Faculty of Science and Technology, Department of Physics and Technology

Machine learning approach for identification and tracking of coherent structures in turbulent fluids and plasmas

—

Leander Kirkeland

EOM-3901 Master's Thesis in Energy, Climate and Environment 30 SP - December 2022

Abstract

In a fusion reactor, coherent structures of hot and dense plasma can drift radially outwards due to the conditions of the edge plasma and can cause erosion of the outer walls. This erosion can release impurities into the plasma and harm equipment at the walls. This thesis presents two methods of tracking blobs in the boundary region of fusion experiments. The first model is a simple Long Short-Term Memory model with few layers. The second model is a more advanced transformer structure, with more depth and parameters. The performance of the models is evaluated with synthetic data, and compared on experimental data with a pre-trained model. The generation of synthetic data with different distributions in amplitude, size, velocity, and the number of blobs is also presented to better understand when the model is viable. Calculations of the velocities, amplitudes, and sizes of structures found in synthetic and experimental data are presented, where results on experimental data are compared to published results from earlier studies. The transformer-based model shows promising results on synthetic data that has low intermittency. It is shown that higher parameter variation results in worse model performance. Predictions on experimental data show that the model has some problems, including differentiating between blobs and predicting large structures from 0-values. Size and velocity estimates in experimental data are found to be in the same order of magnitude as in previous studies. The Long Short-Term Memory model shows promising results in segmenting the shape of the blobs, but lacks the capacity to differentiate them correctly.

Acknowledgments

A major thanks to all my supervisors for getting me through this thesis. Thanks to Odd Erik Garcia for helping me understand the complex concepts of plasma physics. Thanks to Gregor Decristoforo for helping me understand mechanisms in the scrape-off layer and guiding me through the thesis. Thanks to Filippo Maria Bianchi for guidance in model development and training. You have all been very helpful in making this thesis, and I am forever grateful for the opportunity to learn so much from you. Thanks to MIT for sharing experimental data to be used in this thesis.

Thanks to friends and family for all the support and patience in the last couple of years, and a special thanks to Trine for bearing with me these last few months.

Table of Contents

Abstract i

Acknowledgments ii

Abbreviations ix

Part I / Introduction 1

1 Introduction 1

2 Thesis Outline 1

Part II / Technical Background 2

3 Fusion experiments 2

- 3.1 Fusion reaction 2
- 3.2 Magnetically confined plasma 3
- 3.3 Scrape-off Layer 4
- 3.4 Plasma filaments 5
- 3.5 Alcator C-Mod 7

4 Machine learning 10

- 4.1 Segmentation 10
  - 4.1.1 Semantic Segmentation 10
  - 4.1.2 Object detection 11
  - 4.1.3 Instance segmentation 12
  - 4.1.4 Panoptic segmentation 12
- 4.2 Metrics 12
  - 4.2.1 Accuracy 12
  - 4.2.2 Intersection over union 13
  - 4.2.3 Dropout 13
- 4.3 Normalization 14
  - 4.3.1 Batch Normalization 14
  - 4.3.2 Layer Normalization 15
- 4.4 Activations 15
  - 4.4.1 Sigmoid 16
  - 4.4.2 Tanh 16
  - 4.4.3 ReLU 17
  - 4.4.4 Gelu 18
  - 4.4.5 Softmax 19
- 4.5 Recurrent neural network 20
  - 4.5.1 Gradients 20
  - 4.5.2 Long Short-Term Memory 21

4.6 Convolutional neural network 23
4.6.1 Maxpooling 25
4.7 Transformers 25
4.7.1 Encoder 27
4.7.2 Decoder 27
4.7.3 Attention 27
4.7.4 Backbone 29
4.7.5 Positional encoding 29

Part III / Proposed Method 30

5 Methodology 30
5.1 Hardware 30
5.2 Synthetic data generation 30
5.3 Models 30
5.3.1 LSTM 30
5.3.2 Transformers 33
5.3.3 Recurrent All-Pairs Field Transforms (RAFT) 37
5.4 Preprocessing data 39

Part IV / Results 42

6 LSTM-based model 42
6.1 Training set 42
6.2 Training 42
7 Transformer 46
7.1 Small scale model 46
7.2 Large scale model 46
7.2.1 Intermittency 49
7.2.2 Different sizes 51
7.2.3 Different velocities 54
7.2.4 Different amplitudes 57
7.2.5 Different x-velocities and sizes 57
7.3 Experimental data 59
8 Discussion 63
8.1 RAFT 63
8.2 LSTM 63
8.3 Transformer 63
8.4 Comparison 64

Part V / Conclusions 67

8.5 Conclusion 67
8.5.1 Future work 68

List of Tables

1 Number of parameters in the major parts of the transformer. This shows that the transformer part of the architecture has the most parameters. 34
2 Accuracy and IoU score for the first model for each of the classes. We see that the model is better at finding the background and the first blobs. 43
3 Accuracy and IoU score for the second model for each of the classes. 43
4 Accuracy and IoU score for the third model for each of the classes. The model is better at finding the background and the first blobs. 43
5 Accuracy score for the fourth model for each of the classes. We see that the model is better at finding the background and the first blobs. 45
6 Accuracy of the small Transformer model. This shows promising results compared to the LSTM. 46
7 Accuracy and IoU on the test set for the large scale transformer model. The table presents overall measurements, as well as the individual measurement for each label. 47

List of Figures

1 D-T reaction.[7] The result of fusing a Deuterium atom with a Tritium atom is a Neutron with high energy and a helium atom. 3

2 Illustration of a magnetically confined plasma in a tokamak configuration [8]. 4

3 Sketch of a tokamak divertor configuration [10]. 5

4 Radial drift of a blob structure in the SOL [14]. 5

5 A representation of how blobs are stretched in the SOL region, while being trapped by magnetic field lines [17]. 7

6 Illustration of the Gas-puff Imaging diagnostic in the Alcator C-Mod device. The left figure is from the side and thew right is the top view. Figure taken from [20] . 8

7 Experimental data with the borders of the SOL regime plotted in. The x-axis describes the distance from the center of the reactor. The y-axis is the distance to the outboard midplane. These distances are measured in meters. The red lines are the boundaries of the SOL, where the left line is the LCFS and the right one is the limiter shadow. 9

8 Segmentation 11

9 Intersection over Union [40]. 13

10 Sigmoid activation function. 16

11 Tanh activation function. 17

12 ReLU activation function. 18

13 GELU activation function. 19

14 Softmax activation function. 20

15 A single LSTM cell [65]. Here the gates described in section 4.5.2 are presented. On the left side of the cell, the forget gate is presented. In the middle is the input gate. The Right gate is called the output gate. 22

16 The repetitive nature of a recurrent neural network [66]. 22

17 How the kernel is applied to patches of the image to create new representations[72]. 24

18 U-net architechture. A segmentation method that reduces the spatial dimensions of the image to get different representation in different scales. The differerent representation is concatenated together to get better segmentation results [29]. 24

19 Max pooling example on an 4x4 image [75]. 25

20 The Transformer model architecture 26

21 Multi-head attention 28

22 One frame of the synthetic data used to train the models. In this example, there are two blobs. 31

23 The same frame, showing how the labels are presented for the blobs. 31

24 Model with described layers. In this figure, you can see the input and output values of each layer and which layers that are connected to each other. 32

25 Visualization of the sizes of the image in each layer. The yellow box is the size of the input data, and the red boxes is the LSTM layers, where one is on the whole data set, and one is on half-size images. The green boxes, described as "TimeDistributed" is the max-pooling and upsampling layers. The blue layers are the convolutional 2D layers for classification. 33

27 The three main parts of the Transformer architecture. 34

26 All the layers in the transformer architecture. It is shown how all the layers are divides and put through the transformer, then concatenated back together to get the best possible segmentation results. 35

28 Backbone structure of the transformer architecture. Using an FPN model to get mappings of the data in different resolutions. 36

29 The transformer part of the architecture. Here we have the lowest resolution level from the backbone put into the encoder layer with positional embedding. Then the output of the encoder and an array of zeros is used as input into the decoder layer. Both the encoder and the decoder are output and used later in the model. . 37

30 The head of the transformer architecture. Here we concatenate the outputs of the encoder and decoder, then concatenate all the levels of the backbone for better segmentation. 38

31 Outline of the RAFT model. 39

32 Original data. 40

33 Data after subtracting mean values over time. 40

34 Data after applying gaussian filter. 41

35 Cutoff set to 10. 41

36 Cutoff set to 50. 41

37 The input example used for comparing the different LSTM models. 44

38 Shows the prediction at timestep 70 for the second model. 44

39 Test 3 result 44

40 Multi-head attention 45

41 Accuracy graph over epochs for training and validation data. 47

42 Loss graph over epochs for training and validation data. 47

43 Histogram of x-velocities. The velocity is given in pixels per frame. 48

44 Histogram of y-velocities. The velocity is given in pixels per frame. 49

45 Histogram of amplitudes. 50

46 Histogram of sizes. Sizes are given in pixels. 50

47 Number of blobs found compared to the number of blobs in the data set. 51

48 Label showing the example of a blob changing label over time. The yellow color represents label 2, i.e. blob number 2. 52

49 The prediction of the model on the same frame. Here we can see that the model has predicted the blob as two different blobs. Yellow color in this figure represents label 3, while the green is label 2, which is the original label for the blob. This results in more blobs in examining intermittency when looking at the data set and counting the number of different labels. 52

50 Label of an example where there are a lot of blobs. Here the overlap of blobs is shown. 52

51 Prediction of the same frame as the left image. Here we can see that the model struggles when there is overlap, and given more blobs, there is bound to be overlap. 52

52 Overall accuracy of blobs and background for intermittency. A trend corresponding to the number of blobs and the accuracy is shown. The negative trend shows that the model performs better when there are fewer blobs and less overlap. 53

53 IoU of the blobs. Confirming what the accuracy graph suggests, that the more blobs, the less correct the model predict. 53
54 Accuracy graphs of different sigma values in the size parameter. There is a trend where higher variance results in lower accuracy. 53
55 IoU graphs of different sigma values in the size parameter. The same trend as shown in the accuracy is presented where the higher variance results in a lower score. 53
56 Overall accuracy of blobs and background when comparing different sizes. 54
57 IoU of the blobs comparing different sizes. 54
58 Overall accuracy of blobs and background when comparing different sizes when the labels are matched with the best predictions. 55
59 IoU of the blobs comparing different sizes when the labels are matched with the best predictions. 55
60 Input of the model, the synthetic data. 55
61 Labels given to the model. 55
62 Models prediction. Here it is shown how the model cannot predict correctly, and hence give bad result when there are large overlapping blobs. 55
63 Overall accuracy when looking at velocities sampled from different distributions. This analysis is done with all blobs' labels set to one. 56
64 IoU when looking at velocities sampled from different distributions. 56
65 Overall accuracy of blobs and background when comparing different x-velocities. 56
66 IoU of the blobs comparing different x-velocities. 56
67 Overall accuracy of blobs and background when comparing different x-velocities. 57
68 IoU of the blobs comparing different x-velocities. 57
69 Overall accuracy of blobs and background when comparing different amplitudes. 58
70 IoU of the blobs comparing different amplitudes. 58
71 Overall accuracy of blobs and background when comparing different x-velocities and sizes. 58
72 IoU of the blobs comparing different x-velocities and sizes. 58
73 The raw output of the model on experimental data, shown for timestep 192. 59
74 The input of the model. it is shown that the blobs, even after preprocessing do not have gaussian distributed shape. 59
75 Histogram of amplitudes for experimental data. 60
76 Histogram of x-velocities for experimental data. 61
77 Histogram of y-velocities for experimental data. 61
78 Histogram of sizes for experimental data. 62
79 Comparison between the models' segmentation. In this example, the models find the same blob. The LSTM and Transformer predictions are somewhat larger than RAFT. 64
80 Another good example, except for the small part where the transformer classified a small blob outside of the larger one. 64
81 Comparison between the models' segmentation. In this example the models predict different blobs, and the LSTM predicts a random part not presented in the input. This is the same behavior that the Transformer has on some of the blobs. 65
82 An example of the "large structures from nothing" problem with the transformer. It is shown in the figure that the LSTM predict a small blob in the middle and RAFT predicts nothing, while the transformer predicts large structures across the image. 65

Abbreviations

Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FPN	Feature Pyramid Networks
GeLU	Gaussian Error Linear Units
GRU	Gated recurrent unit
ITER	International Thermonuclear Experimental Reactor
LCFS	Last closed flux surface
LSTM	Long Short-Termed Memory
MIT	Massachusetts Institute of Technology
MLP	Multilayer perceptron
NLP	Natural Language Processing
RAFT	Recurrent All-Pairs Field Transforms
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SOL	Scrape-off layer

Part I / Introduction

1 Introduction

This thesis investigates tracking of coherent structures in the scrape-off layer (SOL), the boundary of a magnetically confined plasma. These structures are often referred to as ”blobs”, and they move radially outward due to the conditions of edge plasma and can cause erosion of the outer walls.

The main aspect of the thesis is to use deep-learning methods to identify and differentiate between these blobs.

Synthetic data is generated by a two-dimensional model of advecting and dissipating structures. This data is used to train the models presented and analyze how well the models are performing.

The first model presented is a Long Short-Term Memory based model. This model is trained on synthetic data and later analyzed with accuracy and intersection over union measurements. This model is trained on different data sets to improve performance. Experimental data predictions are also explored for this model.

The second model is a transformer-based model for a smaller resolution data set. To save time, this model is implemented to check how the architecture performs on this type of data. This model is very memory heavy and not suitable for larger data sets. A simpler version of this architecture is presented and used on larger synthetic data sets more suitable for experimental data. An extensive analysis of this model is presented, and this model is tested on experimental data.

In this thesis I will be comparing these two models as well as utilizing a baseline method called Recurrent All-Pairs Field Transforms (RAFT) [1, 2]. This comparison will reveal where the models agree and how well the models presented in this thesis perform.

2 Thesis Outline

Section 3 presents the theory about fusion reactors and filament propagation.

Section 4 presents the theory about machine learning and methods used in this thesis.

Section 5 presents an overview of the methodology the thesis uses.

Section 6 introduces the Long Short-Term Memory-based model, with testing on experimental data.

Section 7 introduces the Transformer-based model. Testing on synthetic and experimental data is presented.

Section 8 presents a discussion of how well the models perform compared to RAFT.

Section 8.5 presents the conclusion of the thesis.

Part II / Technical Background

3 Fusion experiments

In a world with increasingly growing energy demand, new thinking and better solutions are always the way to go for increased efficiency. Although not a new idea, fusion energy shows promising results in the energy produced due to low quantities of fuelling materials required and significantly improved environmental pollution footprint.. The idea of harnessing energy from fusion reactions is almost a century old. However, there are still some challenges that need to be further investigated to have a stable fusion reactor that puts electricity on the grid. One of these challenges is heat fluxes drifting across magnetic field lines to the main chamber walls.

The idea of nuclear energy arose from the heat generation of radioactive isotopes used in x-rays. This speculation led to the investigation of how to harness this heat. In 1911, Ernest Rutherford commented on the vast amount of heat produced from radioactive isotope [3], but there was little understanding of what was causing this heat exhaustion.

Rutherford continued to experiment on atoms, and in 1934 he fused deuterium into helium and discovered tritium. This is considered the first nuclear fusion process created by humans. This reaction created massive amounts of energy and gave birth to the idea of harnessing energy from fusion.

In the 1950s, scientists worldwide explored the possibility of harnessing this energy source. One of the first fusion reactors was the tokamak, which Andrei Sakharov and Igor Y. Tamm proposed. The first tokamak reactor was created in 1958 [4], and called T-1.

In the following years, several approaches to fusion reactors were proposed. The most discussed ones are the tokamak and the stellarator. There are differences between them, but they both use the torus as the shape of the plasma. While the tokamak uses a toroidal current to generate a poloidal field to contain the plasma, the stellarator uses external non-axisymmetric coils. In this thesis, experiments from tokamaks will be the center of attention.

In more recent years, the research has been focusing on improving the already existing reactor types. This includes enhancing the equipment used in a reactor, such as magnets and sensors.

3.1 Fusion reaction

There are different possibilities as to which materials to use as fuel in a fusion reaction, but the most promising of them is Deuterium-Tritium (D-T). Deuterium (D) is a stable isotope of hydrogen (^{2}H) and can be found in seawater, approximately 0.153 to 0.156 ‰[5]. When it comes to tritium(T), which is radioactive, one can produce it from lithium metal

[6]. This reaction is shown in figure 1, where the amount of energy released in the process is also represented. An equation for the reaction is given as

$$D + T \rightarrow He(3.52MeV) + N(14.06MeV) \tag{1}$$

where D and T are Deuterium and Tritium, and He and N are Helium and a Neutron. For each reaction, 17.6 MeV is produced, which is used to fuel further reactions and produce heat to run a turbine for electricity generation.

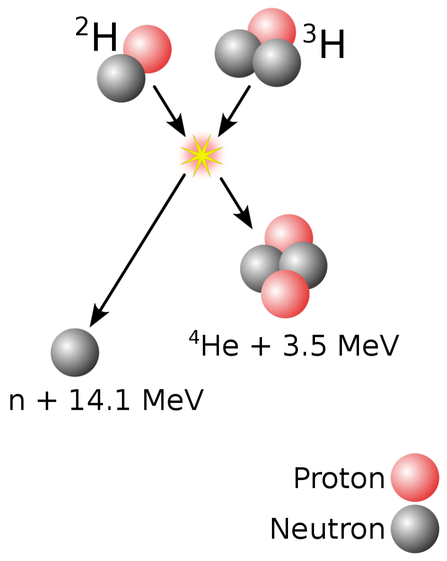

Figure 1: D-T reaction.[7] The result of fusing a Deuterium atom with a Tritium atom is a Neutron with high energy and a helium atom.

3.2 Magnetically confined plasma

A plasma is a state of matter where particles are ionized, i.e., split into electrons and ions. A magnetically confined plasma is defined as a plasma controlled by a magnetic field. This makes the species susceptible to the Lorentz force, and this force traps the charged particles along magnetic field lines. In the context of this thesis, the confined plasma is in a fusion reactor with limited space for the plasma to expand. The tokamak configuration is shown in figure 2, where the torus shape of the reactor is illustrated. This configuration uses the poloidal coils shown in the figure to generate a toroidal field. This toroidal magnetic field traps the plasma in the direction of the magnetic field. The Lorentz force and the increase of the temperature accelerate the plasma, and traps it along the magnetic field lines. The central solenoid creates a poloidal field that accelerates the plasma in the poloidal direction. The result of this configuration is a helical movement of the plasma shown as the black line in figure 2. In the figure, there are outer poloidal field coils, which help the configuration shape and position of the plasma.

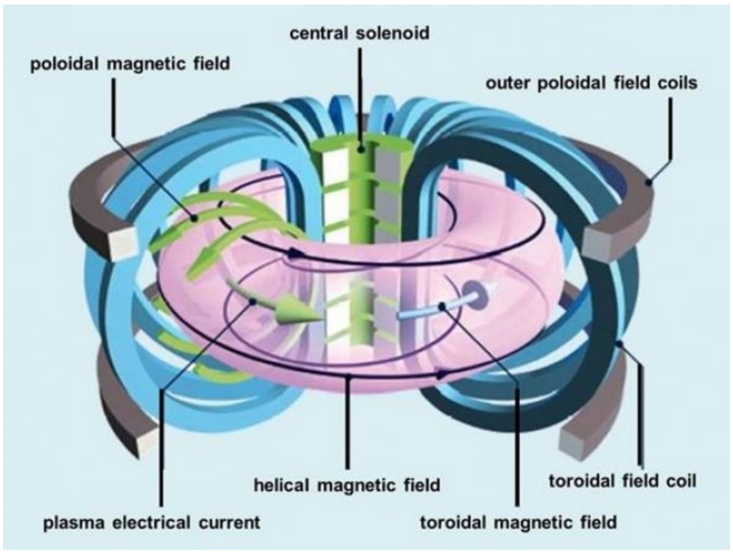

Figure 2: Illustration of a magnetically confined plasma in a tokamak configuration [8].

3.3 Scrape-off Layer

The scrape-off layer (SOL) is defined as the plasma region in the outer edge, characterized as the region with open magnetic field lines that intersect with the divertor plates. In a reactor, divertor plates are solid plates used as a collector of ash and heat transported in the SOL. The separation of the open and closed field lines is called the separatrix, or the last closed flux surface (LCFS). This surface is also where the main plasma is separated from the SOL plasma. Ideally, all particles that diffuse across the separatrix are transported to the divertor plates to be collected there instead of radially drifting the main chamber walls and causing plasma–wall interactions close to the main plasma column [9].

However, this is not always the case, where plasma filaments, described in section 3.4, could radially drift in the SOL and reach the walls. This section is crucial in having a stable reactor where filaments do not reach the wall and damage equipment. Figure 3 shows a tokamak configuration, where the SOL is the region outside the separatrix line, shown in a different color. This figure illustrates how the divertor plates intersect with the open field lines in the SOL.

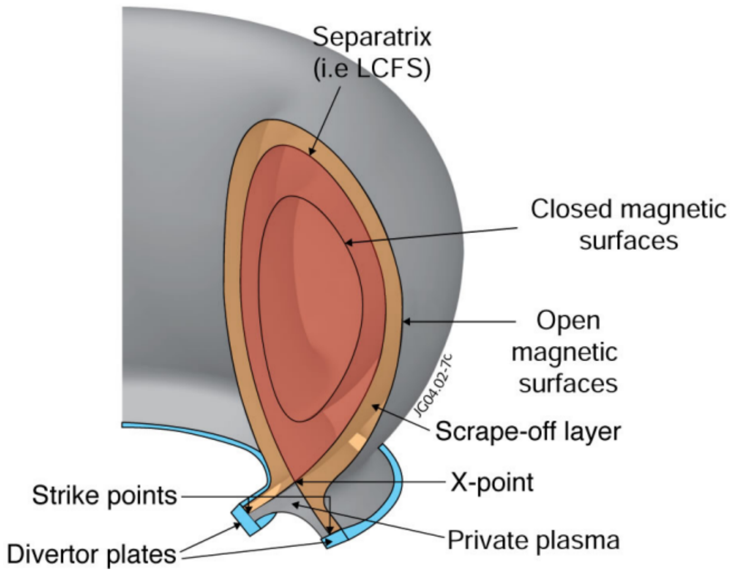

Figure 3: Sketch of a tokamak divertor configuration [10].

3.4 Plasma filaments

A plasma filament, often referred to as a blob in the plane perpendicular to the B field, is a magnetic-field-aligned plasma structure denser than its surrounding plasma [9, 11]. These blobs are often generated close to the separatrix and move through the SOL radially, transporting heat and particles to the outer edges of the SOL [12]. In theory, these blobs seem to be created by nonlinear saturation of turbulence or MHD instabilities in the edge plasma. They have been experimentally observed to be formed near the boundary between open and closed field lines[13].

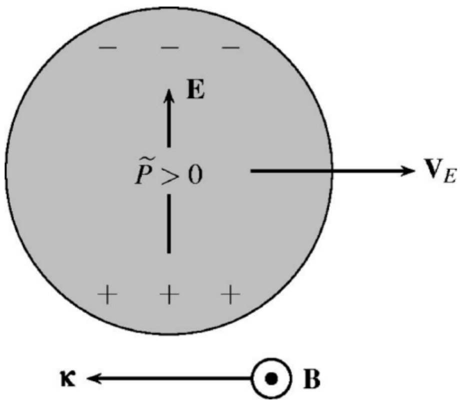

Figure 4: Radial drift of a blob structure in the SOL [14].

In figure 4, the mechanism of the radial transport of the blobs is visualized, and the drift of the blobs is shown as $\mathbf{V}_E$. The inhomogeneous magnetic field in the reactor induces a drift that is charge dependent [15], often referred to as the ∇B drift. The equation of the ∇B drift is described as

$$\mathbf{u}_{\nabla B} = \frac{mv_\perp^2}{2q}\frac{\mathbf{B} \times \nabla_\perp B}{B^2}, \tag{2}$$

where $\mathbf{u}_{\nabla B}$ is the ∇B drift, m is the mass, $v_\perp$ is the high frequency gyro motion velocity, $\mathbf{B}$ is the magnetic field, $\nabla_\perp B$ is the gradient of the magnetic field, and q is the charge of the particle. The charge dependency of this drift causes the electrons and ions to drift in opposite directions, polarizing the blob. This polarization creates an electric field inside the blob, shown in figure 4 as $\mathbf{E}$. The electric field generated inside the blob combined with the magnetic field of the reactor creates an $E \times B$ drift outwards, described as

$$\mathbf{V}_E = \frac{\mathbf{E} \times \mathbf{B}}{B^2}. \tag{3}$$

In this equation, $\mathbf{E}$ is the induced electric field, and $\mathbf{B}$ is the magnetic field in the toroidal direction. It is this drift, shown as $\mathbf{V}_E$ in figure 4 and equation 3, that transports the blobs radially outwards. The $\mathbf{E} \times \mathbf{B}$ drift transports ions and electrons in the same direction due to no charge dependency.

