Research Internship

Specialization: Information and Communication Sciences and Technology

Academic Year: 2018-19

Deep learning for predicting ship motion from images

This is a non-confidential and publishable report

Author:
Nazar-Mykola KAMINSKYI

Promotion:
2020

ENSTA Paris Tutor:
David FILLIAT

Internship Tutor:
Natalia DIAZ RODRIGUEZ

Internship from 14/05/2019 to 20/08/2019

Host Organization: ENSTA Paris

Address: 828, Boulevard des Maréchaux, 91762 Palaiseau Cedex

Confidentiality Notice

This is a non-confidential report and publishable on the Internet.
The codes associated with this report can be used freely.

Abstract

Today, artificial intelligence penetrates into all areas of human activity. Developed methods and recent advances in machine learning show good results and will gradually replace other *"traditional"* approaches. AI allows to automate processes thereby improving efficiency and accuracy of tasks in co-operation with people. I worked on the problem of predicting ship motion from images of the sea surface.

First of all, using 3D graphics generator Blender, I created a dataset, which simulates the ship's movement through sea waves. This software also gives information about parameters of the boat (pitch and roll in our case). Data streams were structured in sequences and normalized for further processing. Then 9 different neural networks were developed and tested. As I worked with time-ordered image sequences, I decided to use convolutional neural networks (CNN) for processing images and long short-term memory (LSTM) networks for time series processing. Finally, I run the hyperband algorithm to find the best model hyperparameters and then all experiment results were analyzed. Also, some possible improvements are suggested.

Keywords: Computer Vision, Deep learning, ship motion, pitch and roll prediction, image processing, Blender, CNN, LSTM, Hyperband

Résumé

Aujourd'hui, l'intelligence artificielle pénètre dans tous les domaines de l'activité humaine. Les méthodes mises au point et les progrès récents de l'apprentissage automatique donnent de bons résultats et remplaceront progressivement les approches *"classique"*. AI vous permet d'automatiser divers processus améliorant ainsi l'efficacité et la précision des tâches dans la coopération avec les gens. J'ai travaillé sur le problème de prédiction du mouvement du navire à partir d'images de la surface de la mer.

Tout d'abord, en utilisant un générateur graphique 3D Blender, j'ai créé un ensemble de données, qui est une simulation du mouvement du navire dans les vagues de la mer. Ce logiciel donne également des informations sur les paramètres du bateau. Les données ont été structurées en séquences et normalisées en vue de travaux ultérieurs. Puis 9 réseaux neuronaux différents ont été développés et testés. J'ai décidé d'utiliser le convolution neural network (CNN) pour le traitement des images et le long short-term memory (LSTM) pour le traitement des séries chronologiques. Finalement, j'ai lancé l'algorithme de l'hyperbande pour trouver les hyperparamètres du meilleur modèle et les résultats de toutes les expériences ont été analysés. Enfin, j'ai suggéré quelques améliorations possibles.

Mots-Clés: l'apprentissage profond, Blender, CNN, LSTM, Hyperband

Acknowledgements

I would like to thank David Filliat, professor and director of the Computer Science and System Engineering Laboratory [1] at ENSTA ParisTech and my internship tutor at ENSTA - Natalia DIAZ RODRIGUEZ, for the opportunity to work on an interesting research project. Constant support and advice on various issues helped me achieve good results.

[1] http://u2is.ensta-paristech.fr/

Contents

1 - Introduction

In today's world, more and more systems are developing where artificial intelligence is involved. Every day thousands of engineers are working to improve these systems. People have made significant progress in building and using autonomous cars, but water transport only develops own autonomous systems. Engineers are faced with the difficult task of controlling the boat in real time in the sea. To make this process effective, safe, flexible and accurate you need to use many different systems.

One of the most important parameters of the ship control is ship motion. There are many ship operations that require ship motion prediction. For example:

- The landing aircraft and helicopters on carriers or destroyers (In difficult weather and sea states conditions, like a storm or a hurricane the landing operation may be quite dangerous);
- Side by side cargo transfer is another application where ship motion is extremely necessary. One crane may off-load some dangerous ammunition from a big ship to a small ship. Again the operation can be quite dangerous in high sea states.
- In some naval operations, there are some "mating" operations between a large transport ship and some small ships. The large ship is floating far from a shore and the small ships go back and forth between the large ship and the shore. During high sea states, it may be difficult for the small ships to go inside the large ship.

In all the above applications, ship motion prediction will be very important because an accuracy and a coordination are important to avoid damage to boats or aircraft, and to ensure the safety of operations reducing all risks to a minimum [1].

1.1 Internship mission

The main goal of the internship is to develop deep learning models to predict vessel movements from sea surface images. A desirable requirement by operators

would be being able to predict ship motion 30 seconds ahead, as accurately as possible.

1.2 Related work

The main idea of this work is connected and a continuation of work presented in [2]. The author considers the basic problem of ship motion prediction and offered some solutions using artificial intelligence methods like Convolutional Neural Network and Long Short Term Memory neural network. A ship motion simulation [3] can be a useful tool to aid our main task. Thanks to the simulation, one can understand both the sea and the boat behavior, but still this software can simulate only simple cases, because the assumptions had been made about some of the physical parameters of the waves, so such simulation can not provide all possible scenarios. The sea state estimation algorithm presented in [4] predicts the onsite sea state well with limited knowledge of the history of the vessel motion and with small computational effort, using Welch cross-spectral estimation method and wave direction estimate and system of linear equations. In [1], a model for ship motion prediction using ship motion history was proposed. A prediction algorithm based on MCA (Minor Component Analysis) was compared with Neural networks (NN), Auto-regressive models (AR) and Wiener Predictor. In [5, 6, 7] the main work is focused on the processing of information from the ship sensors and analysis of the ship motion history for further waves reconstruction and ship motion prediction. Various methods have been proposed such as Nonlinear auto-regressive exogenous (NARX) network [5] for time series modeling, a wavelet neural network [6], or a sine-summation algorithm [7]. However, none of the above approaches use images to make predictions, and the information from the images can help with sudden gusts, rapid weather changes and other unforeseen situations that can not be quickly detected by the history of ship motion from the sensors.

1.3 Plan

The report is organized as follows: Chapter 2 gives an understanding of the data we generated to work with. An overview of the models is provided in Chapter 3, and training aspects are discussed in Chapter 4. Test results and analysis of the results are given in Chapter 5. Chapter 6 concludes this work and gives some ideas for improvement.

2 - Data

2.1 Ship motions

Ship motions are defined by the six degrees of freedom that a ship, boat or any other craft can experience [Wikipedia[1]]. There are two groups of movements (see Figure 2.1):

- Rotational motions
  - Pitch
  - Roll
  - Yaw
- Translational motion
  - Heave
  - Sway
  - Surge

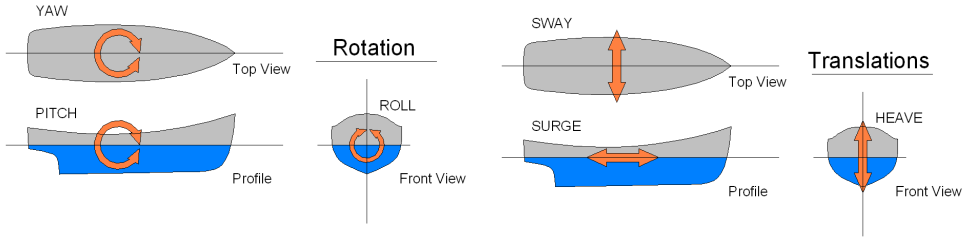


Figure 2.1: Ship motions from Wikipedia[2]

Our problem is based on the prediction of rotational movements (see Figure 2.2):

[1] https://en.wikipedia.org/wiki/Ship_motions
[2] https://en.wikipedia.org/wiki/Ship_motions

- The **longitudinal/X** axis, or **roll** axis, is an imaginary line running horizontally through the length of the ship, through its center of gravity, and parallel to the water line. A roll motion is a side-to-side or port-starboard tilting motion of the super structure around this axis.
- The **transverse/Y** axis, lateral axis, or **pitch** axis is an imaginary line running horizontally across the ship and through the center of gravity. A pitch motion is an up-or-down movement of the bow and stern of the ship.
- The **vertical/Z** axis, or **yaw** axis, is an imaginary line running vertically through the ship and through its center of gravity. A yaw motion is a side-to side movement of the bow and stern of the ship.

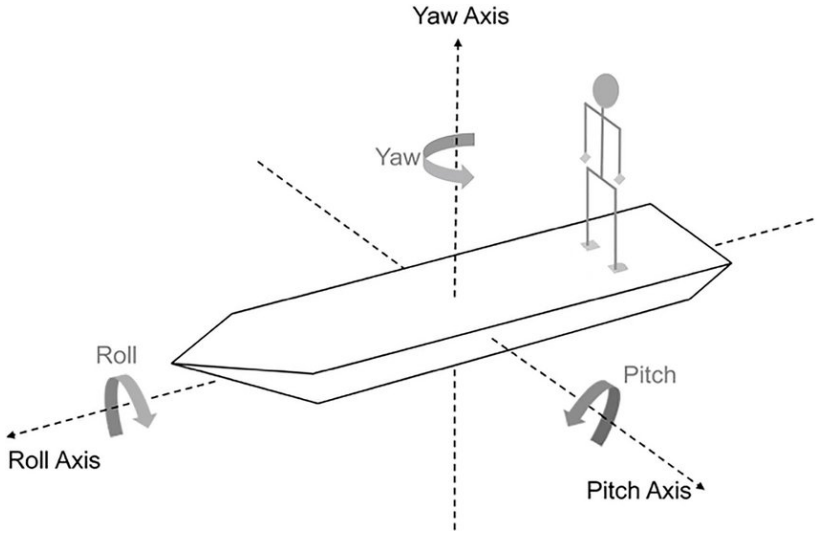


Figure 2.2: Illustration of ship motion, showing roll, pitch, and yaw, the three rotational degrees of freedom from ResearchGate [4]

.

Since the *Yaw axis* is controlled by a human or GPS module as these movements help to get to the destination, in this work it will be a fixed trajectory defined by a Bezier circle, therefore, for this study and prediction were two of the three rotational degrees of freedom were selected - **pitch** and **roll.**

2.2 Data generation

As mentioned above, for our experiments pitch, roll and a video/image sequence are needed. However, as we do not have access to real data, we use

[4] https://www.researchgate.net/figure/Illustration-of-ship-motion-showing-roll-pitch-and-yaw-t
fig1_311662283

Blender simulator[5] to generate a wide variety of it. Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and 2D design.

Thanks to the work presented in [8], we can use ready-made scripts to generate the necessary data. All needed is to specify the necessary parameters of the ocean, a camera position and an image expansion. In this case:

- Camera is fixed on the simulated ship with next parameters:
  - height - *5 m*
  - rotation x - *76 rad*
  - rotation y - *90 rad*
  - rotation z - *75 rad*
- Image resolution
  - height - *54 pixels*
  - width - *96 pixels*

Other ocean parameters as choppiness, wave scale, wind velocity, and wave alignment were chosen randomly, in a realistic looking manner.

Also, to save memory, we chose to render 2 frames per second. This is the best option so that the differences between two frames are noticeable, and they do not "duplicate" the information (see Figure 2.3).

The created dataset is available here[6]. In general, we generated 540 episodes (400 images per episode) = 216,000 images = 1800 minutes = 30 hours of simulations.

[5] https://www.blender.org/

[6] https://drive.google.com/file/d/1capC4lXDIyx4kKDvfH-USu5OCT5VGRLo/view?usp=sharing

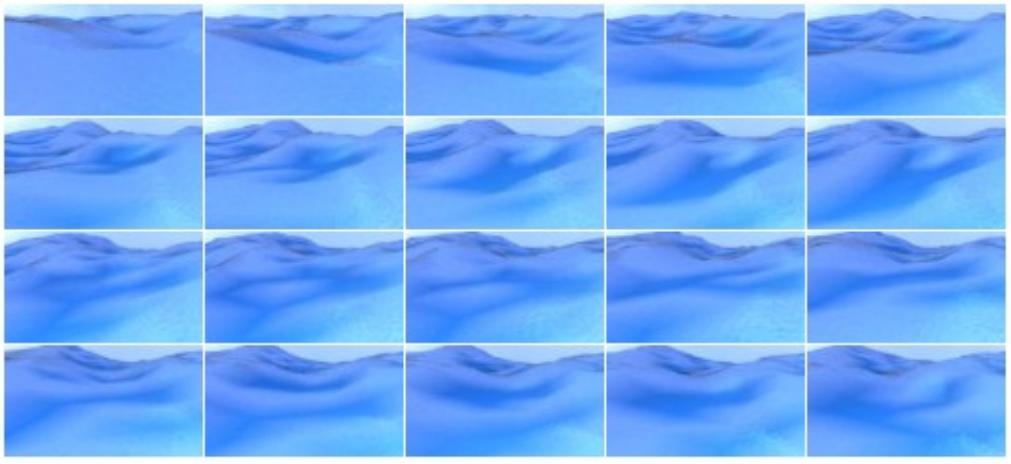


Figure 2.3: Example of 10 seconds generated data from our ship motion Blender simulator repository [8].

