Alma Mater Studiorum - University of Bologna

Computer Science and Engineering - DISI

Artificial Intelligence

A study on tackling visual odometry by a transformer architecture

Master degree thesis

Supervisor

Prof. Luigi Di Stefano

Co-supervisor

Luca De Luigi

Candidate

Xiaowei Wen

Academic Year 2021-2022- Second session

Dedicated to my parents.

Summary

This dissertation describes a deepening study about Visual Odometry problem tackled with transformer architectures. The objectives were: create a synthetic dataset using BlenderProc2 framework, try different kind of transformer architectures which includes: ResNet feature-extractor with encoder and a small MLP, ResNet feature-extractor with encoder-decoder and a MLP, ResNet-feature extractor with encoder-decoder and pose Auto-encoder.

“Dio benedica quelle persone che quando incroci il loro sguardo per sbaglio, sorridono.”

Thanks

First, I would like to express my deepest gratitude to Professor Luigi Di Stefano and Luca De Luigi for the guidance and support during the internship

Second, I would like to thank my parents for their moral support and for their patience.

Third, I would like to thank my girlfriend and friends for being patient with me and to put up with my complaining.

Bologna, 06 October 2022 Xiaowei Wen

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this section we will present the summarized content of the whole thesis.

1.1 Background

In recent years, the field of computer vision has been growing in complexity and the number of applications has been increasing, main applications are: image classification, object detection, face recognition, image segmentation, Simultaneous Localization and Mapping (SLAM) and visual odometry which is a task in which the robot is able to understand where it is and how it is oriented.

The development of computer vision has been a long process, the growth is favoured by the development of new hardware components and new challenges, about the latters, we have CIFAR-10 (Doon et al. [3]), Fashion-MNIST(Xiao et al. [14]), MS-Coco (Lin et al. [9]) and ImageNet (Deng et al. [2]). These datasets are often used as benchmark for novel models.

For the architectures, starting from AlexNet (Krizhevsky et al. [8]), then VGG (Simonyan et al. [10]), Inception-V1 (Szegedy et al. [11]), Inception-V2(Szegedy et al. [12]), ResNet (He et al. [7]), etc., the complexity of the models has increased enormously. Each of these models introduced some innovations and improved the performance on the benchmarks, for example:

- AlexNet introduced the concept of the *convolutional neural network* (CNN) and use of the separation of the models into two different GPUs.
- VGG introduced the concept of stage, which repeated more times, composes the model.
- Inception-V1, Inception-V2 and V3 which are based on the concept of *inception*

module which was composed by different paths that the input has to go through to reach the output.

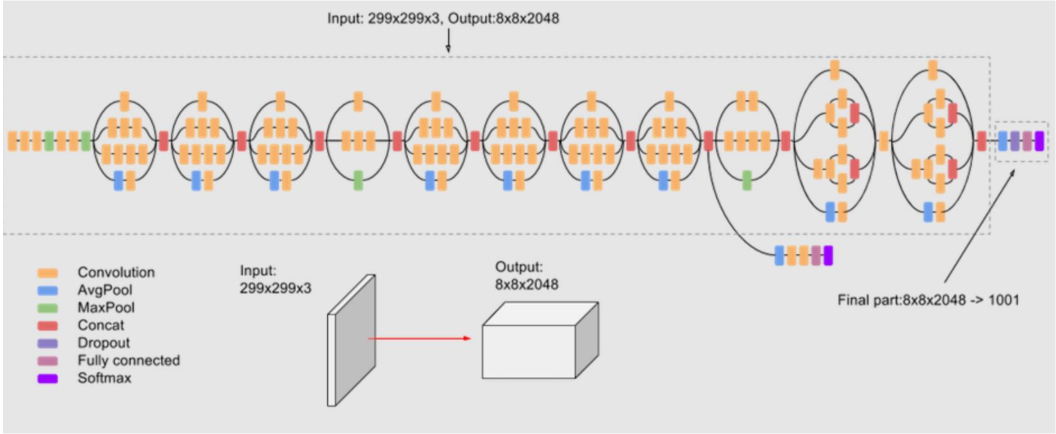


Figure 1.1: Inception V3 Structure

- ResNet is a model that is based on the concept of *residual network* which is composed by several blocks of the same type with the skip connections:

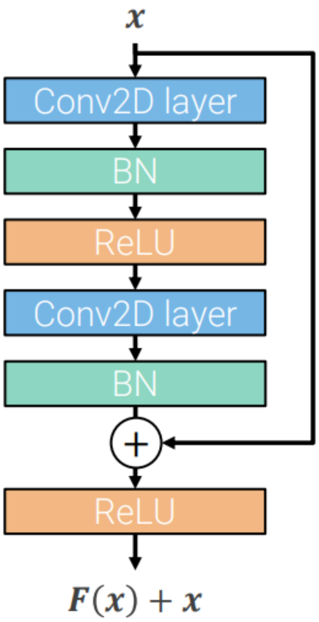


Figure 1.2: Skip connection

Basically, the input of the block is added to the output before feeding it to the next block, in this way, we can avoid the vanishing gradient problem making easier the training process.

After this, the computer-visionists lend the Transformer architecture (Vaswani et al. [13] from Natural Language Processing (NLP), bringing up ViT (Dosovitskiy et al. [4]) which is based on the Multi-Head Attention (MHA) mechanism. A multi-head attention is a module for attention mechanisms which runs through an attention mechanism several times in parallel. In this way, the model can attend to the different parts of the input, forming, in this way, the cross-attention over different parts of the input.

1.2 Problem

The term "odometry" originated from two Greek works *hodos* (meaning "journery" or "travel") and *metron* (meaning "measure"). This derivation is related to the estimation of the change in a robot's pose (translation and rotation) over time. Mobile robot use data from motion sensor to estimate their position relative to their initial location, this is called odometry. VO is a technique used to localize a robot by using only a stream of images acquired from a single or multiple camera. There are different ways to classify the typology of Visual Odometry:

- based on the camera setup:
  - Monocular VO: using only one camera;
  - Stereo VO: using two cameras;
- based on the information:
  - Feature based method: which extracts the image feature points and tracks them in the image sequence;
  - Direct method: a novel method which uses the pixel intensity in the image sequence directly as visual input.
  - Hybrid method: which combines the two methods.
- Visual inertial odometry: if a Inertial measurement unit (IMU) is used within the VO system, it is commonly referred to as Visual inertial odometry.

We can represent the pose in different ways, for example: **euler angles**, **quaternions**, **SO(3)**, **rotation matrices** combined with **translation vectors**.

The goal is to create a Neural network (NN), using a **ResNet** to extract features from images and the **transformer** presented by Vaswani et al. [13], which is able to estimate a sequence of camera's pose given a sequence of images.

