University of Bergen
Department of informatics

Deep Learning Methods for Automated Classification of Fish Behavior

Author: Mathias L. Madslien
Supervisors: Ketil Malde (UiB) and Tonje Knutsen Sørdalen (UiA)

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

May, 2022

Abstract

Machine learning is starting to get footholds in marine science and has demonstrated encouraging potential in analyzing ecological data, for example, counting and classifying species from image and video data. This study will investigate the prospects of building a behavior classifier trained on video data from wild nesting sites of the corkwing wrasse, provided by the Institute of Marine Research (IMR). The proposed models are based on 2D and 3D convolutional neural networks (CNN) and recurrent neural networks (RNN) to extract spatial and temporal features to detect and classify behavioral events in noisy surroundings. The behavior classes focused on was the corkwing wrasse's spawning and chasing events. The suggested models do not exceed F1-scores above 0.34, and the analysis implies that the models struggle to differentiate relevant spatial information from noise. The results demonstrate the difficulties of employing deep learning solutions with limited quantities of noisy wild-life data in intricate tasks like action recognition.

Acknowledgments

Firstly, I would like to thank my companions in the reading room, Knut Holager, Hans Martin Johansen, Halvor Barndon Helland, Emir Zamwa, Johanna Jøsang, John Isak Villanger. A special thanks to my fellow sandwich creators, each day striving to move the boundaries for what two slices of bread can be elevated to.

A big thanks to my supervisors Ketil Malde and Tonje Knutsen Sørdalen, for invaluable advice and guidance throughout the work. I would also like to thank my girlfriend, family, and friends.

Mathias L. Madslien
May, 2022

Contents

List of Figures

List of Tables

1 | Introduction

1.1 Background

In animal ecology, population models that estimate abundances rely on capture-mark-recapture techniques (CMR), where individuals are tagged with a readable identifier (e.g., pit tags, collars, tattoos). This means that animals need to be caught and handled, which can cause stress to the animal and cause changes to their natural behavior, which could result in unreliable data [18]. Besides monitoring the abundance of species for healthy marine ecosystems, marine scientists also need to adequately understand the behavior and reproduction habits and how they may change under different circumstances [36]. Climate changes and increasing water temperatures will inevitability lead to behavioral responses of the inhabitants [36]. This can for example cause species to slow down swimming speed and increase their susceptibility to being eaten. Additionally, the most significant increase in water temperature changes will occur in the shallow waters, where nesting sites of many commercially and ecologically important fish species are based [36]. Behavior linked to reproduction is of particular concern because of the direct link to population recruitment [44]. Tracking species' behavioral patterns over different temperature scales can therefore improve our understanding of how climate changes will affect reproduction and the overall population abundance.

The need to monitor the ocean more closely has led to an increase in underwater video and image data usage, which emphasizes the need for efficient analysis techniques [18]. The evolution of camera technologies and increasing magnitudes of data have opened for applications of machine learning and computer-vision methods in marine science [37]. Such solutions could automate the analysis of ecological data and restrict the need for human interference during data collection. Especially the use of spatial deep learning methods for image data has increased and proved to be very effective in tasks like species classification in natural environments [50, 59]. Nevertheless, similar applications for video analysis are not that common yet. Videos are a primal source of ecological data; thus, reliable systems to efficiently process and analyze video data could mitigate the need for manual processing and increase performance [28]. However, the classification of video data is more demanding than still images since videos require more computational resources and have visual variations within each data sample. In addition to spatial features, video classification models usually need to capture temporal dynamics during several frames to extract discriminative features, elevating the task complexity compared to images.

However, the employment of machine learning methods for such comprehensive tasks demands sufficient amounts of high-quality, annotated data to develop robust and reliable models. Annotation work is expensive and tedious, often requiring taxonomic experts. Despite high-quality programs committed to annotation work, this process is still a bottleneck in the advancements toward automation [35].

1.2 Challenges

Action recognition of fish behavior will intuitively be a more demanding task than human actions since fish does not have as distinct poses as humans do. Also, external factors like lightning and colors are dependent on water turbidity, weather, and depth which might increase variations in the surrounding saturation and hue [40]. These factors will seemingly enhance the importance of extracting temporal dynamics, for example, the changing orientation of fish throughout time. Moreover, such variations may increase the data quantities required to contextualize the videos' relevant information, independent of unrelated environmental factors.