2.3 Data preprocessing

Before using the generated data, it needs to be processed. Since we used an artificial generator, all images have the same size and scaling or cropping is not required. However, data needs to be normalized, and so, every pixel value is converted into values between -1 and 1 (see Formula 2.1).

$$p_{normalized} = \frac{p_{original} \cdot 2}{255} - 1 \tag{2.1}$$

Since we are going to use pitch and roll values for some models, we normalize it with absolute values. The absolute values of these angles (pitch and roll) are -90 and 90 degrees, and so the following Formula 2.2 is used to normalize the data in [-1, 1]:

$$angle_{normalized} = \frac{angle_{original} - (-90)}{90 - (-90)} \cdot 2 - 1 = \frac{angle_{original} + 90}{180} \cdot 2 - 1 \tag{2.2}$$

Since we work with image sequences and want to predict sequences of ship motion, the images order is important[9].

[8] https://github.com/Nazotron1923/ship-ocean_simulation_BLENDER

[9] Our data is time-ordered so this fact was taken into account when creating the data loader: e.g. we use 5 seconds to predict 5 seconds; data used must be strictly consistent and from one single episode

3 - Proposed Models

To solve our pitch and roll prediction problem, existing model architectures will be used, whose advantages will be combined to achieve the best result. To begin, we consider the basic architectures for images - Convolutional Neural Networks [9] see Section 3.1 and time series (in our case the simulation of the sea surface) - Long Short-Term Memory networks [10] see Section 3.2.

3.1 Convolutional Neural Networks

In deep learning, a Convolutional Neural Network (CNN, or ConvNet) [9] is a class of deep neural network, most commonly applied to analyze visual imagery and extract visual features. CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfit the data. Typical ways of regularization include adding some form of magnitude measurement of the weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical patterns in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are lower level compare to Fully Connected (FC) neural network [Wikipedia[1]].

3.2 Long Short-Term Memory networks

Long Short-Term Memory networks [10] – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn;

LSTM has the form of a chain of repeating modules of neural networks. The repeating module has four interacting layers in a very special way (see Figure

[1]Convolutional Neural Networks - https://en.wikipedia.org/wiki/Convolutional_neural_network

3.1).

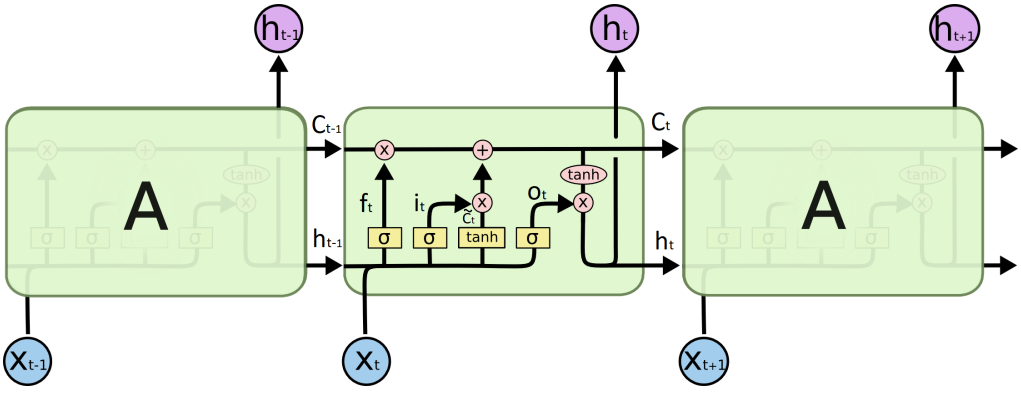


Figure 3.1: The repeating module in an LSTM contains four interacting layers: 1) sigmoid layer (f_t); 2) sigmoid layer (i_t); 3) tanh layer ($\widetilde{C}_t$); 4) sigmoid layer (o_t). [Understanding-LSTMs[3]]

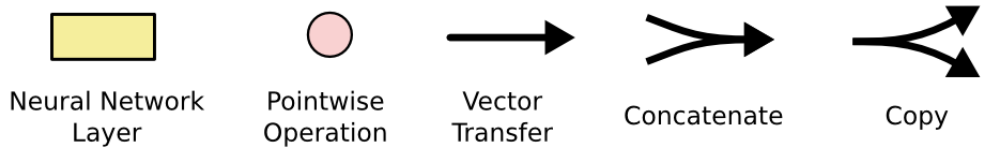


Figure 3.2: LSTM notation [Understanding-LSTMs[3]]

Cell state is the most important part of the LSTM architecture, here we can store information extracted from input, add new or delete old. All this is allowed by a special structure called gates.

To forget information replies a sigmoid layer called the **forget gate layer** (see Eq 3.1).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.1}$$

To store information two layers will be combined: sigmoid layer called the "input gate layer" decides which values we'll update; a hyperbolic tangent *tanh* layer creates a vector of new candidate values, $\widetilde{C}_t$, that could be added to the state. (see Eq 3.2 and Eq 3.3).

$$i_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_i) \tag{3.2}$$

[3]Understanding LSTMs - https://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.3}$$

The information is updated as follows: we forget certain information $f_t \cdot C_{t-1}$ and add a new one $i_t \cdot \widetilde{C}_t$ see Eq 3.4.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C}_t \tag{3.4}$$

Finally, the output will be based on our cell state, but will be a filtered version see Eq 3.5 [Understanding LSTM Networks[4]].

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.5}$$

$$h_t = o_t \cdot tanh(C_t) \tag{3.6}$$

3.3 Autoencoders

An autoencoder [11] is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. An autoencoder is a neural network that learns to copy its input to its output. It has an internal (hidden) layer that describes a code used to represent the input, and it is constituted by two main parts: an **encoder** that maps the input into the code, and a **decoder** that maps the code to a reconstruction of the original input [Wikipedia [5]].

To assess different baselines, we decided to use an autoencoder (see Figure 3.3) to extract characteristics (features) from an image into a vector. Let us describe the architecture of this autoencoder; a model with 12 layers:

[4] http://colah.github.io/posts/2015-08-Understanding-LSTMs
[5] Autoencoder https://en.wikipedia.org/wiki/Autoencoder

Input	Frame [54 x 96 x 3]							
Encoder								
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Output Padding*	*Input Size*	*Output Size*
Conv 1	Convolution ReLu	8 -	5 x 5 -	1 -	2 -	- -	54 x 96 x 3	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	-	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution ReLu	16 -	3 x 3 -	1 -	1 -	- -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	-	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution ReLu	32 -	3 x 3 -	1 -	1 -	- -	14 x 24 x 16	14 x 24 x 32
Pooling 3	Max pooling	1	3 x 3	2	1	-	14 x 24 x 32	7 x 12 x 32
FC μ	FC						1 x 2688	1 x 1024
FC σ	FC + ReLu						1 x 2688	1 x 1024
$\mathcal{Z}$	FC						1 x 1024	1 x 2688
Decoder								
TConv 1	Transposed Convolution ReLu	16 -	3 x 3 -	2 -	1 -	1 -	7x12x32	14x24x16
TConv 2	Transposed Convolution ReLu	8 -	3 x 3 -	2 -	1 -	(0, 1) -	14x24x16	26x48x8
TConv3 **Output**	Transposed Convolution Tanh	3 -	4 x 4 -	2 -	1 -	0 -	26x48x8	54x96x3

Table 3.1: An autoencoder architecture description.

After training this model, the first part of the autoencoder (see Figure 3.4) will be used in the created models as a pre-trained component. This approach allows us to achieve greater performance. Saved configurations can be found in the project repository [6].

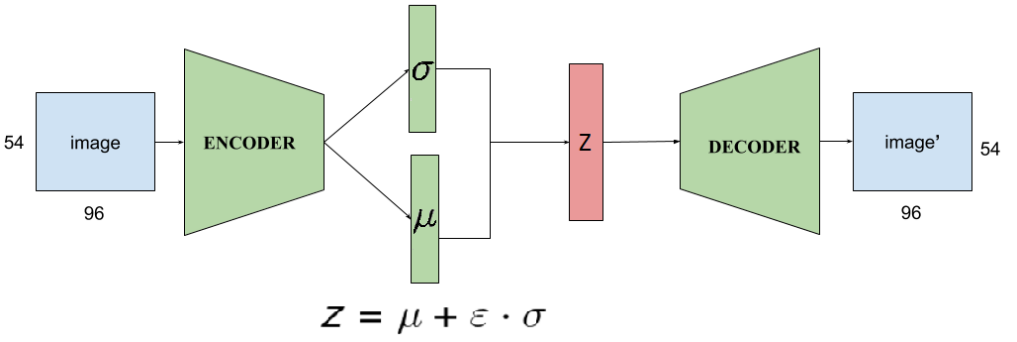


Figure 3.3: Illustration of the architecture of an autoencoder [8], where μ and σ - mean and variance of the distribution, ϵ - random variable .

[6] https://github.com/Nazotron1923/PRE-summer-2019-/tree/master/Pre

[8] https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

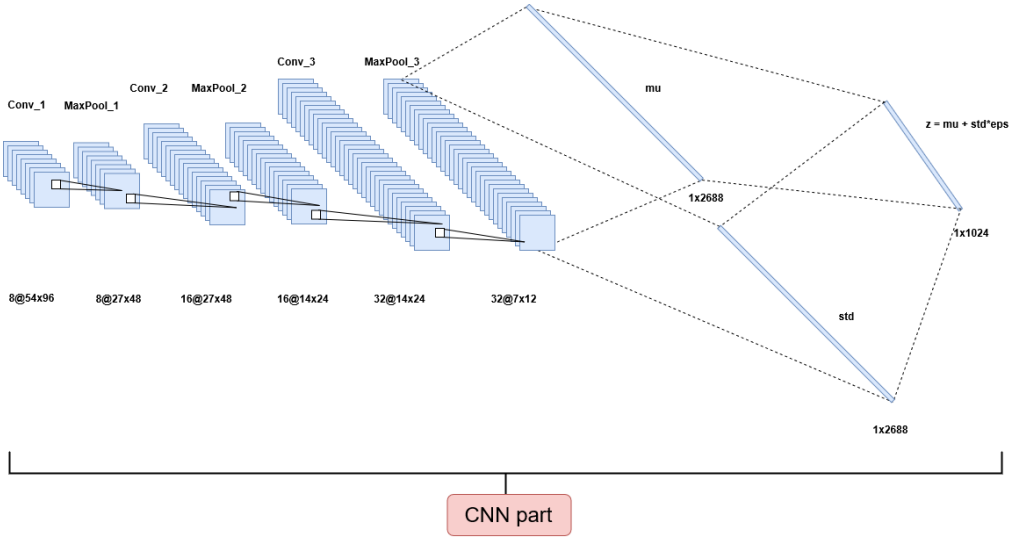


Figure 3.4: Encoder part of an autoencoder

3.4 Proposed Model

In this section I will describe the models that have been created - its architectures and features. We introduce some notations that will help identify a particular model by its name (see Table 3.2).

Designation	Explanation
N	Number of past frames used as input (in the stream history sliding window)
M	Number of future predicted frames by the model
CNN	A model containing a CNN architecture
LSTM	A model containing an LSTM architecture
FC	A model containing Fully Connected layers
PR	A model where the original pitch and roll are used as input for its prediction M frames in the future
images	A model where images are input and used for prediction
stack	A model where several images stacked are used as input. Typically, images have to be submitted separately;
encoder	An LSTM architecture is used as part of the model to encode an input vector
img-encoder	An individual encoder is used to encode a sequence of images
PR-encoder	An individual encoder is used to encode a sequence pitch and roll
decoder	An LSTM architecture is used as part of the model to decode a latent vector

Table 3.2: Models nomenclature for each module component.

3.4.1 CNN stack FC model

This model is an improvement of architecture proposed in [2]. The architecture of the CNN part was changed by adding more layers (see Table 3.3), as well

as the number of images used for prediction was expanded.

Also 2 variants of the model were tested:

- **First version** - predict the specific second forward (e.g. 5^{th} second directly (see Table 3.3 and Figure A.1))
- **Second version** - predict a sequence of seconds ahead pitch and roll starting from the 1^{st} second (see Table 3.4 and Figure A.2)

Input	Stack of **N** frames [54 x 96 x 3]						
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution BatchNorm ReLu	8 - -	5 x 5 - -	1 - -	2 - -	54 x 96 x 3	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution BatchNorm ReLu	16 - -	3 x 3 - -	1 - -	1 - -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
FC 1	Dropout (0.3) + FC + ReLu + Dropout (0.4)					1 x 5376	1 x 1024
FC 2	FC + ReLu + Dropout (0.4)					1 x 1024	1 x 128
Output	FC + Tanh					1 x 128	**1 x 2**

Table 3.3: CNN stack FC **first version** model architecture.

