TIPANGLE: Accurate Pan and Tilt angle determination of Traffic Cameras

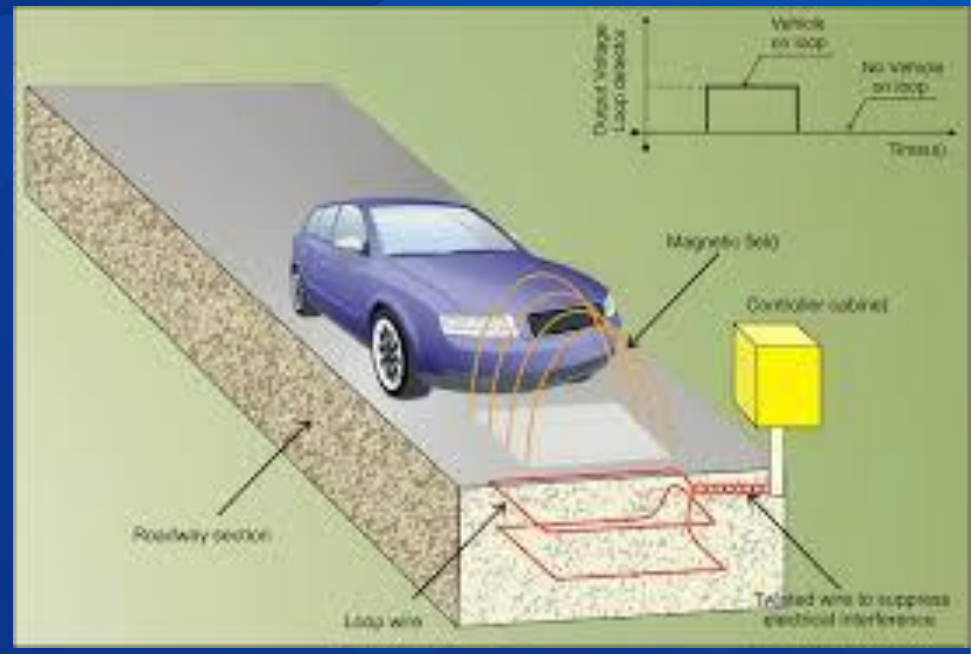
Shreehari Jagadeesha, Edward Andert, and **Aviral Shrivastava**

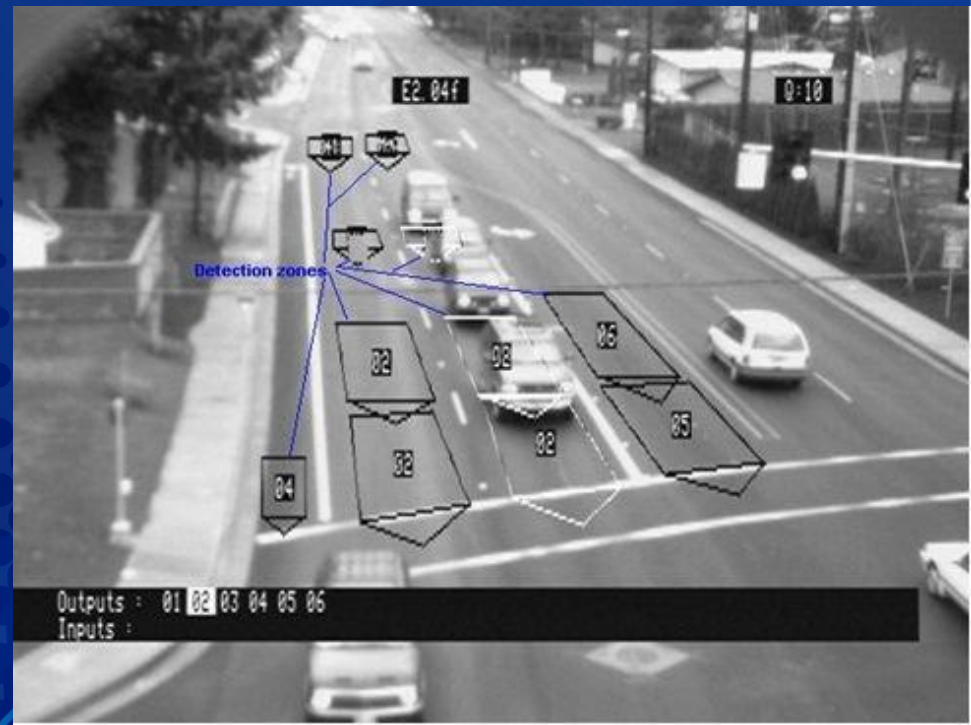
Arizona State University

Camera-based Traffic Monitoring

- Main use – Sensor for traffic lights
  - Previously used inductive loops
  - Installation and maintenance is intrusive
  - Damaged by road damage
  - Cannot differentiate between vehicles and cannot detect light vehicles or pedestrians
- Traffic cameras
  - Traffic sensing for signal timing
  - Accident detection and verification, public safety
  - Improve emergency response
  - Traffic law violation, movement of perpetrator
- No of traffic cameras are on the rise

- More than 400 cameras in Arizona.
- More than 300 traffic cameras in Bengaluru.