1.3 Research Question

Millions of wild-caught wrasses are harvested and transported to Norwegian salmon pens yearly [1]. Sustainability of this fishery requires well-adjusted management regulations on the minimum and maximum sizes, technical regulations of fishing gear, and catch quotas, which demand the need for knowledge and research in population dynamics; population growth, population sizes, and the underlying reproduction and mating system [19, 27]. This thesis will explore the prospects for automating the behavior classification of corkwing wrasse (*Symphodus melops*) from video data with spatio-temporal deep learning algorithms. Corkwing wrasse is a fish species habitual in shallow waters and selectively harvested in Norwegian aquaculture as a cleaner fish to reduce salmon lice infestations [19]. Environmental changes can affect the behavior of individuals and subsequently the sustainability of harvesting from wrasse populations. Thus, automation of behavioral analysis can help understand and predict how future changes will impact corkwing wrasses' sexual selection and reproduction, and how fisheries should abide by this [27]. The data consist of a video catalog provided by the Institute of Marine Research (IMR), obtained from nesting sites of corkwing wrasse, collected during nesting periods 2018-2020 outside Austevoll of the west coast of Norway. In addition to the video data, annotations are provided to describes the behavioral events taking place in the videos [15], employed to train a behavior classifier.

The main objective of this thesis was to explore if deep learning models can extract relevant features to detect and discriminate behavioral events of a wild fish species, despite the videos involving a considerable amount of natural noise. The aim was to process the data to be employable in deep neural networks and develop a behavior classifier model for the wild corkwing wrasse. The videos were divided into shorter sequences used as input, and then the proposed models were trained and tested to explore if they were capable of recognizing behavior events occurring throughout the streams.

Research question: Are spatio-temporal deep neural network models able to capture relevant features from the provided underwater video dataset to detect and accurately classify behavioral events?

1.4 Thesis Outline

Theoretical Background Formulates technical background theory related to machine learning fundamentals and relevant deep learning methods.

Related Works Contains a brief review of relevant literature to the study.

Materials and Methods Explain the materials and methodical approach and considerations taken throughout the work.

Results Provide main findings of the work significant to our research question.

Discussion Interprets the findings from the result chapter and addresses the implications and limitations of the work.

Further Works and Conclusion Describes the actions that can be done further in light of this research, and summarizes how the main findings answered the research question.

2 | Theoretical Background

2.1 Machine Learning Basics

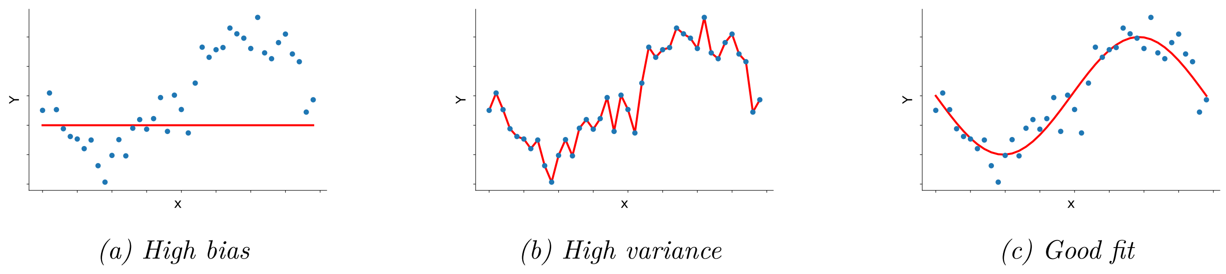
Machine Learning (ML) is a branch of Artificial Intelligence (AI) where the computer learns to make predictions or decisions from data and statistics. Compared to fixed human-written algorithms, machine learning algorithms learn from experience by processing historical data [17]. Such methods open up great potential for solving new types of tasks by making decisions in a human-like manner without being explicitly programmed to.

Machine Learning tends to be categorized as either supervised or unsupervised learning tasks. Both will learn from data features described by the input, but supervised learning algorithms will also process target values corresponding to every data point, acting like a supervisor striving to reduce the discrepancy between predictions and targets. The main objective for supervised machine learning algorithms is to learn a predictive function $\hat{f}$ that should be able to make accurate estimates $\hat{y}$ for unseen data. Supervised learning tasks are often used to solve classification and regression tasks, where linear regression and K-nearest neighbor are examples of supervised learning algorithms.

An unsupervised learning algorithm does not learn from target values but instead looks for statistical patterns in the data without guidance from target values. Clustering or density estimation are examples of typical unsupervised learning tasks. Unsupervised learning algorithms can be beneficial for analyzing unknown structures of data. Still, results from unsupervised methods may not be as easy to interpret as supervised since there are no provided target values to compare with. Moreover, the mitigation of manual labeling is favorable to unsupervised methods.