1.3 Solution

To solve the problem of visual odometry, we tried different approaches to feed the data into the model, and to construct the model itself. We tried following approaches to feed the data:

- Feeding the sequence into the model directly and presenting the pose as *euler angles*.
- Feeding the sequence into the model directly and presenting the pose as *rotation matrix* so with twelve numbers and *translation vector*.
- Feeding the sequence into the model where the first frame is the origin of the reference frame and presenting the pose as *euler angles*.
- Feeding the sequence into the model where the first frame is the origin of the reference frame and presenting the pose as *rotation matrix* and *translation vector*.
- Feeding the sequence into the model where the first frame is the origin of the reference frame, and using the auto-regressive model to predict the pose.

We used these input strategies to feed the sequence, we tried different variants of models:

- Using the different version of ResNet (producing embedding of size 512 and 2048) model as feature extractor attached to the transformer with only encoder part.
- Using a Multi Layer Perceptron (MLP) for the images divided into patches to extract the features, and concatenating all the features as a single embedding then feed it into the only-encoder version of transformer.
- Using a MLP for the images divided into patches to extract the features, and concatenating all the features as a single embedding then feed it into the encoder-decoder version of transformer.
- Using the different version of ResNet model as feature extractor attached to the transformer with encoder and decoder.
- The same model as the previous but implemented in auto-regressive mode.

1.4 Results

1.4.1 Full sequence prediction

With different models a variety of results are achieved, but the most important result is that an important baseline for the future development of this kind of systems has been settled down. After a few trials with few models, which produced some circular trajectories, with the encoder-only version of transformer, and feeding the *sequence 3* of Kitti (for more details § 3.1) where the first image is considered as origin, we showed that the model is able to learn a single sequence in over-fitting, but fails when trying to over-fit a more complex sequence.

The encoder-decoder version achieved the same results as the previous version, but the model was able to learn also the *sequence 7* of Kitti, but it fails when trained with both *sequence 3* and *sequence 7*.

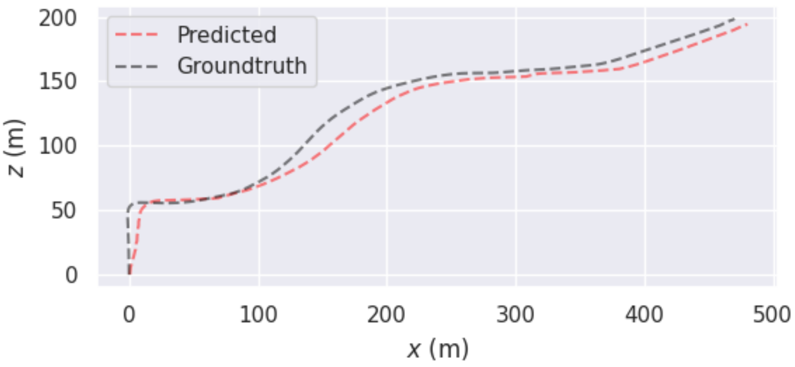


Figure 1.3: Good prediction sequence 3

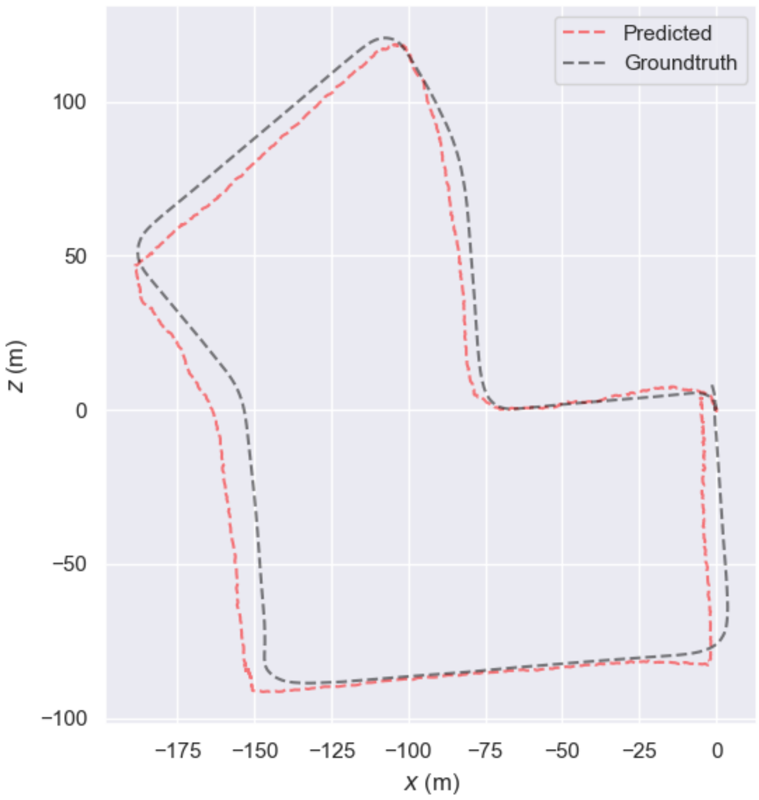


Figure 1.4: Good prediction sequence 7

1.4.2 Autoregressive models

We implemented only the encoder-decoder version of the transformer in the autoregressive way, and most of the time the prediction of the network during the training on seq 3 is just a straight line. So the model is **not** able to predict the simplest sequence in over-fitting. The model could not predict any reasonable trajectory, predicting only a linear trajectory as the follow:

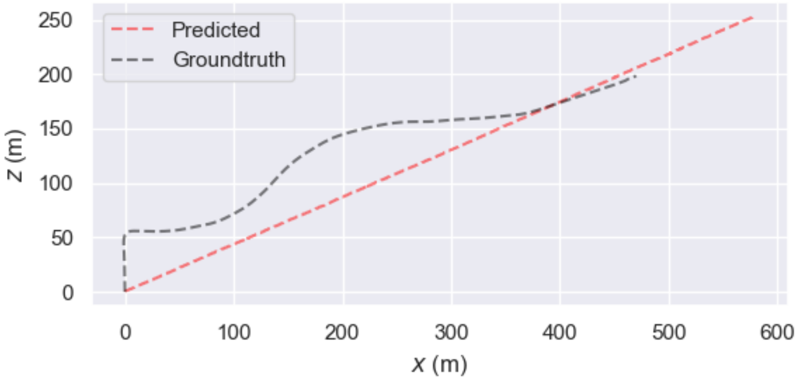


Figure 1.5: Bad prediction sequence 3 of autoregressive model

Although the model has been trained for more than two hundred epochs, the network cannot understand the goal, and this maybe is due to the loss function.

1.5 Thesis Organization

First chapter introduces the general content about thesis and gives a short presentation of the project and the results;

Second chapter a deepening about the theoretical foundations used during the stage and the project;

Third chapter presents the datasets used during for the training and the testing of the model;

Fourth chapter presents the state-of-the-art of Visual Odometry;

Fifth chapter presents the experiments did during to develop the system;

Sixth chapter presents the different implementations of the system;

Seventh chapter discusses about the results and possible future developments.