In the SOL plasma, the blobs are stretched along the field lines. This is shown in figure 5, where blobs are trapped along magnetic field lines. The interchange motion caused by a non-uniform magnetic field can be further analyzed to be dependent on different parameters. Studies show that blob velocity scales with the size and amplitude of the blobs [16]. These velocities scale differently depending on the regimes. The first regime is a plasma with no sheath current due to no divertors or small-sized blobs and the plasma is moving radially outward, and the other is where there are divertors and the plasma is moving towards the divertors. In tokamaks, the divertor plates generate sheaths on them which would otherwise be on the outer walls. The sheath currents in the SOL reduce the radial velocities of the blobs. This sheath dissipation helps the configuration control where the plasma ends up, and the particles move rapidly toward the divertors instead of the walls. With the sheath current, the velocity scaling is given as

$$\frac{V}{C_s} = \left(\frac{2l}{R}\frac{\Delta n}{N}\right)^{\frac{1}{2}} \tag{4}$$

where V/C_s is the initial velocity scaling, C_s is the acoustic speed, l is the cross-field blob size, R is the major radius of the reactor, and $\Delta n/N$ is the relative amplitude. From equation 4 it is shown that the velocity depends on the square root of amplitude and size of the blobs. Without sheath currents in the SOL, the velocity scaling is described as

$$\frac{V}{C_s} = \frac{2L_\parallel}{R}\frac{\rho_s^2}{l^2} \tag{5}$$

where $L_{parallel}$ is the magnetic connection length and ρ_s is the sound gyration radius. The velocity scaling in this regime is inversely proportional to the cross-field blobs size, while the relative amplitude unaffected the results.

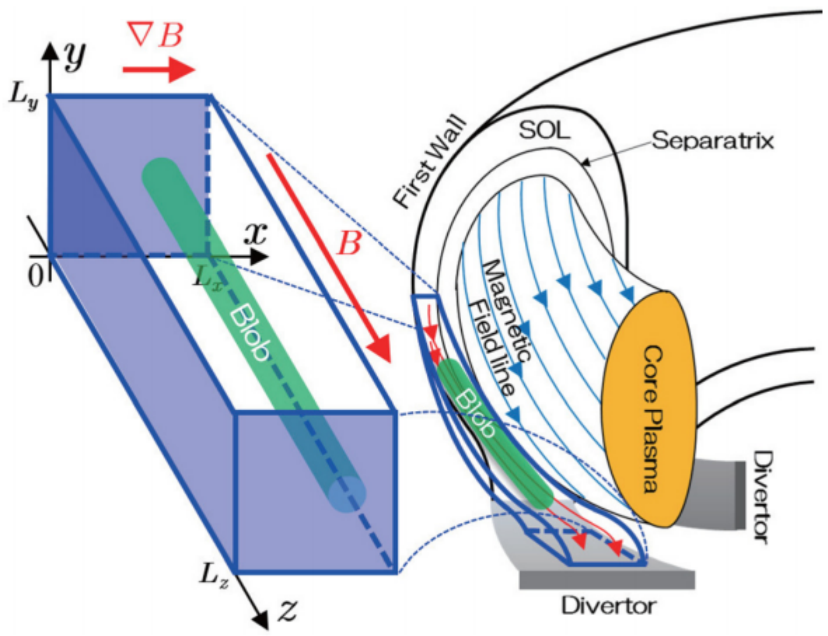

Figure 5: A representation of how blobs are stretched in the SOL region, while being trapped by magnetic field lines [17].

3.5 Alcator C-Mod

In this thesis, experimental data from the Alcator C-Mod tokamak will be used to compare the blob tracking models. C-Mod is the third reactor in a series of reactors built by the Massachusetts Institute of Technology (MIT), first deployed for experiments in 1993 [18]. This reactor is larger than the previous Alcator reactors but is still relatively small compared to other reactors. C-Mod has a high particle density, given a high B/R. This allows the reactor to perform analysis on fully equilibrated electrons and ions. These specifications make the reactor especially suitable for exploration in the SOL region, divertor physics and technology studies. For this thesis, the edge characteristics are the most important parts, where the C-Mod reactor reaches levels comparable to the expected values of International Thermonuclear Experimental Reactor (ITER) [19]. This makes experiments from C-Mod highly relevant. The reactor was shut down in 2016, but the results are still being explored and included in scientific work today.

In these types of experimental data, there can be a lot of different resolutions depending on the equipment used to capture it and the purpose of the data. How the data is collected is shown in figure 6, where the camera is placed at the midplane of the reactor.

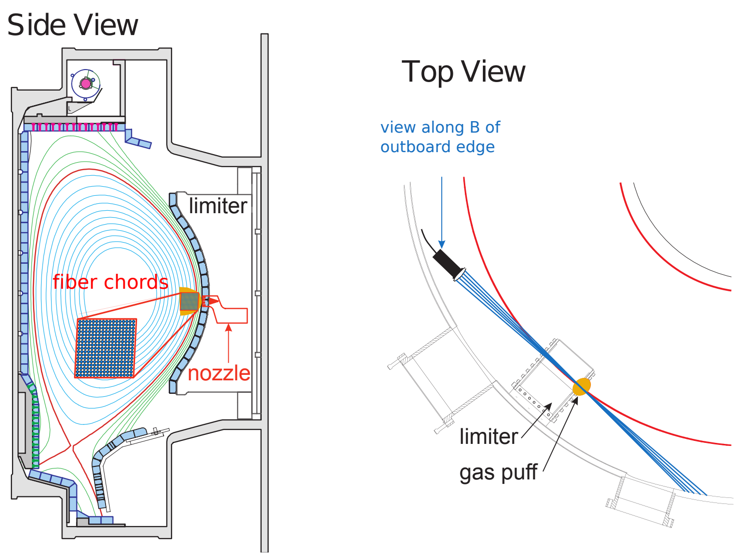

Figure 6: Illustration of the Gas-puff Imaging diagnostic in the Alcator C-Mod device. The left figure is from the side and thew right is the top view. Figure taken from [20]

Collected data from Gas puff imaging (GPI) is used as the experimental data in this thesis. This diagnostics method uses neutral gas, such as helium or deuterium, to enhance local emissions caused by collisions in the plasma edge [21, 22]. This method produces a higher line radiation intensity where the density of particles is higher than the surrounding parts. This opens the opportunity to follow structures with larger particle densities over time for analysis and experimentation in the SOL.

There is a lot of information to be gathered from these experimental data, but the main focus in this thesis is velocities, amplitudes, size, and intermittency. Different methods have been implemented to compute these variables, such as Recurrent All-Pairs Field Transforms (RAFT) [1, 2]. RAFT will be described in later sections.

The raw data from the experiments discussed in this thesis has a dimension of 64x64. The experimental data will be run on different models trained in this thesis, as well as a baseline model used in other papers as a comparison. In figure 7, the raw data, the LCFS, and the limiter shadow are plotted to give an overview of where the SOL region is located. The limiter shadow is the region where magnetic field lines interact with vessel walls. In this example, the noise of the experimental data is presented, which can induce problems for machine learning models.

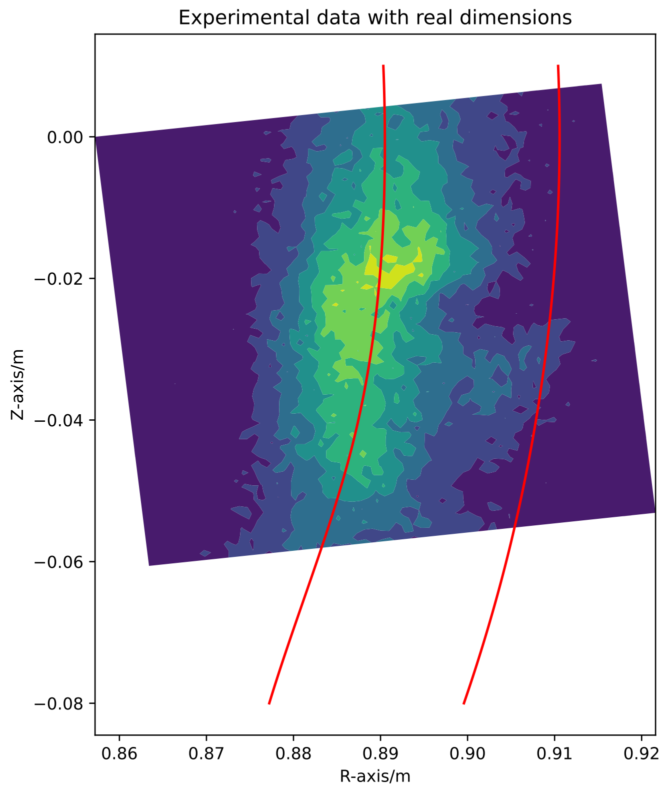

Figure 7: Experimental data with the borders of the SOL regime plotted in. The x-axis describes the distance from the center of the reactor. The y-axis is the distance to the outboard midplane. These distances are measured in meters. The red lines are the boundaries of the SOL, where the left line is the LCFS and the right one is the limiter shadow.

4 Machine learning

Machine learning was first used by Arthur L. Samuel in “Some studies in machine learning using the game of checkers” [23]. Machine learning sprung out of the idea of machines learning to represent data and predict results from the given data.

In machine learning, there are a lot of branches, and one of these branches is supervised learning. Supervised learning builds on the idea that the model will learn the representation based on a set of data and labels of the data. This method requires a training set that consists of data that is going to be learned and corresponding labels that describe what we want the model to do. Examples of supervised learning are classification and regression [24], which are discussed later in this thesis.

Another branch of machine learning is unsupervised learning where there is no need for data labels, i.e., the desired output of a classification/regression task. The main idea of unsupervised learning is to find interesting transformations of data without the help of labels. This type of learning is used for data visualization, data compression, or data denoising, or to get a better understanding of the data. Examples of unsupervised learning are clustering and dimensionality reduction [25].

This thesis will cover CNN-based architectures that use supervised learning, as well as transformers which are self-supervised. A self-supervised architecture uses both supervised and unsupervised methods. The transformer architecture uses unsupervised as a pretraining method and then supervised learning to fine-tune the model.

4.1 Segmentation

For this task, visualization is key. Therefore, segmentation is the tool of choice. Segmentation is a tool to discriminate between instances called segments, divisible by some feature. In this thesis, different segmentation methods will be described, but they all have in common that they divide the image into different segments by pixel-by-pixel classification. The reason for using segmentation is to create a more meaningful and easier-to-understand representation of the videos for analysis purposes.

Within segmentation, there are three branches, all with their own methods. These three branches are described below, as well as object detection, which is closely related to the subject but not under the segmentation category.

4.1.1 Semantic Segmentation

The first branch, semantic segmentation, is a fairly simple segmentation method, which only separates the classes. In the instance of this thesis, we would like to separate between two classes: blob or background. Semantic segmentation is used to differentiate between classes that distinguish from each other in features. Each pixel is classified as a given

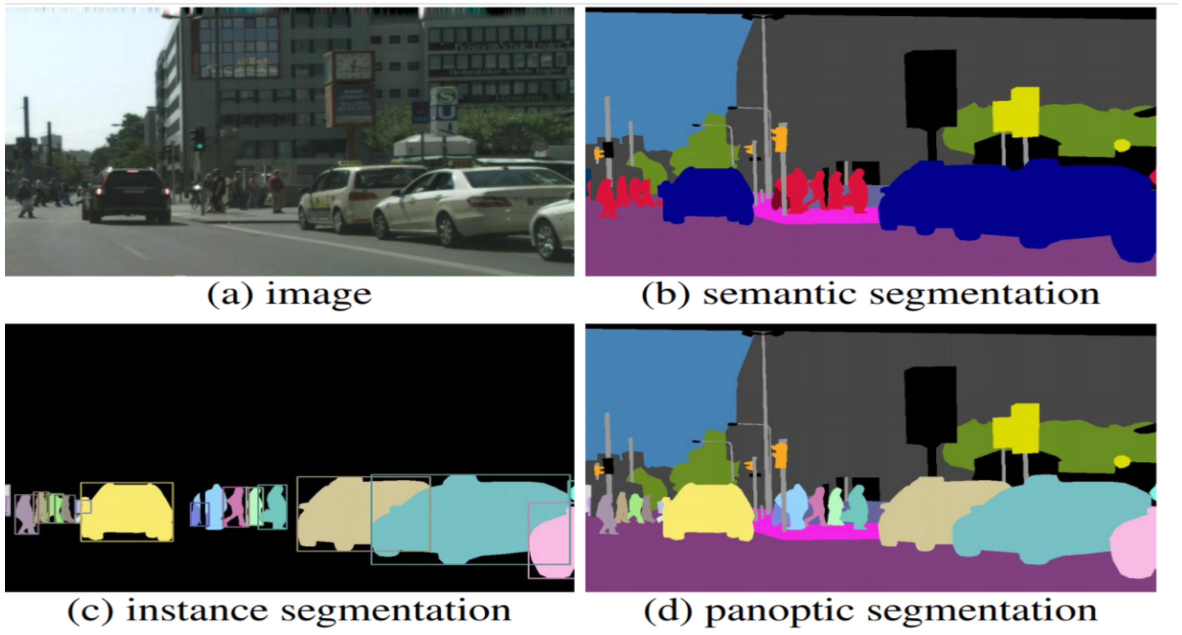
(a) image
(b) semantic segmentation
(c) instance segmentation
(d) panoptic segmentation

Figure 8: Different segmentation types. (a) shows the input image, (b) shows semantic segmentation, (c) shows instance segmentation, and (d) shows panoptic segmentation. Image is taken from [26].

class, and when you recreate the image or video, you can see where the different classes are in the image or video.

In figure 8 semantic segmentation is presented in (b). Here you can see that all the objects are classified as the same class, and each instance is not differentiable.

There are many implementations of semantic segmentation. Among them TMANet [27], Segmenter [28] and the most famous U-Net [29].

4.1.2 Object detection

Although not a segmentation method, object detection is closely related to the subject. In object detection, you want to find objects in the image or video, classify them and get their position. This is done by bounding boxes, where a set of 4 numbers represent the box. There are different approaches to how you build the boxes, but one example is bbox = (start_x, start_y, width, height), where the first point is the starting pixel on the x-axis, the second point is the starting pixel on the y-axis, the third point is the number of pixels along the x-axis (width), and the last point is the number of pixels along the y-axis (height). This method is different from semantic segmentation because it only finds objects of interest and does not care about the background. This method is often used to separate both classes and instances. An instance in this meaning could be one person in a crowd of people, for example.

In figure 8 the bounding boxes are shown in (c), as the rectangular box around each object.

Examples of implementations are presented in Fast R-CNN [30] and Faster R-CNN [31].

4.1.3 Instance segmentation

Instance segmentation combines both semantic segmentation and object detection to segment instances. In this branch, the model predicts the bounding boxes, the mask (the segmentation), and the class of each separate instance. In this thesis, there is only one class, the blob.

In figure 8, instance segmentation is shown in (c). It is shown how each car and person has its color and bounding box, meaning they have their own representation, and you can analyze them separately instead of in one chunk. However, you can see that it only detects and segments foreground objects and does not care about the background.

Examples of implementations are presented in Mask R-CNN [32], Istr [33], and TT-SRN [34].

4.1.4 Panoptic segmentation

Panoptic segmentation combines the best of semantic and instance segmentation in that it can differentiate between instances within a class as well as classify the background. Panoptic segmentation was presented in [26], as a complete task for real-life applications. This method is not as well tested as the others, but it would be ideal for this type of problem.

In figure 8 panoptic segmentation is shown in (d). It is shown that each person and car has different colors, and you can separate each instance.

Examples of implementation are Panoptic SegFormer [35], UPSNet [36], and Panoptic-deeplab [37].

4.2 Metrics

Better and more comprehensive model benchmarking is needed to better understand and develop models [38]. In this part, a few metrics will be presented to be used as a comparison between the different models used.

4.2.1 Accuracy

Accuracy is the most used metric in comparison because of its simplicity. This metric is a great starting point to compare models in how they perform on a given data set. However, in this thesis, accuracy is not the most desired metric to use as it only shows the whole data set's comparison on pixel-wise classification. An example of how accuracy is not always practical is in the case of an unbalanced data set. Given a data set that consists of 90% of class one and 10% of class 2, the model would give a 90% accuracy when classifying the whole image as class 1. Generally speaking, an accuracy of 90% is good, but in this example you want to find class 2, and therefore accuracy does not tell us how well the model performs. There are augmentations to accuracy, such as class-wise accuracy, which will also be used in this thesis.

4.2.2 Intersection over union

Intersection over Union (IoU) is generally used in segmentation tasks. This metric has the advantage over accuracy in that it compares the overlap of the model's prediction with the label, and we would like to maximize this number to get the best predictions. The equation for IoU is given as

$$IoU = \frac{A \cap B}{A \cup B}, \tag{6}$$

where A is the area of the label, B is the area of the prediction, and ∩ and ∪ are intersection and union, respectively. In this equation, the intersection, i.e., the overlapping area of the model's prediction and the ground truth, is divided by the union, i.e., the area of both combined [39]. By doing this calculation, one can generate the rate at which the items overlap and how much of the space is not classified correctly. A visualization of the metric is shown in figure 9, where you see the idea behind this metric.

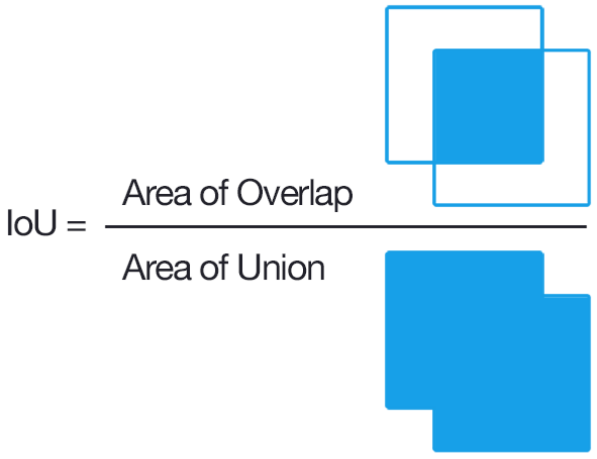

Figure 9: Intersection over Union [40].

4.2.3 Dropout

Dropout is primarily used to reduce overfitting in larger models by assigning each neuron with a probability of keeping it active. Dropout removes some neurons from the forward pass with a probability p_{drop}, so the model does not rely on a small number of neurons to activate, but instead the whole model. Before training you need to define the hyper-parameter p_{drop}, and this is usually found during the validation tests of the model [41]. This method can better generalize the model, hence increasing the model's performance on unseen data [42]. In a recurrent neural network dropout has been shown to give a bad performance in recurrent layers, so it usually only gets applied to the input and the output layer [43].

4.3 Normalization

Normalization of data is known to be beneficial to neural network training in the sense that it reduces the training time of the model. The main objective of normalization is to scale data to the same range of values for smoother and faster training [44].

4.3.1 Batch Normalization

Batch Normalization builds on the same idea as normalization. However, instead of normalizing the whole dataset, it normalizes a batch. A batch is a small portion of the whole data set. Batch normalization is done using two parameters, the mean and standard deviation of the batch. The equation for the mean is given by

$$\mu_b = \frac{1}{B}\sum_{i=1}^{B} x_i \tag{7}$$

where x_i is the input data, B is the batch size, and μ_b is the mean of the batch. The equation for the standard deviation is given by

$$\sigma_b^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_b)^2 \tag{8}$$

where σ_b is the standard deviation of the batch. The batch normalization, $\hat{x}_i$, is then given by

$$\hat{x}_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2}}, \tag{9}$$

where $\hat{x}_i$ is the normalized data. A small value ϵ is often added to σ to avoid division by 0. While training a network, the parameters of the model layers are constantly changing to get the best results. This process complicates the training of models because the inputs of each layer are dependent on the parameters of the previous layer [45].

Some of the benefits of batch normalization include enabling higher learning rates which in turn reduces training time, and better network generalization due to a wider minima in the loss function [45, 46]. In training machine learning models, you want to minimize the loss function. Having a wider minima in the loss function makes it more likely that the model finds the best representation of the data, since it is easier for the model to find the minima using larger learning rates. However, batch normalization has some limitations. If the batch sizes are small, the batch mean and standard deviation could not be representative of the whole data set, raising the issue of changing the input data of certain layers in each batch, resulting in the model constantly changing the parameters of that layer and the model not learning anything. It also depends on batch statistics, meaning it is less suited for sequential data such as the data used in this thesis. Batch normalization is often used in CNN-like networks.

4.3.2 Layer Normalization

Layer normalization was first proposed in [47] as a replacement for batch normalization for sequential models. Layer normalization, similar to batch normalization, reduces training time by normalizing parts of the training set. Layer normalization normalizes mean and variance over all features on a single training case. This means that the model calculates the statistics at each time step instead of in each batch. Layer normalization is done using two parameters, the mean and standard deviation of the batch. The equation for the mean is given by

$$\mu_l = \frac{1}{H}\sum_{i=1}^{H} x_i \tag{10}$$

where x_i is the input data, μ_l is the mean of all the hidden units in a layer, and H is the number of hidden units in a layer. The equation for the standard deviation is given by

$$\sigma_l^2 = \frac{1}{H}\sum_{i=1}^{H}(x_i - \mu_l)^2 \tag{11}$$

where σ_l is the standard deviation of all the hidden units in a layer. The layer normalization, $\hat{x}_i$, is then given by

$$\hat{x}_i = \frac{(x_i - \mu_l)}{\sqrt{\sigma_l^2}} \tag{12}$$

where $\hat{x}_i$ is the normalized data.

The main difference between batch and layer normalization in models is the dependency on batch size and their suitability concerning which data the model is trained for.

4.4 Activations

Activations are useful in machine learning models to increase the complexity of the model to better map complex input data such as images, videos, etc. Without activation, the model acts as a Linear Regression Model with limited performance and power [48]. When using the activation function we introduce non-linearity into the model, which is recommended in most data sets. Activation functions are usually differentiable because of the implementation of backpropagation in the model to calculate the loss for training. In the following sections, we introduce five activation functions used in this thesis.

4.4.1 Sigmoid

The sigmoid activation is a bounded, differentiable, real-valued function [49] widely used in many machine learning models, such as LSTM described in 4.5.2. The equation for this activation function is given by

$$f(x) = \frac{1}{e^{-x}}. \tag{13}$$

A visualization of the sigmoid function is shown in figure 10, where you can see the non-linearity and continuity of the function. The figure also shows how the function is bounded within values of 0 and 1, where negative values are < 0.5.

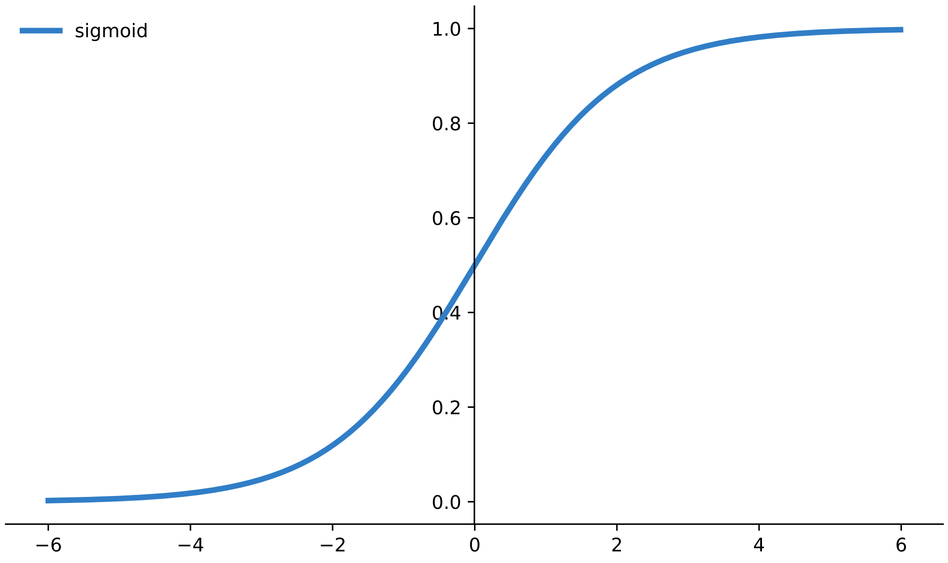

Figure 10: Sigmoid activation function.

4.4.2 Tanh

Hyperbolic tangent or tanh is a sigmoidal activation function, only different from the sigmoid activation function in that it is bounded within -1 and 1 instead of 0 and 1. This activation function is also used in LSTM models described in 4.5.2. This function is used over the sigmoid function when different signs are required in the activation. The equation for this activation function is given by

$$f(x) = 2 \cdot \text{sigmoid}(2x) - 1. \tag{14}$$

A visualization of the tanh function is shown in figure 10, where you can see the non-linearity and continuity of the function. The figure also shows how the function is bounded within values of -1 and 1, where negative values are < 0.0. One of the main benefits of tanh over the sigmoid activation function is that the mean is closer to 0 for the input of the next layer. Therefore, this activation function is often used in the hidden layers of machine learning models.