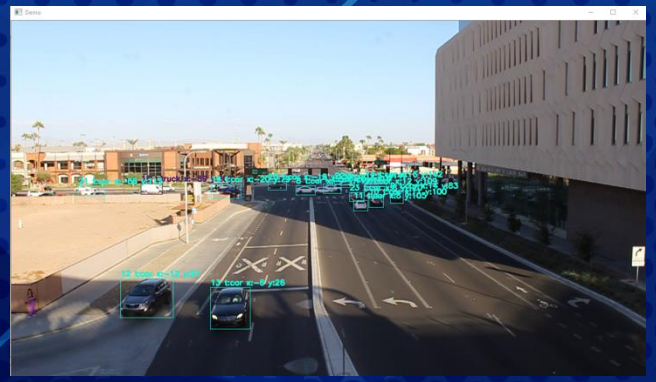


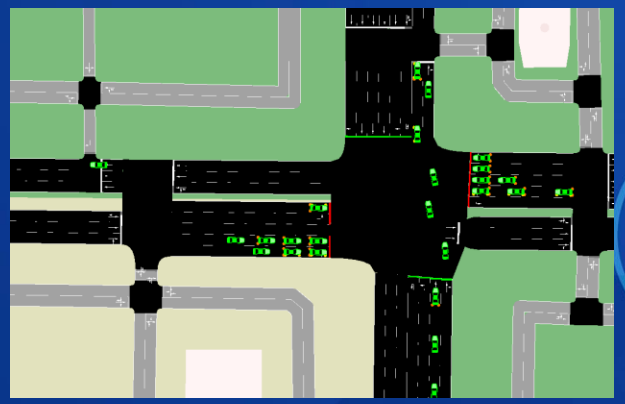
Pan and Tilt Cameras

- Pan and Tilt and Zoom cameras (PTZ) provide
  - Remotely controlled to pan horizontally, tilt vertically, and zoom in or out, providing a much wider field of view than fixed cameras.

- Track moving objects
- Detailed observation
- Incident verification
- Event monitoring

Fine-grain Traffic Mapping





- Camera-based traffic monitoring can enable fine-grain traffic modeling
  - Means we can track each vehicle, its position, its type, its trajectory etc.
  - As opposed to the classical flow-based traffic modeling
  - This allows for more accurate traffic modeling
  - Accounts for diversity in vehicles, model individual vehicle behavior

Project with Arizona Department of Transportation (ADOT) to map live traffic onto traffic Simulator

As a part of that project, we got access to direct camera feed from 4 cameras in the City of Tempe near ASU.

All of them were PT cameras (no zoom).

Challenge: Changing camera pose!!

- Traffic monitoring personal may have changed the pose to track the construction at a site and might have forgotten afterwards.
- Sometimes - natural causes such as wind and rain.

- Localization of vehicles on the map necessitates continuous calculation of camera pose to reduce errors.
- PTZ traffic cameras often lack sensors in them to determine their pose with respect to the world in turn challenging their data venerability.
  - New PTZ cameras enable querying for PTZ metadata through Pelco and VISCA protocols.

Existing Approaches and their Limitations

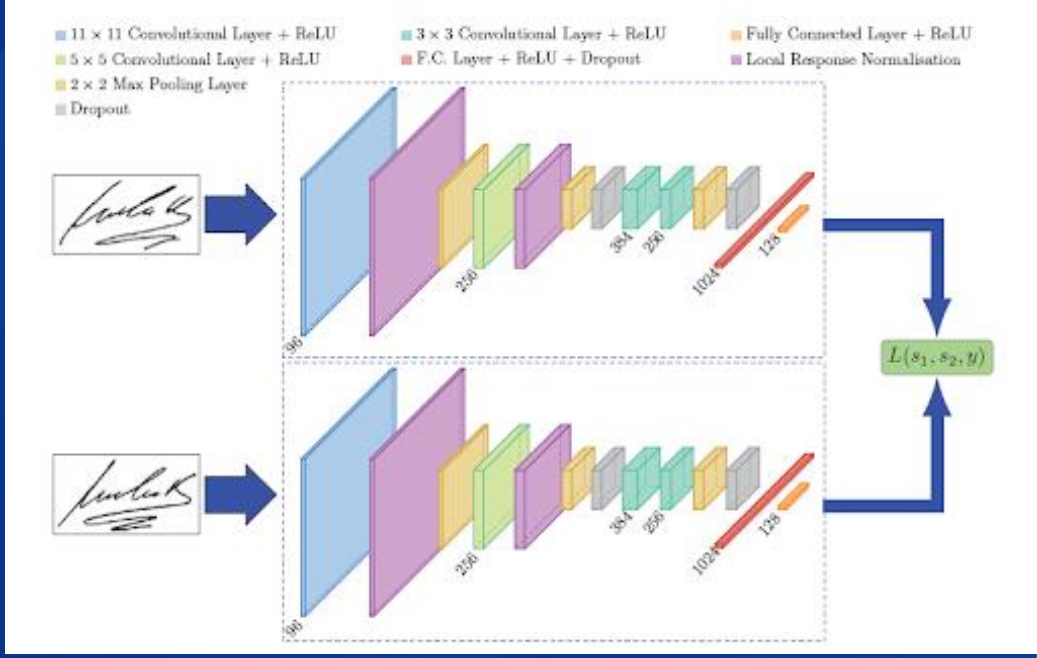
- M. Pollefeys, R. Koch and L. Van Gool, "Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters," *Sixth International Conference on Computer Vision, 1998.*
  - The pose is estimated by computing the focal length and optical center
  - This approach is computationally intensive procedure involving determination of focal length and optical center.
- K. . -T. Song and J. . -C. Tai, "Dynamic Calibration of Pan–Tilt–Zoom Cameras for Traffic Monitoring," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 5, pp. 1091-1103, Oct. 2006
  - Uses images of lane markings on road which is later fed to DNN for pose estimation.
  - Not applicable at intersections lacking lane markings.
- H. Nagayoshi and M. Pollefeys, "Estimating Camera Pose Using Trajectories Generated by Pan-Tilt Motion," 2014 2nd International Conference on 3D Vision, Tokyo, Japan
  - Tracks using a marker placed on target camera being observed by other cameras.
  - Requires the use of multiple high-definition cameras and target marker.
- H. Dinh and H. Tang, "Camera calibration for roundabout traffic scenes," 2014.
  - Uses the elliptical equation on the roundabout to determine the pose of the camera.
  - Only applicable to roundabouts.