During the drafting of the essay, following typography conventions are considered:

- the acronyms, abbreviations, ambiguous terms or terms not in common use are defined in the glossary, in the end of the present document;
- the first occurrences of the terms in the glossary are highlighted like this: word;
- the terms from the foreign language or jargon are highlighted like this: *italics*.

Chapter 2

Theoretical foundations

In this chapter we will present the main theoretical knowledge useful to understand the content from successive chapters.

2.1 Deep Learning

Deep learning method is part of machine learning methods based on artificial neural network with representation learning. The learning process can be supervised, semi-supervised, or unsupervised.

There is a very large variety of deep learning architectures, some of them are specialized in some fields meanwhile others have a broader usage, especially, there are Convolutional Neural Network and Transformers.

2.1.1 Convolutional Neural Network

The Convolutional Neural Network (CNN) is a class of artificial neural network, it is used in almost every imagery related task, such as image classification, object detection, image segmentation, etc.

The CNN take an input image, assign importance (learnable weights and biases) and process the input image by using the convolution operation extracting features. There are two important parameter in the convolution operation, the kernel size and the stride. The kernel is a matrix which is used to perform the convolution operation, the stride is the number of pixels the kernel slides over the input image to produce a new pixel. With stride, we can control the size of the output image, if the stride is equal to 1, the output image will have the same size of the input image, if the stride is equal to 2, the output image will have half the size of the input image.

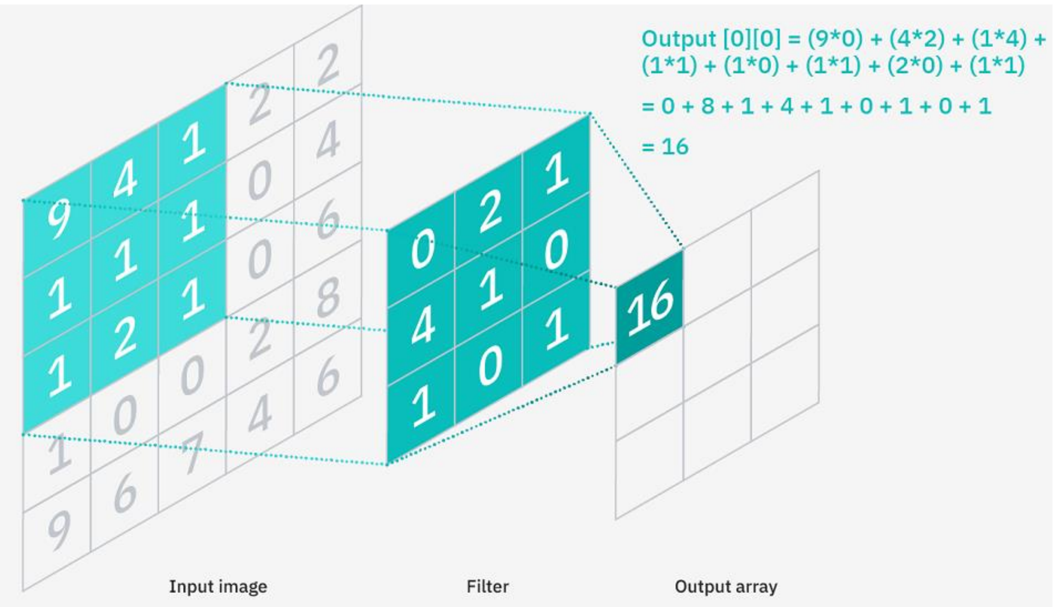


Figure 2.1: Convolutions: every single element of the output feature map is obtained by summing the element-wise product between the elements from the input feature map and the kernel. The whole feature map is then obtained sliding the kernel over the input feature map.

Increasing the number of layers and combining the pooling layers, the CNN is able to extract more and more complex features, such as edges, lines, shapes, etc. Then, there are pooling layers, usually max-pooling and average pooling, which can reduce the dimensionality of the feature maps by setting strides >= 2, which is useful to reduce the computational cost. For example, max-pooling is computed as showed in the image:

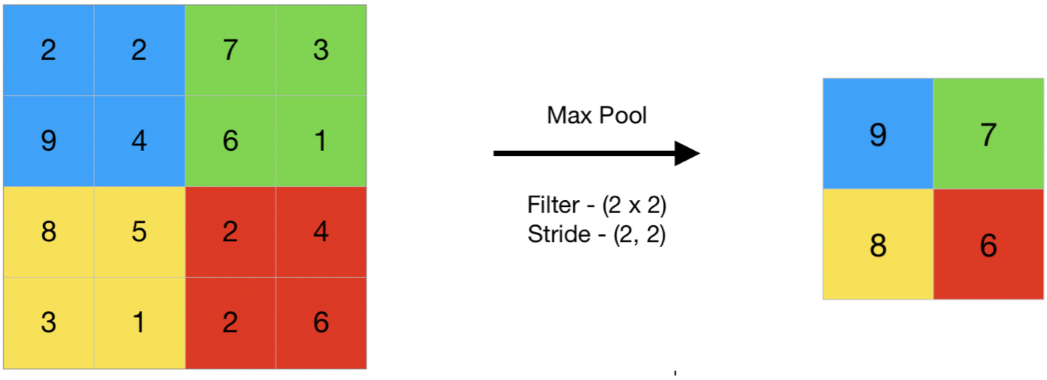


Figure 2.2: Max-pooling: essentially, it strides over the input image and takes the max value of the area covered by the kernel.

With stride = 2, sliding over the input feature map and taking the maximum value of the window, the dimensionality of the feature map is reduced. Another important component is the activation function, Rectified Linear Unit (ReLU) is the most used one, it is defined as:

$$ReLU(x) = \max(0, x)$$

Figure 2.3: ReLU activation function.

Which guarantees the non-linearity of the network, allowing the network to learn more complex features. These are the main components of a CNN, but there are other components, such as batch normalization and dropout which are used to improve the performance of the network reducing the over-fitting.

Currently, the most used CNN architecture is the ResNet which will be used in the project as feature-extratctor.

2.1.2 Transformer

The transformer architecture is a class of neural network architecture, born for the task of machine translation, but it has been used in many other vision tasks.