Input	Stack of **N** frames [54 x 96 x 3]						
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution BatchNorm ReLu	8 - -	5 x 5 - -	1 - -	2 - -	54 x 96 x (**N**·3)	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution BatchNorm ReLu	16 - -	3 x 3 - -	1 - -	1 - -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution BatchNorm ReLu	32 - -	3 x 3 - -	1 - -	1 - -	14 x 24 x 16	14 x 24 x 32
Pooling 3	Max pooling	1	3 x 3	2	1	14 x 24 x 32	7 x 12 x 32
FC μ	FC					1 x 2688	1 x 1024
FC σ	FC + ReLu					1 x 2688	1 x 1024
FC 1	FC + ReLu + Dropout (0.4)					1 x 1024	1 x 512
FC 2	FC + ReLu + Dropout (0.4)					1 x 512	1 x 128
Output	FC + Tanh					1 x 128	1 x (**M**·2)

Table 3.4: CNN stack FC **second version** model architecture.

3.4.2 CNN stack PR FC model

As indicated in Section 1.2, authors used the ship motion history to predict future motion. So was decided to add additional information - pitch and roll to the previous architecture (see Table 3.5 and Figure A.3) to try to improve accuracy of prediction.

Input	Stack of **N** frames [54 x 96 x 3]						
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution BatchNorm ReLu	8 - -	5 x 5 - -	1 - -	2 - -	54 x 96 x (**N**·3)	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution BatchNorm ReLu	16 - -	3 x 3 - -	1 - -	1 - -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution BatchNorm ReLu	32 - -	3 x 3 - -	1 - -	1 - -	14 x 24 x 16	14 x 24 x 32
Pooling 3	Max pooling	1	3 x 3	2	1	14 x 24 x 32	7 x 12 x 32
FC mu	FC					1 x 2688	1 x 1024
FC sigma	FC + ReLu					1 x 2688	1 x 1024
Adding sequence of **N** pitches and rolls [1 x 2]							
FC 1	FC + ReLu + Dropout (0.4)					1 x (1024 + **N**·2)	1 x 512
FC 2	FC + ReLu + Dropout (0.4)					1 x 512	1 x 128
Output	FC + Tanh					1 x 128	1 x (**M**·2)

Table 3.5: CNN stack PR FC model architecture.

3.4.3 CNN PR FC model

In this configuration, the input changed. Now the images are not served in a stack, but separately, i.e., each image will be passed through the CNN part of model, after which the obtained features will be concatenated into one vector (see Table 3.6 and Figure A.4).

Input	Sequence of **N** frames [54 x 96 x 3]						
N CNN parts							
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution BatchNorm ReLu	8 - -	5 x 5 - -	1 - -	2 - -	54 x 96 x 3	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution BatchNorm ReLu	16 - -	3 x 3 - -	1 - -	1 - -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution BatchNorm ReLu	32 - -	3 x 3 - -	1 - -	1 - -	14 x 24 x 16	14 x 24 x 32
Pooling 3	Max pooling	1	3 x 3	2	1	14 x 24 x 32	7 x 12 x 32
FC μ	FC					1 x 2688	1 x 1024
FC σ	FC + ReLu					1 x 2688	1 x 1024
Adding sequence of **N** pitches and rolls [1 x 2]							
FC 1	FC + ReLu + Dropout (0.4)					1 x (**N**·(1024 +2))	1 x (**N**·(1024 +2))/2
FC 2	FC + ReLu + Dropout (0.4)					1 x (**N**·(1024 +2))/2	1 x 1024
FC 3	FC + ReLu + Dropout (0.4)					1 x 1024	1 x 128
Output	FC + Tanh					1 x 128	1 x (**M**·2)

Table 3.6: CNN PR FC model architecture.

3.4.4 LSTM encoder decoder PR model

Since our task is to use a sequence of past frames to predict a sequence of the following future frames, we can use sequence to sequence prediction approaches, for example, an encoder-decoder architecture from Pytorch tutorial[9]. Therefore, this architecture was implemented (see Table 3.7 and Figure A.5).

Input	Sequence of **N** pitches and rolls [1 x **N**·2]					
Name	*Layer*	*Number Layers*	*Batch First*	*Hidden Size*	*Input Size*	*Output Size*
Encoder	LSTM	1	True	1 x 300	1 x **N**·2	1 x 300
Decoder	ReLu + LSTM	1	True	1 x 300	1 x 300	1 x 300
FC 1	FC + ReLu				1 x 450	1 x 225
Output	FC + Tanh				1 x 225	1 x (**M**·2)

Table 3.7: LSTM encoder decoder PR model architecture.

3.4.5 CNN LSTM img-encoder PR-encoder decoder model

This model combines CNN and LSTM architectures. It was made to achieve better results by taking advantage of 2 types of architectures.We did sparation of LSTM component for different types of information: images and angles (see Table 3.8 and Figure A.6).

Initially, the sequence of frames will be processed by CNN part to extract features. After that, the features are combined into a vector and fed to the img-encoder (LSTM part). The pitch and roll sequence is fed separately to the PR-encoder (LSTM part). One latent vector is generated and decoded by decoder (LSTM part). The output is a predicted sequence of pitch and roll.

Input	Sequence of **N** frames [54 x 96 x 3]						
N CNN parts							
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution ReLu	8 -	5 x 5 -	1 -	2 -	54 x 96 x 3	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution ReLu	16 -	3 x 3 -	1 -	1 -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution ReLu	32 -	3 x 3 -	1 -	1 -	14 x 24 x 16	14 x 24 x 32
Pooling 3	Max pooling	1	3 x 3	2	1	14 x 24 x 32	7 x 12 x 32
FC mu	FC					1 x 2688	1 x 1024
FC sigma	FC + ReLu					1 x 2688	1 x 1024
LSTM part: adding sequence of **N** pitches and rolls [1 x 2]							
Name	*Layer*		*Number Layers*	*Batch First*	*Hidden Size*	*Input Size*	*Output Size*
Img Encoder	LSTM		1	True	1 x 600	1 x **N**·1024	1 x 600
PR Encoder	LSTM		1	True	1 x 300	1 x **N**·2	1 x 300
Decoder	ReLu + LSTM		1	True	1 x 450	1 x 900	1 x 450
FC 1	FC + ReLu					1 x 450	1 x 225
Output	FC + Tanh					1 x 225	**1 x (M·2)**

Table 3.8: CNN LSTM img-encoder PR-encoder decoder model architecture.

[9] https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

3.4.6 CNN LSTM encoder decoder images PR model

Given the complexity of the previous model, it was decided to combine img-encoder and PR-encoder in one encoder element (see Table 3.9 and Figure A.7).

Input	Sequence of **N** frames [54 x 96 x 3]						
N CNN parts							
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution ReLu	8 -	3 x 3 -	1 -	2 -	54 x 96 x 3 -	54 x 96 x 8 -
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution ReLu	16 -	3 x 3 -	1 -	1 -	27 x 48 x 8 -	27 x 48 x 16 -
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution ReLu	32 -	3 x 3 -	1 -	1 -	14 x 24 x 16 -	14 x 24 x 32 -
Pooling 3	Max pooling	1	3 x 3	2	1	14 x 24 x 32	7 x 12 x 32
FC μ	FC					1 x 2688	1 x 1024
FC σ	FC + ReLu					1 x 2688	1 x 1024
LSTM part: adding sequence of **N** pitches and rolls [1 x 2]							
Name	*Layer*		*Number Layers*	*Batch First*	*Hidden Size*	*Input Size*	*Output Size*
Encoder	LSTM		1	True	1 x 700	1 x **N** $\cdot$ (1024 + 2)	1 x 700
Decoder	ReLu + LSTM		1	True	1 x 408	1 x 700	1 x 408
FC decoder	FC + ReLu					1 x 408	1 x 204
Output	FC + Tanh					1 x 204	**1 x (M·2)**

Table 3.9: CNN LSTM encoder decoder images PR model architecture.

3.4.7 CNN LSTM encoder decoder images model

Modification of the previous model without using additional information current ship motion data. For details see Table 3.10 and Figure A.8.

Input	Sequence of **N** frames [54 x 96 x 3]						
N CNN parts							
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution ReLu	8 -	5 x 5 -	1 -	2 -	54 x 96 x 3	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution ReLu	16 -	3 x 3 -	1 -	1 -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution ReLu	32 -	3 x 3 -	1 -	1 -	14 x 24 x 16	14 x 24 x 32
Pooling 3	Max pooling	1	3 x 3	2	1	14 x 24 x 32	7 x 12 x 32
FC mu	FC					1 x 2688	1 x 1024
FC sigma	FC + ReLu					1 x 2688	1 x 1024
LSTM part							
Name	*Layer*		*Number Layers*	*Batch First*	*Hidden Size*	*Input Size*	*Output Size*
Encoder	LSTM		1	True	1 x 1024	1 x (**N**·1024)	1 x 1024
Decoder	ReLu + LSTM		1	True	1 x 1024	1 x 1024	1 x 1024
FC 1	FC + ReLu					1 x 1024	1 x 512
Output	FC + Tanh					1 x 512	**1 x M·2**

Table 3.10: CNN LSTM encoder decoder images model architecture.

3.4.8 CNN LSTM images PR model

A modification of the model proposed in Section 3.4.6. We decided to combine all LSTM elements into one to reduce the parameters of the model and check the efficiency of this configuration. For details see Table 3.11 and Figure A.9.

Input	Sequence of **N** frames [54 x 96 x 3]						
N CNN parts							
Name	*Layer*	*Number filters*	*Filter Size*	*Stride*	*Padding*	*Input Size*	*Output Size*
Conv 1	Convolution ReLu	8 -	5 x 5 -	1 -	2 -	54 x 96 x 3	54 x 96 x 8
Pooling 1	Max pooling	1	3 x 3	2	1	54 x 96 x 8	27 x 48 x 8
Conv 2	Convolution ReLu	16 -	3 x 3 -	1 -	1 -	27 x 48 x 8	27 x 48 x 16
Pooling 2	Max pooling	1	3 x 3	2	1	27 x 48 x 16	14 x 24 x 16
Conv 3	Convolution ReLu	32 -	3 x 3 -	1 -	1 -	14 x 24 x 16	14 x 24 x 32
Pooling 3	Max pooling	1	3 x 3	2	1	14 x 24 x 32	7 x 12 x 32
FC μ	FC					1 x 2688	1 x 1024
FC σ	FC + ReLu					1 x 2688	1 x 1024
LSTM part: Adding sequence of **N** pitches and rolls							
Name	*Layer*		*Number Layers*	*Batch First*	*Hidden Size*	*Input Size*	*Output Size*
Decoder	ReLu + LSTM		1	True	1 x 1000	1 x (**N**·(1024 + 2))	1 x 1000
FC 1	FC + ReLu					1 x 1000	1 x 500
Output	FC + Tanh					1 x 500	**1 x (M·2)**

Table 3.11: CNN LSTM images model architecture.

4 - Model Training

This problem belongs to the class of supervised learning - regression problems. Supervised learning [12] is the machine learning task of learning a function that maps an input to an output based on examples of input-output pairs.

Consider the main components to perform training process. For the implementation of the models we used Pytorch[1].

4.1 Parameters

To customize the training of our models a number of parameters needs to be configured. The main parameters for our models are:

- **Past window size** - the number of seconds of simulation is taken to predict pitch and roll (sequence of frames);
- **Future window size** - the number of seconds for which pitch and roll will be predicted (sequence of frames).
- **Number of episodes** - the number of episodes used as input data ;
- **Latent vectors** - The dimensions of the latent vectors for the LSTM parts of the model (one (see Section 3.4.8), two (see Section 3.4.4) or three (see Section 3.4.5) - depending on the model;
- **Learning rate** - A hyperparameter which determines to what extent newly acquired information overrides old information [Wikipedia[2]];
- **Weight decay** - A hyperparameter which consists of adding a penalty to the error function that depends on the magnitude of the weights that connect neurons to each other. A regularization technique used to limit overfitting in a neural network [Wikipedia[3]];

[1] https://pytorch.org/
[2] https://en.wikipedia.org/wiki/Learning_rate
[3] https://fr.wikipedia.org/wiki/Weight_decay

- **Batchsize** - a hyperparameter that defines the number of samples to work through before updating the internal model parameters [Machine Learning Mastery[4]];
- **Number of Epochs** - A hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset;
- **Frames per second (fps)** - A data generation parameter. We worked with 2 fps and 1 fps. More fps is needed for the video, in our case 1 or two is enough, when between 2 frames the difference still can be detected.

4.2 Data loader

Preparing data is a big issue of machine learning. To work, we need data supplied to the algorithm in a strictly defined format. For each model an input is a sequence of images. That is, the generated episodes in Section 2 were separated into sequences of different sizes depending on the models. This work uses two types of episode splitting into sequences:

- For models without **LSTM** module - the length of the sequence will be equal to the past window size;
- For models with **LSTM** module - the length of the sequence will be of size [60, 72, 120] frames. This separation is caused by the specific use of the LSTM model. The passage through this sequence will be carried out step by step (see Figure4.1). Blue images - past window, red images future window and together it's one step through the sequence.