Problem Statement

- Automatically determine the Pan and Tilt of a PTZ camera, while preferably:
  - The solution is not camera-dependent, i.e., you do not need to know the intrinsic camera properties i.e., focal length and optical center.
  - Does not require additional hardware, e.g., IMU or encoder units embedded in the camera.
  - Does not depend on the lane markings painted on the road.
  - Does not require the use of multiple cameras and markers placed on target camera.
  - Works for all intersections and is not only for roundabouts.
  - Is computationally efficient.
  - Requires minimum calibration.

Solution Overview

- Cannot use a CNN/DNN, since they are extremely computation intensive – we want this to work on a microprocessor – Raspberry Pi.



- We use a Siamese Neural Network (SNN)
  - Efficient and effective for recognition systems
  - E.g., face and signature recognition over a small-ish set of people.
  - SNN just checks if two images are similar
  - Requires less training
  - Computationally efficient

- Main Idea
  - Take images at all pan and tilt angles during calibration
  - During operation phase, check which pan and tilt image does the given image most resembles.

Step 1 - Image Collection - 1

Pan 7.5, Tilt – 0.5

Pan 7.5, Tilt – 3

Pan 7.5, Tilt – 5.5

Pan 7.5, Tilt – 8

Pan 7.5, Tilt – 10.5

Pan 0, Tilt – 0.5

Pan 5, Tilt – 0.5

Pan 15, Tilt – 0.5

Pan 25, Tilt – 0.5

Pan 30, Tilt – 0.5

MPS-LAB
Make Programming Simple

Step 1 - Image Collection - 2

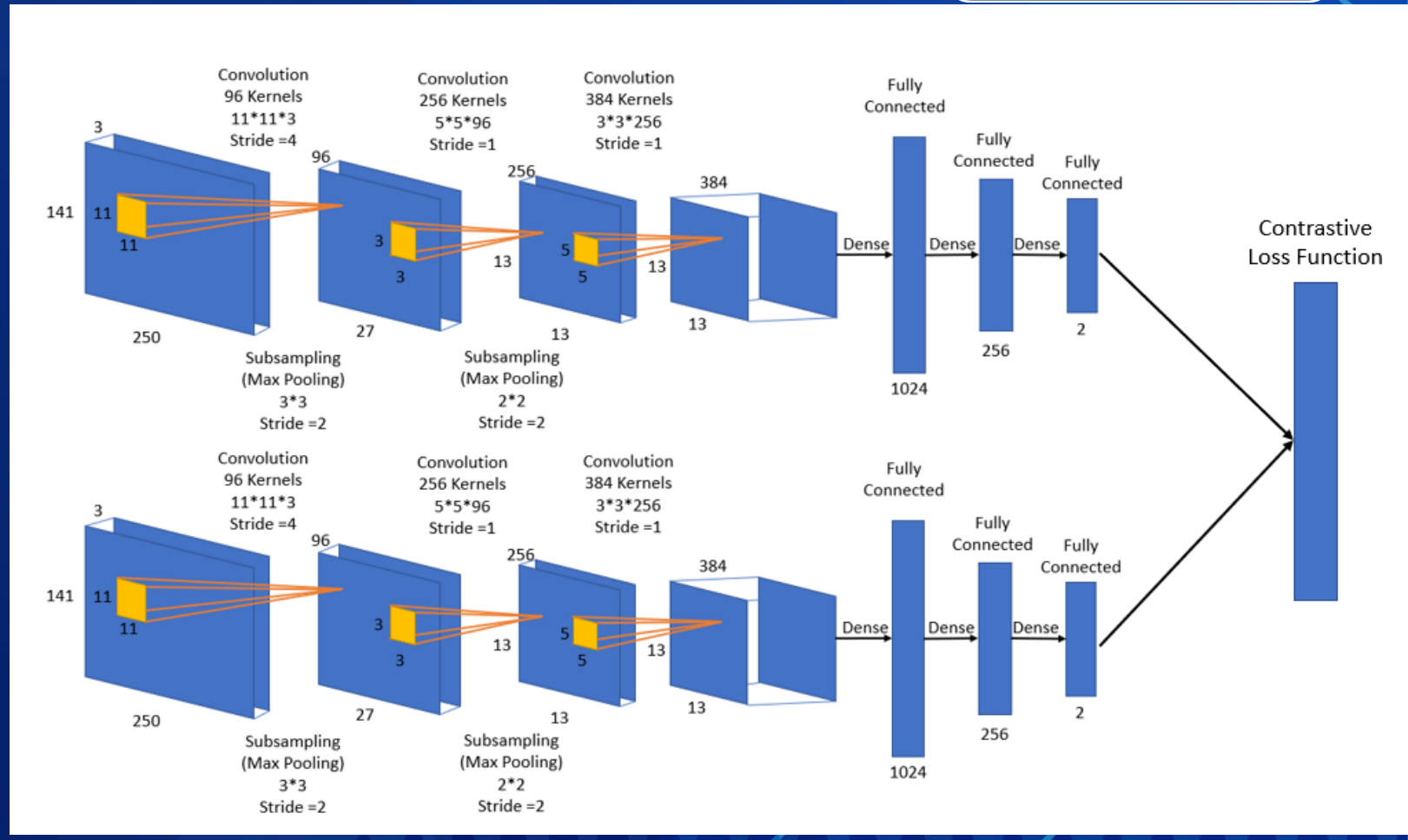
- In total, collected 165 images for training
  - The Pan angle varies from 0° to 30° with a step size of 2.5°
  - The Tilt angle varies from -0.5° to -10.5° with the same step size of 2.5°
  - Morning, evening, and Night
  - Over 3 days
- *Additionally collected 90 images at random angles for testing