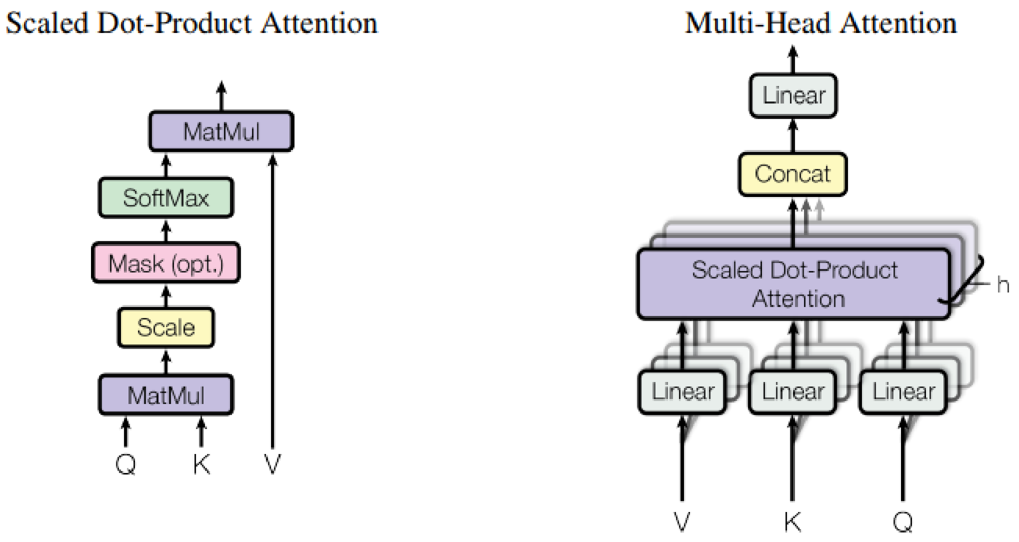
As introduced in [13], the Transformer is a model architecture based entirely on attention mechanism. The first step of attention mechanism is to compute the Q, K and V vectors, by multiplying the input vector x by the weight matrices W_q, W_k and W_v. Then, the attention weights are computed by using the scaled dot product attention, which is the softmax of the dot product between the query and the key vectors divided by the square root of the dimensionality of the key vector. Finally, the attention weights are multiplied by the value vector to obtain the output vector. The output vector is then passed through a feed-forward neural network, which is composed by two linear layers with ReLU activation function, added to the input vector and normalized by the layer normalization. The self-attention module is then repeated N times, where N is the number of layers.

The whole process can be summarized as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Figure 2.4: Scaled dot product attention.

And the graphical representation is:


Scaled Dot-Product Attention

Multi-Head Attention

Figure 2.5: Attention mechanism: (Left) scaled-dot-product. (right) Multi-head attention which is obtained by combining many scaled-dot-product attention.

Another important notion introduced is the multi-head attention, which is obtained by using more scaled dot product attention, each one with different weights, and concatenating each output vectors. Then, using a slightly modified version of the self-attention module, the encoder module is used as decoder, which takes as input also the output sequence, and repeating the number of encoder and decoder modules, we obtain the whole transformer architecture, as follows:

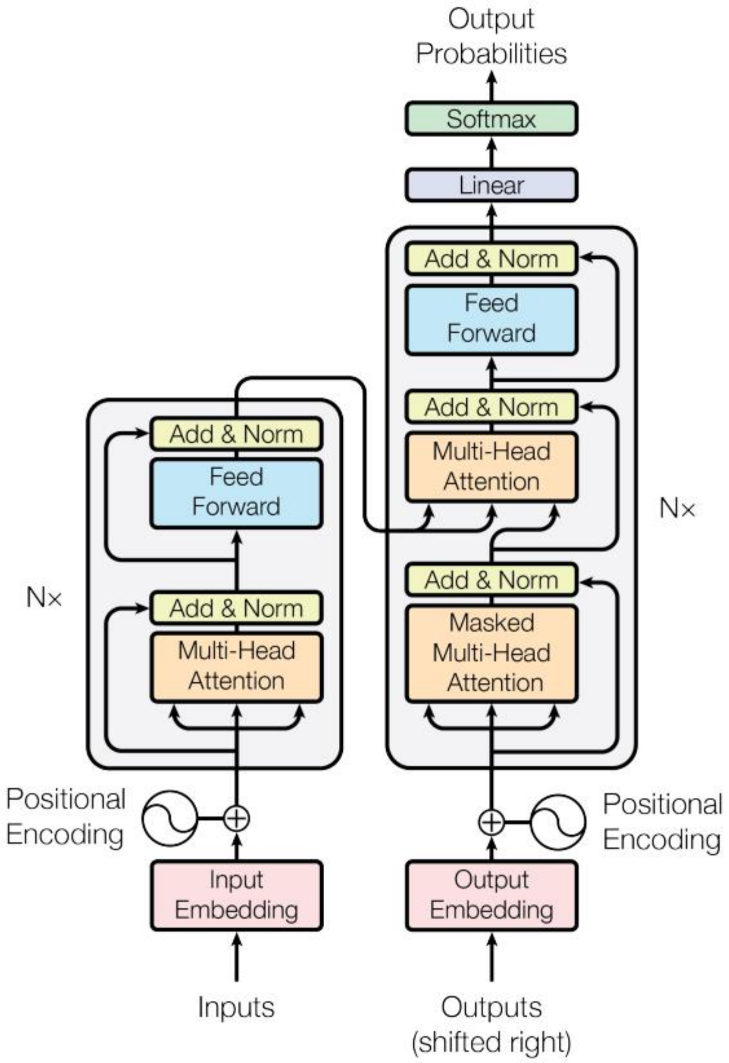


Figure 2.6: Transformer architecture: the encoder and decoder modules are composed by a self-attention module and a feed-forward neural network repeated N times.

With this architecture the new state-of-the-art results have been achieved in Natural Language Processing, especially in machine translation.

Then the adapted version applied to vision tasks also brought very good results, such as in object detection, image captioning, etc.

2.2 Odometry

As introduced in §1.2, the problem of odometry is about the estimation of the change in a robot's pose over time.

The odometry, also known as self-localization, an be classified in different ways, in §2.2.1 there is a more detailed description of the different types of odometry.

2.2.1 Taxonomy

There different types of odometry, which based on the classification of [1] can be divided into two main categories: *GNSS available* and *GNSS not available*.