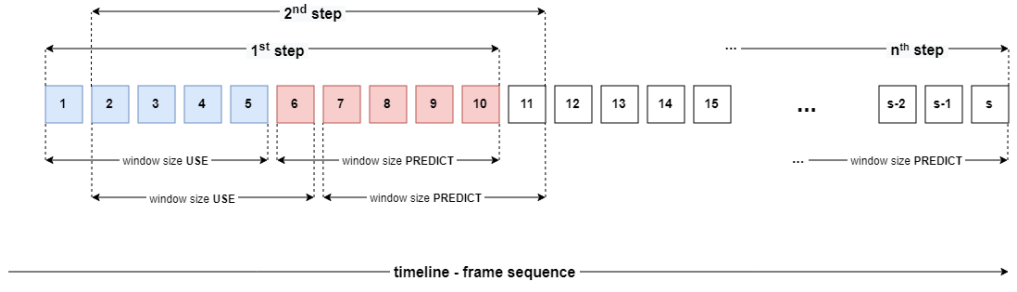


Figure 4.1: Iterate through the sequence step by step

Splitting details: every 10 percent of generated data are allocated separately and are split randomly into 3 groups: training data, validating data and test data. This is done in order, if the data are generated with increasing complexity of the sea state (wave height, wind speed increases, etc.), then all 3 categories

[4] https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

will get data of varying complexity, this improves the training of models and their ability to generalize data.

4.3 Optimizer

A key component of training the model is the optimizer used. We use Adam [13], [14], an adaptive learning rate optimization algorithm designed specifically for training deep neural networks, showing huge performance gains in terms of speed of training. We experimented with **S**tochastic **G**radient **D**escent **SGD** for comparison, but not getting better results.

4.4 Loss Function

Since I am solving the regression problem, we use **M**ean **S**quare **E**rror (MSE) as one of the most commonly used regression loss function. MSE is the sum of squared distances between our target variable and predicted values (see Eq. 4.1, where $\hat{angle}$ is the predicted angle and $angle$ is the original simulated ground truth one):

$$loss = MSE = \frac{\sum_{i=1}^{number frames}(angle_i - \hat{angle}_i)^2}{number frames} \tag{4.1}$$

Since we predict the angles for a sequence of frames, the error is assumed to be for the whole sequence. Later, you can calculate the error for each specific frame in the sequence that was predicted.

4.5 Early Stopping

In machine learning, early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration. Up to a point, this improves the learner's performance on data outside of the training set. Past that point, however, improving the learner's fit to the training data comes at the expense of increased generalization error. Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit. Early stopping rules have been employed in many different machine learning methods, with varying amounts of theoretical foundation [Wikipedia [5]]. We used this technique to reduce memory resources and prevent the problem of overfitting, and a ready-made implementation, which can be found on repository[6]

[5] https://en.wikipedia.org/wiki/Early_stopping
[6] https://github.com/Bjarten/early-stopping-pytorch

4.6 Hyperband

Setting system hyperparameters is one of the most important and complex aspects of deep learning, even a state of art can have poor performance with parameters chosen randomly and vice versa a good configuration can make even a simple model the most effective solution. Finding the right parameters takes a lot of work. There are several approaches that can help:

- Grid Search
- Random Search
- Bayesian optimization
- Evolutionary optimization

Each of these methods has its advantages and disadvantages, but the main problem is a large computing power (computational operations) is needed.

HyperBand algorithm [15] for hyperparameter search is a mix of grid search and random search designed in specific way to increase performance and to speed up the calculations. The ready implementation [7] of the algorithm was used.

[7] https://github.com/zygmuntz/hyperband

5 - Model Testing

First, the basic settings (given in table 5.1) were tested;only for good models we searched hyperparameters using Hyperband algorithm (see Section 4.6). For all tests, the parameters were saved here[1] for future representation. r

Parameters	
Number episodes	540
Number epochs	50
Learning rate	1e-04
Weight decay	1e-03
Frame per second (fps)	2
Past window size	10 s
Future window size	12 s

Table 5.1: Basic parameters for all models

5.1 CNN stack FC first version model test

First of all, we tested the architecture proposed in [2] on new data (see Section 3.4.1). The evolution of the loss function can be seen in Figure 5.1, and the results of the experiment placed in Table 5.2.

CNN stack FC first version	Result
Test loss avg	0.0045643
Validation loss avg	0.0034879
Test loss avg	0.0033126

Table 5.2: Results for [CNN stack FC first version] model

As can be seen from the presented results, the model does not converge well. In addition, there was forecast for only a specific moment of the future, here 12^{th} second. So, the model is clearly not suitable for us.

[1] https://drive.google.com/file/d/1capC4lXDIyx4kKDvfH-USu5OCT5VGRLo/view?usp=sharing

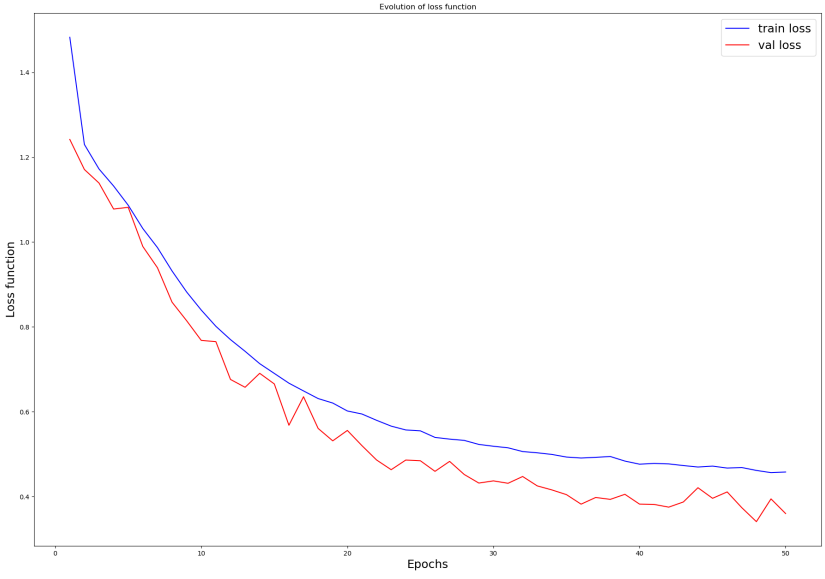


Figure 5.1: Loss function for CNN stack FC first version model. (Value of loss function $\cdot 10^{-2}$)

5.2 CNN stack FC model VS CNN stack PR FC model test

After modification of the first model two more models with some modification were created: **[CNN stack FC second version] model**(see Section 3.4.1 and **[CNN stack PR FC] model**(see Section 3.4.2).

In the [CNN stack PR FC] model, the current ship motion was used as additional information. The sequence of pitch and roll has been verified to affect the accuracy of the model. We also tested the parameters of the models : learning rate, past window size, future window size in the form of grid search. The results are shown on the Figures 5.2 and 5.3.

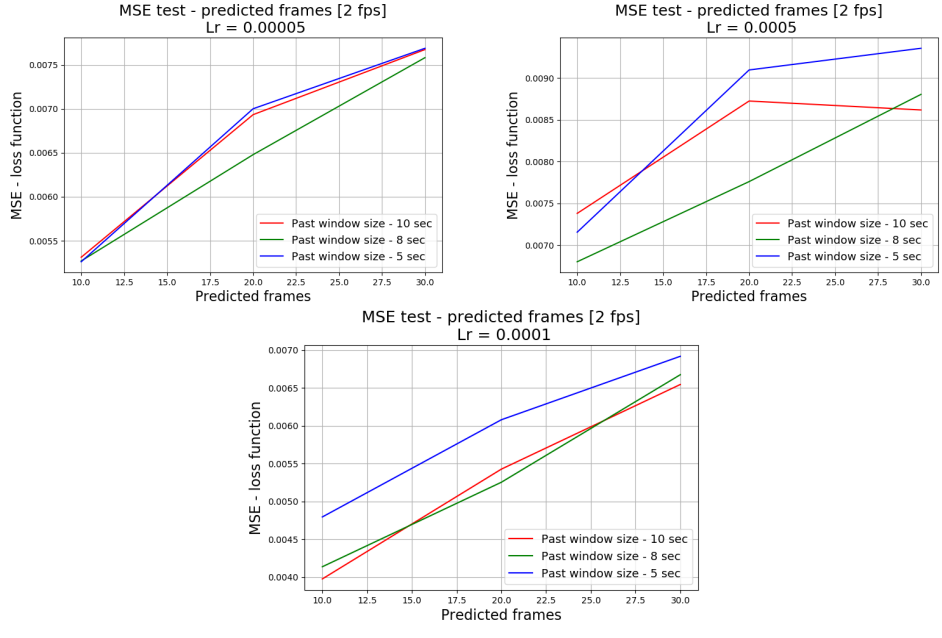


Figure 5.2: MSE test (test loss) evolution CNN stack FC model with different learning rate

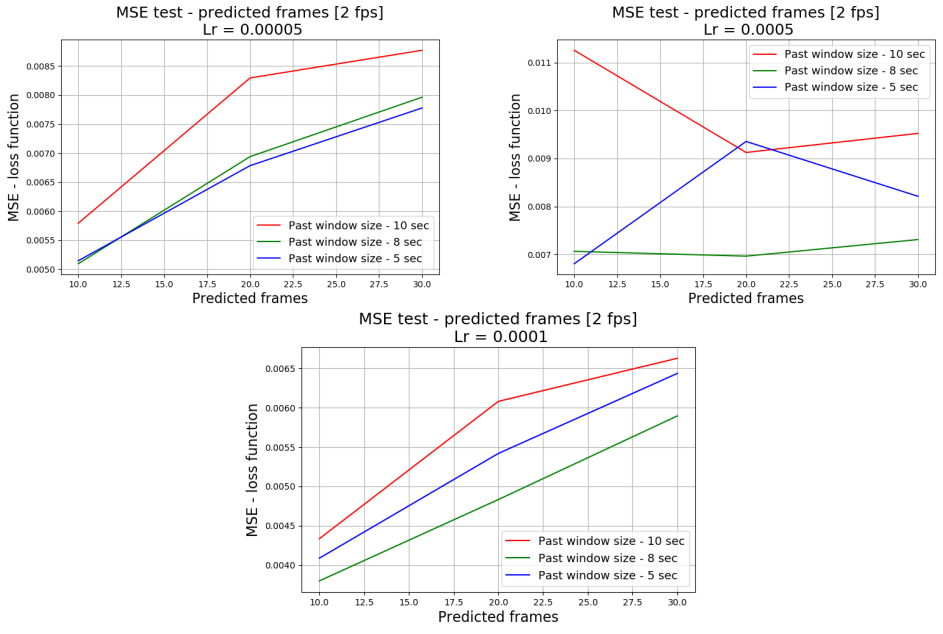


Figure 5.3: MSE test (test loss) evolution CNN stack PR FC model with different learning rate

Results show that using ship motion as additional information increases accuracy. Although the value of the loss function has increased, these models predict pitch and roll for a sequence of future frames, and thus this increase is justified.

5.3 CNN PR FC model test

It was decided to slightly change the structure of the input to extract more information from all frames in sequence. Therefore, for each image in the sequence is used separately CNN (see Section 3.4.3). Result of testing is placed in Table 5.3 and in Figure 5.4.

CNN PR FC	Result
Test loss avg	0.0037869
Validation loss avg	0.0025979
Test loss avg	0.0025291

Table 5.3: Results of experiment for CNN PR FC model

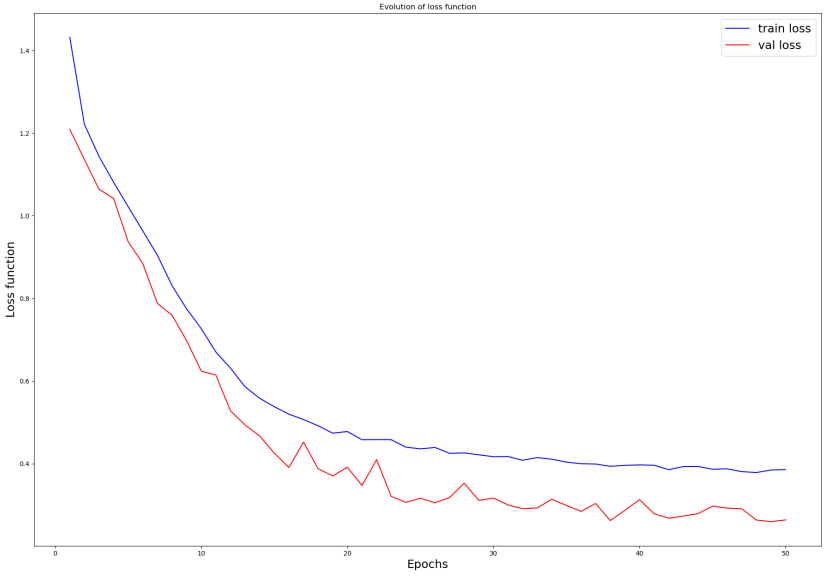


Figure 5.4: Loss function for CNN PR FC model

We see a loss function reduction, therefore, an accuracy increases. However, the results of the model are still not satisfactory; there is a poor convergence, which indicates that the system is not well tuned. However, the approach of splitting frames from the original input stack greatly improves performance.