Camera setup for image collection

0,-0.5	2.5,-0.5					15,-0.5					27.5,-0.5	30,-0.5
0,-3.0	2.5,-3.0					15,-3.0					27.5,-3.0	30,-3.0
0,-5.5	2.5,-5.5					15,-5.5					27.5,-5.5	30,-5.5
0,-8.0	2.5,-8.0					15,-8.0					27.5,-8.0	30,-8.0
0,-10.5	2.5,-10.5					15,-10.5					27.5,-10.5	30,-10.5

Step 2 – Train the Siamese Neural Network

- Used contrastive loss between two image embeddings.
- This loss function is used to learn embeddings in which two similar points have a low Euclidean distance and two dissimilar points have a large Euclidean distance.
- Distance between images embeddings from different days and time, but same pan and tilt should be low.



$$(1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m - D_W)\}^2$$

Used Siamese Neural Network Parameters

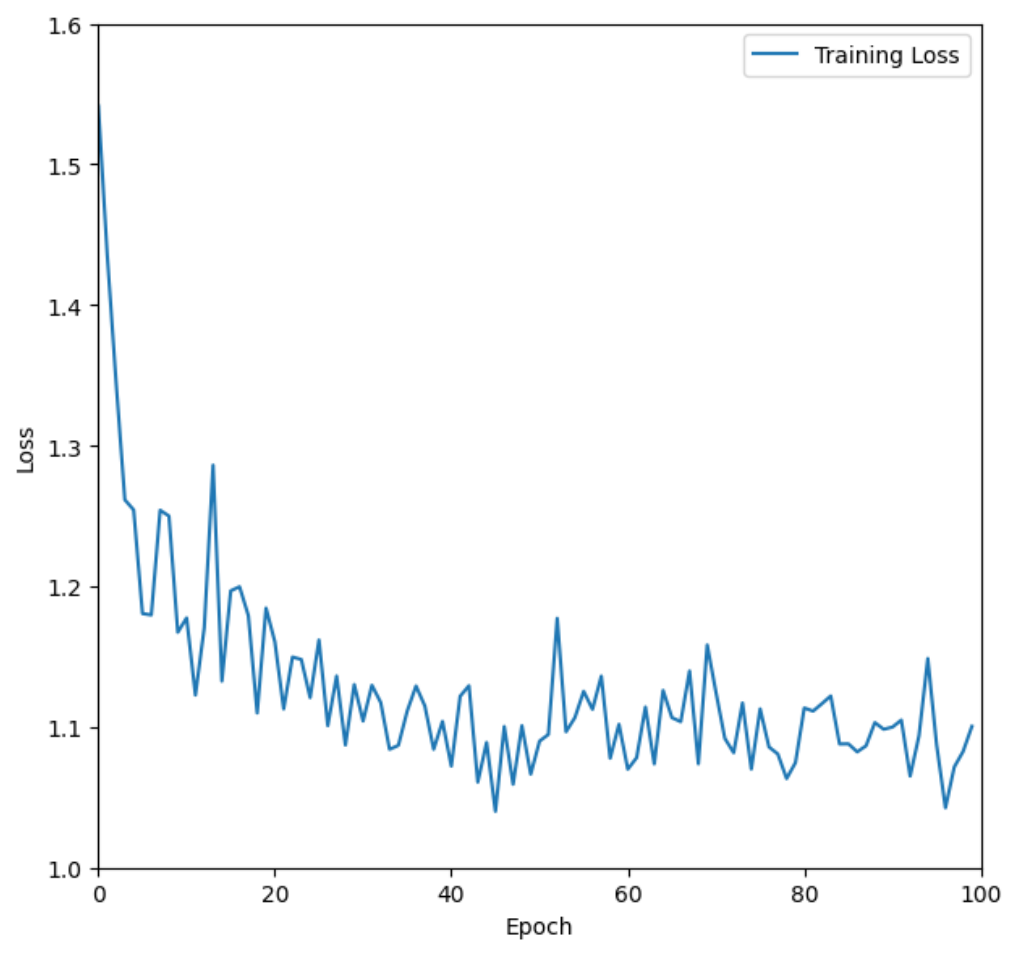
SIAMESE NEURAL NETWORK LAYER DETAILS.