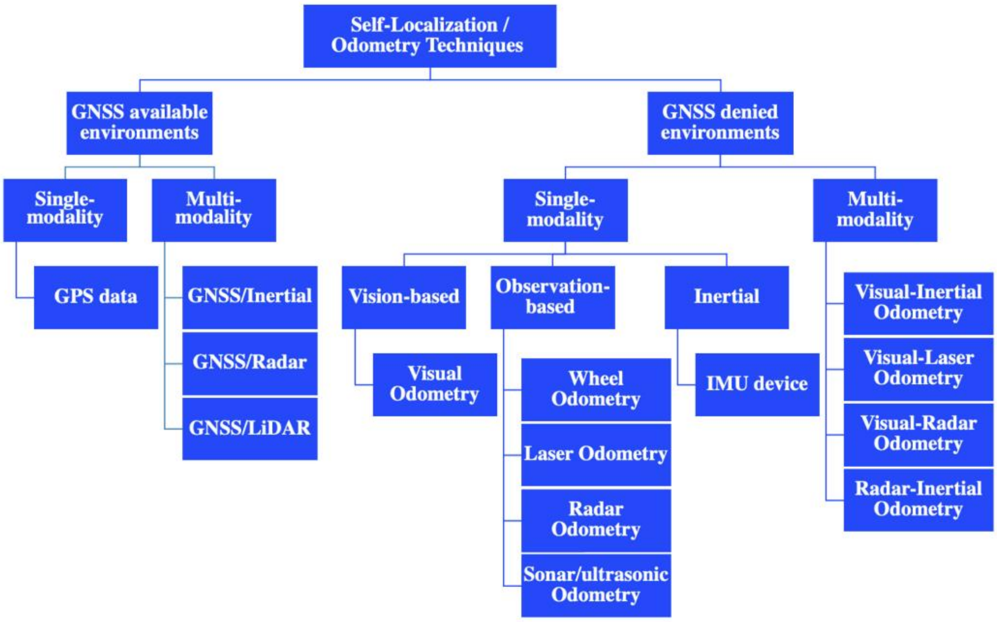


Figure 2.7: Taxonomy of odometry techniques

2.2.2 Reference systems

To tackle the problem of odometry, we need as to choose the representation system to adopt. There are many way of representing the pose of the camera or the robot, but the most common are the *Euler angles* and the *quaternions* and *rotation matrix* combined with *translation matrix.*

2.2.3 State of the art

2.3 Literature protocol

In the literature, when we need to develop a new project, there is a protocol which should be followed to increase the possibility of success. This protocol is composed by a sequence of steps, the success of every step is fundamental for the continue of the next steps. The steps are:

1. Study of the state of the art and deepening about the other projects' results.

2. Seeking for the dataset, we should look for a dataset which fits to our purpose, we should understand the characteristics of the datasets.

3. We should find some projects in oder to use them for the comparison

4. Validate the dataset using the other models, trying to reach the same results as the authors'.

5. Build the model and use it as baseline.

6. Over-fit the model with a single prediction target class, in our case a single sequence to verify the network capacity.

7. Over-fit the model with two and more prediction target classes, in this way, we are verifying that the model can learn more than one target, which is useful for us to understand which is the limit of the network in term of capacity.

8. Train the model with the whole dataset, trying to improve the results achieved by the baseline, by changing the hyper-parameters or by changing the model.

9. Fine-tuning, perform a fine tuning of the neural network can squeeze the last drops of performance of the network.

10. Compare the results with the state of the art, discussion about the results and the possible improvements.

Chapter 3

Datasets

In this chapter we will present the datasets created and used for the visual odometry.

3.1 Kitti

The odometry benchmark consists of 22 stereo sequences, saved in loss less png format: 11 sequences are provided with ground truth trajectories for training and 11 sequences (11-21) without ground truth trajectories for evaluation.

This odometry benchmark is a subset of KITTI Vision Benchmark suite [6].

3.1.1 Scene

The images represents a various of scenes from mid-sized city, rural areas and on highways.

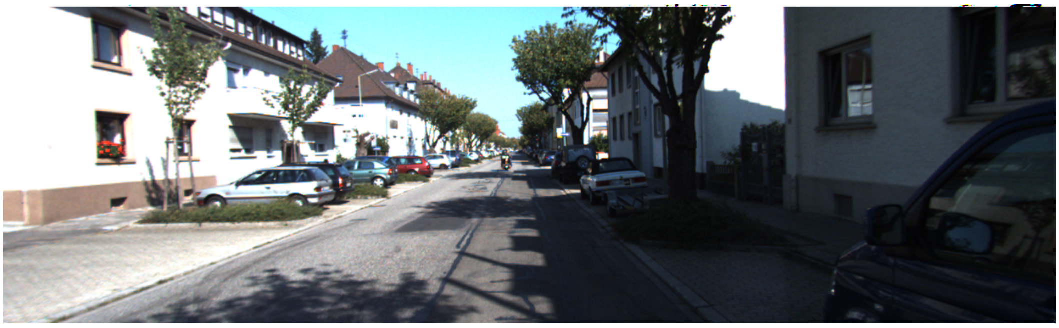


Figure 3.1: KITTI - example of scene

3.1.2 Image generation

Each sequence of the KITTI dataset is composed of by four sequences of images: left-coloured, right-coloured, left-grey and right-grey. Each one is captured by a camera mounted on the top of vehicle. They calibrated the four video cameras intrinsically and extrinsically and rectified the input image. Then they computed the 3D rigid motion parameters which relate the coordinate system of the laser scanner.

Meanwhile, the ground-truth is directly given by the output of the GPS/IMU localization unit projected into the coordinate system of the left camera after rectification.

3.1.3 Dataset statistics

The dataset consists of 22 stereo sequences, with a total length of 39.2 km, which was the longest in the time of the publication of the paper. In the dataset, there are no specifically indicate which sequence is used for training, validation or testing, but in this work, the dataset is split as this:

Sets	**N. of Sequence**	**N. Image**
Training set	8	20.098
Validation set	2	1.902
Test set	1	1.201
Total	11	23.201

Table 3.1: KITTI - dataset statistics

The images dimensions about 1240x370 are slightly different, generally varying for few pixels.

3.1.4 Usage

This dataset is the one mainly used, as it is the one of the most famous and most used in the literature.

The sequence **3** and **7** are used for evaluation and testing, because they are the easier ones.

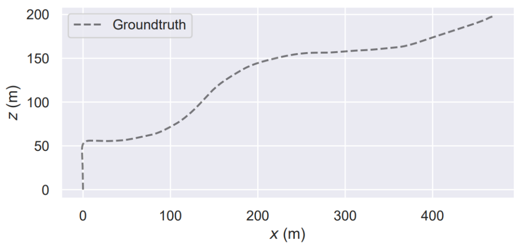


Figure 3.2: KITTI - sequence 3

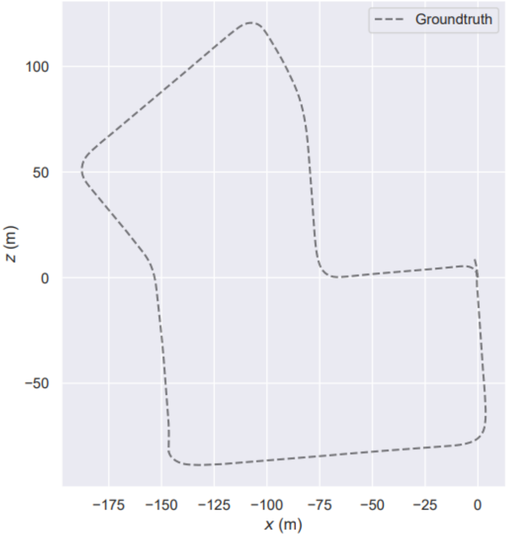


Figure 3.3: KITTI - sequence 7

Initially, to test the model's capacity, the model was trained and evaluated on the same sequence, to see if it's able to reproduce the ground truth. Then, the model was fed with more complex sequences.