5.4 Testing LSTM models

Next we implemented and tested models which are synthesis of CNN and the LSTM parts: [**LSTM encoder decoder PR**], [**CNN LSTM img-encoder PR-encoder decoder PR**], [**CNN LSTM encoder decoder images PR**], [**CNN LSTM encoder decoder images PR**], [**CNN LSTM images PR**], which are presented in the sections 3.4.4 - 3.4.8. The tests were carried out with the previously set parameters (see Table 5.1). The results and comparison of the models are given in the next section.

5.5 Models Comparison

The results of the experiments can be found in Table 5.4, which shows the normalized average MSE of the sum of pitch and roll over the predicted sequence (of 24 frames of length).

MODELS	MSE train x 10^{-2} avg over sum pitch and roll	MSE val x 10^{-2} avg sum over pitch and roll	MSE test x 10^{-2} avg sum over pitch and roll
CNN stack FC 3.4.1	0.6359306	0.6478805	0.6190951
CNN stack PR FC 3.4.2	0.556660	0.47093	0.483350
CNN PR FC 3.4.3	0.3786881	0.2597921	0.2529106
LSTM encoder decoder PR [baseline] 3.4.4	0.8555552	0.8697197	0.7235203
CNN LSTM image-encoder PR-encoder decoder 3.4.5	0.0442216	0.1040735	0.0806697
CNN LSTM encoder decoder images PR 3.4.6	0.0393415	**0.0890370**	0.0729175
CNN LSTM encoder decoder images 3.4.7	**0.0388357**	0.0909325	**0.0715274**
CNN LSTM decoder images PR 3.4.8	0.1264467	0.1380148	0.1082258

Table 5.4: Results for all models with the parameters precised in Table 5.1. *Red line* - our baseline **LSTM encoder decoder PR** model, worst result; *Light green line* - the best result at the moment; *Strong green line* - second result but this model will be tested more see Section 5.6

For better understanding, we consider also the evolution of the loss function for different models during training on the training (see Figure 5.5) and validation datasets (see Figure 5.6). The last plot also shows the MSE value on the test dataset (dashed line).

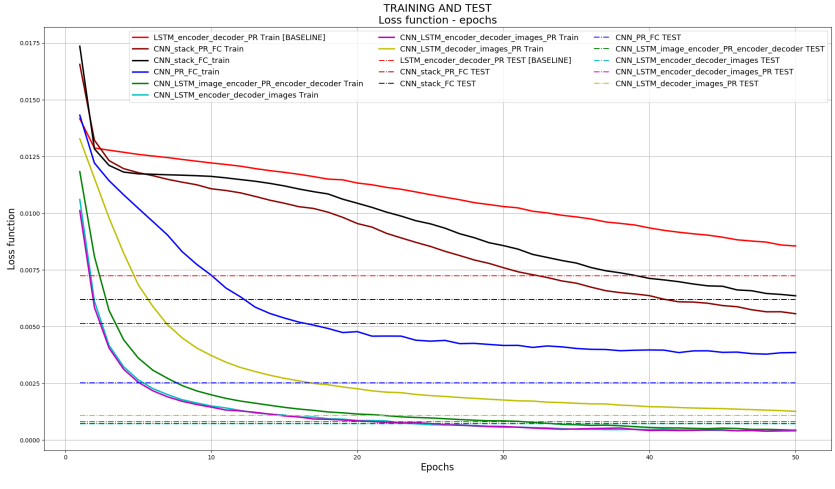


Figure 5.5: Loss function on the **training** data with **test** results for all models

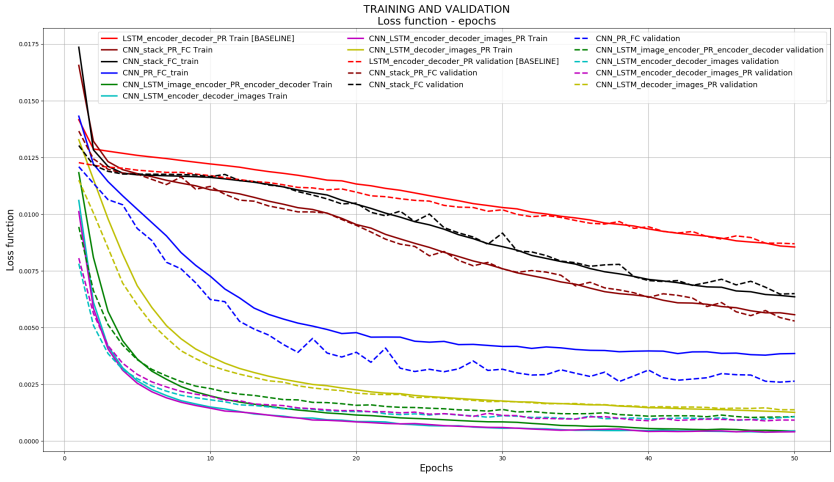


Figure 5.6: Loss function on the **training** and the **validation** data for all models

Let us explore further the components of our prediction - pitch and roll. Training results denormalized MSE values at the 10^{th} predicted second can be seen in Table 5.5.

MODELS	MSE pitch at 10^{th} sec [denormalized]	MSE roll at 10^{th} sec [denormalized]
CNN stack FC 3.4.1	28.9790454	28.9440312
CNN stack PR FC 3.4.2	21.1823023	22.5687344
CNN PR FC 3.4.3	11.3655497	11.2846873
LSTM encoder decoder PR [baseline] 3.4.4	30.9973885	29.4405860
CNN LSTM image-encoder PR-encoder decoder 3.4.5	3.56111801	2.83715593
CNN LSTM encoder decoder images PR 3.4.6	**2.89517881**	2.70937074
CNN LSTM encoder decoder images 3.4.7	3.42248999	**2.32273956**
CNN LSTM images PR 3.4.8	4.22731776	4.16475212

Table 5.5: Testing results for all models. Denormalized MSE for pitch and roll at 10s in predicted sequences. *Red line* - our baseline and the worst result; *Light green line* - the best result at the moment; *Strong green line* - second result

As can be seen from the table, the results are not so unambiguous: it can be seen that the prediction of pitch angle is a more complex task, almost always the error of this prediction is larger than for the roll angle. We identified two models

with the best performance in pitch and roll that we predict. **CNN LSTM encoder decoder images PR model** for pitch and **CNN LSTM encoder decoder images model** for roll were the best solutions. Since the results are more balanced (there is no huge difference between pitch MSE and roll MSE for the model) , the work was mainly done to improve the **CNN LSTM encoder decoder images PR model**.

Results of actual angles predictions of **LSTM encoder decoder PR model [baseline]** are in Figure 5.7, and for **CNN LSTM encoder decoder images PR model** in Figure 5.8.

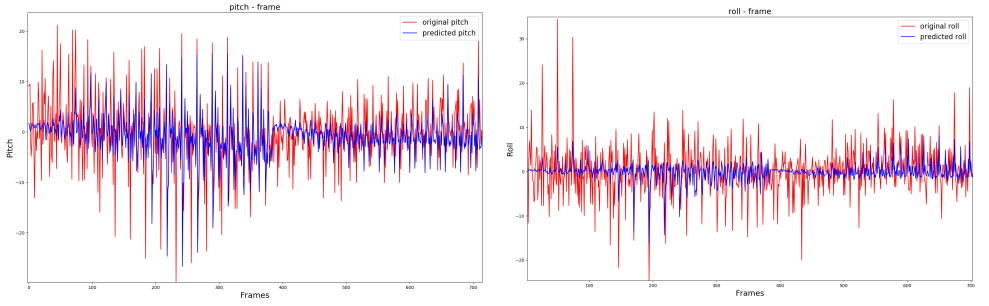


Figure 5.7: Original and predicted pitch (left) and roll (right) at 10^{th} second using L**STM encoder decoder PR model**

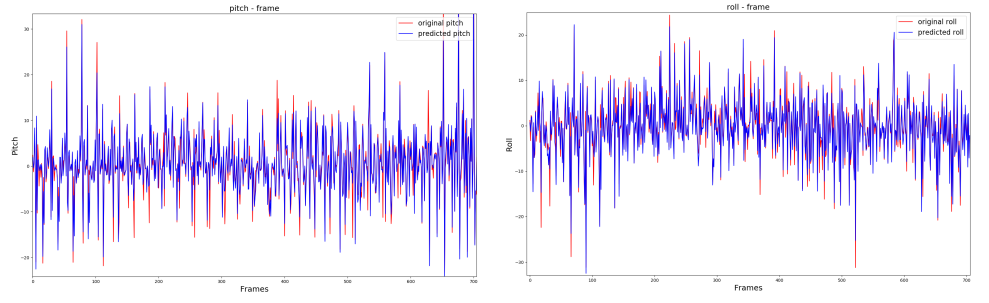


Figure 5.8: Original and predicted pitch (left) and roll (right) at 10^{th} second using C**NN LSTM encoder decoder images PR model**

As can be seen from the above figures, the results of those two models are significantly different. Despite **CNN LSTM encoder decoder images PR model** showing good results, it is clear that there are peaks that are poorly predictable. The model weakly generalizes to new data. (red peaks in the Figure 5.8)Since we predict not one value, but a whole sequence of angles in time, it would be interesting to see the evolution of MSE through this sequence. One loss function adding pitch and roll was created to facilitate calculations Results for the three most promising models according to table 5.4 are in Figure 5.9.

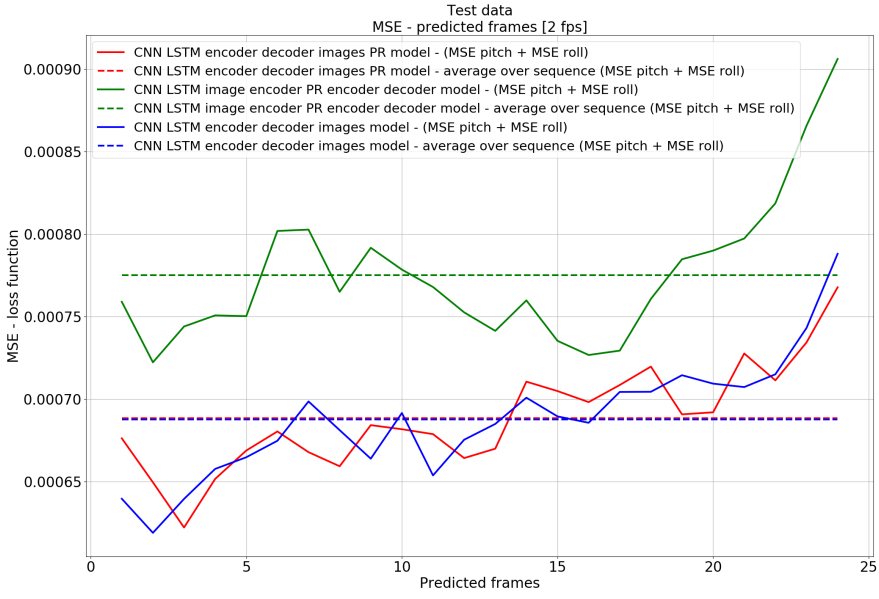


Figure 5.9: Evolution of sum of pitch MSE, roll MSE, and average value over predicted sequence for three models

Consider the results for pitch and roll. Figure 5.10 shows the evolution MSE on test data also for 3 models for two angles separately. **CNN LSTM encoder decoder images PR model** shows one of the most balanced results. For more details see Figures A.10 - A.13.