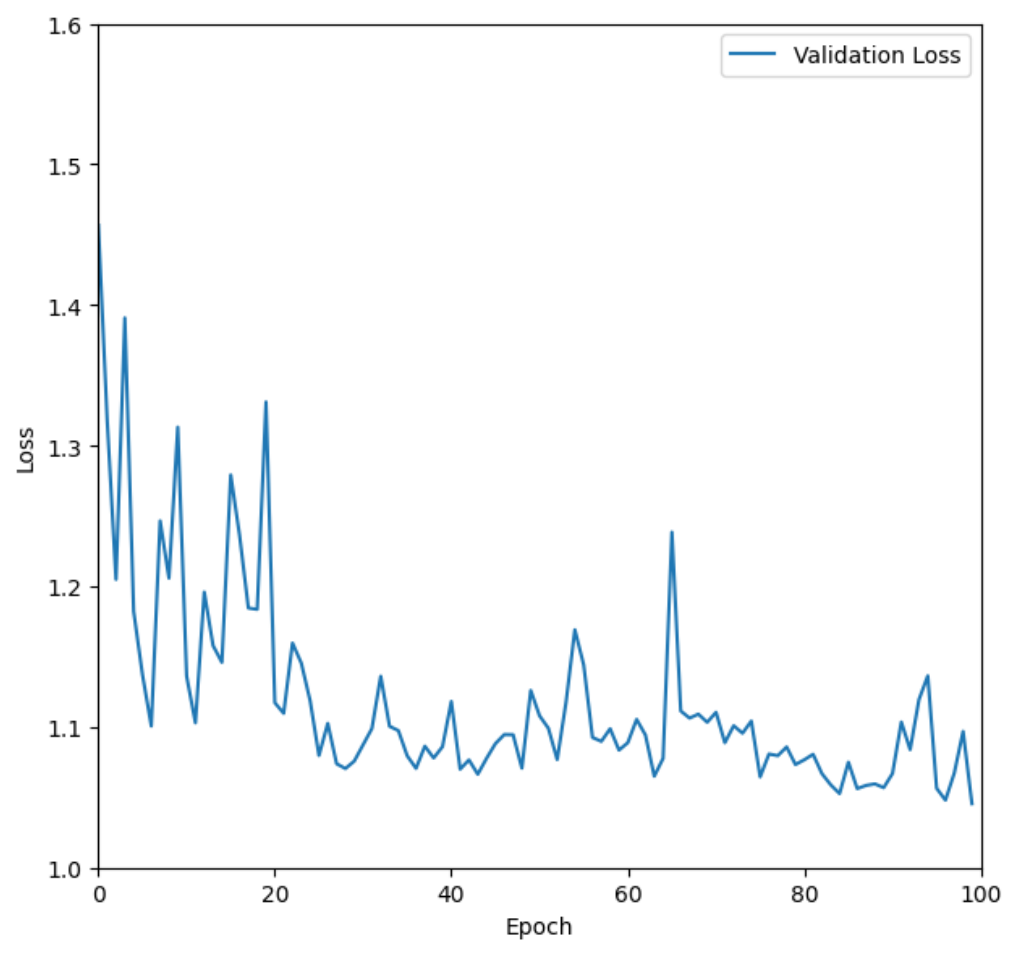
#	Layer Type	Kernel	Stride	Input Channels	Output Channels
1	Convolutional	11x11	4	1	96
2	Max Pooling	3x3	2	-	-
3	Convolutional	5x5	1	96	256
4	Max Pooling	2x2	2	-	-
5	Convolutional	3x3	1	256	384
6	Linear	-	-	384	1024
7	Linear	-	-	1024	256

TRAINING PARAMETERS

Parameters	Values
Activation Function	ReLU
Learning Rate	0.0005
Optimizer	Adam Optimizer
Loss Function	Contrastive Loss
Loss Margin	2.0
Batch Size	64
Epoch Count	100