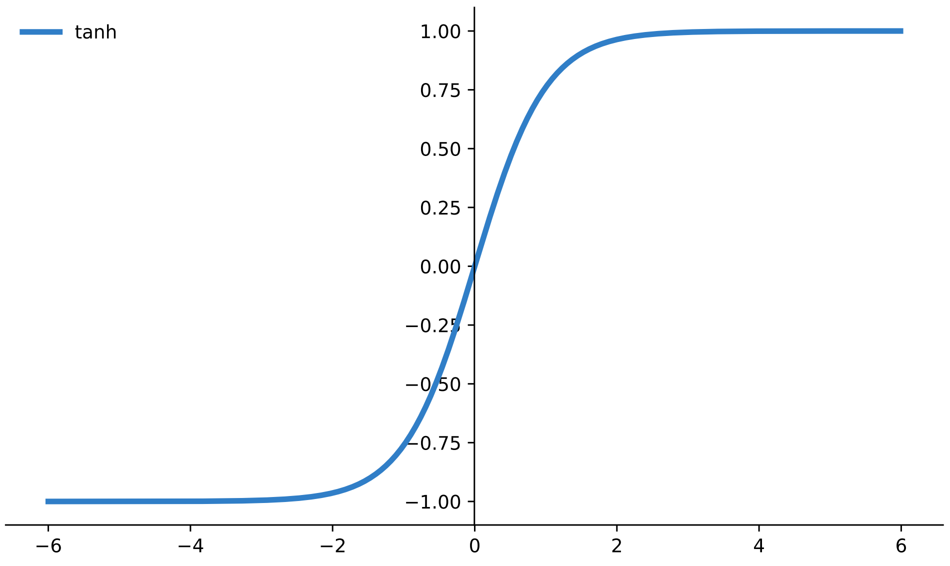

Figure 11: Tanh activation function.

4.4.3 ReLU

ReLU stands for Rectified Linear Unit and is widely used in machine learning models. This activation is used to negate negative activation values, hence reducing the number of neurons active at a given time [48]. This activation was proposed in [50] and has since been the most used activation function for deep learning models [51]. The equation for this activation function is given by

$$f(x) = max(0, x). \tag{15}$$

Another reason why ReLU is so frequently used in the hidden layers of the model is the lack of expensive mathematical operations compared to sigmoidal functions. In this activation function, the only calculations are to check if the value is larger than 0. A visualization of the activation function is shown in figure 12, where the negative values are given the value 0, and positive values are kept as it is.

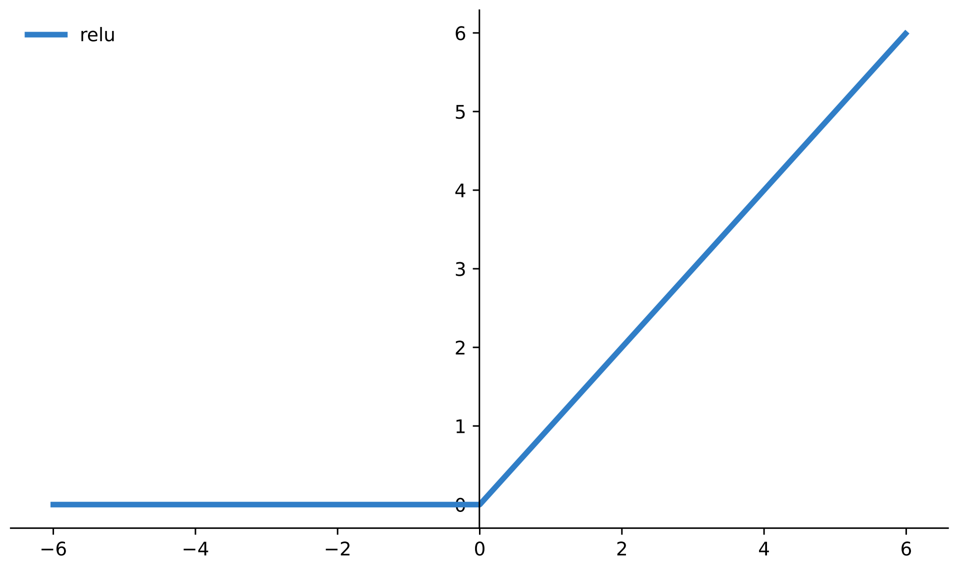

Figure 12: ReLU activation function.

4.4.4 Gelu

Gelu is an adaptation of the ReLU activation function and stands for Gaussian Error Linear Units. This activation was first proposed in [52] as an improvement to the standard, well-implemented Relu. The gelu activation function weighs the input based on the value rather than gating it by its sign. This weighing gives a performance boost and is widely used in NLP and transformer structures. The equation for this activation function is given by

$$f(x) = \frac{1}{2}x(1 + erf\left(\frac{x}{\sqrt{2}}\right) \tag{16}$$

where erf is an error function creating the bulge around 0 shown in figure 13.

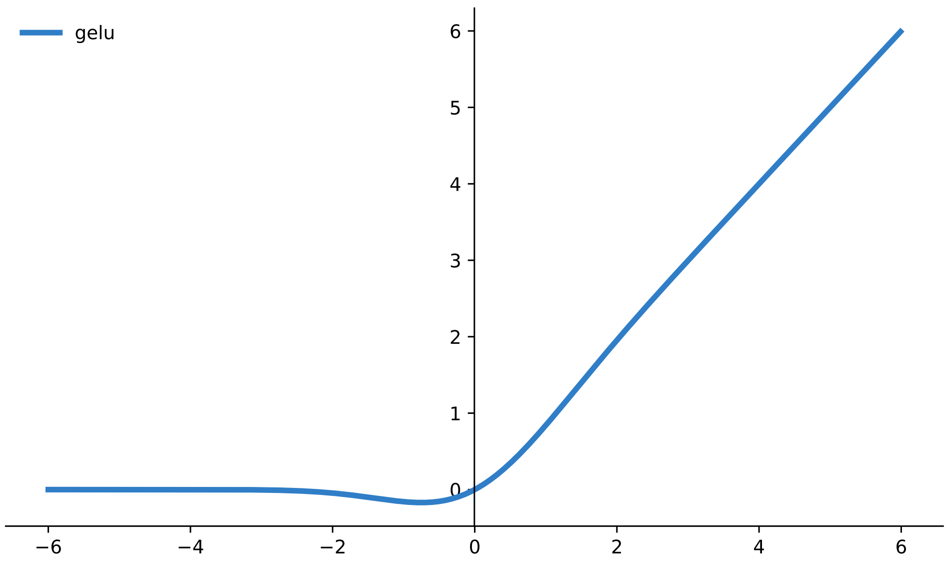

Figure 13: GELU activation function.

4.4.5 Softmax

The softmax activation function was first proposed as an activation function in machine learning in [53] as an idea from the Boltzmann distribution. This activation function is a combination of multiple sigmoid functions to create an activation function well suited for the last layer, as it represents prediction confidence between 0 and 1. In multiclass problems, the final layer of the classification is usually a softmax function with the number of classes as the channels, where the value of the channel is the confidence of the class. A great benefit of using softmax is that the sum of all the values are equal to one. The equation for this activation function is given by

$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}. \tag{17}$$

In figure 14 you can see a visualization of the softmax activation function, where the higher the number, the more confident the model is that the point is of that class.

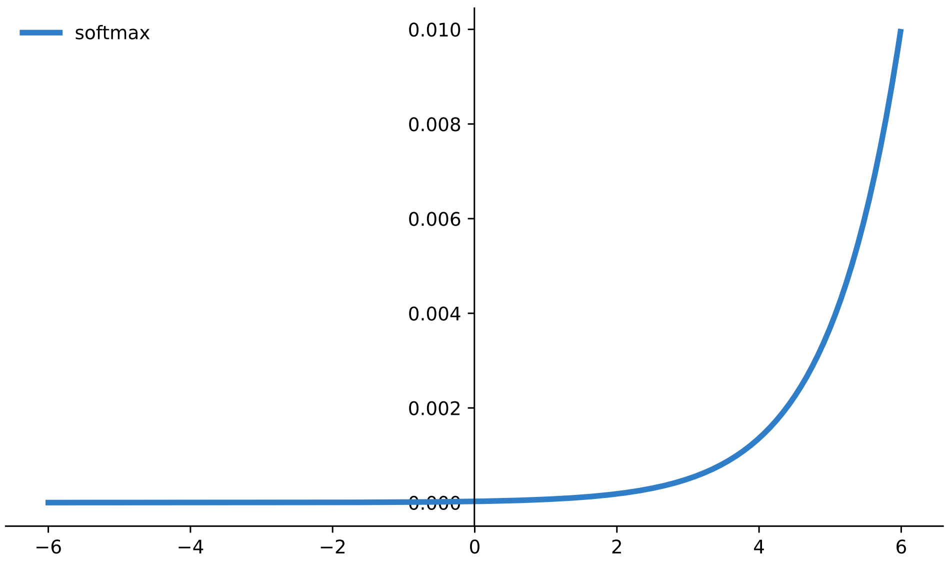

Figure 14: Softmax activation function.

4.5 Recurrent neural network

The regular Artificial neural network (ANN) uses a feed-forward mechanism that is limited to static classification. This creates a static mapping between input and output, hence it is not very useful in the task of time-dependent classification [54]. Here the Recurrent neural network shines. Recurrent neural networks (RNN) are networks that can account for temporal dependencies in the data [55]. This means that RNNs can process sequential data. There are many recurrent neural network architectures, but we will focus on Long Short-term Memory (LSTM) in this thesis.

4.5.1 Gradients

One of the main issues with regular RNNs is the problem of exploding or vanishing gradients [56]. This happens due to the feedback signal either vanishing or exploding. This limits the regular RNN only to be effective on 10 consecutive time steps [54]. The backpropagating method used in RNN is called backpropagation through time (BPTT) [57–59]. During BPTT in a regular RNN, changes in the weights are dependent on the previous timestep. This means that if the weight adjustment in timestep t were small, the adjustment in timestep t-1 would be even smaller. After a certain number of steps, the adjustments at the earliest time steps will be insignificant. This happens due to non-linearities in the computation of the hidden state of the cell, h. The standard RNN models use sigmoid or tanh, which is less than 1. A signal that goes through this type of non-linearities becomes smaller and smaller. This is called vanishing gradients and leads to a halt in the model's training because the weights are not changing at earlier stages. To illustrate this, we could consider the loss function

$$\frac{\partial L[t]}{\partial W} = \sum_{\tau} \frac{\partial L[t]}{\partial h[t]} \frac{\partial h[t]}{\partial h[\tau]} \frac{\partial h[\tau]}{\partial W} \tag{18}$$

where L is the loss function, W is the network parameters, h is the hidden state of the LSTM cell, and t and τ are the end and initial time of the data in question respectively. In equation (18) the partial derivatives of the states with respect to their previous values can be factorized as

$$\frac{\partial h[t]}{\partial h[\tau]} = \frac{\partial h[t]}{\partial h[t-1]} \dots \frac{\partial h[\tau-1]}{\partial h[\tau]} = f'_t \dots f'_\tau. \tag{19}$$

This equation, and hence the system, is stable when $|f_t| < 1$. However, in this case for equation (19) the product expansion exponentially converges to 0 when t-τ increases. This results in equation 18 being dominated by terms of short-termed dependencies, and the vanishing gradient effect occurs [60]. The layers are then less updated as the gradient is backpropagated through the network.

On the other hand, the changes in weights could be large during BPTT, corresponding to $|f_t| > 1$. This leads to exploding gradients due to the repeated multiplication of large values to the gradients. This could cause values to grow exponentially larger and cause numerical errors. To prevent these problems, multiple methods have been proposed, among these regularization and soft constraint [60–62] for exploding gradients and Gated Neural Networks and gradient norm clipping [60, 62, 63] for vanishing gradients.

To prevent vanishing gradients, the Gated Neural Networks (GNN) were proposed [63]. Some examples of these networks are Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU), where LSTM will be used in this thesis and is further explained in 4.5.2. The key property of the LSTM is that they have an internal memory cell, c_t that does not go through a non-linearity when transiting from one time step to the next (see figure 15, where no non-linearities on the path connecting c_{t-1} with c_t). This, allows the memory content to neither vanish nor explode during the BPPT. The non-linearities are only in the gates that are used to update the cell with the input and to produce the output. In particular, the gates are used to forget or keep information from the previous timesteps so that the model forgets unimportant information but keeps essential information that is helpful for the problem.

4.5.2 Long Short-Term Memory

The long short-term memory architecture was first proposed in [64] as a method to have dependencies over longer time intervals. The LSTM architecture can remember up to 1000 time steps depending on the complexity of the data [64]. One of the main reasons to use LSTM over the standard RNN is to avoid having to deal with the non-linearity problems and hence the vanishing gradient problems discussed in section 4.5.1. In figure 16 you can see that each node in the LSTM makes a loop for each timestep in the array.

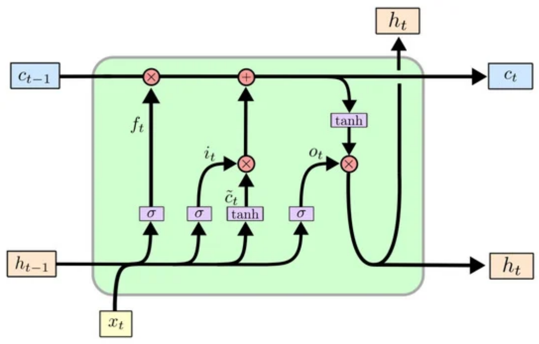

Figure 15: A single LSTM cell [65]. Here the gates described in section 4.5.2 are presented. On the left side of the cell, the forget gate is presented. In the middle is the input gate. The Right gate is called the output gate.

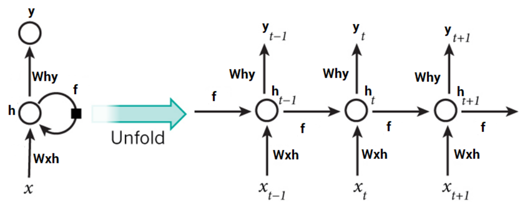

Figure 16: The repetitive nature of a recurrent neural network [66].

One cell in an LSTM architecture has 3 gates, forget-, input-, and output-gate, which is shown in figure 15. This cell is repeated, as shown in figure 16. Each gate in the LSTM has a different mathematical calculation, and they have different tasks. The furthest gate on the left is the forget gate. This gate produces a number between 0 and 1 and calculates the rate the model forgets information. This gate is used to reset the cell's content given new information. When the new information renders the old cell content less important, the forget gate is activated [54]. The equation for the forget gate is given by

$$f_t = \sigma(x_t U_f + h_{t-1} W_f), \tag{20}$$

where σ is the sigmoid function, shown in section 4.4.1. In this equation, x_t is the input data, U_f is the weight associated with the input data, h_{t-1} is the hidden state from the previous timestep and W_f is the weight matrix for the hidden state. If $f_t = 0$ the gate forgets all the information from the previous step, and if the value is 1 it keeps all information. In the early stages of training f_t is close to 1 to prevent the model from forgetting before it has learned.

The gate in the middle of figure 15 is called the input gate. This is divided into two activation functions given by the equations

$$i_t = \sigma(x_t U_i + h_{t-1} W_i), \tag{21}$$
$$\tilde{c}_t = \tanh(x_t U_c + h_{t-1} W_c), \tag{22}$$

where tanh is the hyperbolic tangent function shown in section 4.4.2. U_i and U_c are weight matrices for the respected input and W_i and W_c are Weight matrix for their respected hidden state. i_t and $\tilde{c}_t$ are the outputs of the input gate, which are multiplied together and added to the product of the forget gate and the current state from the previous cell. This gives the equation

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t \tag{23}$$

where c_t is the current state of the cell. This current state is fed into the next cell.

The last gate we have in the LSTM cell is called the output gate, and this gate is shown as the right side of figure 15. This gate produces the hidden state of the cell. This hidden state is used as input into the next cell, and can also be used as the output of the cell depending on the problem at hand. The first operation in this part is an activation given by the equation

$$o_t = \sigma(x_t U_o + h_{t-1} W_o). \tag{24}$$

The calculations are the same as the forget gate, but with new weights U_o and W_o. The output of this activation, o_t, is multiplied by the current state and activated by a tanh function. This produces the hidden state, which is given by the equation

$$h_t = \tanh(c_t) \cdot o_t. \tag{25}$$

The hidden state is used as the output of the model, either the hidden state of the last cell, or each hidden state. The first option is used in classification problems where you need to generate an output (e.g., the class label) for the whole time series, while the last is used in problems like segmentation, where you need to generate an output at each time step.

4.6 Convolutional neural network

A convolutional neural network (CNN) is built by convolutional layers. An idea of pattern recognition in models sprung out in [67], where the recognition of patterns in images, unaffected by their position, is proposed. This is often considered the first step toward the convolutional networks we use today. The real breakthrough of the convolutional layers was when a CNN won the ImageNet Classification Challenge in 2012 with a large margin[68]. This network is popularly called AlexNet and is described in [69]. CNN structures also won later competitions of the ImageNet Classification Challenge, this began the era of CNN as the standard model for image classification. The later models had more layers and became more and more accurate, and the winner of the three next competitions was won by ZFNet(2013), VGGNet(2014)[70], and GoogLeNet(2015)[71].

A convolutional layer works in the way that kernels are applied to patches of the image and parsed through the image. These kernels have different representations and alter the image in a way that benefits the classification task at hand. When parsing through the image, the model learns d amount of kernels, which can be activated depending on the different structures in the images. All these kernels help the model find where and whether there are structures that are important for the downstream task at hand. How the kernels are applied to each patch is shown in figure 17, where you multiply each kernel to the patch to produce a single pixel. The result of the kernels going through the image is a mapping that is a feature representation of the image.

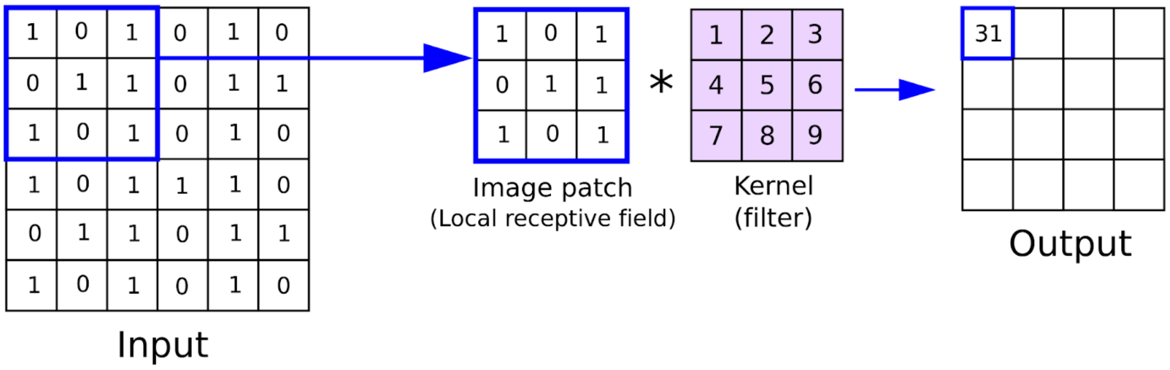

Figure 17: How the kernel is applied to patches of the image to create new representations[72].

For this thesis, we are interested in using convolutional layers for segmentation. This was first proposed in the thesis [73], but the most popular model for this is the U-Net [29]. This architecture creates a model with a U-like architecture, shown in figure 18. This architecture uses convolutional layers and max-pooling to reduce the receptive field and capture patterns in images as features in different levels, then convolutional layers and upsampling to rescale the features into the original size of the image. In this thesis, we are going to adopt the idea of this model, but use Transformers or LSTM-based models for the temporal part.

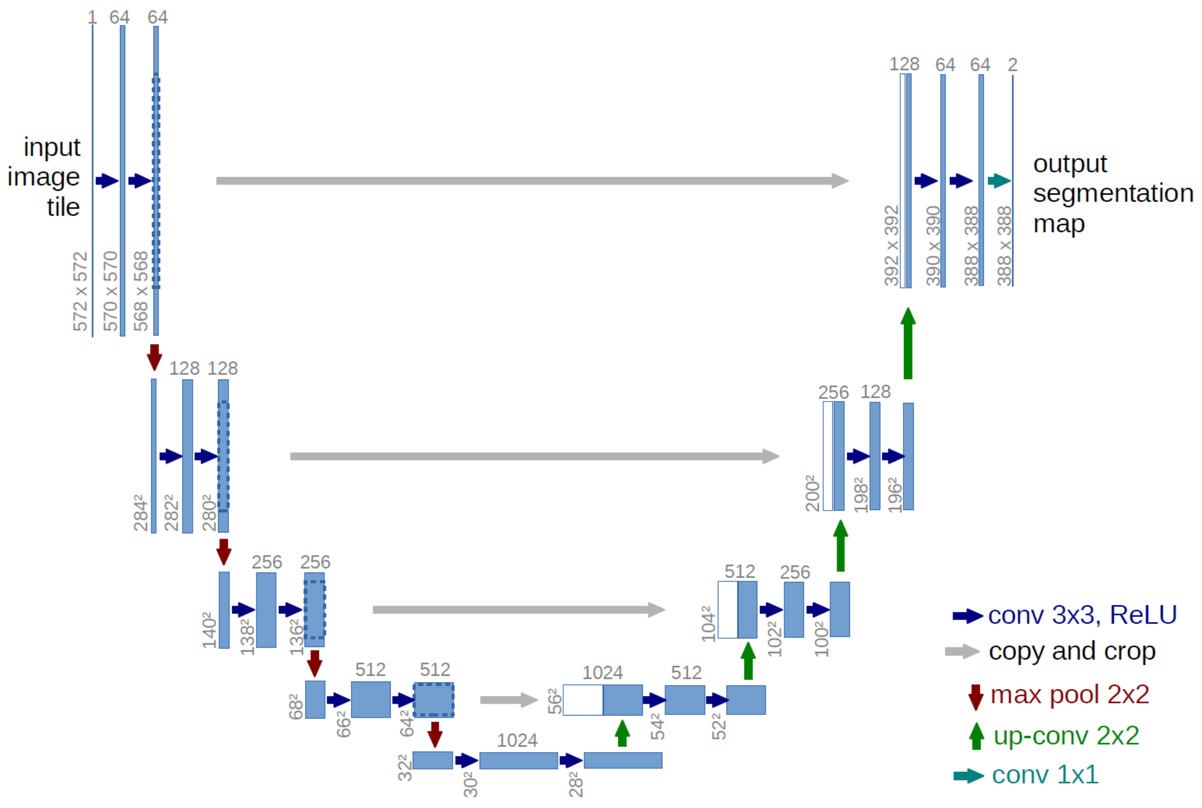

Figure 18: U-net architechture. A segmentation method that reduces the spatial dimensions of the image to get different representation in different scales. The differerent representation is concatenated together to get better segmentation results [29].

4.6.1 Maxpooling

The main objective of maxpooling in CNN is to reduce the spatial dimensions by downsizing the image [74]. Max pooling has no learnable parameters, therefore a significantly lower computational cost compared to a convolutional layer.

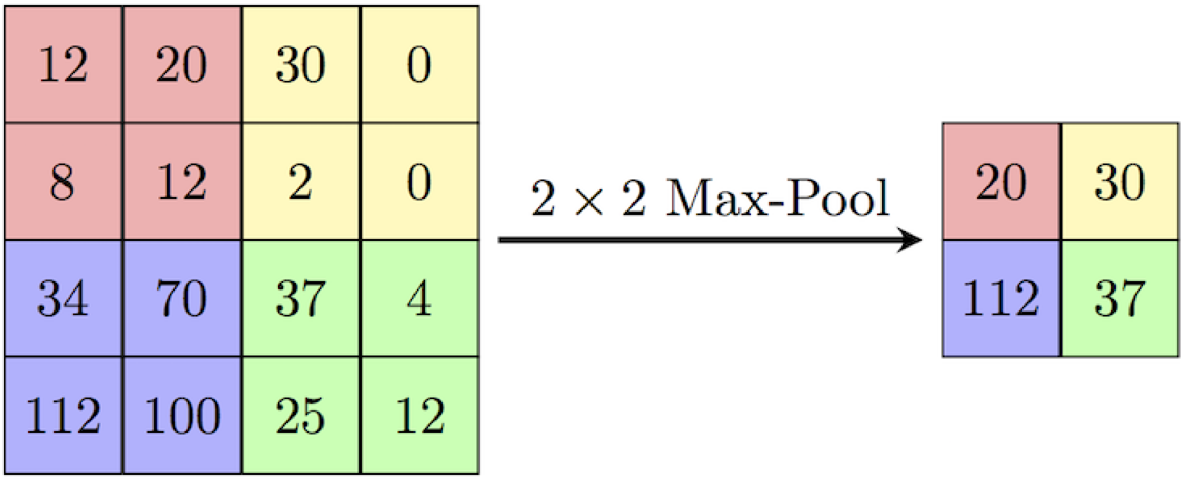

Figure 19: Max pooling example on an 4x4 image [75].