3.2 Synthetic

As in there are few real-life datasets for visual odometry, we decided to create a synthetic dataset by using BlenderProc2 framework, which is a procedural photo-realistic rendering framework, and it allows to:

- **Loading**: **.obj*, **.ply*, **.blend*, *BOP*, *ShapeNet* etc.
- **Objects**: set or sample objects poses, apply physics and collision checking.
- **Materials**: set or sample physically-based materials and textures.
- **Lighting**: set or sample lights, automatic lighting of 3D-front scenes.
- **Cameras**: set, sample or load camera poses from file.
- **Rendering**: RGB, stereo, depth, normal and segmentation images/sequences.
- **Writing**: *.hdf5 containers, *COCO* and *BOP* annotations.

3.2.1 Scene

To create the synthetic dataset, the first thing is to create a scene with customized objects, material and textures.

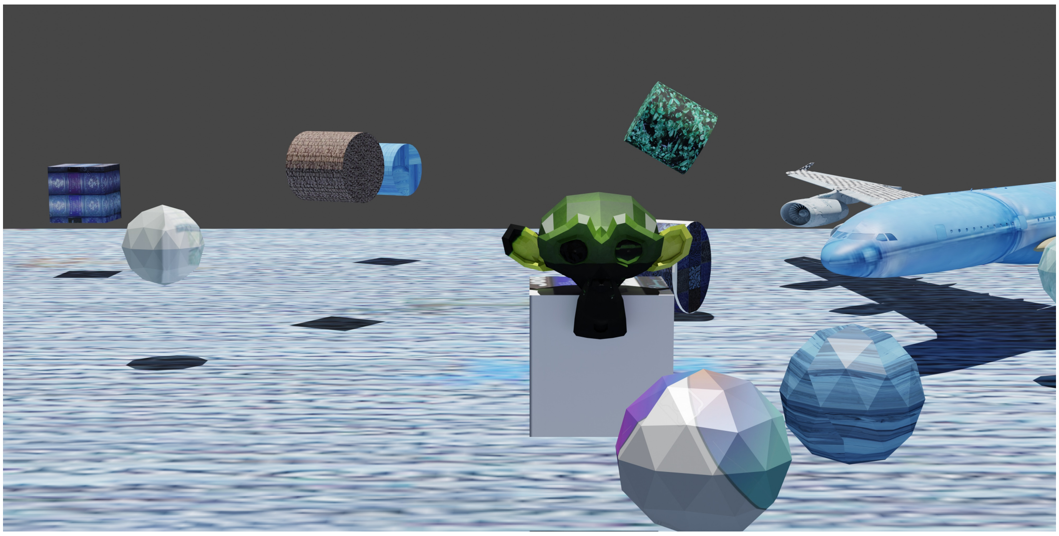


Figure 3.4: Example of scene

The scene is composed by a set of objects, more precisely:

- **a monkey**: which is at the centre of the scene over a cube.
- **a plane**: which is at left corner of the scene.
- **a set of cubes and spheres**: which are placed randomly in the scene.

When rendering the scene, the textures are loaded *randomly*, in the way that in different sequences the textures are different.

3.2.2 Image generation

To generate the sequences, we need to choose the camera position in the scene to do so, we choose randomly a position sampler from the following set for each new pose:

- **disk**: samples a point on a circle or on a 2-ball or on an arc/sector with an inner angle less or equal to 180 degrees.
- **sphere**: samples a point from the surface or from the interior of a solid sphere.
- **part-sphere**: samples a point from the surface or from the interior of a solid sphere which is split in two parts.

- **shell**: samples a point from the volume between two spheres (with radius of the spheres given as parameters).

once we have the next position of the camera, we compute the rotation matrix to be applied to the camera in the way that the camera is always looking at the POI (Point Of Interest) which corresponds to the centre of the scene.

```
rotation = bproc.camera.rotation_from_forward_vec(poi - new_
    position)
```

Code 3.1: Computes the rotatition matrix for the camera.

Then, we apply the rotation matrix to the camera and we generate the image, and by setting a certain number of frames between two poses, the framework renders a sequence of images with relative intermediate poses.

But sometime, it happens that the new camera pose is too close to an object of the scene, so we set two conditions that need to be satisfied, otherwise the sampled camera pose is skipped. The first condition checks if there are obstacles in front of the camera which are too far or too close based on the given *proximity_ checks*, while the second evaluates the interestingness or coverage of the scene.

```
def check_pose(c2w_m, special_obj, bvh_tree):
    if not bproc.camera.perform_obstacle_in_view_check(c2w_m,
        {"min": 5.0}, bvh_tree):
         return False
    if bproc.camera.scene_coverage_score(c2w_m, special_
        objects=special_obj) < 0.7:
         return False
    return True
```

Code 3.2: Checks whether the camera pose satisfies the conditions.

But when the new position is too far away from the old position, the rotation of the camera assumes a wrong value during the transition, because it rotates counter-clockwise instead of clockwise, or vice-versa. For example: If we sample the camera position from a disk at 0, 90, 180, 270 degrees, the rotations should be as in the figure 3.1:

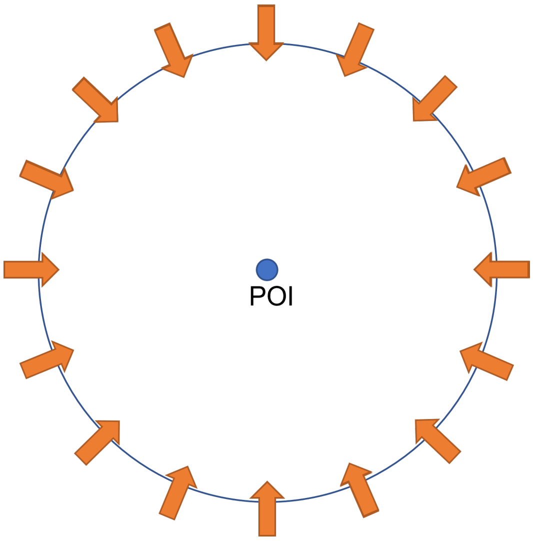


Figure 3.5: Correct transition on the disk

But, the transition from 180 to 270 degrees we obtain is a wrong rotation which is like in figure 3.2:

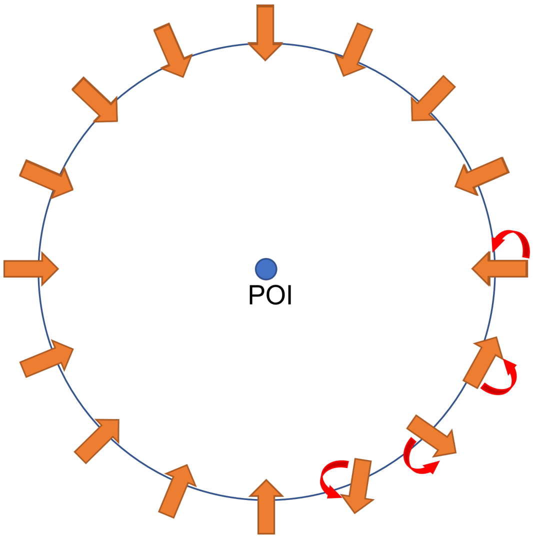


Figure 3.6: Wrong transition on the disk

To solve this problem, we tried different solutions: as first, we sample the next pose near to the previous one, but this solution sometime still fails. The final solution was to sample the new position as previously defined with samplers, but instead of letting the framework to compute the intermediate poses, we manually interpolate them, and by setting frame number to one, we obtain a sequence of correctly rotating images.