2.1.1 Bias-Variance Trade-off

One of the biggest challenges of ML models is to build a model with both accuracy and flexibility. The main objective for a model is a good performance on unobserved input; hence it is said to generalize well [17, p. 110]. Data are usually split into three distinct datasets to ensure reliable optimization and evaluation. A model is optimized (trained) on a fixed subset containing most of the observations, referred to as the **training set**. The remaining data separated from training are split into two partitions called the **validation-** and **test set**. The validation set's utilization is to evaluate our model during training, which helps configure hyperparameters and keep track of progression. The test set is also used for evaluation, but it is not applied until the training phase is completed to provide an unbiased estimate of the model performance. Throughout the training, a model is optimized to reduce the error of predictions on training data samples, referred to as the **training error**. If ML tasks did not have generalization as an objective, there could simply be trained a complex model as possible to minimize the training error. However, a low training error does not necessarily represent good generalization abilities. The measurement of generalization is estimated by the **generalization error**, which computes the model's performance on the test, supposed to resemble the model's capability on unseen real-world data.



(a) High bias *(b) High variance* *(c) Good fit*

Figure 2.1: Model fits (red line) on the data.

The main objectives for a machine learning algorithm are to minimize the training error and simultaneously keep the distance between training and generalization error small. This brings us to the challenges of **underfitting** and **overfitting**. When the training error gets low, the model learns a fit very close to the data points for a particular dataset. Consequently, it might perform worse on unseen data points and is then overfitted to the training data. **Variance** can be viewed as the amount of how the model will alter due to changes in the training data [17, p. 129]. In a model with high variance (figure 2.1b), there is likely to be a considerable gap between training- and generalization error; thus, the model is overfitted to the training set. Underfitting occurs when a model fails to capture the structural trends in the training data and will not be able to reach a low training error. **Bias** refers to the deviation measure of the model predictions and target values [17, p. 129]. A model with high bias (figure 2.1b) will systematically make wrong assumptions and underfit. The key to reducing risks of overfitting and underfitting is to choose a model with suitable capacity with respect to the complexity of the task and data quantity [17, p. 112]. More complex models like deep neural networks will have an increased risk of overfitting, while a simpler model like linear regression is more prone to underfitting.

2.2 Feed-forward Neural Networks and Gradient-Based Learning

Artificial Neural Networks (ANN) are the base of all deep learning algorithms and are loosely related to the structure of biological brains. By combining multiple connected functions, they can recognize complex structures and patterns to make predictions. This section will describe how such networks process data and learn from it.

2.2.1 Multilayer Perceptron

Multilayer Perceptron (MLP) (also called Feed-forward Neural Networks) is one of the most basic examples of ANNs. An MLP is simply a composition of layers forming a directed acyclic graph (figure 2.3), each layer consisting of several non-linear functions, referred to as **neurons**. The composition of neurons and layers allows MLPs to learn complex representations of the input data [33]. Each neuron activation h can be computed by the non-linear activation of a weighted sum of input x:

$$h = a(\sum_{j}^{J} w_j x_j + b) \tag{2.1}$$

where a is the activation function applied element-wise, w weights, x the input, and b the constant bias.

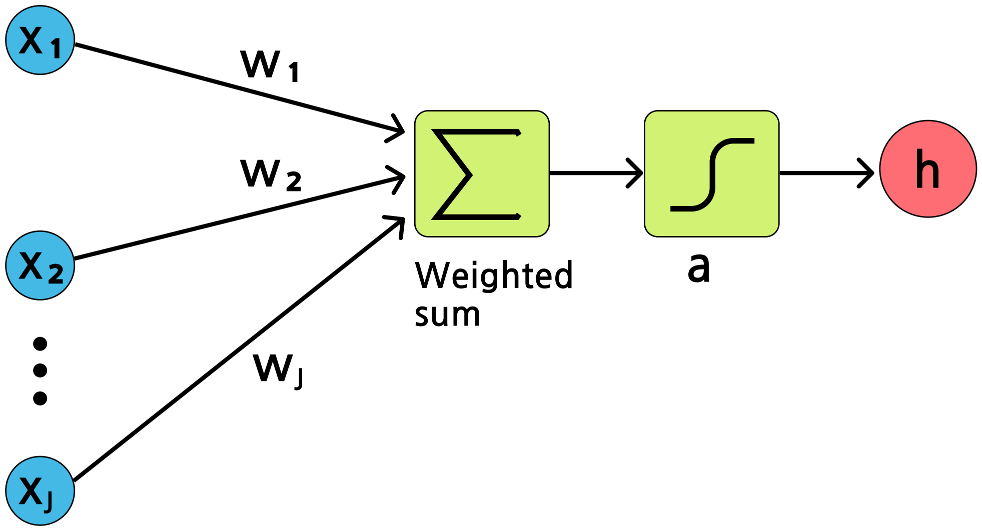


Figure 2.2: Computation of neuron output h (Equation 2.1).