Training Results



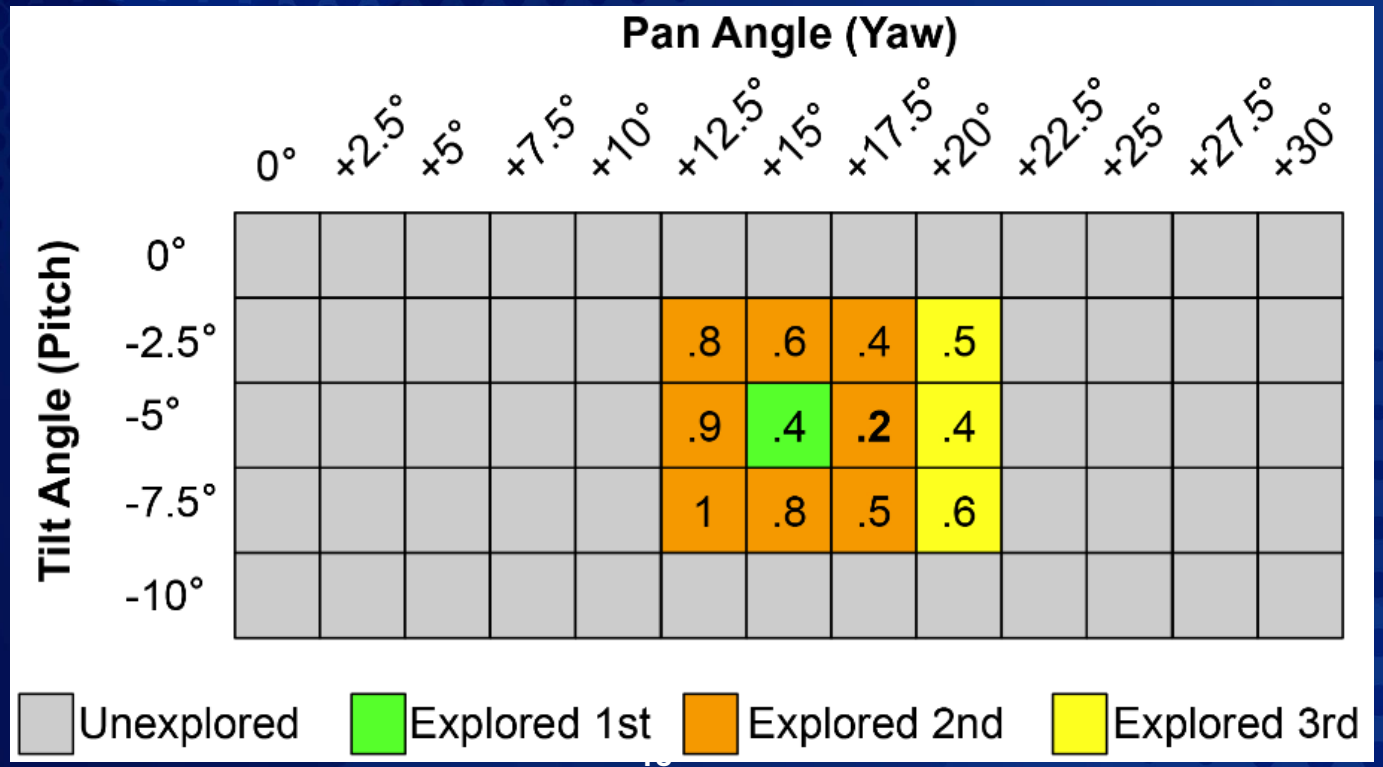


Step 3 - Inference

- The same 195 images collected for the training are used as search space images.
- An image is provided to the network for determining its pose.
- Euclidian Distance between Input image embedding, and search space images embeddings is calculated.
- The search space image embedding having least Euclidian Distance with input image embedding would be the nearest matching image.
- PAN and TILT angle from the nearest matching image is obtained as the solution.

Step 3 – Gradient Descent for Fast Inference

- The search space consists of 195 images.
- Traversing all the images iteratively for inference is inefficient.
- Gradient Descent is used to make the inference procedure faster.