3.2.3 Dataset statistics

In total, we have generated 14 sequences, which are divided as follow:

Sets	**N. of Sequence**	**N. Image**
Training set	12	29.100
Validation set	1	1.002
Test set	1	1.003
Total	14	31.105

Table 3.2: Synthetic dataset statistics

Each image has dimension of 1024x308 pixels with 3 RGB channels. The whole dateset has dimension **1.69 GB**.

3.2.4 Usage

By using the dataset at training time the loss function is highly variable reaching values of **thousands**, also because the **Kitti** dataset is much fluid as the trajectory and the camera rotation angles are very small, so, the sequences generated are not similar to the real dataset.

Chapter 4

Experiments

In this chapter we will discuss about different models and different prediction strategies.

4.1 Models

We designed different models, each one with a different architecture. These models will be used to test the different prediction strategies.

4.1.1 encoder-only model

With only-encoder, we mean that the network has only the encoder part, the decoder part is not present.

We tried to use the encoder part of the network to predict the pose of the camera, using both ResNet18 and ResNet50 as feature extractor. The main difference of ResNet18 and ResNet50 is the number of the output embedding, in fact, ResNet18 has output embedding dimension of 512, while ResNet50 has output embedding dimension of 2048. We also tried with different depth of the network, depth of **6** and **12** layers of encoder. Then, to obtain prediction, we used a fully connected layer to reduce the dimension of the embedding to the desired number of neuron, depending on the prediction strategy. In the figure 4.1 the model architecture is shown.

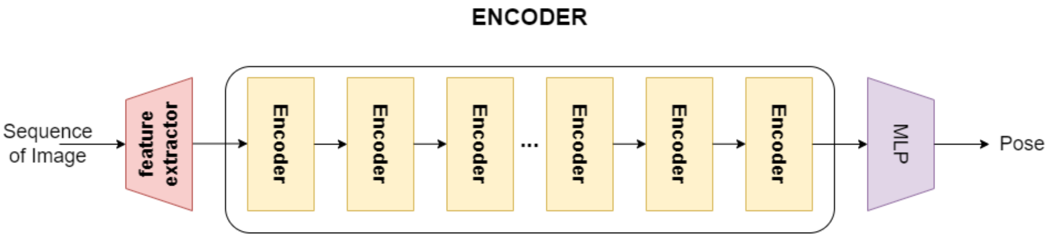


Figure 4.1: Encoder-only transformer

4.1.2 Encoder-decoder

For the encoder-decoder, we used the same encoder of the previous section, but we added a decoder part, which also takes in input a vector parameter as *memory* of the network.

In this version of the network, we tried the same configurations of the encoder-only model but the feature extractor ResNet50, because the network requires more memory than those available on GPU.

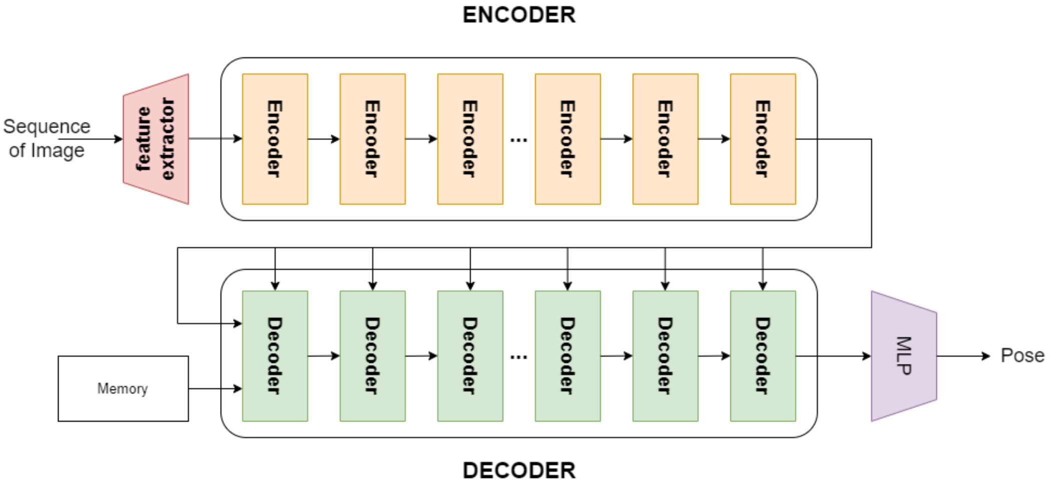


Figure 4.2: Encoder-Decoder transformer

4.1.3 Encoder-Decoder with Auto-encoder

In this version of the model, we used the same encoder-decoder model of the previous section, but we added a pose auto-encoder, which given the ground-truth of the pose, it increases the dimensionality of the input to 512 or 2048 and back to the original dimensionality. The auto-encoder is split into two parts: first part brings the dimensionality from 6 to 512 and 2048, the second part brings the dimensionality back to 6.

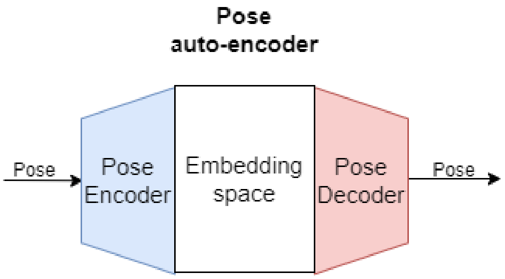


Figure 4.3: Pose auto-encoder

In the Figure 4.3, we can see the architecture of the pose auto-encoder

We used the first part to create the ground-truth of the pose, the second part to get the prediction of the pose from 512 or 2048 dimensionality. So, the final structure of the model is shown in Figure 4.4.