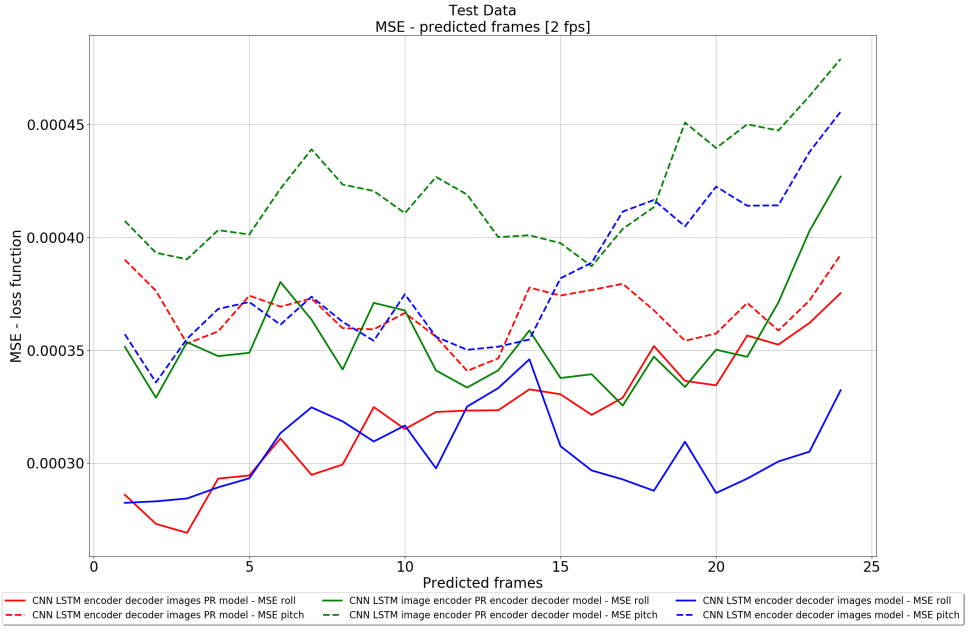


Figure 5.10: Pitch MSE evolution and Roll MSE evolution for 3 models

5.6 Results

Model **CNN LSTM encoder decoder images PR** was chosen for further analysis as the most suitable because of good convergence properties, it is balanced in prediction of pitch and roll and the loss function reaches one of its minimal values. The Hyperband algorithm described in Section 4.6 was used to search hyperparameter: to search for hyperparameters, the frame rate was changed to **1 fps**. This was done for several reasons: first, to save time; second, to save memory. This reduces the number of images fed into the model at the same past window size in seconds, however, the principal architecture remains the same.

After preliminary tests, the following hyperparameters and limits were selected to find the optimal model (see Table 5.6):

Hyperparameters	Distribution	[min, max] interval
Encoder latent vector	uniform	[500, 1500]
Decoder latent vector	uniform	[250, 500]
Learning rate	uniform	$[9{\cdot}10^{-6}, 2{\cdot}10^{-3}]$
Weight decay	uniform	$[9{\cdot}10^{-5}, 3{\cdot}10^{-3}]$

Table 5.6: Hyperparameters search space.

After the search was completed, the following best configuration (in Table 5.7) was found:

Hyperparameters	Best configuration
Encoder latent vector size	700
Decoder latent vector size	408
Learning rate	$1.937 \cdot 10^{-4}$
Weight decay	$9.47 \cdot 10^{-5}$

Table 5.7: Best configuration for CNN LSTM encoder decoder images PR model.

Then, using the best configuration, the final test of the model was carried out to predict **30 seconds ahead**. Results are presented below.

	At 15^{th} second	At 30^{th} second
Pitch MSE [denormalized]	2.9282012	3.7539778
Roll MSE [denormalized]	3.0667821	4.9226913

Table 5.8: Final test result.

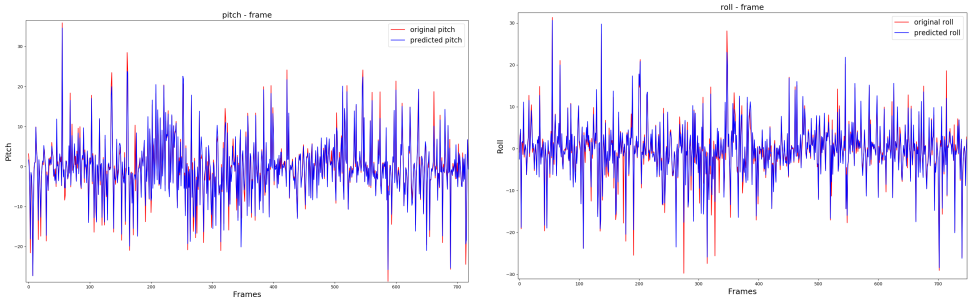
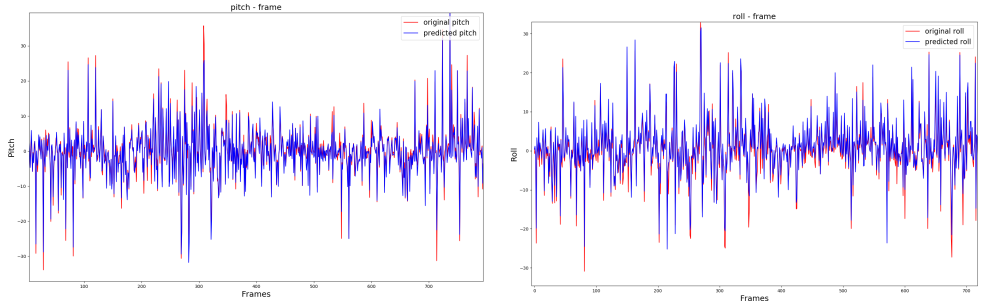


Figure 5.11: Original and predicted pitch (left) and roll (right) at 15^{th} seconds in the future, using CNN LSTM encoder decoder images PR model.



2

Figure 5.12: Original and predicted pitch (left) and roll (right) at 30^{th} second using CNN LSTM encoder decoder images PR model

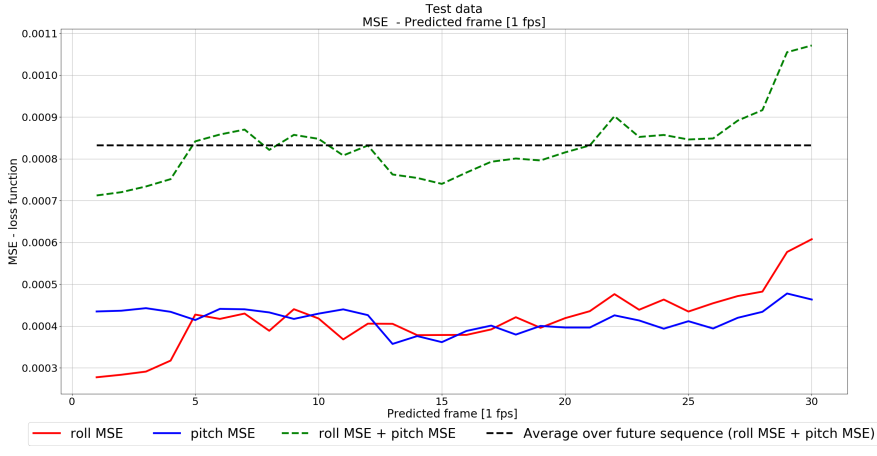


Figure 5.13: Final test. Evolution pitch and roll MSE for the best CNN LSTM encoder decoder images PR model configuration

As can be seen from the above results (see Figures 5.11 - 5.13), the model still has problems. In general, the behavior is stable and consistent with the trends but there are difficulties in predicting the exact peak values. Figure A.14 shows that the model overfitted a little and thus, poorly generalizes.

6 - Conclusion

In this work, in order to predict ship motion from images, nine deep neural network models were created and tested. Such variety of models is caused by the complexity of the problem. Empirical results show that models with LSTM parts and using additional information such as the current ship motion improve the pitch and roll prediction accuracy. The best result was achieved by a [CNN LSTM encoder-decoder images PR] model. The best combination of parameters was found using Hyperband algorithm (learning rate, weight decay, encoder latent vector size and decoder latent vector size). In general, the model normally exhibits fluctuations, and sometimes skips large peaks (not being accurately enough for the angle value). The problem is still not completely solved and can have many improvements. Even in the best version of the created model there are problems such as overfitting, poor generalization, etc. Still, not solved the problem with the data, to achieve a better result and good working model real data from the ship is needed. However, reasonable predictions are achieved with the proposed models, which code is available in repository [1].

6.1 Future Improvements

Working on this project a number of configurations and architectures, number of aspects were investigated to achieve the best possible result. It is found that the use of CNN and LSTM models together improves the result. In the future we should continue to work on CNN and LSTM models. The model input also needs to be supplied with a separate sequence of images as regular LSTMs require, rather than connecting them in a stack as input. In future work new sequence to sequence prediction model presented in [16] could be tested. The most influential parameter was found to be the learning rate, and to achieve even better results, different strategies should be applied during training. Parameters: learning rate, weight decay, past/future window size were tested but still could be exploring with other hyperparameters: batchsize, optimizer, etc.The size of the latent vectors influences the amount of information that will be ”remembered” from past sequences, and therefore affects the final result.

[1] https://github.com/Nazotron1923/PRE-summer-2019-/tree/master/Pre

A number of improvements can be done to continue project. After working on the problem I want to offer the following ideas that may be useful:

- Generate better datasets. The best option would be using real data from different ships. If the previous option is not available, Unreal Engine graphics generator could provide a realistic simulation. When using it ,is important to work on the physics model of the ship, which greatly affects the behavior of the ship at sea.
- Use more ship parameters: yaw, heave, sway, surge, weight, dimensions, physical features, etc. All these parameters affect the behavior of the boat and ship motion prediction. It could be possible to use meteorological data that affects the formation of waves.
- Use video directly as input [17, 18]; this way, we can more efficiently use time coherence and potentially boost performance.
- Using extra information, the variation of possible neural network architectures will increase. New models and configuration of hyperparameters for old ones should be tested, as well as more complex/effective CNN and LSTM modules, etc.

Bibliography

[1] George Zhao, Roger Xu, and Chiman Kwan. "Ship-motion prediction: Algorithms and simulation results". In: vol. 5. June 2004, pp. V–125. ISBN: 0-7803-8484-9. DOI: 10.1109/ICASSP.2004.1327063.

[2] Zhi ZHOU. *Predicting Ship's Motion Based on Machine Learning*. URL: https://drive.google.com/file/d/1A0Jgqq1ElIiHYhDEImBzrKtDDpGBwnfp/view.

[3] Shyh-Kuang Ueng, David Lin, and Chieh-Hong Liu. "A ship motion simulation system". In: *Virtual Reality* 12 (Mar. 2008), pp. 65–76. DOI: 10.1007/s10055-008-0088-8.

[4] Astrid H. Brodtkorb, Ulrik D. Nielsen, and Asgeir J. Sørensen. "Online wave estimation using vessel motion measurements." In: *IFAC-PapersOnLine* 51.29 (2018). 11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018, pp. 244–249. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.09.510. URL: http://www.sciencedirect.com/science/article/pii/S2405896318321992.

[5] Guoyuan Li et al. "Neural-network-based modelling and analysis for time series prediction of ship motion". In: *Ship Technology Research* (Apr. 2017). DOI: 10.1080/09377255.2017.1309786.

[6] Wenjun Zhang and Zhengjiang Liu. "Real-Time Ship Motion Prediction Based on Time Delay Wavelet Neural Network". In: *Journal of Applied Mathematics* 2014 (Aug. 2014), pp. 1–7. DOI: 10.1155/2014/176297.

[7] Christian Bolander and Douglas Hunsaker. "A Sine-Summation Algorithm for the Prediction of Ship Deck Motion". In: Oct. 2018. DOI: 10.1109/OCEANS.2018.8604888.

[8] Manuel Cortés Batet. *Boat and sea wave Blender simulator*. URL: https://github.com/manubatet/Ship-simulator.

[9] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *ArXiv e-prints* (Nov. 2015). URL: https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks.

[10] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[11] Yunchen Pu et al. "Variational Autoencoder for Deep Learning of Images, Labels and Captions". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 2352–2360. URL: http://papers.nips.cc/paper/6528-variational-autoencoder-for-deep-learning-of-images-labels-and-captions.pdf.

[12] Wikipedia. *Supervised learning*. URL: https://en.wikipedia.org/wiki/Supervised_learning.

[13] Vitaly Bushaev. *Adam — latest trends in deep learning optimization.* URL: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

[14] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[15] Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *CoRR* abs/1603.06560 (2018). arXiv: 1603.06560v4. URL: http://arxiv.org/abs/1603.06560.

[16] Oriol Vinyals and Quoc V. Le. "A Neural Conversational Model". In: *CoRR* abs/1506.05869 (2015). arXiv: 1506.05869. URL: http://arxiv.org/abs/1506.05869.

[17] A. Ullah et al. "Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features". In: *IEEE Access* 6 (2018), pp. 1155–1166. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2778011.

[18] Junhyuk Oh et al. "Action-Conditional Video Prediction using Deep Networks in Atari Games". In: *CoRR* abs/1507.08750 (2015). arXiv: 1507.08750. URL: http://arxiv.org/abs/1507.08750.