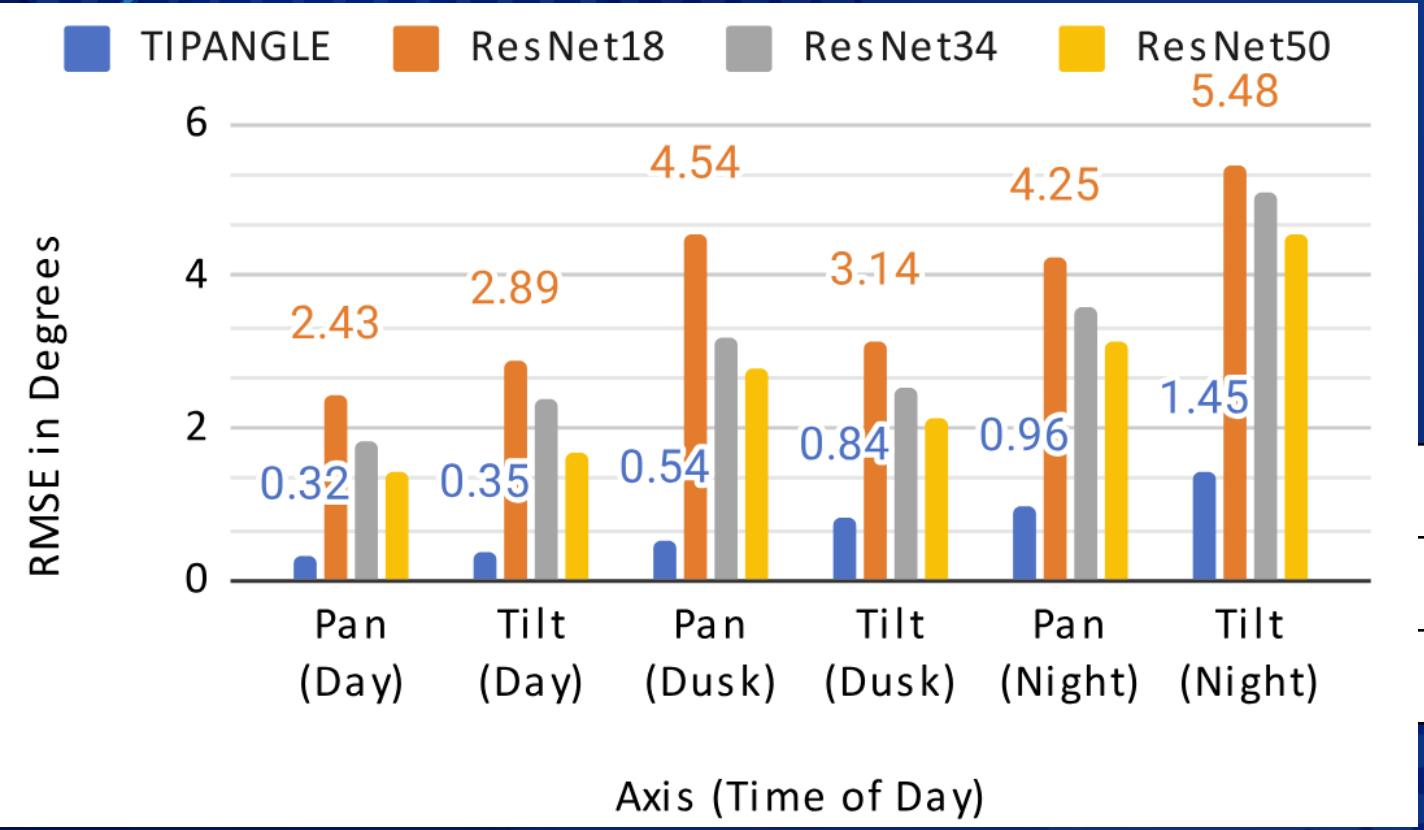


Algorithm 1: Gradient Descent Algorithm

Input : lastPose, imageGrid
Output: pose

```
smallestDistance ← ∞ localMinima ← None
  unexploredPositions[]
  unexploredPositions.append((gridPosition(lastPose),
  inferDistance(lastPose)))
while True do
    unexploredPositions.Sort(using [1])
    testGridPosition, testDistance =
     unexploredPositions[].pop()
    if testDistance < smallestDistance then
        localMinima ← testGridPosition
    if testDistance < (2*smallestDistance) then
        for neighbors of testGridPosition do
            unexploredPositions.append(gridPos(neighbor)
             , inferDistance(neighbor))
    if length(unexploredPositions) == 0 then
        return LocalMinima
```

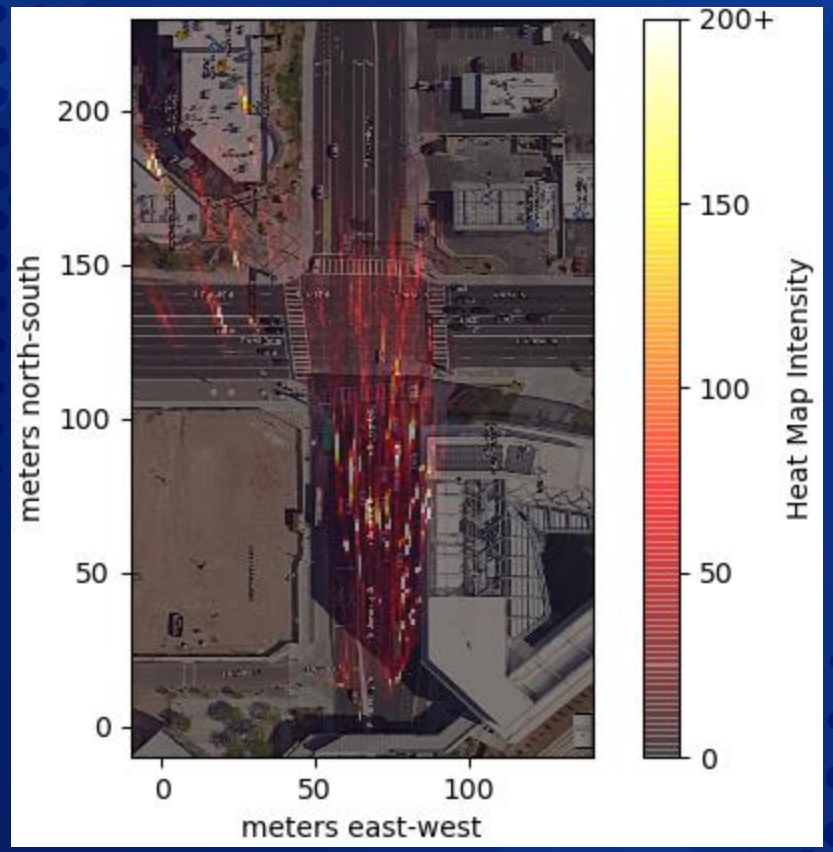
TIPANGLE predicts PAN and TILT angles 4X more accurately and 3x faster than DNN–based approaches