Figure 19 shows an example of a 2x2 maxpooling operation on an 4x4 image. It shows that each of the 2x2 squares with different colors in the 4x4 image becomes one pixel, and it takes the max value of the square. Max pooling can be useful in CNN since it selects the pixels with the highest value in a given region, hence extracting the sharpest features in the image and can help the model differentiate between objects.

4.7 Transformers

Transformers were first proposed in [76] and have been used as the primary problem-solver in Natural Language Processing (NLP). However, the early stages of Transformers only consisted of an encoder-decoder architecture. The advantage of this architecture over RNN is that it looks at the example as a whole instead of sequence-by-sequence. This allows the model to avoid recurrence and have a global dependency between the input and output, and this is the main reason the model does not suffer from long dependencies. Transformers also avoid the BPTT in training, removing the problems with the vanishing gradients. A multilayer perceptron (MLP), or a feed-forward network, also has the advantage of not using BPTT. The difference between an MLP and a transformer is the use of attention and positional embedding in the transformer to control the positions in the sequence. The original transformers were used for NLP, but in recent studies transformers have been applied to many different problems. In this thesis, a transformer will be used to segment blob-like structures in videos. In the following sections, the main aspects of the Transformer architecture and how it can handle long sequences will be discussed.

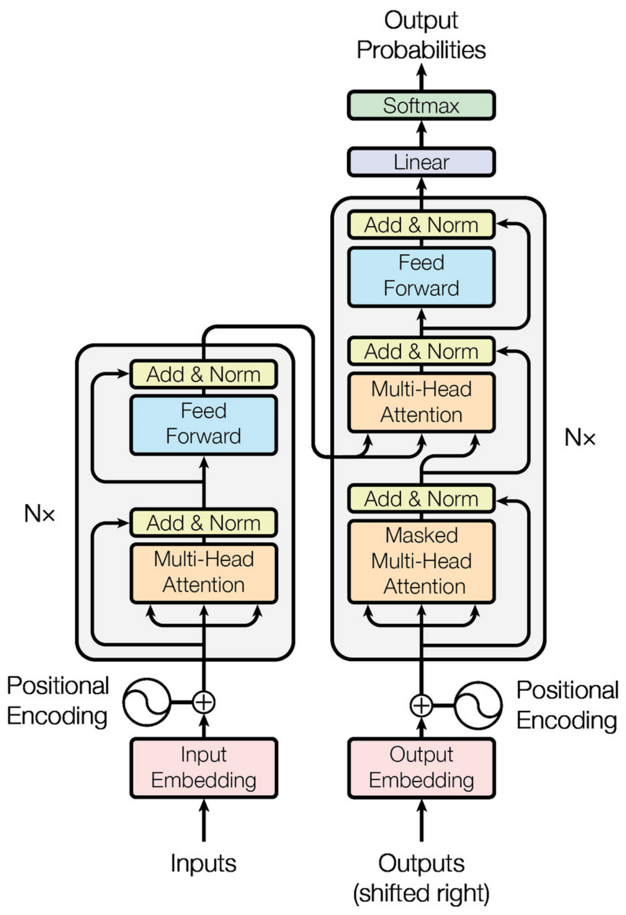

Figure 20: The Transformer model architecture. The figure is taken from [76].

The first to prove that the transformer architecture can work as a classification method is shown in “An image is worth 16x16 words: Transformers for image recognition at scale”[77], where they use a patch of 16x16 pixels as a ”word” used in the NLP problems, and this shows state-of-the-art performance results. The way they did this was to implement the idea of embedding patches of the image, compared to embedding each word in the NLP problems. Each patch in the image gets a new representation, with more information describing the patch. Later this model was tested, and there have been created more robust and efficient models.

4.7.1 Encoder

In an Encoder-Decoder architecture, the encoder creates a representation of the input, where $\mathbf{x} = (x_1, ..., x_n) \rightarrow \mathbf{z} = (z_1, ..., z_n)$. Here $\mathbf{x}$ is the input, and $\mathbf{z}$ is the representation generated by the encoder.

The left part of figure 20 shows the encoder part of the architecture. The main components of this part are a self-attention mechanism and a feed-forward neural network. The feed-forward neural network is a simple ANN, and the information in this part only moves in one direction, from the input nodes to the hidden nodes and finally to the output nodes. The function for this operation is given by

$$\mathrm{FFN(x)} = \max(0, xW_1 + b_1)W_2 + b_2. \tag{26}$$

The self-attention mechanism, or Multi-Head Attention, will be explained in section 4.7.3.

4.7.2 Decoder

The Decoder’s job is to decode the representation created by the encoder into the desired output. Given the $\mathbf{z}$ generated by the encoder, the decoder creates an output sequence $\mathbf{y} = (y_1, ..., y_n)$ for one element at a time.

The right part of figure 20 shows the decoder part of the architecture. The structure of the Decoder is much like that of the encoder, except it contains an additional attention mechanism, the Masked Multi-Head Attention. The feed-forward neural network part is explained in 4.7.2, and the attention mechanism will be described in 4.7.3.

4.7.3 Attention

There are several different types of attention, but I will focus on Multi-Head Attention in this thesis. The main idea behind attention is that you have input data that is more important than others. This results in mapping the input through some key and query vectors, where the keys are paired with a value. The output is calculated as a weighted

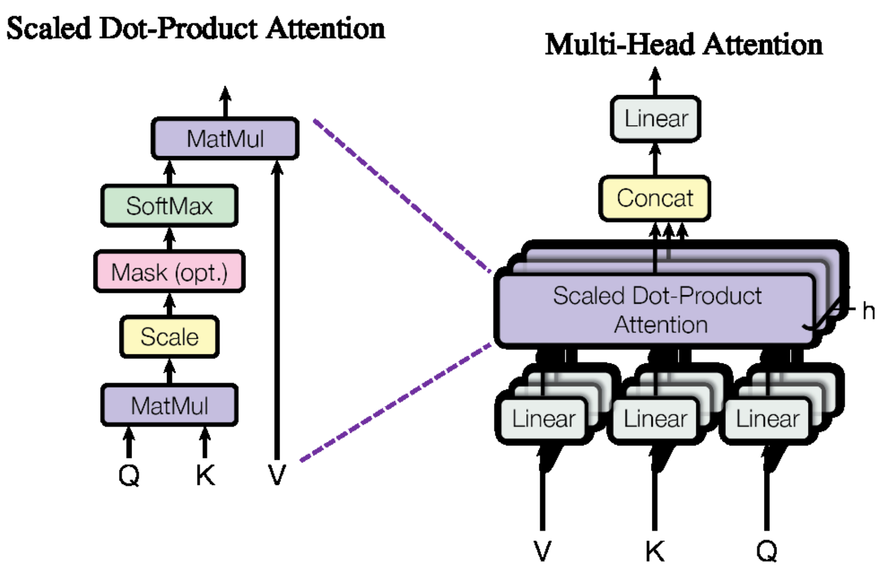

Figure 21: Multi-head attention. The figure is taken from [76].

sum of the values, and the weights are calculated by a function of the query and the corresponding key, called the Scaled Dot-Product attention [76]. In a transformer, the query is the given patch which is being investigated, while the keys are the whole set of patches. This gives a score between each query and the keys, which becomes the values.

In figure 21 the mechanism of multi-head attention is shown. Here you can see the values as V, the keys as K, and the query as Q. These are all put in the Scaled Dot-Product attention which takes the query and the corresponding key and multiplies them, then takes a softmax to create the weight for the value. In this Scaled Dot-Product attention function we get the output

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{27}$$

where d_k is the dimension of the keys. It is suspected that higher values of d_k increase the magnitude of the dot, pushing the softmax function into the region where the gradient is very small [76]. A division of $\sqrt{d_k}$ reduces this magnitude.

The multi-head attention mechanism combines multiple attention mechanisms and then concatenates them. This allows the model to learn different dependencies between inputs. The function of this operation is given as

$$\begin{gathered}\text{Multihead(Q, K, V)} = \text{Concat}(\text{head}_1, ..., \text{head}_h)\text{W}^O \\ \text{where head}_i = \text{Attention}(\text{QW}_i^Q, \text{KW}_i^K, \text{VW}_i^V).\end{gathered} \tag{28}$$

In the equation above, W^O is a parameter matrix, and W^Q, W^K, and W^V are the parameter matrix of the multi-head, query, key, and value, respectively.

4.7.4 Backbone

Transformer models use a backbone to extract features from an image to be further processed in the model. This backbone often reduces the spatial and temporal dimensions of the model, to easier process larger data sets faster. For instance segmentation, there are many different backbones implemented, among them Region-of-Interest operations, Region Proposal Networks, or Feature Pyramid Networks (FPN). In this thesis, FPN will be used as the backbone. FPN is used to get representations of the data in different levels, where the model can better understand structures of different sizes.

4.7.5 Positional encoding

In [76], a method is proposed to keep track of the absolute or relative position of a sequence, called positional encoding. This method was initially used in NLP problems. The positional encoding as a sinusoidal function is described as

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}}) \tag{29}$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}}) \tag{30}$$

where pos is the position, and i is the dimension. With this encoding, the model can easily attend to nearby positions.

Since this thesis uses videos and not time series, it makes more sense to use positional embedding over positional encoding. Positional embedding resembles positional encoding, but it is learnable by the model. Positional encoding codes the position of the word in the sequence for NLP problems, while in images you encode the position of the patches. This means that the model requires more memory, which helps the model learn the representation better.

Part III / Proposed Method

5 Methodology

5.1 Hardware

The first models were trained on one Nvidia GeForce RTX 3060 GPU. This GPU has VRAM of 12GB and constrained the dept of the models. The final model was trained on Nvidia RTX A6000 GPU with 48 GB of memory for larger depth and higher resolution.

5.2 Synthetic data generation

In this thesis, a synthetic data generator has been used for the training data. This model was created by the Complex Systems Modelling (CoSMo) group at UiT, and can be found at https://github.com/uit-cosmo/2d-propagating-blobs [78]. This is a stochastic two-dimensional model of advecting and dissipating blobs. A lot of blob parameters are interchangeable in the generation of these data, such as x- and y-velocity, size, and amplitude. In this thesis, this data will generate training-, test-, and validation data, with different parameters to test how well the models are performing on synthetic data. A frame of the synthetic data is presented in figure 22, where the data labels are presented in 23. The labels are ordered in the way that the first blob that enters the frame gets the lowest number label.

5.3 Models

All models in this thesis are built on the Keras API, and they are all tested on experimental and synthetic data, and trained on synthetic data. All of the models implement semantic segmentation as the method of identifying and tracking blobs.

5.3.1 LSTM

Models created using the LSTM approach were fairly simple and are mostly used to compare the results of a simple LSTM model with the more complex transformer method. A semantic segmentation approach was used in these models since they provide a prediction of each pixel, and postprocessing can be used to find the center of mass velocities, amplitudes, and size of the predicted blobs. The max amount of blobs within 100 frames the model could find was set to 10. Fewer results and testing has been done on these models and is not the main focus of the thesis.

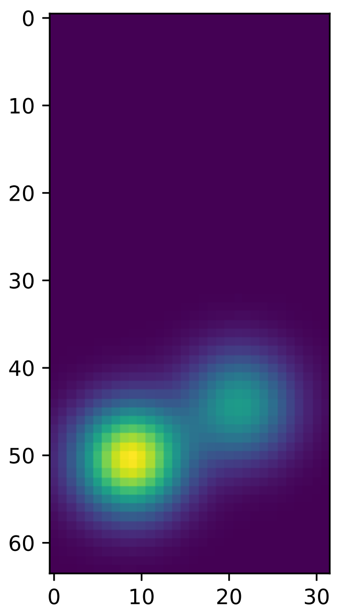

Figure 22: One frame of the synthetic data used to train the models. In this example, there are two blobs.

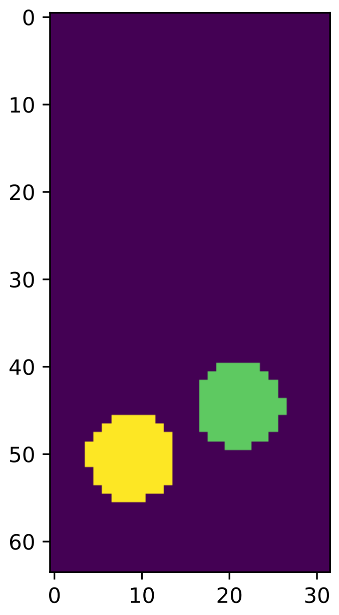

Figure 23: The same frame, showing how the labels are presented for the blobs.

The model consists of two Convolutional LSTM layers and 2 convolutional layers, with max-pooling between the first LSTM and second LSTM layer and upsampling between the last LSTM and first convolutional layer. This model is fairly simple, but it shows the memory restraint of LSTM and recurrent models regarding large data sets. This thesis has not invested too much time in optimizing the hyperparameters for this data set as it did not show as promising results as the transformer model. The model's outline is shown in figure 24, and the size difference in the layers is shown in figure 25. This model has 20,282 trainable parameters but can only handle a batch size of 8. This shows one of the flaws with recurrent networks when you have a large amount of data that needs to be processed.

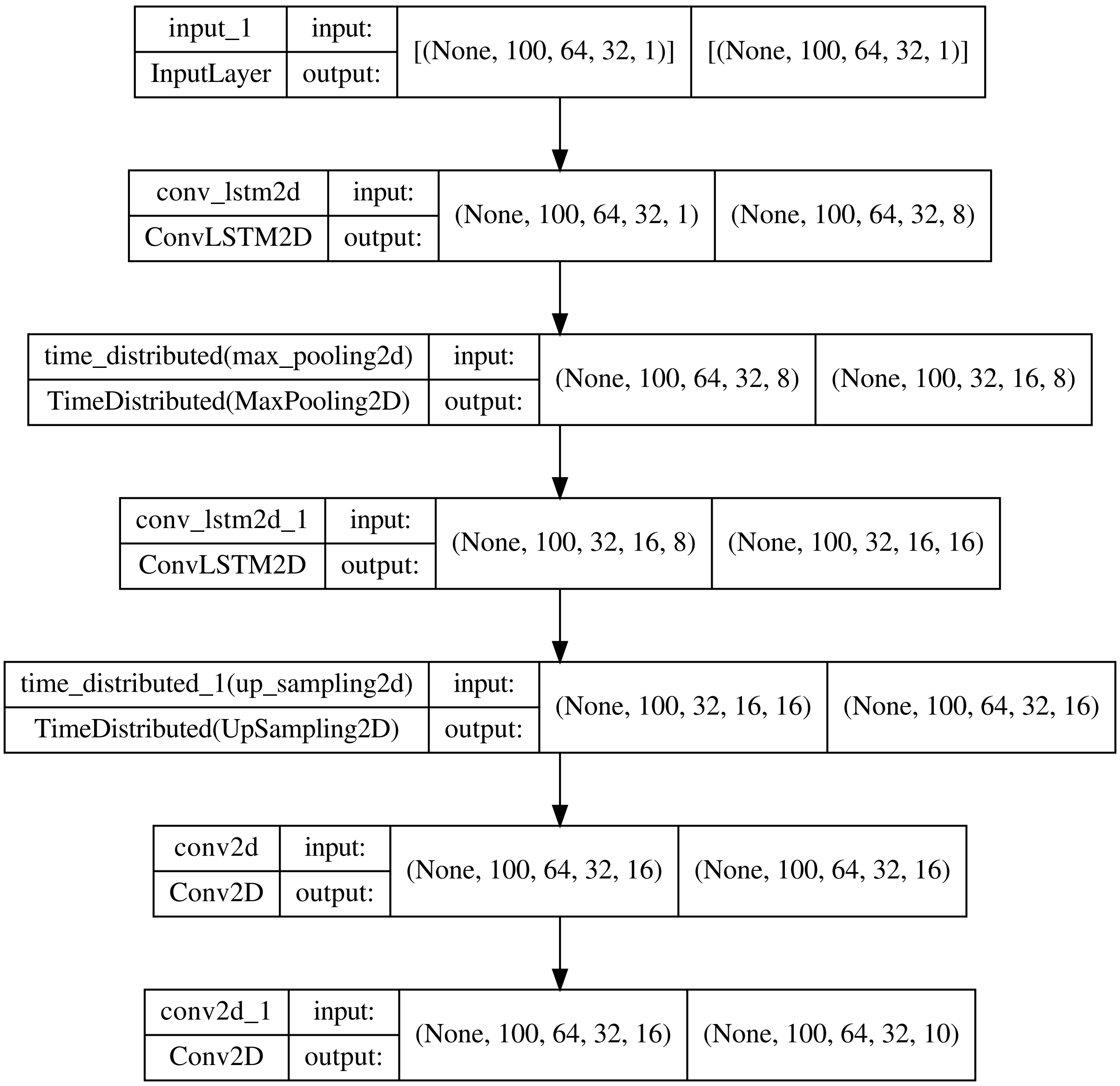

Figure 24: Model with described layers. In this figure, you can see the input and output values of each layer and which layers that are connected to each other.

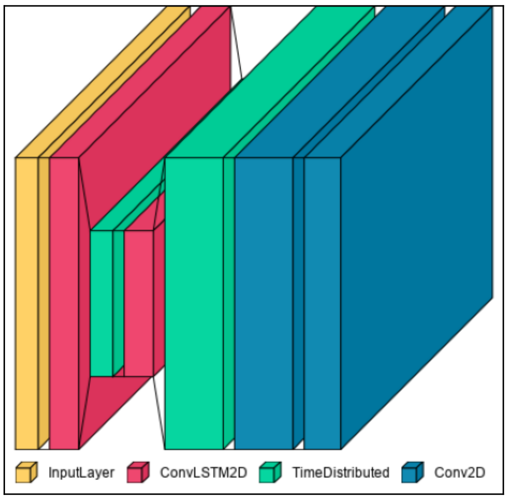

Figure 25: Visualization of the sizes of the image in each layer. The yellow box is the size of the input data, and the red boxes is the LSTM layers, where one is on the whole data set, and one is on half-size images. The green boxes, described as "TimeDistributed" is the max-pooling and upsampling layers. The blue layers are the convolutional 2D layers for classification.

5.3.2 Transformers

The approach for using transformers was to train models on lower-resolution data sets to test if the architecture was able to handle the segmentation tasks. Later a full-scale model was trained, with a few changes to fit the larger resolution videos in the limited memory of the GPU.

However, when looking at the results, there was a problem with not knowing how many blobs were to appear in a given timestep. This is a problem as you have to set the maximum number of classes to classify, and increasing this number too much will decrease the accuracy of the model. When looking at the experimental data and predictions of other models, a limit of 20 per 64 time-frames was set for the final model. Other models found 115 blobs in the time span of 2000 time-frames, so 20 blobs per 64 frames are far within the expectations.

5.3.2.1 Small scale model

For the transformer, a few models were tried out on a smaller data set to learn which layers were important and how to find the best configuration that works on the synthetic data. The training on large, memory-hungry models takes a lot of time, and given the time limitation, faster testing of features was necessary. These models were trained on the same data set, generated from `2D_propagating_blobs`. To test out different model types and how the transformer performs on these types of data, a test set with 16 timesteps and 16x16 images was created to try out different models. This data set of 16 timesteps only contained a maximum of 4 blobs, which is a smaller ratio than 20 per 64, but this was a test model.

The final model tested on this data set had a structure like the one shown in figure 26. In the image, you can see that the model creates three sets of feature maps in three different resolutions. The resolutions used for this are (16,16,16,32), (8,8,8,32), and (4,4,4,32). These feature maps are created in a simple convolutional layer that strides over each map with 2x2x2 kernels, to create a feature map with half the resolution of the previous layer. The first layer is just run through a 1x1x1 kernel to create a feature map from the input data.

These feature maps are encoded, to keep track of the position when put through the transformer layer. The transformer layer is a basic one like the one described in 4.7, where it is recursed over 1,4 and 8 times, with 2,4 and 8 heads.

5.3.2.2 Large scale model

The final model for the transformer architecture was used on the larger data set with 64 time steps and 64x32 image size. This model consists of three major parts, the backbone, the transformer, and the head. The parameters of each of these parts are shown in table 1, and the whole model consists of 1,585,093 parameters, of which 1,584,869 are trainable.

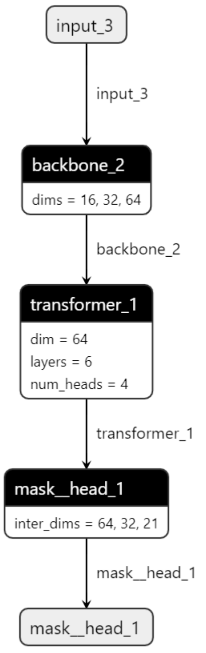

Figure 27: The three main parts of the Transformer architecture.

Layer	Number of parameters
Backbone	69696
Transformer	962560
Head	552837

Table 1: Number of parameters in the major parts of the transformer. This shows that the transformer part of the architecture has the most parameters.

The backbone is a simple feature extractor that extracts features on different levels. These levels are 1/1, 1/2, and 1/4 ratio of the original image. The method uses convolutional 3D layers to reduce spatial and temporal dimensions to decrease the computational cost of the encoder and decoder. The backbone creates patches of (4,4,4) as input into the transformer, where each patch has 64 numbers representing it. This is called image

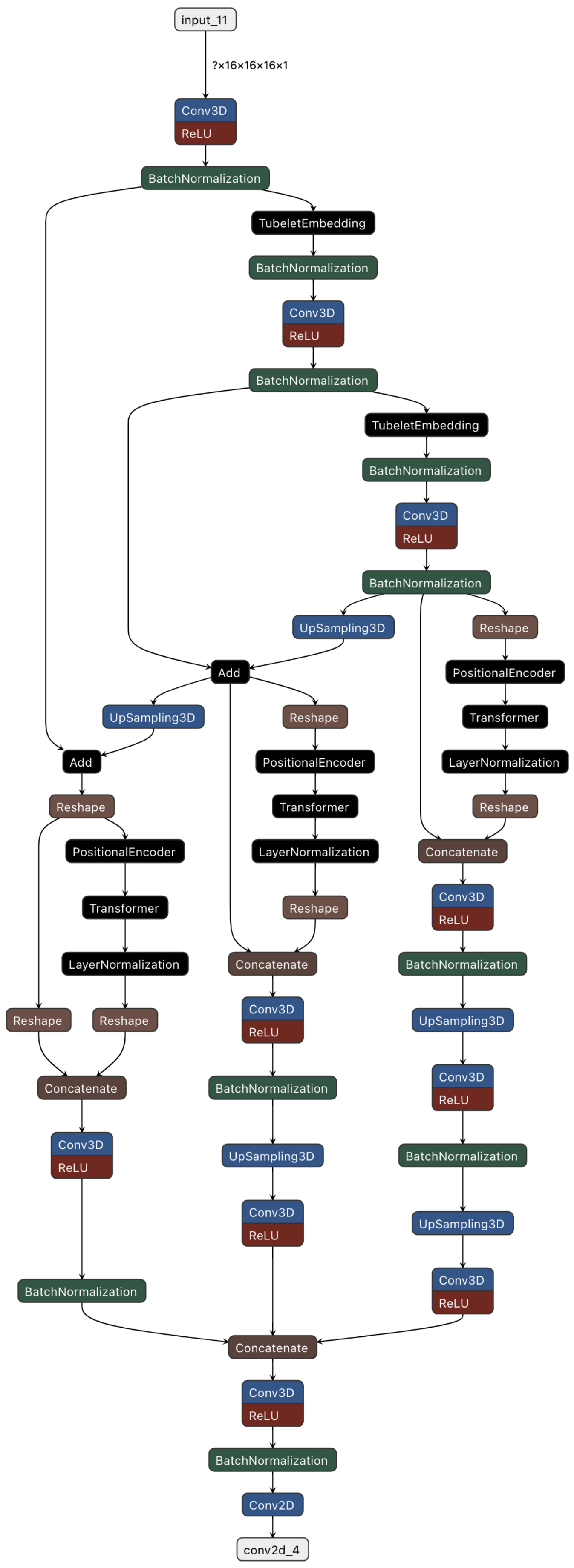

Figure 26: All the layers in the transformer architecture. It is shown how all the layers are divides and put through the transformer, then concatenated back together to get the best possible segmentation results.