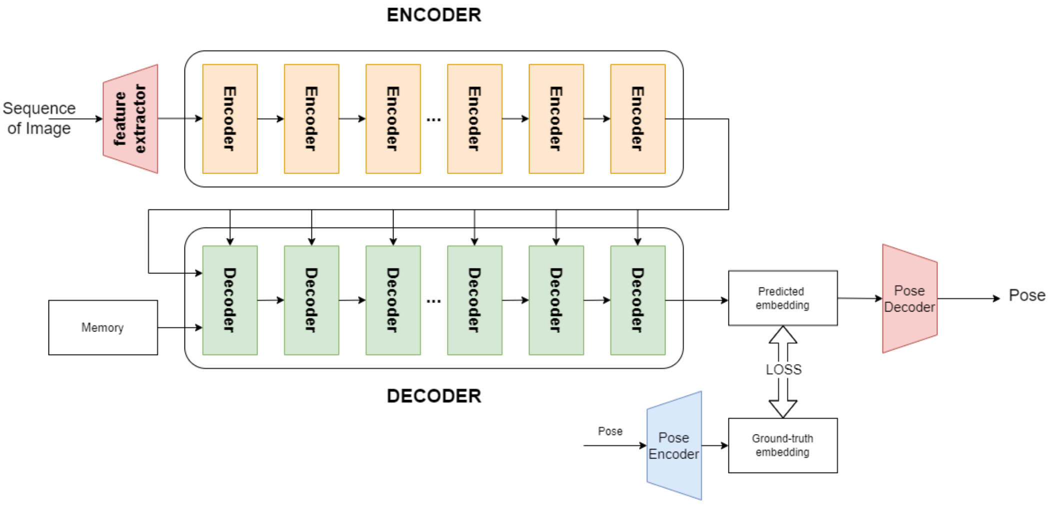


Figure 4.4: Encoder-Decoder with pose auto-encoder

4.1.4 Encoder-Decoder in autoregressive mode

In this model, we replaced the parameter vector *memory* with the output of the first part of the pose auto-encoder because the output is used to generate the ground-truth embedding used as target embedding, and the second part, as usual, is used to calculate the pose from embedding.

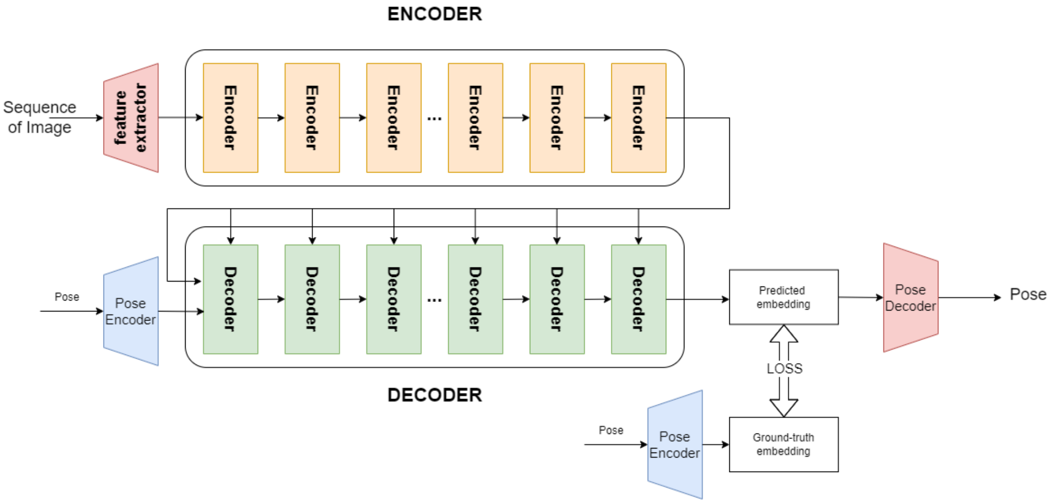


Figure 4.5: Encoder-Decoder with pose auto-encoder in autoregressive mode

In the figure 4.5, we can see the architecture of this model but the main difference between this model and previous ones is in the implementation of training and inference, this will be discussed in the next section.

4.2 Prediction Strategies

Chapter 5

Implementations

In this chapter we will discuss the implementations of different components of the neural network and different versions os the transformer.

5.1 Dataset preprocessing

Before feeding the image into the transformer, we preprocessed the images, as is usual in the literature, we computed the mean and the standard deviation per colour of the datesets, then we performed a normalization of the images.

Originally the RGB values are [zero, 255], we make a normalization dividing all values by 255, bringing the range of possible values to [zero, 1], this helps the neural network because it reducing the value range ...

5.2 Models

5.3 Losses

5.4 Pose Auto-encoder

5.5 Training cycle

Chapter 6

Final discussions

In this chapter we will discuss the results achieved, future developments and personal comments.

6.1 Result Achieved

6.2 Knowledge Acquired

6.3 Future Developments

6.4 Personal Evaluation

Glossary

Auto-regressive A model is defined as auto-regressive when the next predictions depends on the previous ones.. 4, 33

CNN Convolutional Neural Networks are a class of neural network which uses the convolution. The convolution is a LSI operator, which given an input array and a kernel, it performs the summation of the element-wise product between elements of the kernel and those of the input, then it slides the kernel over the whole range of the input. . 35

Embedding Embedding is a vector representation of a data set.. 4, 33

IMU Inertial measurement unit is a device that measures the motion of an object in space.. 35

MHA Multi head attention is a module for attention mechanisms which run through an attention mechanisms several times in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension.. 35

MLP Multi layer perceptron is 35

NLP Natural Language Processing is 35

NN Neural network models are a subset of Machine Learning models. NN is a network based on the concept of artificial neuron and neurons are organized in layers. The aim of the network is to mimic the data that is used in the training time. . 35

ReLU ReLU is 35

SLAM Slam is a computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.. 35

Vanishing gradient problem When there are more layers in the network, the value of the product of derivative decreases until at some point the partial derivative of the loss functions approaches a value close to zero, and the partial derivative vanishes.. 2, 33

Word Example of a term in the glossary. 8, 33

Acronyms

CNN Convolutional Neural Network. 9

IMU Inertial measurement unit. 3

MHA Multi-Head attention. 3

MLP Multi layer perceptron. 4

NLP Natural Language Processing. 3

NN Neural network. 3

ReLU Rectified Linear Unit. 11

SLAM Simultaneous Localization and Mapping. 1

Bibliopraphy

Paper references

[1] Yusra Alkendi, Lakmal Seneviratne, and Yahya Zweiri. "State of the Art in Vision-Based Localization Techniques for Autonomous Navigation Systems". In: *IEEE Access* PP (May 2021), pp. 1–1. DOI: 10.1109/ACCESS.2021.3082778 (cit. on p. 14).

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (2015). DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385 (cit. on p. 1).

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: 25 (2012). URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 1).

[10] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: (2014). URL: https://arxiv.org/abs/1409.1556 (cit. on p. 1).

[11] Christian Szegedy et al. "Going Deeper with Convolutions". In: (2014). URL: https://arxiv.org/abs/1409.4842 (cit. on p. 1).

[12] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: (2015). URL: https://arxiv.org/abs/1512.00567 (cit. on p. 1).