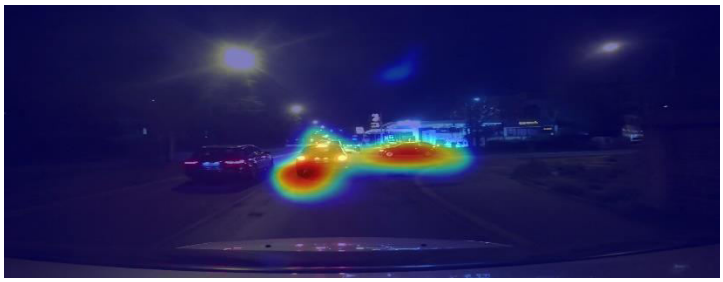


Figure 83 Optical Flow Branch Output

Figure 84 Segmentation Branch Output

Figure 85 Final output

Conclusion

This project presents an implementation for a multi-branch deep architecture that integrates three sources of information: raw video (RGB), motion (optical flow) and scene semantics (semantic segmentation) aiming to predict the driver's focus of attention.

Our main contribution is mainly in three points:

1. Changing the back-end programming language from Theano to Tensorflow.
2. Replacing the optical flow algorithm with a different one with better performance
3. Replacing the semantic segmentation network with a lightweight network that has better performance all of the above modifications aimed to make the response of the model as fast as possible or even near to the real-time response.

Finally; this model is able to predict the driver's focus of attention in real-world driving sequences. Because the model's input is car-centric videos only, it might be integrated with already adopted ADAS technologies.

References

D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In 2015 IEEE International Conference on Computer Vision (ICCV), Dec 2015.

L. Bazzani, H. Larochelle, and L. Torresani. Recurrent mixture density network for spatiotemporal visual attention. In International Conference on Learning Representations (ICLR), 2017

M. Cornia, L. Baraldi, G. Serra, and R. Cucchiara. A Deep Multi-Level Network for Saliency Prediction. In International Conference on Pattern Recognition (ICPR), 2016

A. Palazzi, F. Solera, S. Calderara, S. Alletto, and R. Cucchiara. Where should you attend while driving? In IEEE Intelligent Vehicles Symposium Proceedings, 2017.

https://people.csail.mit.edu/celiu/OpticalFlow/

T. Brox, A. Bruhn, N. Papenberg, and J.Weickert. High accuracy optical flow estimation based on a theory for warping. In European Conference on Computer Vision (ECCV), pages 25–36, 2004.

A. Bruhn, J.Weickert and C. Schn¨orr. Lucas/Kanade meets Horn/Schunk: combining local and global optical flow methods. International Journal of Computer Vision (IJCV), 61(3):211–231, 2005.

C. Liu, W. T. Freeman, E. H. Adelson and Y. Weiss. Human-assisted motion annotation. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1-8, 2008.

Computer Vision for Visual Effects 1st Edition by Richard J. Radke

Aaron Bobik, Irfan Essa and Arpan Chakraborty. Georgia Tech. Introduction to computer vision. Udaciy. https://www.udacity.com/course/introduction-to-computer-vision--ud810

OpenCv documentation

Fisher Yu and Vladlen Koltun: Multi-Scale Context Aggregation by Dilated Convolutions.

https://www.quora.com/In-machine-learning-what-does-multiscale-feature-learning-mean-such-as-that-used-in-deep-belief-network-What-scale-does-it-refer-to

http://bmvc2018.org/contents/papers/0494.pdf

https://arxiv.org/pdf/1611.06612.pdf

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

https://courses.cs.washington.edu/courses/cse576/17sp/notes/Sachin_Tal