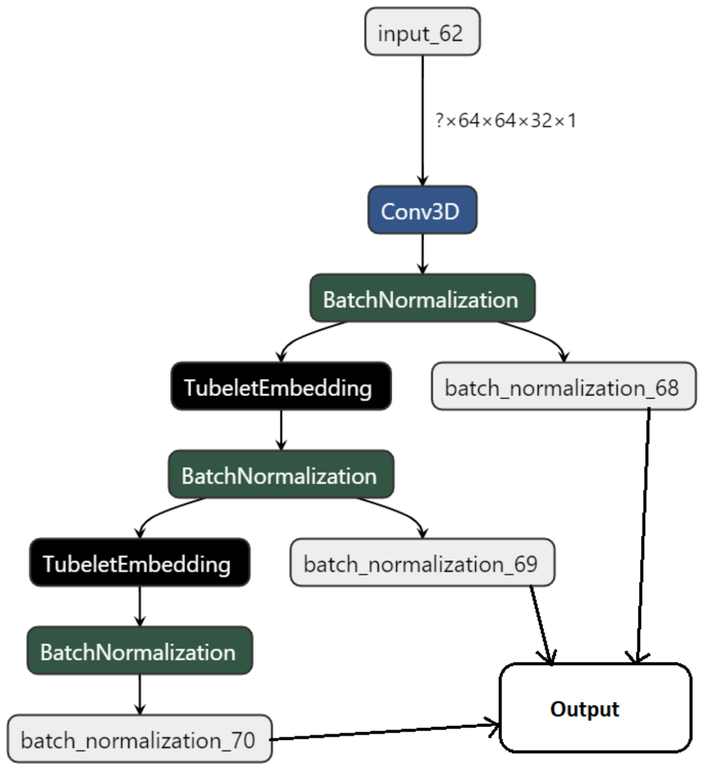

Figure 28: Backbone structure of the transformer architecture. Using an FPN model to get mappings of the data in different resolutions.

embedding and is discussed in section 4.7.5. This means that the output of the backbone is (BatchSize, 16,16,8,64). It is also given a positional encoding, to keep track of the position of the patch within the image. The backbone also creates feature representation for (1,1,1) and (2,2,2) patch sizes used later in the head part of the architecture. The backbone architecture is shown in figure 28.

The transformer used in this model is a basic transformer described in section 4.7 and shown in figure 20. The output of the backbone is flattened to (BatchSize, 2048,64) and is shown as the input in the figure. There are six of these blocks for both the encoder and the decoder. At the first block, a zero array of the same size as the encoder's input is used in the decoder architecture. For the rest of the blocks, the decoder uses the output of the previous block as this input. The steps if there were only one block in the transformer are shown in figure 29.

The task of the head is to recreate a mapping of the encoded and decoded output of the transformer of the same size as the input. This part is done in three steps using convolutional 3D layers. The first step is to reshape the outputs of the transformer back to a size of (BatchSize, 16,16,8,64), then concatenate the final encoder and decoder output. This concatenated array of size (BatchSize, 16,16,8,128) is run through a convolutional 3D layer that finds 64 representation maps for the down-scaled image. These representation maps are then concatenated with the representation map from the backbone with the same size and run through a convolutional 3D layer to find 64 representation maps. Then, the model upsamples the video to a size of (BatchSize, 32,32,16,64). This concatenation

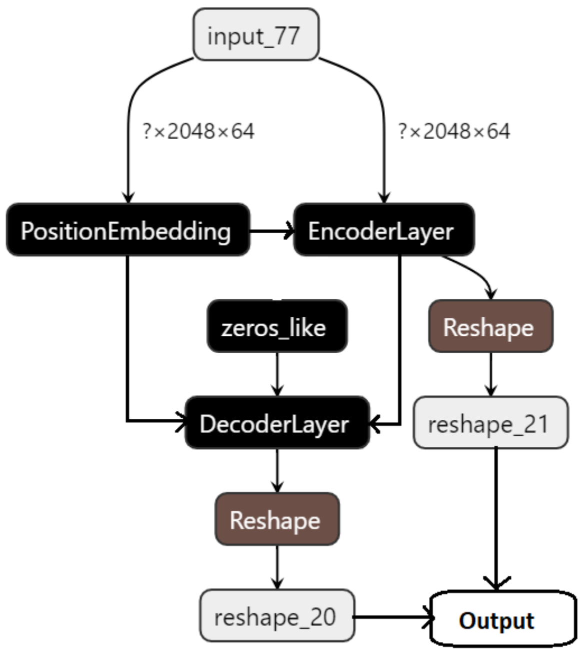

Figure 29: The transformer part of the architecture. Here we have the lowest resolution level from the backbone put into the encoder layer with positional embedding. Then the output of the encoder and an array of zeros is used as input into the decoder layer. Both the encoder and the decoder are output and used later in the model.

and upsampling are repeated until the video is of the original size, with 32 representation maps. The last step in this layer produces a pixel-wise classification with 21 classes, where the first class is the background, and the rest are the different blobs. The architecture is shown in figure 30, where you can see that the five different inputs are concatenated to generate the best possible reshaping of the blobs. Input_24 and input_25 are the output of the encoder and decoder in the transformer part of the model. Input_26 is the lowest resolution output of the backbone, the same input as the encoder uses. Input_27 and input_28 are the other outputs of the backbone. These are concatenated like it is done in U-net [29] to help the model in shaping the output.

5.3.3 Recurrent All-Pairs Field Transforms (RAFT)

This model was presented in [1] and trained for blob tracking in the paper [2]. The outline of the RAFT model is shown in figure 31, where it is shown that the model consists of three main parts: (1) a feature decoder that extracts a feature vector for each pixel, (2) a correlation layer that produces a lower resolution 4D pair-wise pixel correlation of the dimensions HxWxHxW, (3) and a recurrent layer based on a GRU used as an update based on the correlation. GRU is a gated neural network with two gates, comparable to the LSTM which has three gates.

RAFT is run with the default settings to determine how many blobs were detected in this setting. When comparing this model to the other models, a setting of $viou = 0.5$, $amp = 0.75$, and $blob\_life = 15$ is used. VIoU is a cost metric used to describe the pair-wise cost between the current and previous frame. Amp is used to discard detected blobs

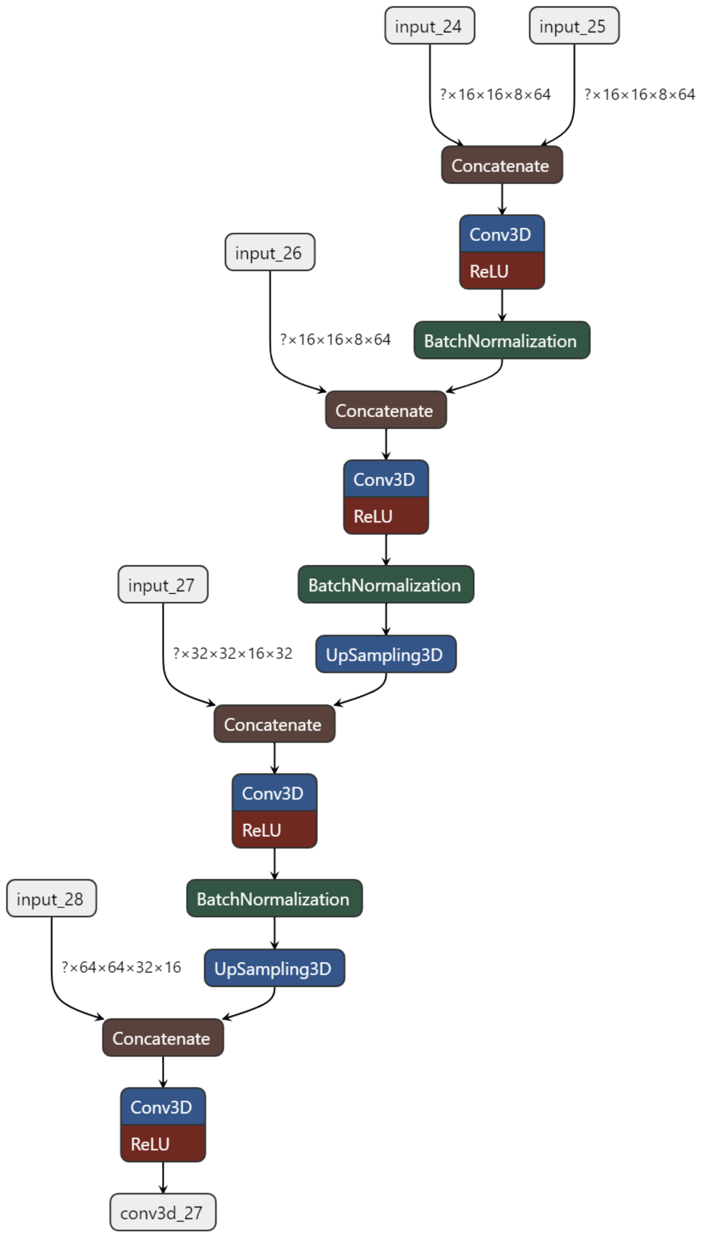

Figure 30: The head of the transformer architecture. Here we concatenate the outputs of the encoder and decoder, then concatenate all the levels of the backbone for better segmentation.

with an amplitude lower than the set value. Blob_life is used to discard short-lived blobs. This method is far more complicated and has more features than the models presented in this thesis.

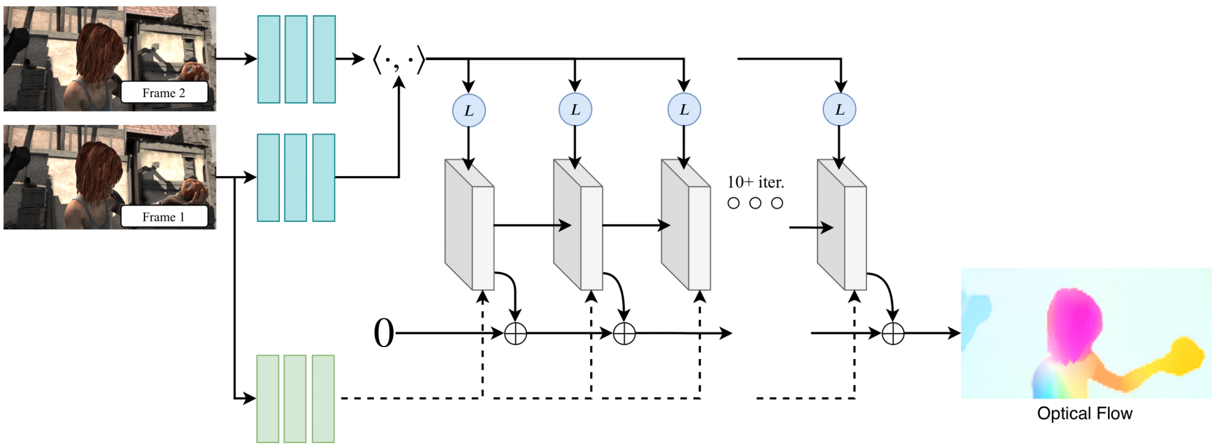

Figure 31: Outline of the RAFT model.

5.4 Preprocessing data

When working with experimental data, preprocessing the data is recommended to predict results better. The main reasons to do this are to make the distributions of different ranges comparable and have the data in a format suitable for neural networks, i.e. have the values in a small, fixed range. Firstly, let's discuss the normalization of the training data. As discussed in 4.3, we don't want the model to be influenced by the magnitude of the values and focus more on the shape of the underlying distributions in the data. Therefore, we normalize the training data with the expression

$$D(t) = \frac{T(t)}{max(T(t))}, \tag{31}$$

where D(t) is the normalized data, and T(t) is the training data. We perform this normalization to keep the amplitudes of the data between 0 and 1. Since this data set does not have any negative values, there is no need to perform a more advanced normalization.

As for the experimental data, there are some more steps. With the synthetic data generation being limited to gaussian shaped blobs, there are limited options for how to make the training data resemble the experimental data. Therefore, the next steps are made to make the data suitable for the model without removing the blobs. This is done to make the different data comparable. The most optimal method had been to have training data that is generated with all the aspects in the experimental data. This was tested, but the program was slow, and after 3 days of running the program crashed. Hence, a compromise was made, where the training data was made as comparable as possible to the experimental data, and some methods were made to the experimental data to generate some results of the models.

Firstly there is a section in the experimental data set that always has high amplitude values, which we do not have in our training set. In figure 32 you can see an example of the original experimental data cropped to only include the rightmost part of the data, where the LCFS is. In this and all similar figures, the axes represent pixels. There are 64 pixels in the y-axis and 32 pixels on the x-axis. The figure shows a large density in a line close to the middle of the image, and here there are persistently high amplitude values across timesteps. In this data, the highest value is 358 and the lowest is 64. In figure 33 the experimental data where the pixel-wise mean over time is subtracted from each pixel is shown, where the maximum value is 115.2 and the minimum value is -51.2. Here you can see more clearly where the blobs are forming. However, since this is experimental data, there is a lot of noise, which is not present in the training data.

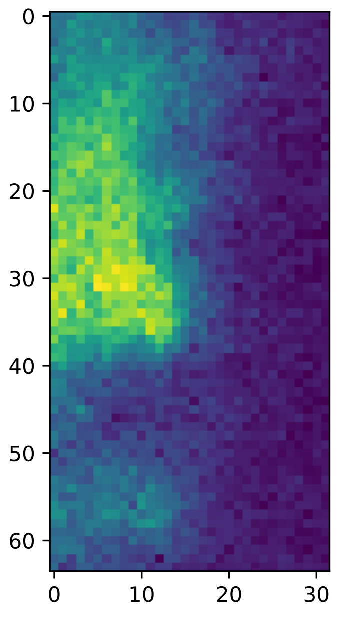

Figure 32: Original data.

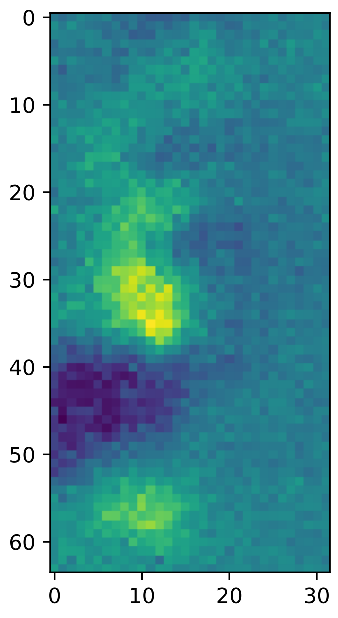

Figure 33: Data after subtracting mean values over time.

We assumed in the training data that amplitude, velocity, and size were gaussian distributed. Therefore a smoothening method provided by scikit learn [79] called gaussian filter has been applied to the data. Scikit learn is a machine learning module that provides tools for data analysis. This gaussian filter reduces extreme values in the data and improves blob tracking. Figure 34 shows the effect of this method on figure 33. In this figure, the maximum amplitude is 76.5 and the minimum is -23.3, meaning that the difference between the highest and the lowest point has been shrunken.

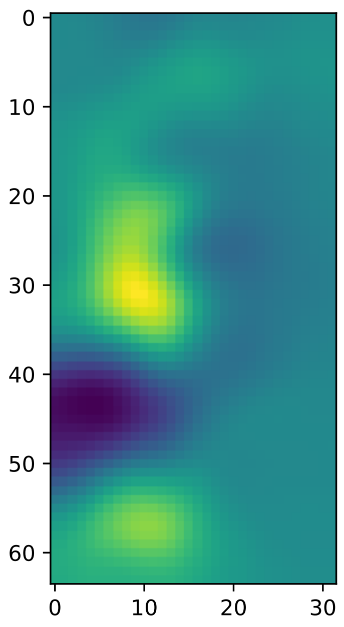

Figure 34: Data after applying gaussian filter.

Another issue here is that there are negative values and in general much noise that could interfere with the model's prediction. Given that the training data does not have any noise, a cut in value has been made to the data. This means that values below a certain threshold are set to 0. This can be done since we only care about the blobs, which have a higher density than the surrounding plasma. In figures 35 and 36 different cutoff values are shown. The final cutoff of the experimental data used was set to 50. This value was deemed reasonable when the maximum value of the experimental data was over 400. Lastly, in the experimental data preprocessing, we use Equation 31 to have values between 0 and 1. This can be done since we have removed all negative values from the array.

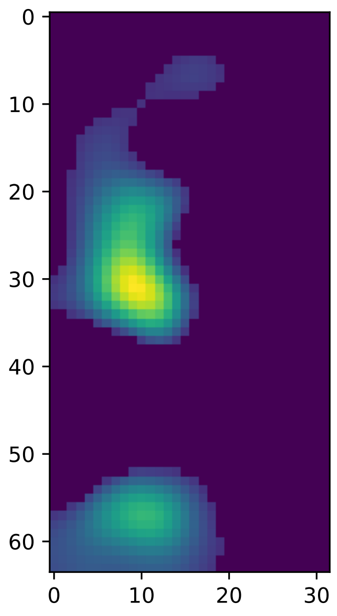

Figure 35: Cutoff set to 10.

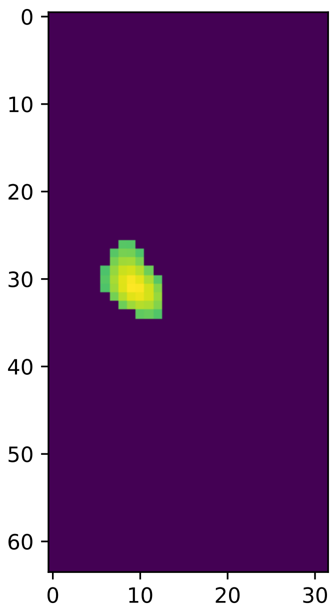

Figure 36: Cutoff set to 50.

Part IV / Results

6 LSTM-based model

When training the LSTM, the models' architecture and the number of parameters stayed the same, as it was the largest possible model runnable on the 12 GB GPU without having memory problems. Therefore, the same model was trained on different data sets to gain performance. Since synthetic data sets can be generated endlessly, the only limitation of training is the memory. My expectation is that the model will generalize better from training on more data. The data sets in this section are almost identically generated, but the GPU can only handle 3000 samples for each run.

6.1 Training set

In training the LSTM model, there were some memory issues when we wanted to use large data sets. Therefore, smaller samples of the data were used. The default parameters for the data sets are described in this section. If the text does not state otherwise, assume the values described. Each model had 3000 generated data samples of time length of 100 frames. Each of these data sets had been generated by 2D_propagating_blobs. Each sample had between 1 and 9 blobs in the time frame, randomly generated by a NumPy function. The time frame of the generated data was 10 time units and the time increment was 0.1 time units. The sizes of the blobs were generated from a gaussian distribution with $\mu = 6$ and $\sigma = 1$. The x velocities were also generated from the same distribution, but the y velocities were generated from a gaussian distribution with $\mu = -0.5$ and $\sigma = 0.2$. The amplitudes were generated from a gaussian distribution with $\mu = 70$ and $\sigma = 10$. Since the only exciting part of the experimental data is in the right part of the 64x64 image, the experimental data was cropped to become a 64x32 image to help with the memory shortage issue. These values were best compatible with the experimental data when I compared them.

The validation set is generated with the same premise as the training set, except for the number of samples. Both the testing- and the validation set have 500 samples.

6.2 Training

First, the models were trained with weights set to 0.1 for class 0, the background, and 0.9 for the rest, the blobs. This is done because of the disproportion of the background compared to the blobs. This was trained over 50 epochs.

The first model was trained with a data set containing up to 8 blobs. This model was trained for 50 epochs. The accuracy and the IoU scores for each class and overall are shown in table 2. This table shows that the model is better at calculating the first blobs,

	0	1	2	3	4	5	6	7	8	overall
Acc	0.9948	0.9058	0.3106	0.2674	0.3211	0.2180	0.1454	0.2826	0.1736	0.9075
IoU	0.9932	0.4807	0.2036	0.1546	0.1616	0.1288	0.0919	0.1357	0.0931	0.2411

Table 2: Accuracy and IoU score for the first model for each of the classes. We see that the model is better at finding the background and the first blobs.

	0	1	2	3	4	5	6	7	8	9	Overall
Acc	0.9942	0.9355	0.7672	0.6839	0.5893	0.5165	0.3634	0.6039	0.4755	0.5163	0.9098
IoU	0.9941	0.6905	0.5274	0.4134	0.3348	0.2765	0.1913	0.2139	0.2103	0.1586	0.3982

Table 3: Accuracy and IoU score for the second model for each of the classes.

and best at calculating the background, which is classified as class 0. The overall accuracy is 90% but given that the data set is greatly imbalanced, this is not a great result. Given the poor performance, this model is not tested on experimental data.

For the next model, we use the ”pretrained” model above as a base, except that we use a maximum of 9 blobs. This model is also trained for 50 epochs. The accuracy and IoU of this model are shown in table 3. In this table, it is shown that the model is performing better on the generated test set with a large margin. This is especially visible in the IoU scores, where the overall IoU went from 0.2411 to 0.3982. This model is tested on the experimental data, and one shot with two blobs, one small and one large, is chosen to show how the model is performing on the experimental data. The normalized shot is shown in figure 37, where the two blobs are visible. Figure 38 shows the output of the model, the mask. Here we can see that the model does not perform well, in that it classifies the same color in different places, and the large blob has multiple labels. There are also smaller points that are just in random colors, and it is concluded that this model is not suitable for use on the experimental data.

For the third try of this model, we used the last model as the ”pretrained” model. A new data set was generated with the same configuration as the last model, which was also trained over 50 epochs. The accuracy and the IoU of this model are shown in table 4. From this table, it is shown that the accuracy and the IoU are lowered compared to the previous model, but the trend is the same as for the other models. The model is better at finding the first blobs and is progressively worse the more blobs. However, we can see from figure 39 that the model more correctly finds the shape of the blobs. It still has the problem of misclassifying the blobs, where the large blob is classified as 3 different blobs.

The fourth time the model was trained, a training set with the same configuration as the last run was generated, and this model was also trained for 50 epochs with the third model as the ”pretrained” model. The accuracy and the IoU of this model are shown in

	0	1	2	3	4	5	6	7	8	9	overall
Acc	0.9942	0.9226	0.6690	0.5567	0.3857	0.4465	0.2146	0.3395	0.0246	0.0042	0.9635
IoU	0.9940	0.6135	0.4159	0.2844	0.2083	0.2129	0.1436	0.1618	0.0220	0.0041	0.3032

Table 4: Accuracy and IoU score for the third model for each of the classes. The model is better at finding the background and the first blobs.

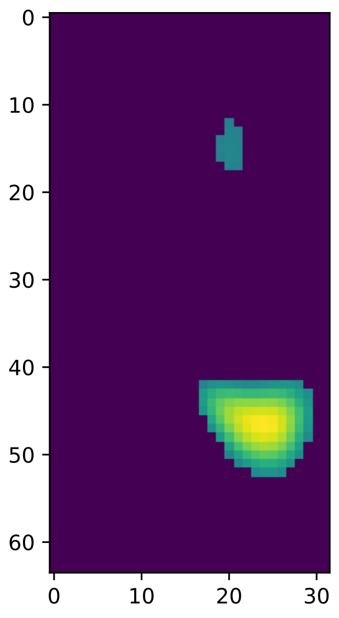

Figure 37: The input example used for comparing the different LSTM models.

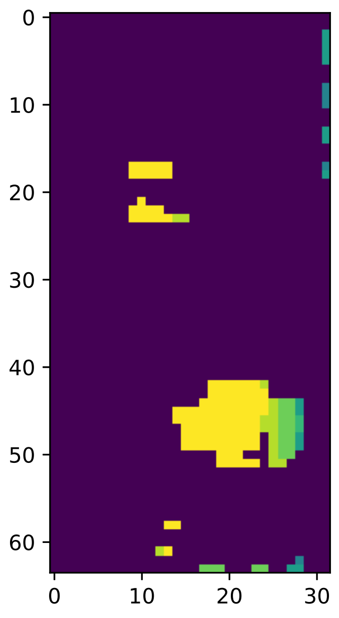

Figure 38: Shows the prediction at timestep 70 for the second model.

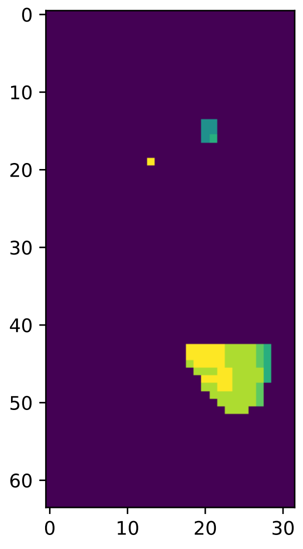

Figure 39: Shows the prediction at timestep 70 for the third model.

	0	1	2	3	4	5	6	7	8	9	overall
Acc	0.9919	0.7159	0.1051	0.02759	0.0026	0.0005	8.0394e-05	0.0	0.0	0.0	0.9217
IoU	0.9889	0.1872	0.0617	0.0223	0.0025	0.0005	7.9329e-05	0.0	0.0	0.0	0.1212

Table 5: Accuracy score for the fourth model for each of the classes. We see that the model is better at finding the background and the first blobs.

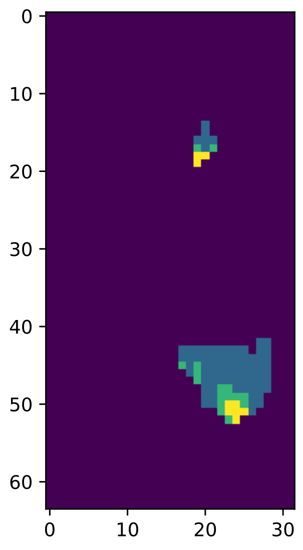

Figure 40: Shows the prediction at timestep 70 for the fourth model.

table 5. Here we can see that the model performs worse than the previous model on the test set. This table shows that the model stops predicting the latter blobs. This is shown by 0.0 accuracy and IoU. When applying the model to the experimental data, we can see from figure 40 that the model still predicts the blob's shape close to the actual shape. Still, the classification is inaccurate, and it has a lot of different classifications inside one blob.