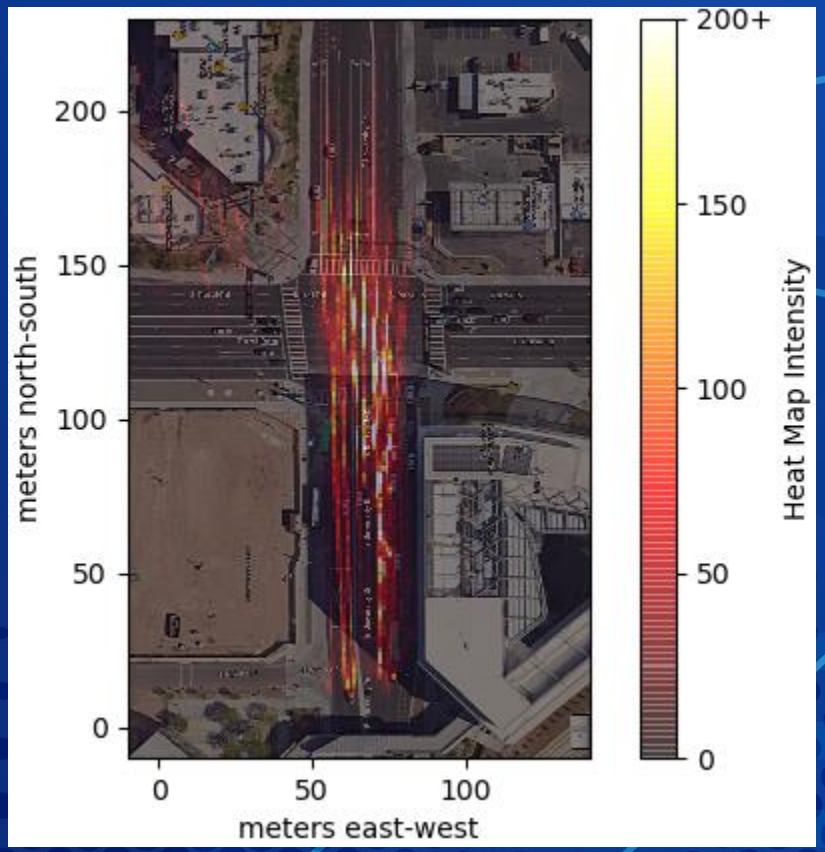
Approach/Time	TIPANGLE	RESNET18
Training Time	6:48 Minutes	37:26 Minutes
Inference Time	0.829s	3.713s

Using the Pose for Localization

Bird's eye view of the Intersection

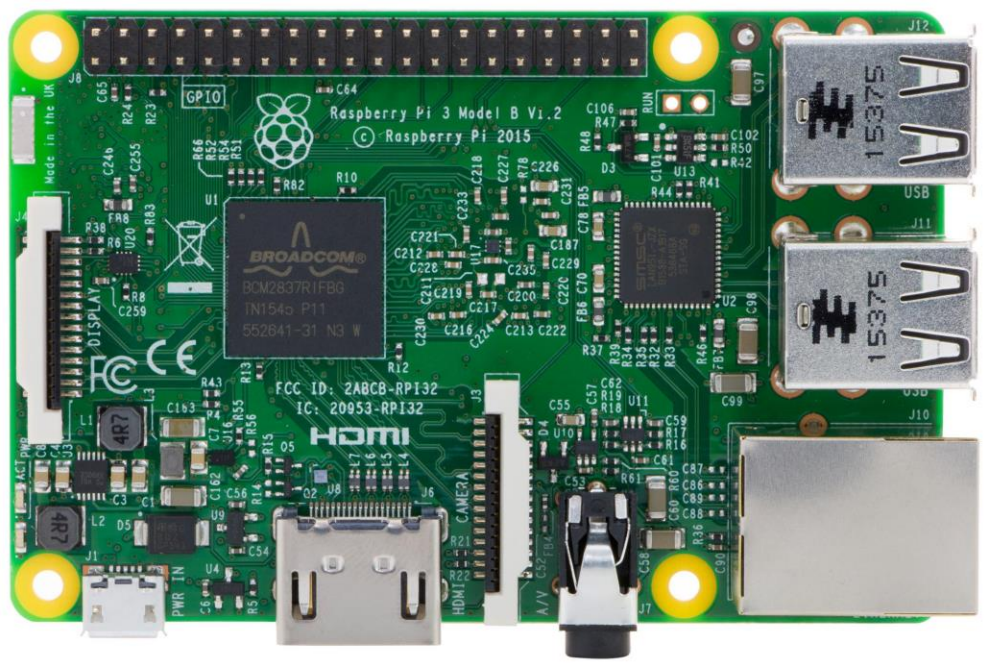


Without TIPANGLE



With TIPANGLE

Embedded Implementation and Optimizations



- Implemented on Raspberry Pi3
- Pruning and Parallelization reduces Inference Time by 1/3rd :
  - Pruning refers to removal of unnecessary nodes whose weights are not helpful in inference.
  - The trained model is pruned for faster inference execution (60.43s to 54.28s).
  - Once pruned the model size is also reduced by 1/3rd.
  - Parallelization on 4 cores is exploited for faster inference (54.28s to 38.18s).

- Clock frequency: 1.2 GHz.
- Chipset (SoC): Broadcom BCM2837.
- Processor: 64-bit quad-core ARM Cortex-A53.
- Memory (SDRAM): 1 GB LPDDR2.

Summary

- Camera-based traffic sensing is replacing induction-loop based flow estimation.
- Just Bengaluru has more than 300 traffic cameras.
- Cameras enable fine-grain traffic analysis by mapping current traffic onto a simulator.
- However, camera angles can change due to various reasons and destroy the ability of cameras to localize vehicles.
- TIPANGLE accurately determines the PAN and TILT angle of a PTZ Traffic Cameras while,
  - Avoiding the need for estimating Intrinsic camera properties i.e., focal length and optical center.
  - Does not require IMU or Encoder units embedded in the camera.
  - Does not depend on the lane markings painted on the road.
  - Does not require the use of multiple cameras and markers placed on target camera.
  - Is ubiquitous and not specific to only roundabouts.
  - Performs 4x better on accuracy and 3x faster in time when compared to ResNet18
- Deployed on embedded system i.e., Raspberry Pi 3 show the light nature of the approach
  - Gradient descent approach used reduce inference time by 2x
  - Pruning and parallel execution of the inference reduces inference time 1/3rd