List of Figures

List of Tables

A - Architectures

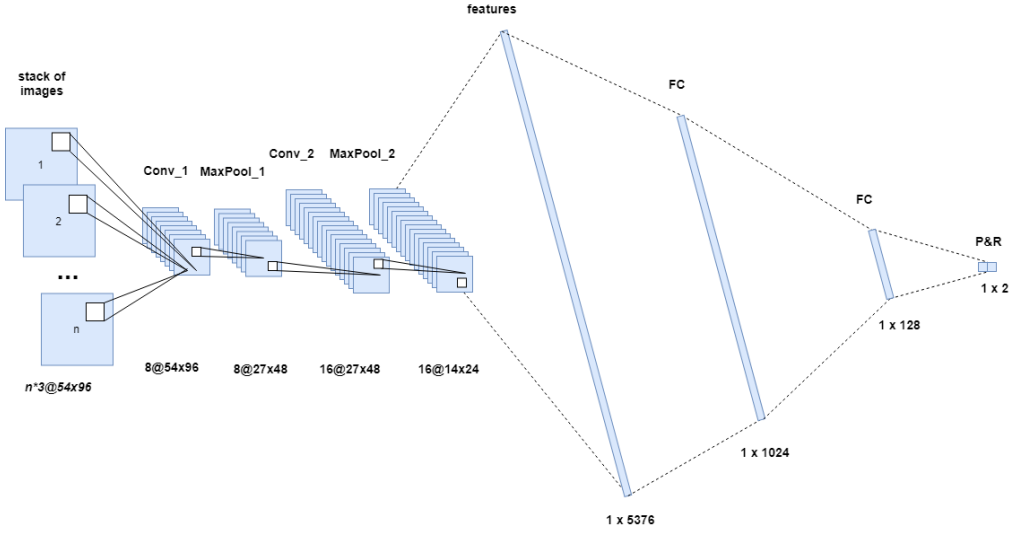


Figure A.1: Architecture of [CNN stack FC **first version**] model

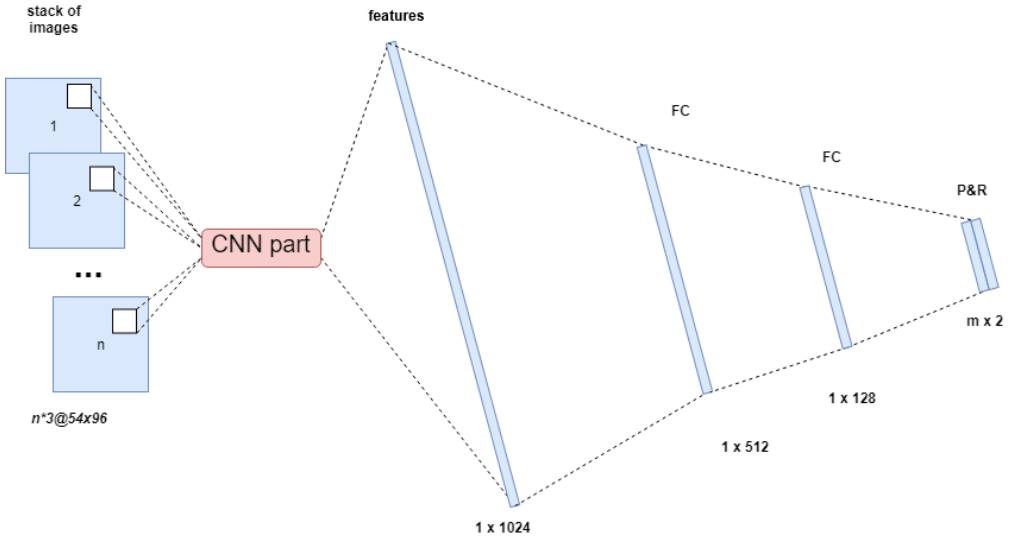


Figure A.2: Architecture of [CNN stack FC **second version**] model

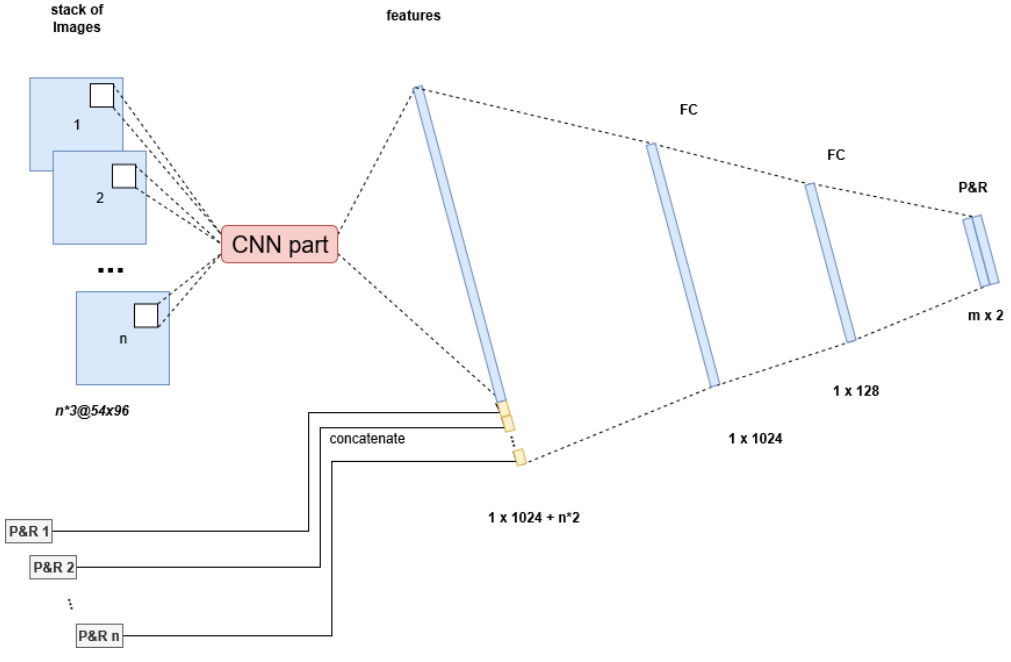


Figure A.3: Architecture of [CNN stack PR FC] model

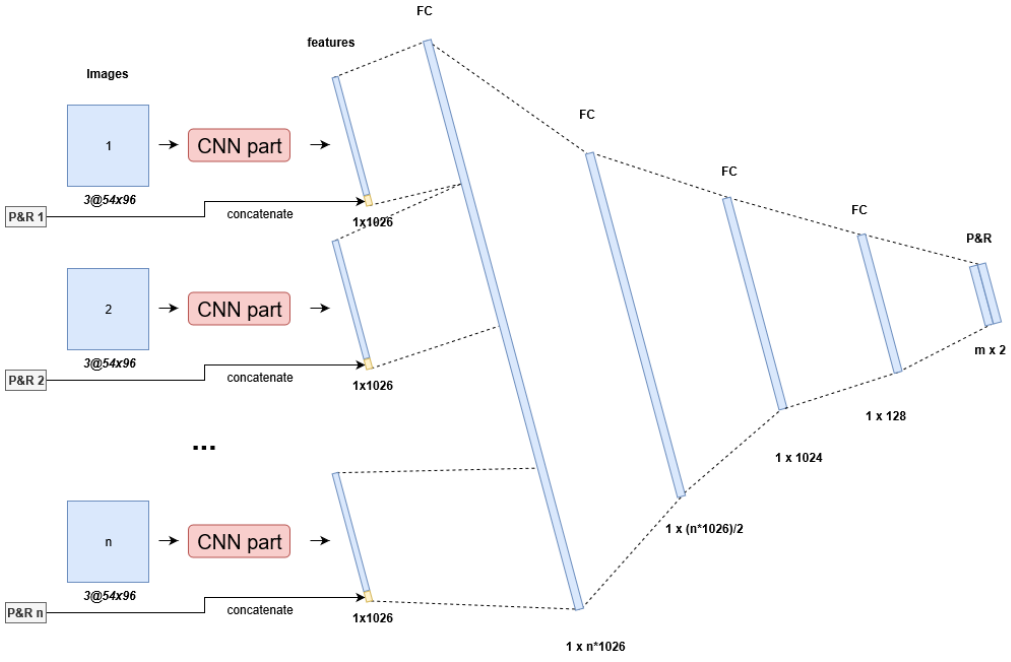


Figure A.4: Architecture of [CNN PR FC] model

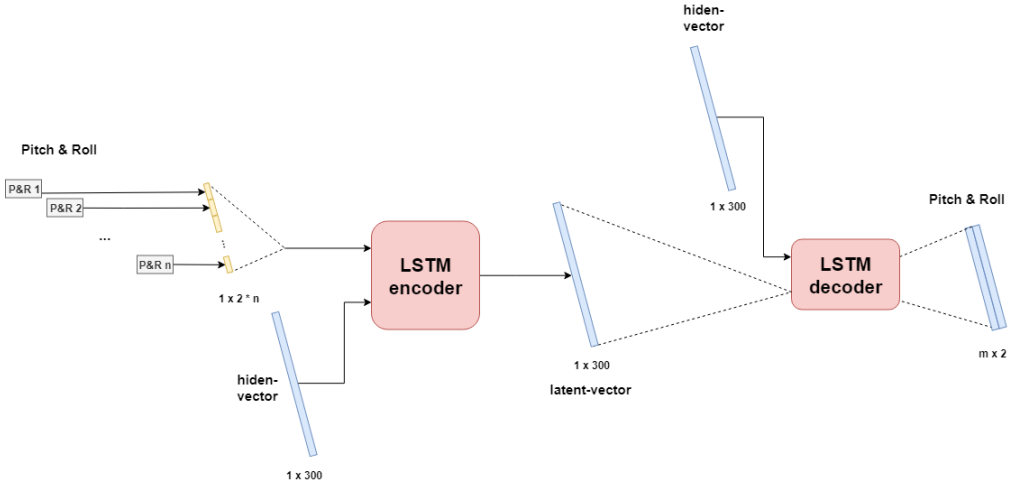


Figure A.5: Architecture of [LSTM encoder decoder PR] model

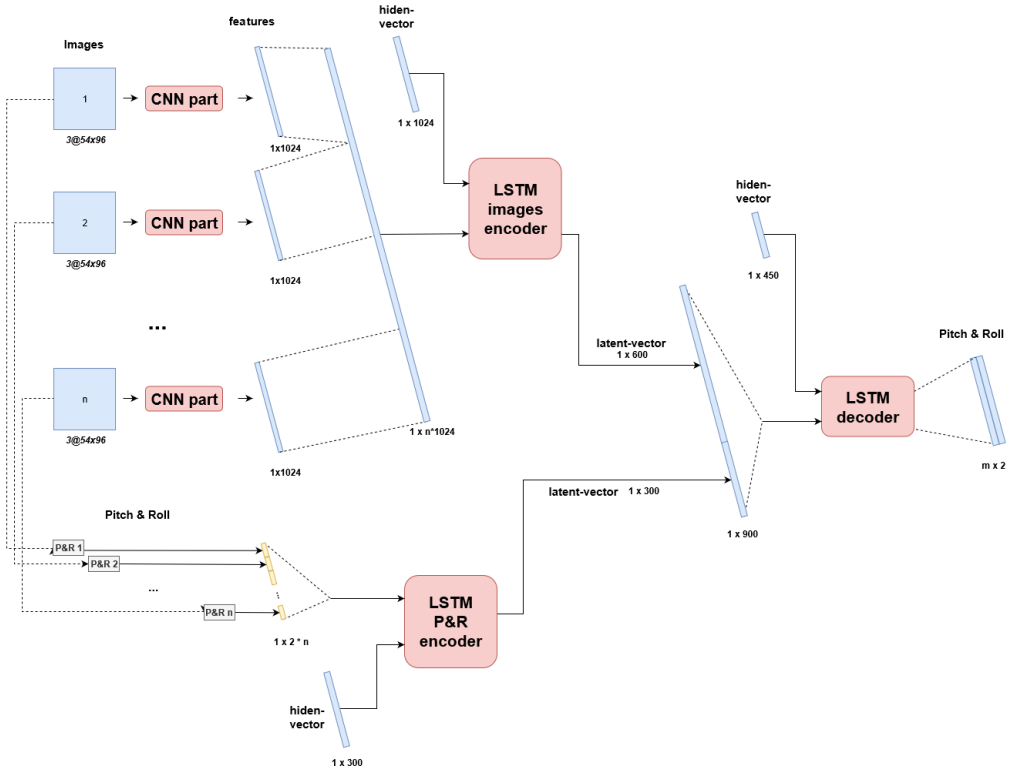


Figure A.6: Architecture of [CNN LSTM img-encoder PR-encoder decoder] model

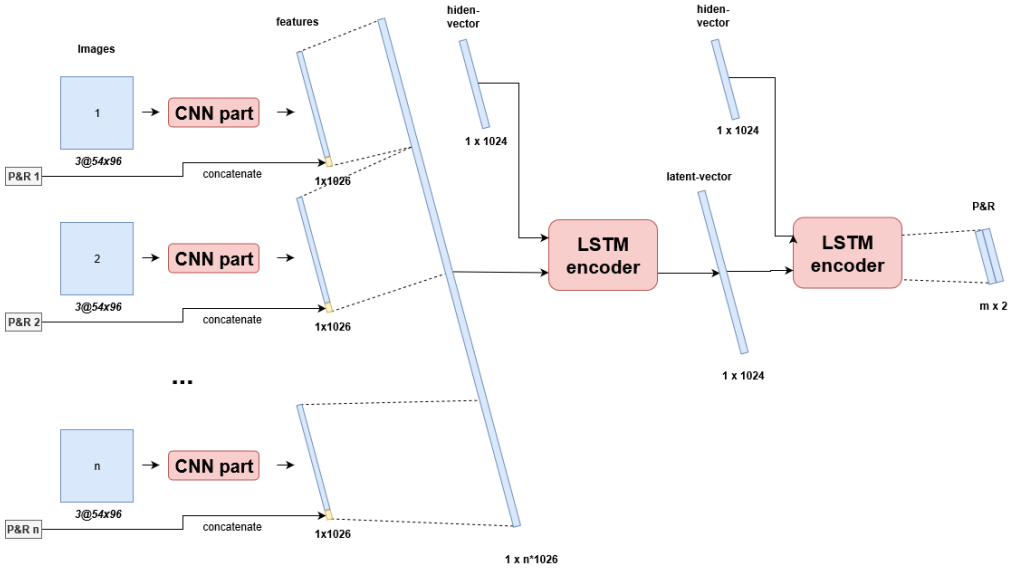


Figure A.7: Architecture of [CNN LSTM encoder decoder images PR] model

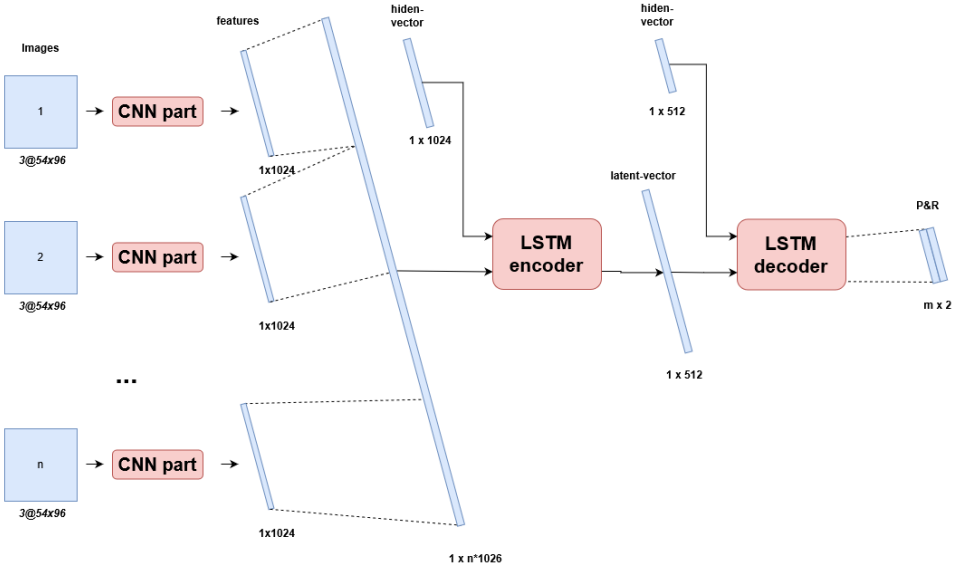


Figure A.8: Architecture of [CNN LSTM encoder decoder images] model