There was one last test on a data set with larger velocities. In this data set, the x-velocity had a gaussian distribution with $\mu = 3$ and $\sigma = 0.5$, while the y-velocity had a gaussian distribution with $\mu = -2$ and $\sigma = 0.5$. This test was supposed to be trained on 50 epochs over the "pretrained" model 2 but it got NaN values for accuracy and loss after about 20 epochs. With that NaN result, the conclusion was that this model was not well suited for this type of task, and there was more potential in other models, such as the transformer model.

The overall conclusion of the tests on these models is that the model could find the shapes of the blobs correctly but it could not correctly distinguish between them. Another aspect to take into consideration is that the training data that was generated only has the option to be a gaussian shape, i.e. a circle, which does not correctly represent the blobs in the experimental data. This could be a factor in why the models, especially model 2, misclassify the blobs even though it predicts the correct shape.

7 Transformer

Since the transformer architecture is relatively new compared to LSTM and I have no prior experience with the architecture, some tests with smaller data sets were done. This gave an insight into how the model is performing on this type of data. The smaller data set allows for faster testing, due to the training time and the memory required being lowered significantly. Therefore, two models are presented in this section, the first on smaller data sets, and a larger one which will be used on experimental data.

7.1 Small scale model

An example of the model's performance is shown here. In this sample, the left side is the model prediction and the right is the ground truth. The model accuracy and IoU are described in table 6. The final loss was at 0.1015. All these measurements are improvements from the LSTM models. With this result, expectations for the transformer are high.

	0	1	2	3	4	overall
Acc	0.9958	0.5943	0.4576	0.5094	0.7378	0.9798
IoU	0.9948	0.5508	0.2510	0.1944	0.2267	0.4435

Table 6: Accuracy of the small Transformer model. This shows promising results compared to the LSTM.

This model is too memory hungry to be used on larger models but it is used as a baseline on how to build the final model.

7.2 Large scale model

In this section, the results of the large scale transformer model on synthetic data are presented. This model can predict 20 blobs, i.e. there are 21 classes. The training and validation accuracy over the 200 training epochs can be shown in figure 41, and the loss is presented in figure 42. This shows that the mode comes to a halt in validation training at around 50 epochs. However, as the accuracy is stable, there does not seem to be an overfitting problem. The accuracy and IoU scores of this model are given in table 7. This table shows that the high intermittency of blobs reduces the model's capability to differentiate between blobs. When the intermittency is high, the blobs tend to overlap, causing problems for the model. Another interesting aspect of this model is that the accuracy of class 20 is much higher than the previous ones. From inspecting the predictions, this is eighter from the last blob being nonoverlapping, or that a lot of the later blobs are just classified as the 20 blobs. Hence the blobs that are blob 20 are classified correctly, giving high accuracy. This can also be seen in the high accuracy vs. the low IoU in table 7.

Tracking blobs can be difficult due to some errors in the model. The model seems to predict 20 more often than other blobs. This results in some strange results when inspecting

Class	Accuracy	IoU
0	0.9962	0.9946
1	0.8063	0.7077
2	0.6504	0.4906
3	0.5429	0.4136
4	0.3642	0.2428
5	0.3337	0.2122
6	0.2940	0.1805
7	0.2352	0.1490
8	0.1876	0.1046
9	0.1723	0.08917
10	0.1808	0.0875
11	0.2169	0.0990
12	0.1109	0.0696
13	0.1191	0.0613
14	0.0808	0.0410
15	0.0493	0.0244
16	0.1316	0.0558
17	0.0915	0.0267
18	0.0988	0.0392
19	0.1914	0.0207
20	0.8793	0.1253
Overall	0.9566	0.2015

Table 7: Accuracy and IoU on the test set for the large scale transformer model. The table presents overall measurements, as well as the individual measurement for each label.

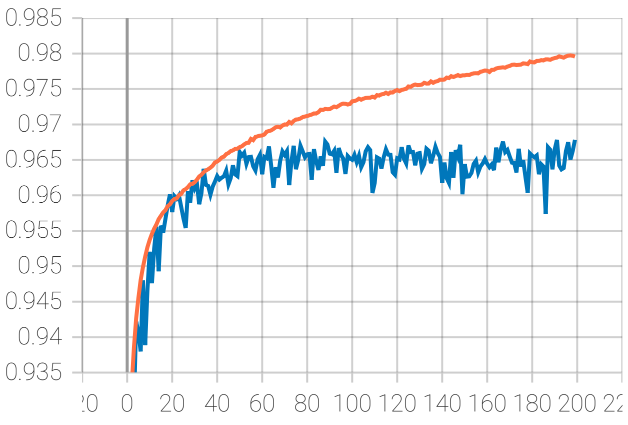

Figure 41: Accuracy graph over epochs for training and validation data.

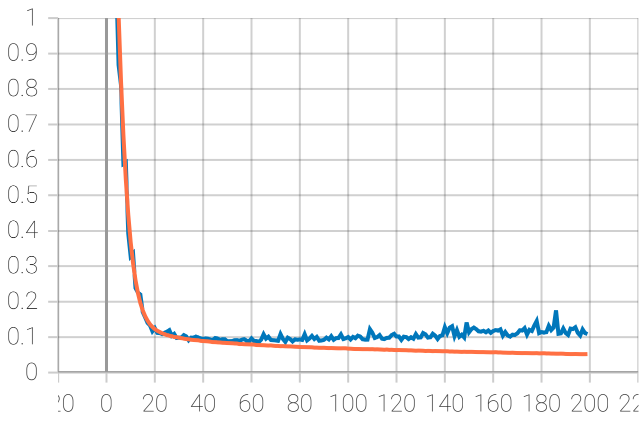

Figure 42: Loss graph over epochs for training and validation data.

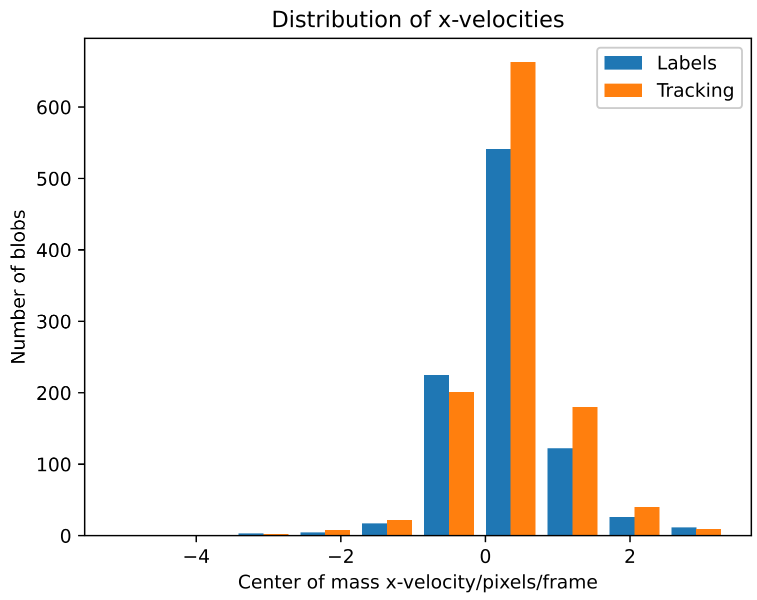

Figure 43: Histogram of x-velocities. The velocity is given in pixels per frame.

the different parameters of the model. For this analysis, a data set with 100 different arrays of 64 timesteps was generated. This data set's parameters were all gaussian distributed, where x-velocity has $\mu = 2, \sigma = 1$, y-velocity has $\mu = -0.3$, $\sigma = 0.5$, size has $\mu = 6, \sigma = 2$, and amplitude has $\mu = 70, \sigma = 10$. The values in the velocities presented next are given in pixels per frame. The sizes are given in pixels, while the amplitude is an arbitrary value.

The results on tracking x-velocity for the blobs gave a mean of 0.3195, where the maximum value was 3.2343 and the minimum -5.2096. Figure 43 shows the histogram of all blobs present in more than two frames. The reason is that we need at least two frames to calculate the difference in the position of the center of mass. A comparison with the labels is also present. The mean of the x-velocities for the labels was 0.2712, which is lower than the model's results. The max velocities were approximately the same, with a value of 3.3445. In the minimum case, there was a significant difference, where the labels had the lowest value of -2.983. Investigating more on the lowest x-velocity for the model, it shows that the model predicts three separate blobs as one. This means that the center of mass is shifted from the first blob to the center of the two other blobs and then between the three blobs when the last one enters the frame. This makes the center of mass move backward at large speeds.

When investigating the y-velocities, the model predicted a mean of 0.2459, while the labels have -0.0675. One reason the model predicts the high values is the highest value of the y-velocity, which was 16.0183 compared to the labels having 10.1089. It also predicts overall higher small values, as shown in figure 44. Here we can see higher values predicted by the model than in the labels. This also comes from the misclassification of the blobs. The lowest values found are -6.4530 for tracking and -5.8083 and labels, which is reasonable.

For amplitudes, figure 45 shows that the difference in amplitude is minimal. One thing to note from this model is that there seems to be easier to find higher amplitude blobs than

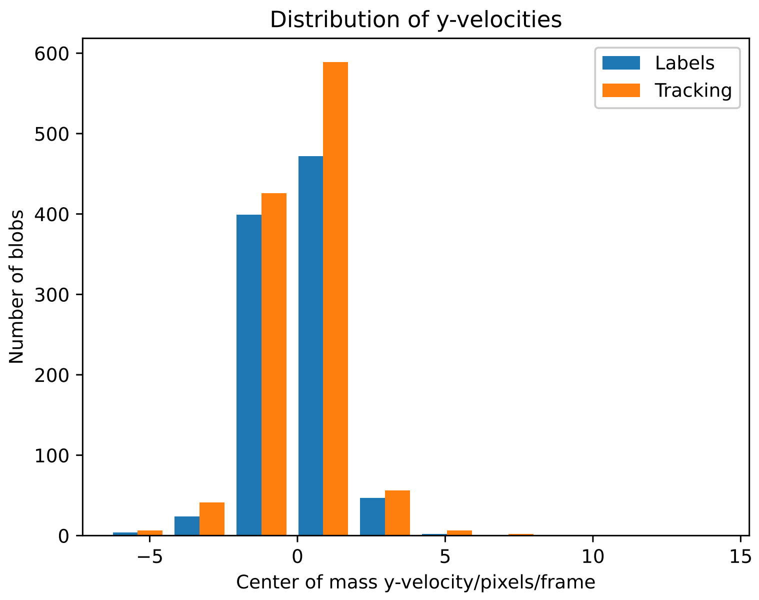

Figure 44: Histogram of y-velocities. The velocity is given in pixels per frame.

lower ones. The mean was 32.6986 and 30.0802 for the prediction and labels, respectively. The maximum amplitude is equal for both, at 69.3690. The lowest found max amplitude has an insignificant value difference, at 0.1355 and 0.0039 for the predictions and labels, respectively.

Investigating the sizes of blobs, shown in 46, reveals that the model often predicts larger structures than what is present in the data set. This becomes clearer from the mean, where prediction gives 102.1168 while the label's mean is 83.0230. The issue of the model combining blobs is also present in the analysis of the blobs. This results in the largest blob structure predicted by the model is at 533, while the label's largest structure is 344. Another thing to mention is that the synthetic data generation only classifies the data that has 70% of the maximum amplitude of the blob as the blob. This could be one of the reasons why the model has larger sizes.

The histograms in this section show all blobs detected by the transformer vs. the labels. The reason why the bar is larger for predictions is that the model finds more blobs than there are in the labels.

7.2.1 Intermittency

Intermittency is one of the most essential parameters regarding blob tracking. Intermittency, in this case, is the measure of blob overlap. To track how well the model performs when the number of blobs in the data set increases, a data set with the number of blobs ranging from 1 to 20 is generated. 100 data sets per number of blobs up to 20 are generated, and each data set has 64 timesteps and 64x32 image size. After the prediction, the number of blobs found by the model is then divided by the number of expected found blobs. The results of this analysis are shown in figure 47. The expected results were that

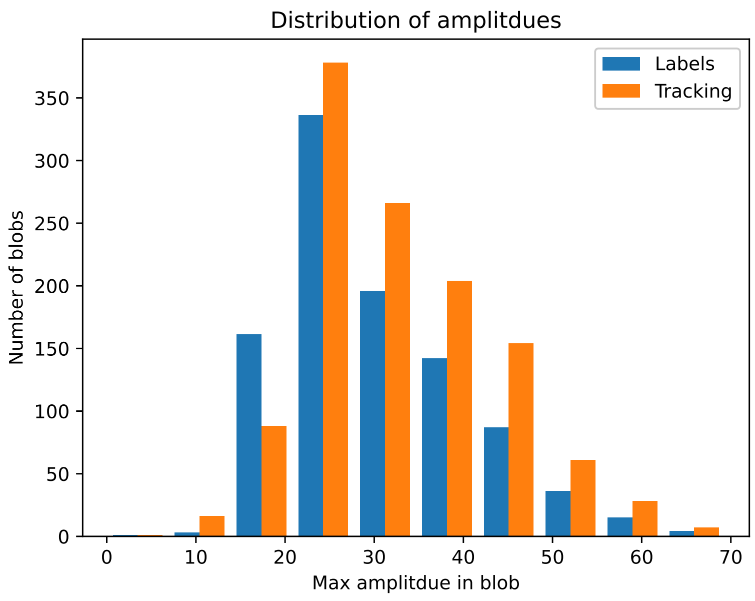

Figure 45: Histogram of amplitudes.

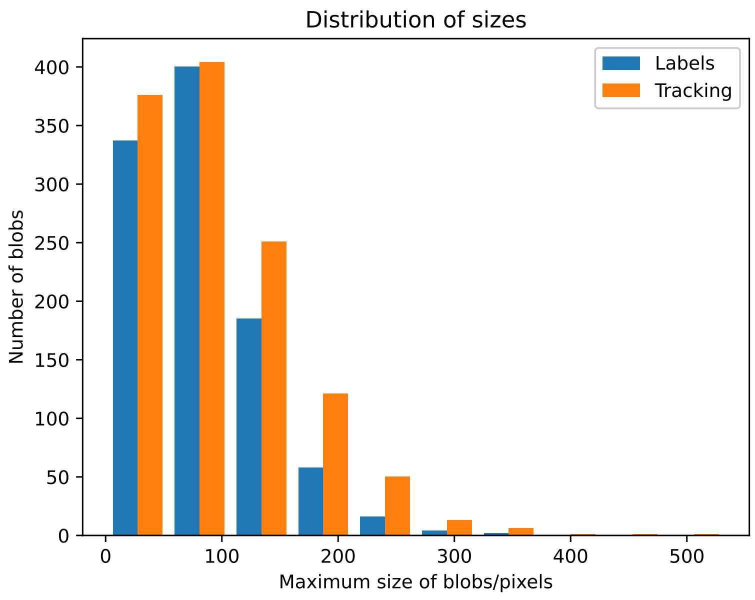

Figure 46: Histogram of sizes. Sizes are given in pixels.

when the number of blobs is increased in the same timeframe, the ratio between detected and real blobs is reduced. The graph is not as expected for the first half. This is because the model often splits the blobs into different blobs over time or creates new blobs when the blobs are overlapping. The first is shown in figures 48 and 49. In this particular incident, blob number 3 becomes number 4 after five timesteps. When there are more blobs in the frame, the model cannot differentiate between the blobs, and the labeling of the blobs becomes wrong. This is shown in figures 51 and 50.

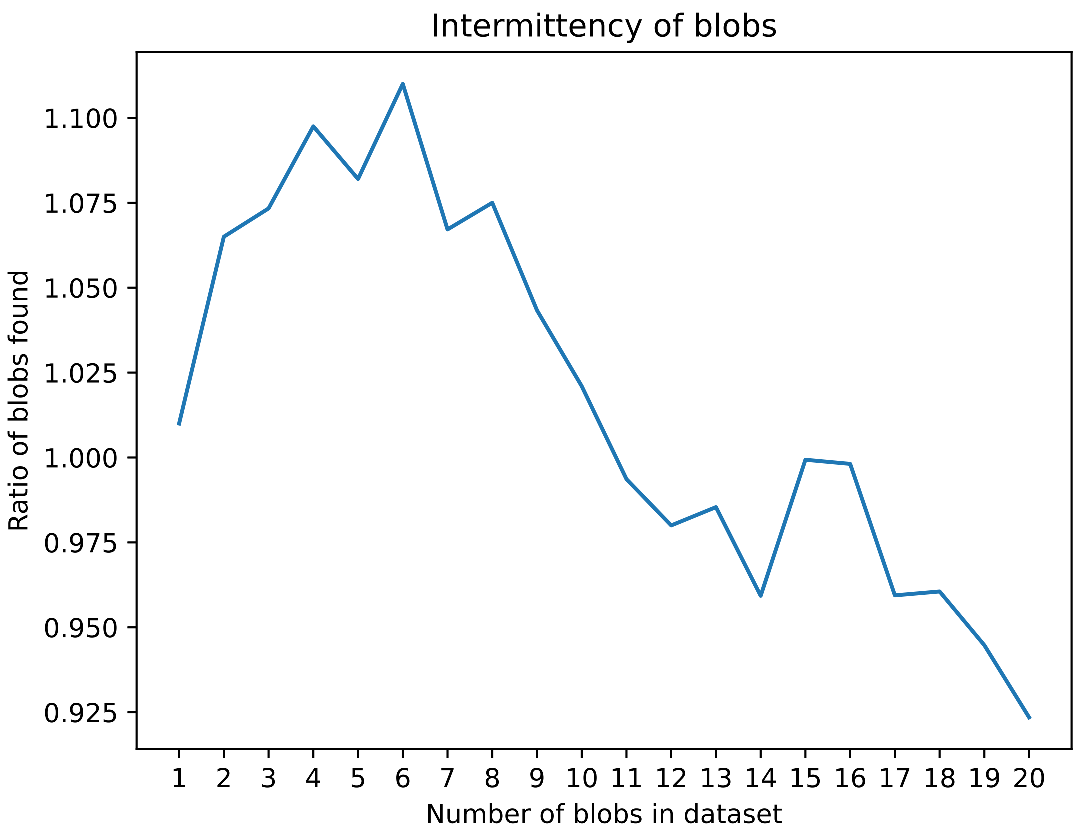

Figure 47: Number of blobs found compared to the number of blobs in the data set.

This analysis produced very limited insight into how well the model performs, so an analysis of the general segmentation (class 0 vs. rest) is done. This is done by checking where the model has predicted something other than class 0 up to where there is a blob. This analysis is done to see if the model can find the blobs in the data set. This accuracy is shown in figure 52, where the expected decay in accuracy with increased blobs is shown. The IoU is also shown in figure 53 where the same trend is present.

7.2.2 Different sizes

To test the segmentation of the model on different sizes, all the blob labels have been set to one. Here we check how the model is affected with no variation in size, a small variation of $\sigma = 1$, a larger variation of $\sigma = 2$, and a very large variation of $\sigma = 5$. The

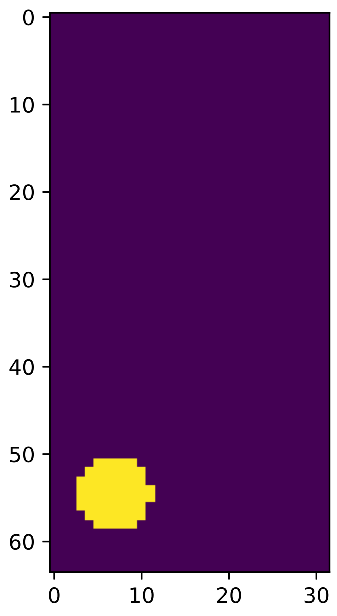

Figure 48: Label showing the example of a blob changing label over time. The yellow color represents label 2, i.e. blob number 2.

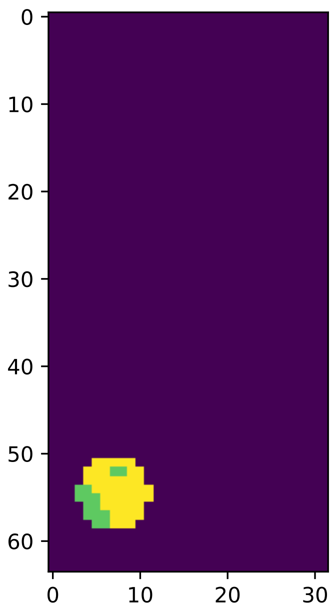

Figure 49: The prediction of the model on the same frame. Here we can see that the model has predicted the blob as two different blobs. Yellow color in this figure represents label 3, while the green is label 2, which is the original label for the blob. This results in more blobs in examining intermittency when looking at the data set and counting the number of different labels.

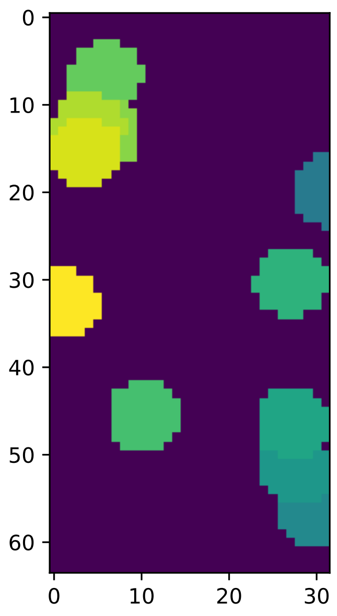

Figure 50: Label of an example where there are a lot of blobs. Here the overlap of blobs is shown.

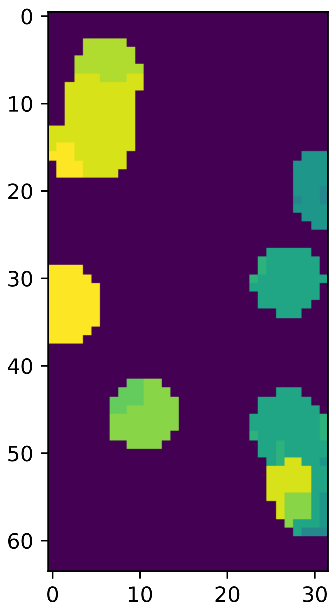

Figure 51: Prediction of the same frame as the left image. Here we can see that the model struggles when there is overlap, and given more blobs, there is bound to be overlap.

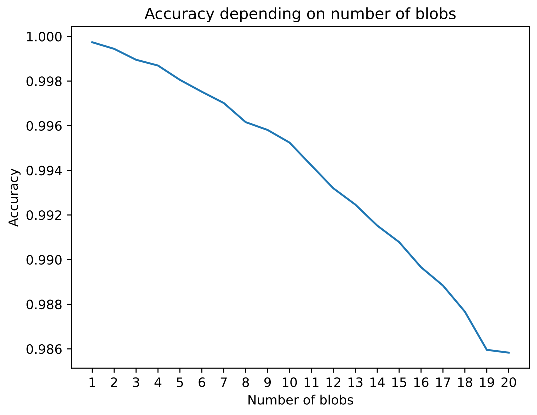

Figure 52: Overall accuracy of blobs and background for intermittency. A trend corresponding to the number of blobs and the accuracy is shown. The negative trend shows that the model performs better when there are fewer blobs and less overlap.

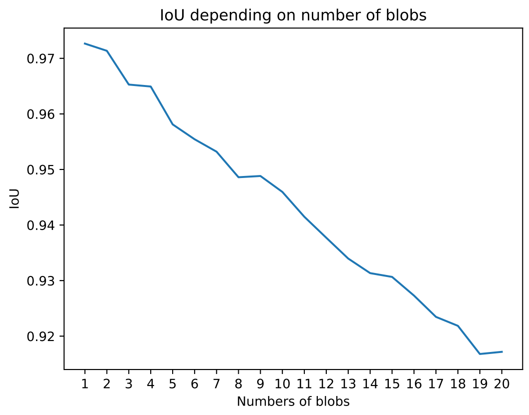

Figure 53: IoU of the blobs. Confirming what the accuracy graph suggests, that the more blobs, the less correct the model predict.

result on one to 20 blobs is shown in figures 54 and 55. This shows the expected results, that when sigma increases, the accuracy and IoU decrease.

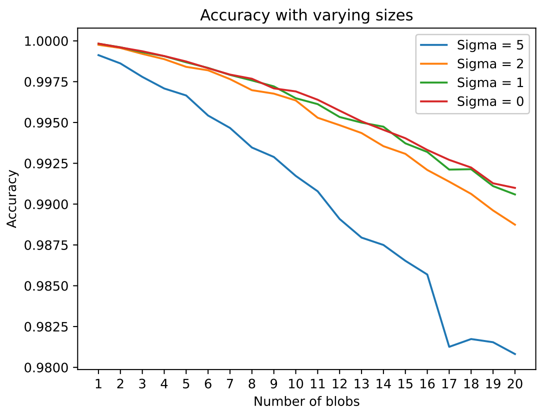

Figure 54: Accuracy graphs of different sigma values in the size parameter. There is a trend where higher variance results in lower accuracy.

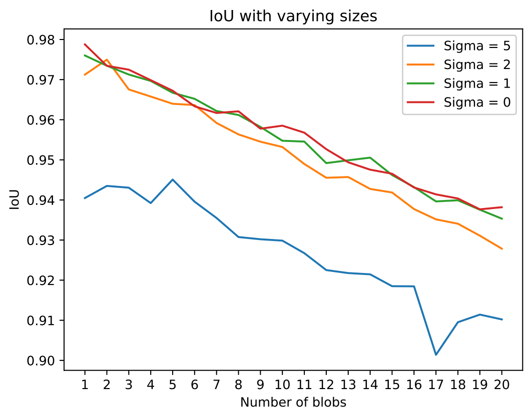

Figure 55: IoU graphs of different sigma values in the size parameter. The same trend as shown in the accuracy is presented where the higher variance results in a lower score.

Since the model is performing so poorly on data sets with many blobs, we look at two blobs generated and how well it handles different sizes of the blobs. To test how well the model works on different sizes of blobs, we generate a data set with 100 data sets with sizes ranging from 2 to 30 with an increment of 2. This data set has constant blobcount, amplitude, and velocities. When the blob has a size of 30, it covers almost the whole frame, so it's expected that the model will perform poorly on these larger blobs.

In figure 56, the overall accuracy when comparing the blob sizes with constant amplitude, number of blobs, and velocity is presented. This figure shows what is expected of the model, where smaller blob sizes are segmented correctly while the larger ones are more wrongly segmented. Another thing to consider in this plot is that the smaller sizes have fewer pixels to segment, hence less room for mistakes in the model. In figure 57, the IoU of the different sizes is presented. In this graph, we can see the same trend as in the accuracy graph. There is shown a small increase at the higher sizes, but I do not know if this is due to a low sample size or something else. There should not be an increase at the end according to the trend. When looking at the predictions of the larger blobs, they are predicting wrongly with a larger margin than the lower-sized ones. However, these are still high accuracy and IoU scores, so the model is performing well in finding where there are blobs.

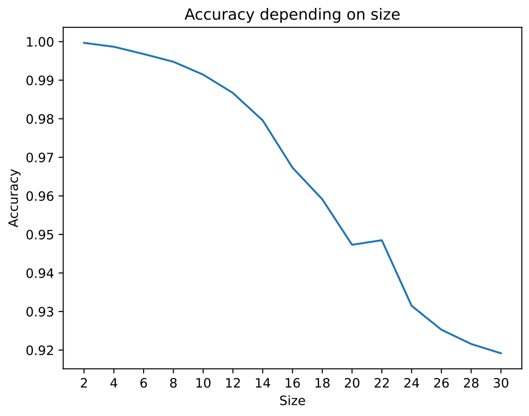

Figure 56: Overall accuracy of blobs and background when comparing different sizes.

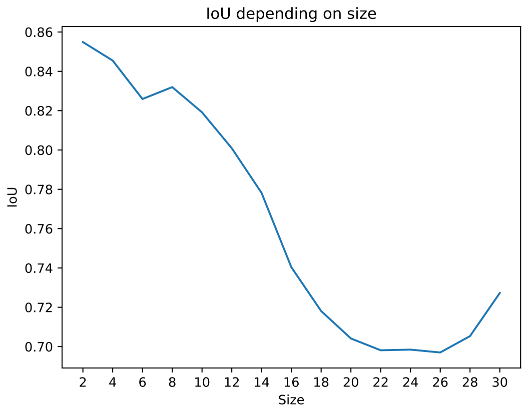

Figure 57: IoU of the blobs comparing different sizes.

To better understand how well the model separates blobs, a method of matching the best-predicted blobs with the labels is made. This is done by separating each of the blobs predicted (each has a different label) into an array and doing the same with the labels. Then check which of the predicted blobs has the best accuracy with the label. The average accuracy of the blobs is shown in figure 58 where we can see a lot of resemblances to figure 56. As stated before, we see that the larger the blob size, the less accurate the model becomes. However, when looking at figure 59, the overall IoU score has decreased significantly. This comes from the overlap of blobs, where when large blobs overlap and cover the whole frame, it is hard to differentiate between them. Another aspect to take from these graphs is that the IoU is at maximum at size 6, which is the mean of the gaussian distribution that the blobs were sampled from. This means that the model has had most of the blobs be of sizes around 6, decreasing when the blob size is increasingly different from 6.

To show why the model is struggling with some of these data sets, figures 60,61 and 62 are added to show that, as a human, it is hard to separate them as well. From the synthetic data, it can seem there is one blob, but we can see from the label that there are two blobs on top of each other. Another aspect of this test is that the model has never seen such large structures in training, as these were gaussian distributed around 6.

7.2.3 Different velocities

The same analysis done on different sizes is applied with different x-velocities. The results are presented in figures 63 and 64. This result is expected, where the higher variation

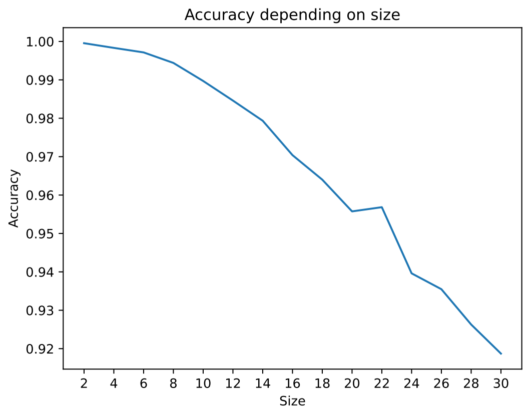

Figure 58: Overall accuracy of blobs and background when comparing different sizes when the labels are matched with the best predictions.

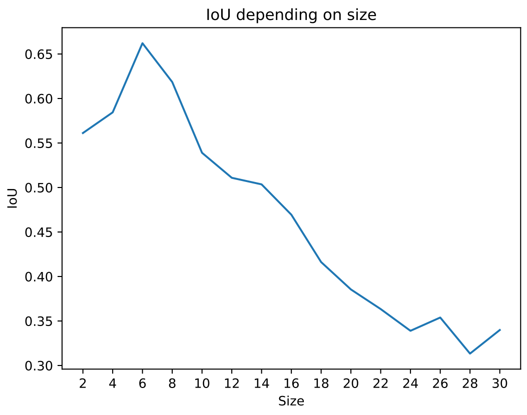

Figure 59: IoU of the blobs comparing different sizes when the labels are matched with the best predictions.

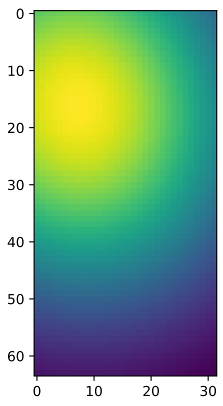

Figure 60: Input of the model, the synthetic data.

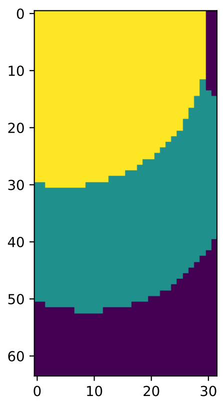

Figure 61: Labels given to the model.

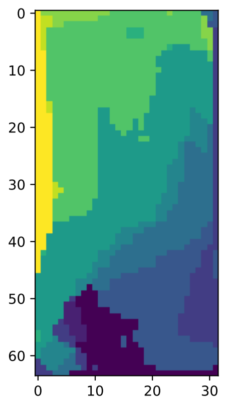

Figure 62: Models prediction. Here it is shown how the model cannot predict correctly, and hence give bad result when there are large overlapping blobs.

reduces the model performance. This is not so well described in the accuracy graph but is shown in the IoU graph, where the higher variance, the lower IoU.

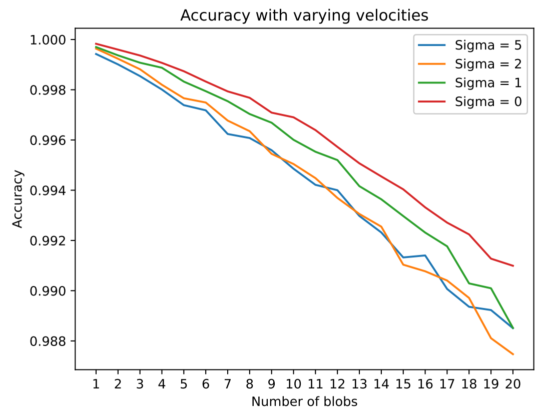

Figure 63: Overall accuracy when looking at velocities sampled from different distributions. This analysis is done with all blobs' labels set to one.

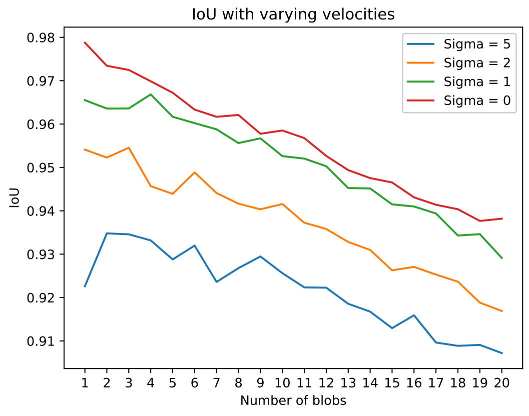

Figure 64: IoU when looking at velocities sampled from different distributions.

Since the model is performing so poorly on data sets with many blobs, we look at two blobs generated and how well it handles different velocities of the blobs. When inspecting the data with different velocities, the highest velocity that could be relevant to inspect is ten. This is because, in higher velocity values, the blobs are only in one frame or none at all. At x-velocity = 10, each blob has two frames. data sets with different velocities of the blobs were created. 100 examples per velocity were generated, all with two blobs.

In figure 65, you can see that the overall accuracy increases with the increase of velocity. The reasoning for this is that the blobs are shorter in the frame. Therefore, there are fewer pixels with blobs in them, and the model is performing better since there is less to segment. Figure 66 does not line up with the expectation of the predictions. The expectations were that the IoU would become less accurate the higher the velocity. Nevertheless, these are high accuracies and high IoU, and when looking through the data sets, they are mostly correctly segmented, with the edges being a bit off.

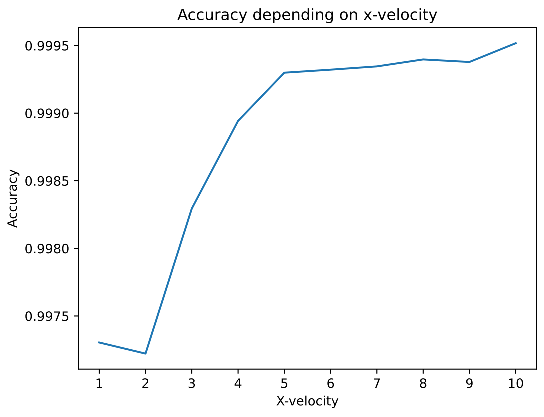

Figure 65: Overall accuracy of blobs and background when comparing different x-velocities.

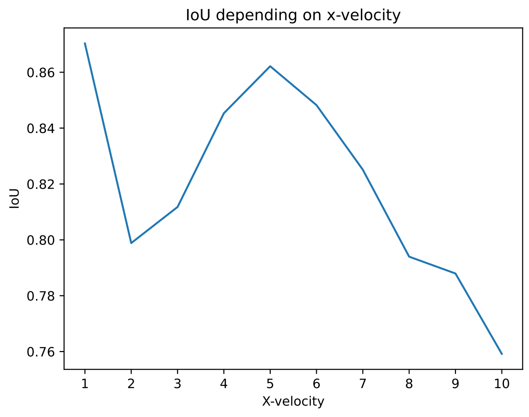

Figure 66: IoU of the blobs comparing different x-velocities.

The same method used on different sizes is applied to this section, where best-fitting predictions were matched. This gives us the result shown in figures 67 and 68. The accuracy graph is almost the same as the one for the whole data, except the values are lower. The interesting thing about these graphs is shown in the IoU graph. The model is struggling to correctly differentiate between the blobs due to the high drop in the IoU score, compared with the one where all blobs had the same labels. It also shows the expected results that the model is less likely to separate correctly between blobs when velocities become high. It is also shown that the model has relatively high IoU scores for blobs with velocities around the velocities given at training, which was gaussian distributed with two as the mean.

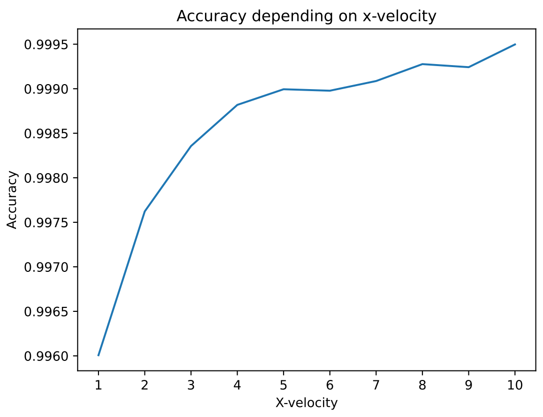

Figure 67: Overall accuracy of blobs and background when comparing different x-velocities.

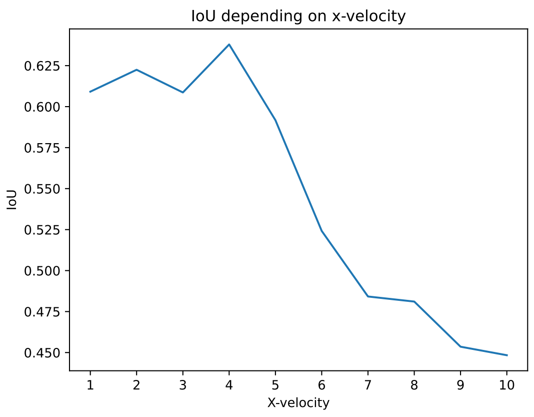

Figure 68: IoU of the blobs comparing different x-velocities.

For y-velocities, we expect the same results as the other parameters, hence no analysis is done on this.

7.2.4 Different amplitudes

A test with two different variations in amplitude has been tested in the same way as the others. In these two data sets, one has no variance in amplitude, while the other has a $sigma = 20$. The accuracies and IoU are presented in figures 69 and 70. These results show the same trend as the other parameters. When the variance increases, the scores decreases.

7.2.5 Different x-velocities and sizes

Lastly, a test where we combine varying sizes and velocities with the same process as before. All labels for blobs are set as the same to check how the segmentation is affected by the variation of parameters. The accuracies and IoU from data sets containing 1 to 20 blobs are presented in figures 71 and 72. These graphs show the expected drop in accuracy and IoU with increasingly more complex data. The same drop with higher variance is shown, which is expected. From this, we can expect that increasing the variance of multiple parameters further complicates the problem and reduces the scores even further.

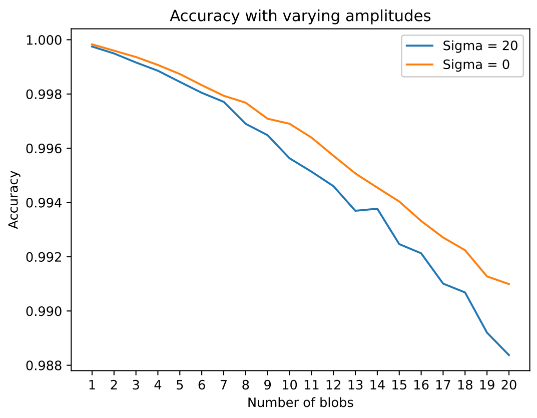

Figure 69: Overall accuracy of blobs and background when comparing different amplitudes.

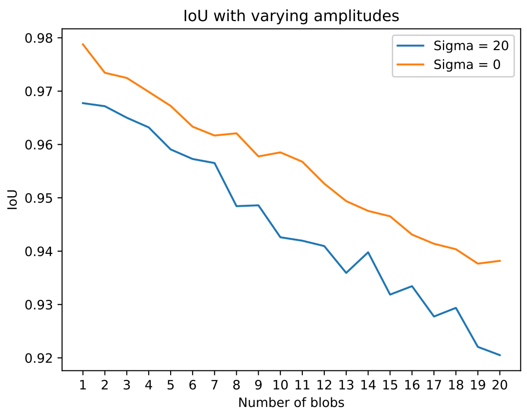

Figure 70: IoU of the blobs comparing different amplitudes.

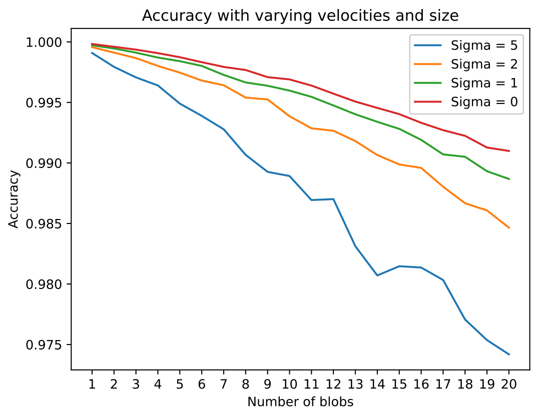

Figure 71: Overall accuracy of blobs and background when comparing different x-velocities and sizes.

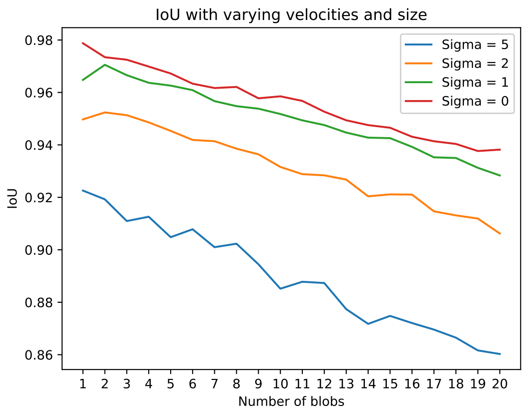

Figure 72: IoU of the blobs comparing different x-velocities and sizes.

7.3 Experimental data

The Transformer does not show great results when predicting experimental data. From observation, within one blob, there are multiple classifications. There are also some other problems that the model encounter, described later.

In figures 73 and 74, the output and the input of the model are presented, respectively. The figures show that the model finds the shape of the blobs, but it labels it with many labels, just like the LSTM model. The large blob in 73 has seven different labels when it should be labeled as one.

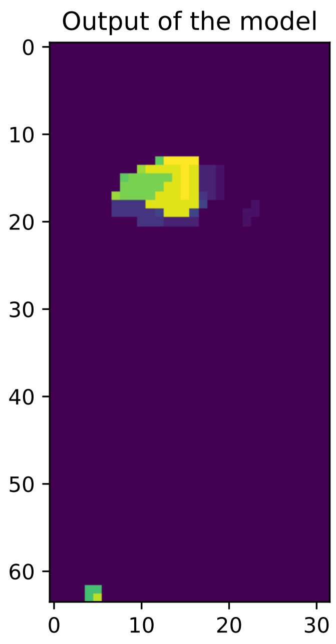

Figure 73: The raw output of the model on experimental data, shown for timestep 192.

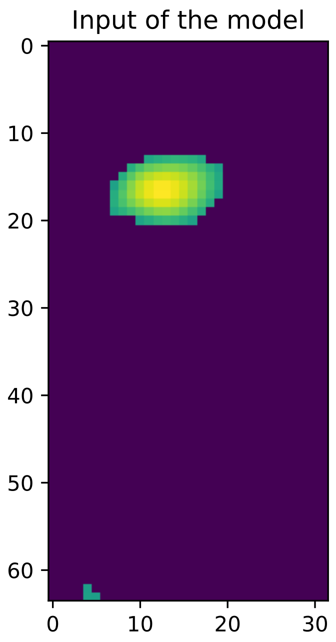

Figure 74: The input of the model. it is shown that the blobs, even after preprocessing do not have gaussian distributed shape.

From the experimental data, there is some strange behavior in the model. The first unusual example is that the model predicts a "halo" around the blob. This is seen when the shape of the blob is not gaussian, and this could be the cause of this problem. The model is trained on perfectly gaussian shaped blobs, and introducing imperfect blobs could cause this behavior.

The second unusual example is that the model could suddenly predict a large structure from nothing. In this example, the input data contains only 0-valued pixels, and the model suddenly predicts larger and larger structures with the highest class possible (20). There doesn't seem to be any logical explanation for why the model would act this way.

The third example is that the model predicts blobs as many different labels. This is the same problem that the LSTM model encountered. This problem is most likely caused by the same issue as the first example, where the model has never seen imperfect Gaussian shapes of the blobs.

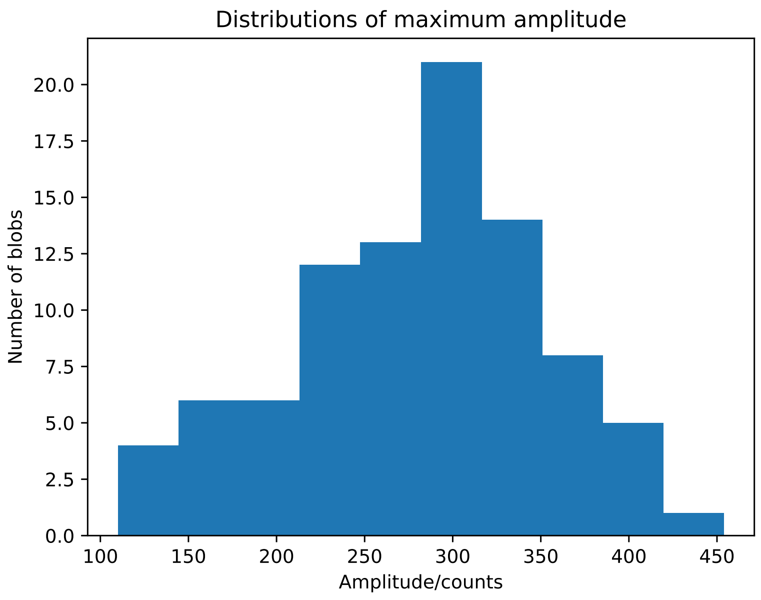

Figure 75: Histogram of amplitudes for experimental data.

Trying to calculate the velocities of the blobs is problematic due to the model prediction of multiple blobs within one blob. With the help of a scipy function [80] called scipy.ndimage.label, we can calculate the amplitude, size, and velocities of the blobs. There is also applied a manual removal of the predictions that are made where the input is equal to zero. When inspecting this, we find that the mean lifetime of the blobs is 16.56 frames. The highest being 103 frames and the lowest one frame. The next parts and include a comparison between the results in this thesis and another paper [16], however these results are on different data sets, so there will be variation between the two.

The amplitudes are plotted in figure 75. This amplitude is taken as the maximum amplitude where the blobs are detected. From this graph, it is clear that the amplitudes are gaussian distributed. As these two data sets are different samples, the amplitudes are not compared.

All the x-velocities calculated from the data are presented in figure 76. This graph shows that the x-velocities are gaussian distributed, with a statistical outliner with a -2043 value. Without this, the mean x-velocity is 23.5453 m/s. This is lower than what is shown in the other paper, where the value is between 150 m/s and 450 m/s [16]. Analyzing the data further, most data is around 0, where eight values are exactly 0. In this analysis, the center of mass had to be rounded to be transformed into the regime of the actual values, so this might be causing the error. Excluding values where the velocity equals 0 gives a mean of 25.8599.

All y-velocities calculated from the data are presented in figure 77. This graph shows that the y-velocities are gaussian distributed. The mean of the y-velocities is -107.5339 m/s. This is closer to the values found in the other paper, at around -150 m/s [16].

The areas of all the blobs are presented in figure 78. From this graph, it is clear that the sizes are exponentially distributed. The mean of the area of the blobs predicted by

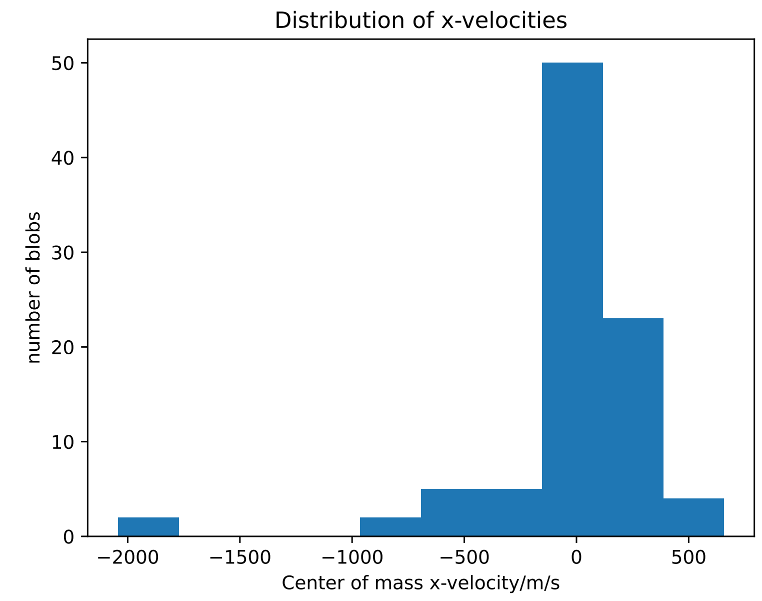

Figure 76: Histogram of x-velocities for experimental data.