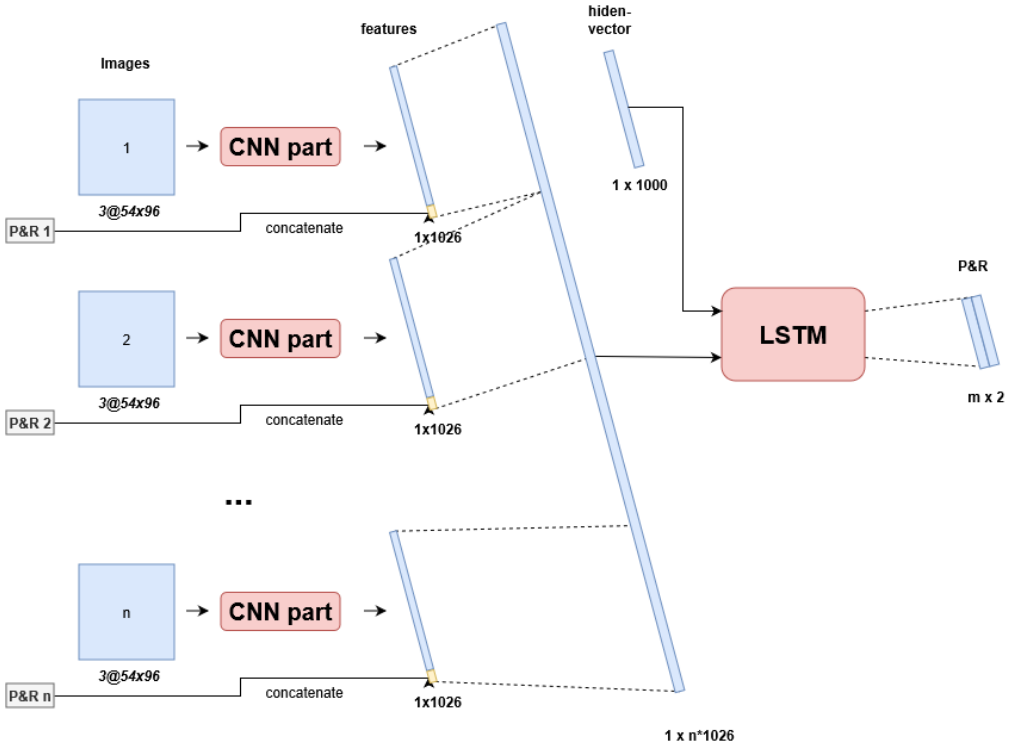


Figure A.9: Architecture of [CNN LSTM images PR] model

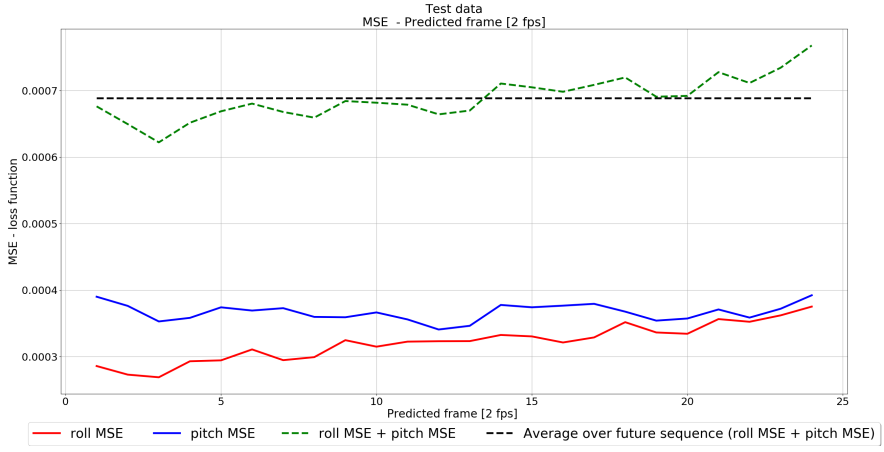


Figure A.10: Evolution of MSE test values with predicted time 12 sec [CNN LSTM encoder decoder images PR model].

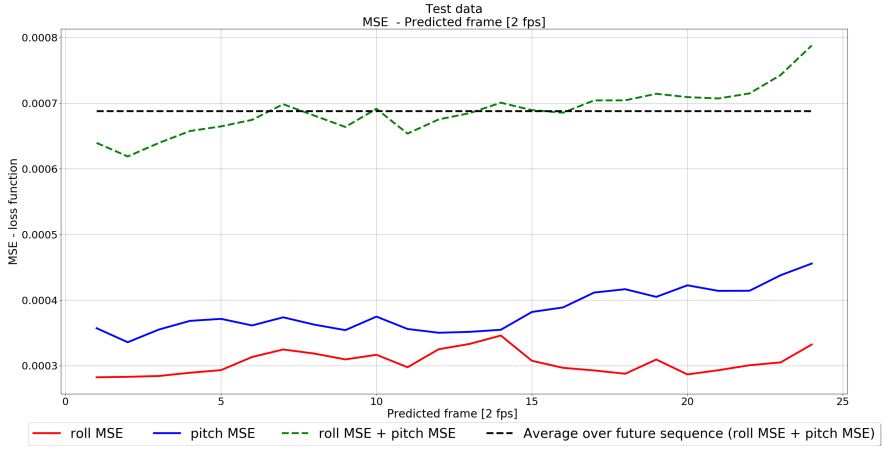


Figure A.11: Evolution of MSE test values with predicted time 12 sec [CNN LSTM encoder decoder images] model.

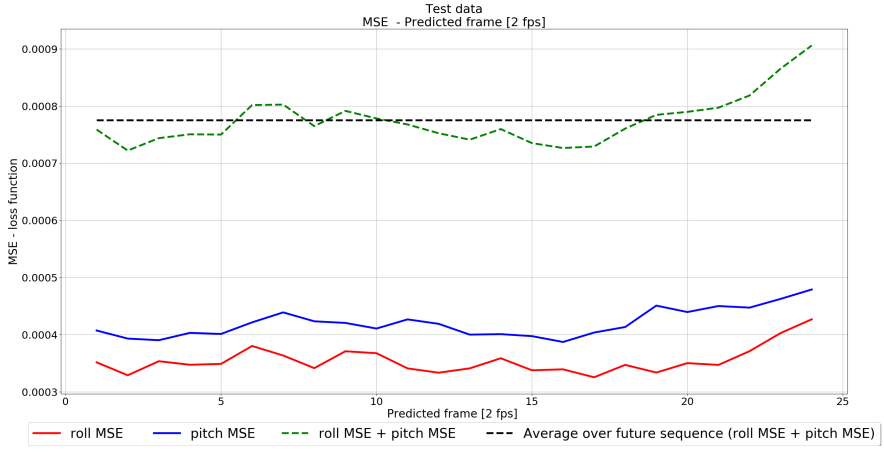


Figure A.12: Evolution of MSE test values with predicted time 12 sec [CNN LSTM img-encoder PR-encoder decoder] model.

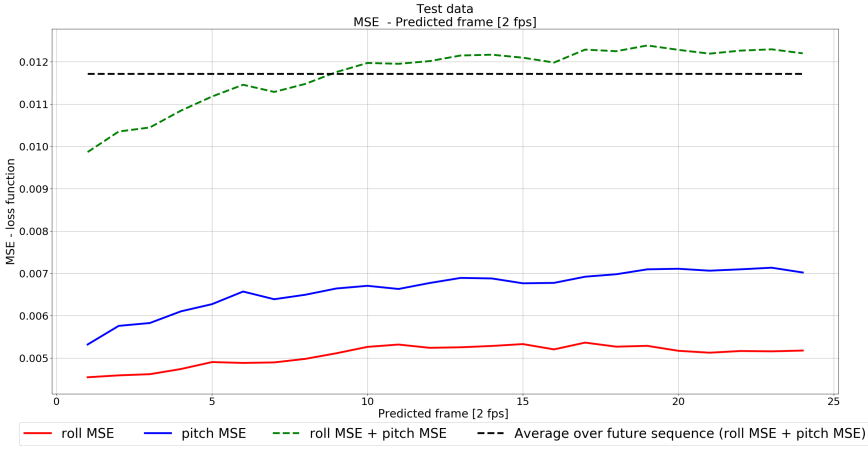


Figure A.13: Evolution of MSE test values with predicted time 12 sec [LSTM encoder decoder PR model].

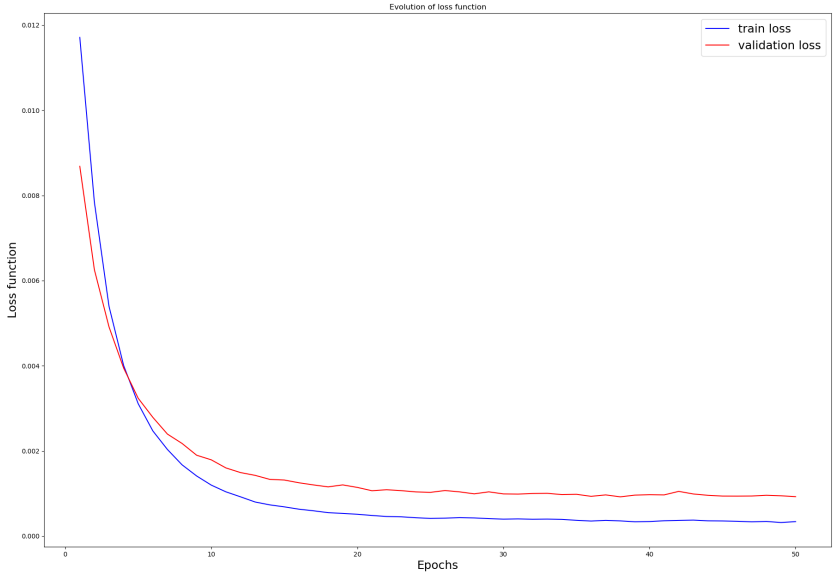


Figure A.14: Evolution of loss function for [CNN LSTM encoder decoder images PR] model. Past window size = 30 seconds (1 fps) , future window size = 20 seconds (1 fps).