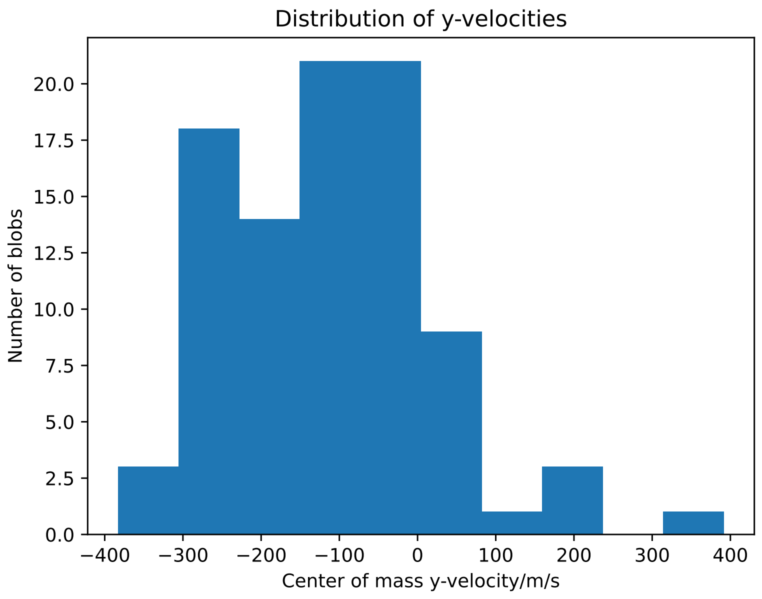

Figure 77: Histogram of y-velocities for experimental data.

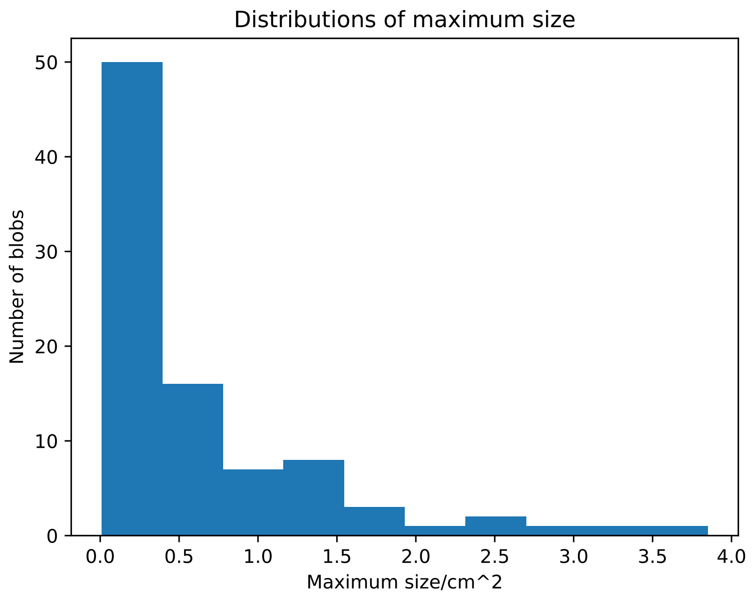

Figure 78: Histogram of sizes for experimental data.

the transformer is 0.6266 cm^2. The comparison of the paper is inaccurate due to the paper stating the length in radial and poloidal directions. However, assuming the shape of a circle, with $l_{pol} \approx l_{rad} \approx 6.5$mm [16], the approximate area of the blobs are 0.3318 cm^2. This result is in the same order of magnitude as the predictions. The larger sizes in the predictions might be attributed to the fact that the model was trained with larger structures.

8 Discussion

To compare the models' performance on the experimental data, we look at the comparison between another model used for the same type of tasks.

8.1 RAFT

The segmentation of this model is shown here, where the original video is on the left side, the flow is shown in the middle, and the mask is shown on the right side. This video was generated with the values discussed in section 5.3.3. When processing the sample above in RAFT, my computer used 17.3 hours to predict the 2000 timesteps and make a video.

8.2 LSTM

The LSTM created in this paper is a relatively simple model to test how an LSTM model compares against transformer models. The model is described further in section 6.

This model had problems with classifying the blobs correctly in the experimental data. For most blobs, there were multiple labels within, and this causes some problems when we want to analyze and compare models.

8.3 Transformer

The transformer model predicted the experimental data with some unusual results. What happens in deep learning models can be hard to comprehend, but there are some logical explanations as to why it behaves the way it does. The problem with the halo around the blob could come from a small activation in the patches corner, which is upscaled in the head part of the transformer. This would cause this halo since the patches are overlapping. Since this is my first time working with Transformers, I did not know this could become an issue. Another reason for this problem could be that the model has not seen shapes that are not gaussian.

The second problem the transformer had was that it would randomly create large structures from nothing in the input data. In this problem, there could be two logical explanations. One possibility is that when the data is first put through the transformer, it is misclassified, and then this spiral out of control. The other explanation is that the model sees into future frames and expects a blob. This should not be possible due to the mask attention in the Transformer, but I am not sure I implemented this correctly.

Other than that, the model has in common with the LSTM model that it cannot classify the blobs correctly, most likely due to the differences in the training data and the experimental data.

8.4 Comparison

Comparing the models' performance on the synthetic data is somewhat skewed as the models can detect the different amounts of blobs and timesteps. The LSTM is set to find a maximum of 10 blobs in 100 frames, while the transformer is set to find a maximum of 20 blobs per 64 frames. In hindsight, this was not the best way of making the models for comparison. This has also decreased the transformer's performance in overestimating the model's power. With this in mind, the results of the transformer on the synthetic data outperform the results of the LSTM.

When it comes to the experimental data, both of the models presented in this thesis perform subpar on the experimental data for separating blobs and classifying them. With this result, the conclusion is that semantic segmentation is probably unsuitable for this problem. It would be better to use more complex methods like instance segmentation or panoptic segmentation.

An example where the models predicted almost the same pixels is shown in figure 79. In this example, all the models found the same blob and segmented it similarly. This is a good comparison, except for the small prediction of the transformer outside the blob. Another example is presented in figure 80, where all the models find the same blob, but there is a difference in size. This could come from the way that the data is preprocessed.

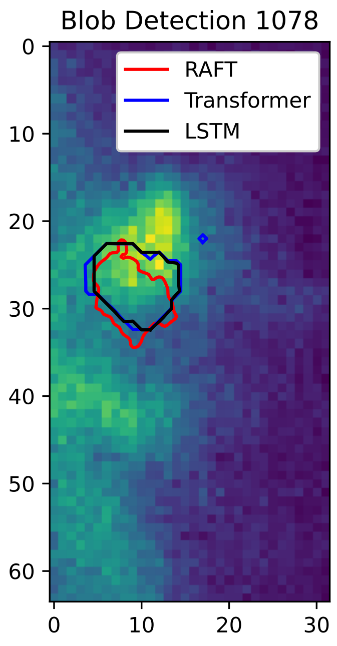

Figure 79: Comparison between the models' segmentation. In this example, the models find the same blob. The LSTM and Transformer predictions are somewhat larger than RAFT.

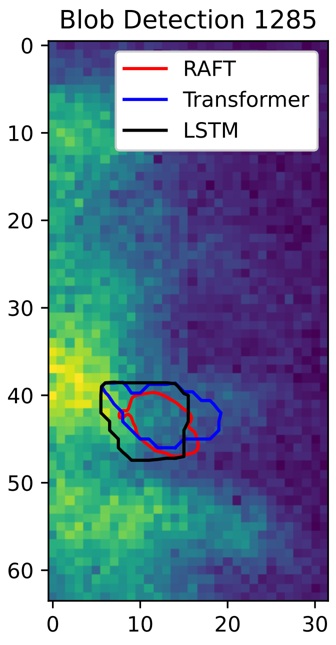

Figure 80: Another good example, except for the small part where the transformer classified a small blob outside of the larger one.

An example where the models predict blobs differently is shown in figure 81. In this example, the models detect different blobs. The LSTM detects two blobs separate from each other, but from the normalized data the blob furthest to the left should not be present as the inputdata is 0-valued. This is the same behavior as the transformer, where points outside the blob are classified as a blob. We can also see here that RAFT detects a different blob than the models presented in this thesis. In figure 82, one of the problems of the transformer is presented. There is nothing to be predicted in this frame, yet the transformer predicts large structures.

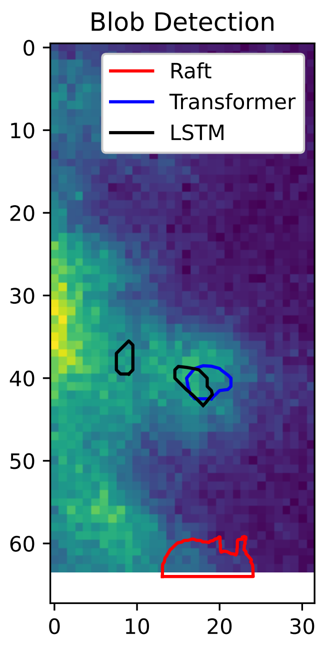

Figure 81: Comparison between the models' segmentation. In this example the models predict different blobs, and the LSTM predicts a random part not presented in the input. This is the same behavior that the Transformer has on some of the blobs.

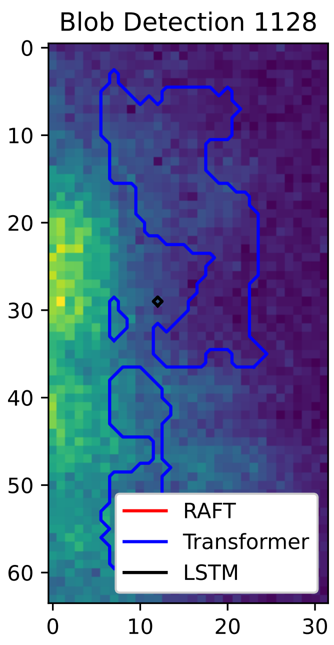

Figure 82: An example of the "large structures from nothing" problem with the transformer. It is shown in the figure that the LSTM predict a small blob in the middle and RAFT predicts nothing, while the transformer predicts large structures across the image.

Another consideration when comparing the models is the time used to predict the results. Here the RAFT model is very slow compared to the models presented in this thesis. On my computer, RAFT used around 5 hours to analyze the 2000 timeframe experimental data, while my models used minutes. This is with the original RAFT, which could run on CPU instead of GPU. At least from the Task Manager, the CPU seemed to be the most loaded. But if the case is that these models are much faster to predict, they could be helpful if fast segmentation is wanted.

It isn't easy to perform a quantitative measure of these data since the models have different presets. For example, the RAFT network looks at the 64x64 image, while the models presented in the thesis only look at the 64x32 image. This means that some of the blobs found by RAFT are not visible for these models. There is also the problem of the misclassification of the blobs by the models, which inflates the number of blobs found when running through the raw output of the models. The models also preprocess the data differently, so the input of Raft differs from the model presented. However, using the scipy [80] method called scipy.ndimage.label some comparison could be made. This method takes connected pixels, no matter the size or shape, and labels them the same. This method can be used as a quick fix for the problems my models have when it comes to differentiating between the blobs. This method can show how many connected structures my models predict and compare them to RAFT. This method gives 1625 labels for the

Transformer and 216 labels for the LSTM. The high number of labels in the Transformer model is because of the large structures and the ”halo” around blobs being classified with multiple labels. A method that sets the label = 0 when the preprocessed data is equal to 0 was tested, and this produced 91 blobs for the transformer, while the LSTM found 86. RAFT found 118 blobs, where some of which are out of frame for the data used for the model presented in this thesis.

Another measure to be used is comparing the overlap between the models. This is done by taking the IoU between the models' predictions compared with RAFT. This analysis shows that the transformer's prediction intersects with the RAFT's prediction in 246 frames, while the LSTM intersects in 208 frames. If we set a requirement of minimum 0.1 IoU, the models intersect in 132 and 117 for the transformer and the LSTM respectively. The transformer's highest IoU gets 0.6626, and the LSTM has 0.7213. These results show that many blobs are not predicted the same by the different models.

Overall for the models presented in this thesis, the LSTM based performed best due to the many strange behaviors of the transformer model. These comparison results are also based on the assumption that raft correctly predicts the blobs. With that said the LSTM model did not correctly predict the labels of the blob as it had the same problem as the transformer in classifying a blob as many different labels. This recurring error is likely caused by the problem-solving method of semantic segmentation and the differences in training and experimental data. RAFT, which uses instance segmentation, is a better solution to this problem.

Part V / Conclusions

8.5 Conclusion

A simple Long Short-Term Memory model has been presented, where the test results are given as accuracy and intersection over union. Test results on experimental data for the model are also shown. This model concludes that the model works fine in detecting where the blobs are but has trouble differentiation between them.

The transformer model performance on the synthetic data shows promising results on the segmentation tasks. However, the task of differentiating between the blobs shows less promising results. The results on experimental data on the transformer and Long Short-Term Memory show that the semantic segmentation approach to this problem is probably not the best course of action. Section 8.5.1 will discuss other possible solutions to this task.

For the transformer architecture, the results on fewer blobs were much better than when the intermittency increased. This explains some of the problems the model had with overlapping blobs. This shows that the model works best if there are fewer blobs.

Looking past the problem of differentiating between the blobs, the results were much more promising when discussing the differentiation between blob and background. This section shows that the variation in parameters influences the results of the model, with higher variance decreasing the results of the model.

Measurements to check when the model is not working correctly based on size, amplitude, and velocities are presented. This shows that when the values differ from the training values, the model's accuracy and intersection over union decrease rapidly, but within the reasonable values, the model performs well.

The results on experimental data are presented. When comparing the prediction in shape to Recurrent All-Pairs Field Transforms, the predictions of my models show the detection of blobs in some cases, while others were not the same. There are some major issues in the model when predicting on experimental data. The comparison with values from another paper shows that the model found some deviations from the estimated blob parameter presented.

The choice of the segmentation method could have been different since instance segmentation seems to have all the aspects that this problem requires. The models should not be used as a method for tracking blobs but could be used as a baseline for future work where a different approach is used. The transformer still showed promising results on the smaller data sets, so with more dept, the model could perform better.

Code is available at https://github.com/leanderkirkeland/Master_thesis_Kirkeland. Some of the results use the same calculations, so the same code is used with the change of datasets.

8.5.1 Future work

When looking at the work of others, the most common solution is to use instance segmentation for this problem. This approach makes sense because we are interested in locating and analyzing the blobs. In instance segmentation, we get a distinct tag on each of the detected blobs, and we can target the blobs alone to do further investigation. This method was experimented with in this thesis, but the time limitation made it outside of the scope of this thesis.

Another concern is the training data used in this thesis. It would be a great baseline for pretraining the models for experimental data, but the synthetic data does not represent all of the dynamics of the experimental data. A method of generating synthetic data that resembles the experimental data was tested, but due to time limitations and errors in the program, this had to be scrapped. Future synthetic data generators could include blobs changing propagation velocity, trajectory, and shapes, as well as measurement noise in the data.

The promising results of the transformer architecture should be further explored with more suitable data and problem sets.

References

[1] Z. Teed and J. Deng. “Raft: Recurrent all-pairs field transforms for optical flow”. In: *European conference on computer vision*. Springer. 2020, pp. 402–419.

[2] W. Han et al. “Tracking Blobs in the Turbulent Edge Plasma of Tokamak Fusion Reactors”. In: *arXiv preprint arXiv:2111.08570* (2021).

[3] D. Bodansky. *Nuclear energy: principles, practices, and prospects*. Springer Science & Business Media, 2007.

[4] R. Arnoux. *Which was the first ’tokamak’-or was it ’tokomag’?* Oct. 1970. URL: https://www.iter.org/newsline/55/1194.

[5] I. Friedman. “Deuterium content of natural waters and other substances”. In: *Geochimica et cosmochimica acta* 4.1-2 (1953), pp. 89–103.

[6] R. V. Petrescu et al. “Some basic reactions in nuclear fusion”. In: *American Journal of Engineering and Applied Sciences* 10.3 (2017).

[7] Wikipedia contributors. *Deuterium–tritium fusion — Wikipedia, The Free Encyclopedia*. [Online; accessed 16-May-2022]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Deuterium%E2%80%93tritium_fusion&oldid=1059652014.

[8] K. Haupt. *SEARCHING FOR THE PERFECT SHAPE*. URL: https://www.iter.org/newsline/-/3037.

[9] G. Decristoforo et al. “Numerical turbulence simulations of intermittent fluctuations in the scrape-off layer of magnetized plasmas”. In: *Physics of Plasmas* 28.7 (2021), p. 072301.

[10] C. G. Theiler. *Basic investigation of turbulent structures and blobs of relevance for magnetic fusion plasmas*. Tech. rep. EPFL, 2011.

[11] G. Decristoforo et al. “Blob interactions in 2D scrape-off layer simulations”. In: *Physics of Plasmas* 27.12 (2020), p. 122301.

[12] C. Killer et al. “Plasma filaments in the scrape-off layer of Wendelstein 7-X”. In: *Plasma Physics and Controlled Fusion* 62.8 (2020), p. 085003.

[13] D. A. D’ippolito, J. R. Myra, and S. J. Zweben. “Convective transport by intermittent blob-filaments: Comparison of theory and experiment”. In: *Physics of Plasmas* 18.6 (2011), p. 060501.

[14] O. E. Garcia, N. H. Bian, and W. Fundamenski. “Radial interchange motions of plasma filaments”. In: *Physics of plasmas* 13.8 (2006), p. 082309.

[15] S. I. Krasheninnikov. “On scrape off layer plasma transport”. In: *Physics Letters A* 283.5-6 (2001), pp. 368–370.

[16] R. Kube et al. “Blob sizes and velocities in the Alcator C-Mod scrape-off layer”. In: *Journal of Nuclear Materials* 438 (2013), S505–S508.

[17] H. Hasegawa and S. Ishiguro. “Study of self-consistent particle flows in a plasma blob with particle-in-cell simulations”. In: *Physics of Plasmas* 22.10 (2015), p. 102113.

[18] E. S. Marmar and Alcator C-Mod Group. “The alcator c-mod program”. In: *Fusion science and technology* 51.3 (2007), pp. 261–265.

[19] M. Greenwald et al. “20 years of research on the Alcator C-Mod tokamak”. In: *Physics of Plasmas* 21.11 (2014), p. 110501.

[20] I Cziegler et al. “Fluctuating zonal flows in the I-mode regime in Alcator C-Mod”. In: *Physics of Plasmas* 20.5 (2013), p. 055904.

[21] S. J. Zweben et al. “Edge turbulence imaging in the Alcator C-Mod tokamak”. In: *Physics of Plasmas* 9.5 (2002), pp. 1981–1989.

[22] J. L. Terry et al. “Observations of the turbulence in the scrape-off-layer of Alcator C-Mod and comparisons with simulation”. In: *Physics of Plasmas* 10.5 (2003), pp. 1739–1747.

[23] A. L. Samuel. “Some studies in machine learning using the game of Checkers”. In: *IBM JOURNAL OF RESEARCH AND DEVELOPMENT* (1959), pp. 71–105.

[24] E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[25] F. Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[26] A. Kirillov et al. “Panoptic segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9404–9413.

[27] H. Wang, W. Wang, and J. Liu. “Temporal memory attention for video semantic segmentation”. In: *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2021, pp. 2254–2258.

[28] R. Strudel et al. “Segmenter: Transformer for semantic segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 7262–7272.

[29] O. Ronneberger, P. Fischer, and T. Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[30] R. Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[31] S. Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28 (2015).

[32] K. He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[33] J. Hu et al. “Istr: End-to-end instance segmentation with transformers”. In: *arXiv preprint arXiv:2105.00637* (2021).

[34] C. Kocagil. *TT-SRN: Transformer-based video instance segmentation framework*. Oct. 2021. URL: https://towardsdatascience.com/tt-srn-transformer-based-video-instance-segmentation-framework-part-i-ae9964126ac0.

[35] Z. Li et al. “Panoptic SegFormer: Delving deeper into panoptic segmentation with transformers”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 1280–1289.

[36] Y. Xiong et al. “Upsnet: A unified panoptic segmentation network”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8818–8826.

[37] B. Cheng et al. “Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 12475–12485.

[38] L. Liu et al. “Benchmarking deep learning frameworks: Design considerations, metrics and beyond”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1258–1269.

[39] Md A. Rahman and Y. Wang. “Optimizing intersection-over-union in deep neural networks for image segmentation”. In: *International symposium on visual computing*. Springer. 2016, pp. 234–244.

[40] Wikipedia. *Jaccard index — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Jaccard%20index&oldid=1086247746. [Online; accessed 10-May-2022]. 2022.

[41] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[42] H. Wu and X. Gu. “Towards dropout training for convolutional neural networks”. In: *Neural Networks* 71 (2015), pp. 1–10.

[43] V. Pham et al. “Dropout improves recurrent neural networks for handwriting recognition”. In: *2014 14th international conference on frontiers in handwriting recognition*. IEEE. 2014, pp. 285–290.

[44] S. Bhanja and A. Das. “Impact of data normalization on deep neural network for time series forecasting”. In: *arXiv preprint arXiv:1812.05519* (2018).

[45] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[46] N. Bjorck et al. “Understanding batch normalization”. In: *Advances in neural information processing systems* 31 (2018).

[47] J. L. Ba, J. R. Kiros, and G. E. Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[48] S. Sharma, S. Sharma, and A. Athaiya. “Activation functions in neural networks”. In: *towards data science* 6.12 (2017), pp. 310–316.

[49] J. Han and C. Moraga. “The influence of the sigmoid function parameters on the speed of backpropagation learning”. In: *International workshop on artificial neural networks*. Springer. 1995, pp. 195–201.

[50] V. Nair and G. E. Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Icml*. 2010.

[51] C. Nwankpa et al. “Activation functions: Comparison of trends in practice and research for deep learning”. In: *arXiv preprint arXiv:1811.03378* (2018).

[52] D. Hendrycks and K. Gimpel. “Gaussian error linear units (gelus)”. In: *arXiv preprint arXiv:1606.08415* (2016).

[53] J. Bridle. “Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters”. In: *Advances in neural information processing systems* 2 (1989).

[54] R. C. Staudemeyer and E. R. Morris. “Understanding LSTM–a tutorial into long short-term memory recurrent neural networks”. In: *arXiv preprint arXiv:1909.09586* (2019).

[55] Z. C. Lipton, J. Berkowitz, and C. Elkan. “A critical review of recurrent neural networks for sequence learning”. In: *arXiv preprint arXiv:1506.00019* (2015).

[56] S. Hihi and Y. Bengio. “Hierarchical recurrent neural networks for long-term dependencies”. In: *Advances in neural information processing systems* 8 (1995).

[57] M. C. Mozer. “A focused backpropagation algorithm for temporal”. In: *Backpropagation: Theory, architectures, and applications* 137 (1995).

[58] AJ Robinson and F. Fallside. *The utility driven dynamic error propagation network.* University of Cambridge Department of Engineering Cambridge, 1987.

[59] P. J. Werbos. “Generalization of backpropagation with application to a recurrent gas market model”. In: *Neural networks* 1.4 (1988), pp. 339–356.

[60] F. M. Bianchi et al. “An overview and comparative analysis of recurrent neural networks for short term load forecasting”. In: *arXiv preprint arXiv:1705.04378* (2017).

[61] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[62] R. Pascanu, T. Mikolov, and Y. Bengio. “On the difficulty of training recurrent neural networks”. In: *Proceedings of the 30th International Conference on Machine Learning.* Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1310–1318. URL: https://proceedings.mlr.press/v28/pascanu13.html.

[63] Y. Hu et al. “Overcoming the vanishing gradient problem in plain recurrent networks”. In: *arXiv preprint arXiv:1801.06105* (2018).

[64] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[65] C. Olah. *Understanding LSTM Networks.* https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Online; accessed 10-May-2022]. 2015.

[66] Open Genus. *When to use Recurrent Neural Networks (RNN)?* https://iq.opengenus.org/when-to-use-recurrent-neural-networks-rnn/. [Online; accessed 10-May-2022].

[67] K. Fukushima and S. Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets.* Springer, 1982, pp. 267–285.

[68] O Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[69] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[70] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[71] C. Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 1–9.

[72] A. H. Reynolds. *Convolutional Neural Networks (CNNs)*. https://anhreynolds.com/blogs/cnn.html. [Online; accessed 10-April-2022]. 2019.

[73] R. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[74] K. O'Shea and R. Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[75] Computerscience Wiki. *Max-pooling / Pooling*. https://computersciencewiki.org/index.php/Max-pooling_/_Pooling. [Online; accessed 10-May-2022]. 2018.

[76] A. Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[77] A. Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[78] G. Decristoforo. *2d_propagating_blobs*. https://github.com/uit-cosmo/2d-propagating-blobs.

[79] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[80] P. Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.