Hence, the input of the intermediate layer n will consist of $\mathbf{h}^{n-1}$ which is the vector of neuron activations computed in the previous layer $n-1$.

The types of layers are distinguished into **input-**, **hidden-**, and **output layers**. Firstly, data are passed into the input layer. Each input neuron corresponds to a feature, then proceeds to the stack of hidden layers responsible for finding the complex patterns and abstractions by computing multiple non-linear functions of the input. Then the abstract representation from the last hidden layer is further introduced to the output layer. The representation is then transformed into the desired output format, for example, probability distributions suitable for classification tasks. The number of hidden layers defines the depth of an MLP, and a deeper network may produce more abstract features and representations.

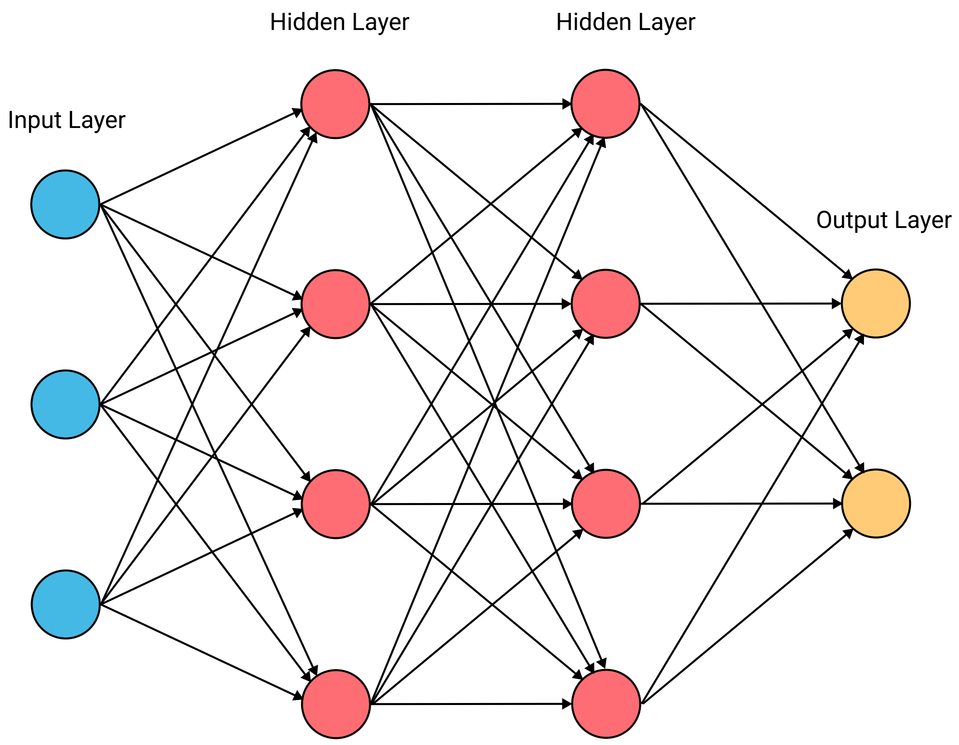


Figure 2.3: MLP with two hidden layers.

The computation of output $\hat{\mathbf{y}}$ for the network illustrated in figure 2.3, given input $\mathbf{x}$ can be calculated as:

$$\begin{aligned}\mathbf{h}^{(1)} &= a^{(1)}(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)})\\ \mathbf{h}^{(2)} &= a^{(2)}(\mathbf{W}^{(2)T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})\\ \hat{\mathbf{y}} &= a^{(3)}(\mathbf{W}^{(3)T}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})\end{aligned}$$

where $\mathbf{h}$ is the hidden neuron activations, a the activation function, $\mathbf{W}$ the matrix of weight parameters, and $\mathbf{b}$ the bias parameters.

No matter the depth, an MLP composed only of linear functions would not represent more than a linear mapping of input to output. For this reason, **activation functions** are introduced to provide non-linearity to the neuron outputs and enable the MLP to solve more complex tasks. Without any activation function in the hidden layers, a network would be unable to represent non-linear functions like XOR, as illustrated by Goodfellow et al. [17, p. 171]. Several types of activation functions are used, but generally, some desired properties are low computational cost, differentiability, and zero-centered, but this depends on the task at hand. The three most known and applied activation functions are Rectified linear unit (ReLU), the Logistic function(σ), and Tanh.

$$\mathrm{ReLU}(z) = max(0, z)$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
$$\mathrm{Tanh}(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

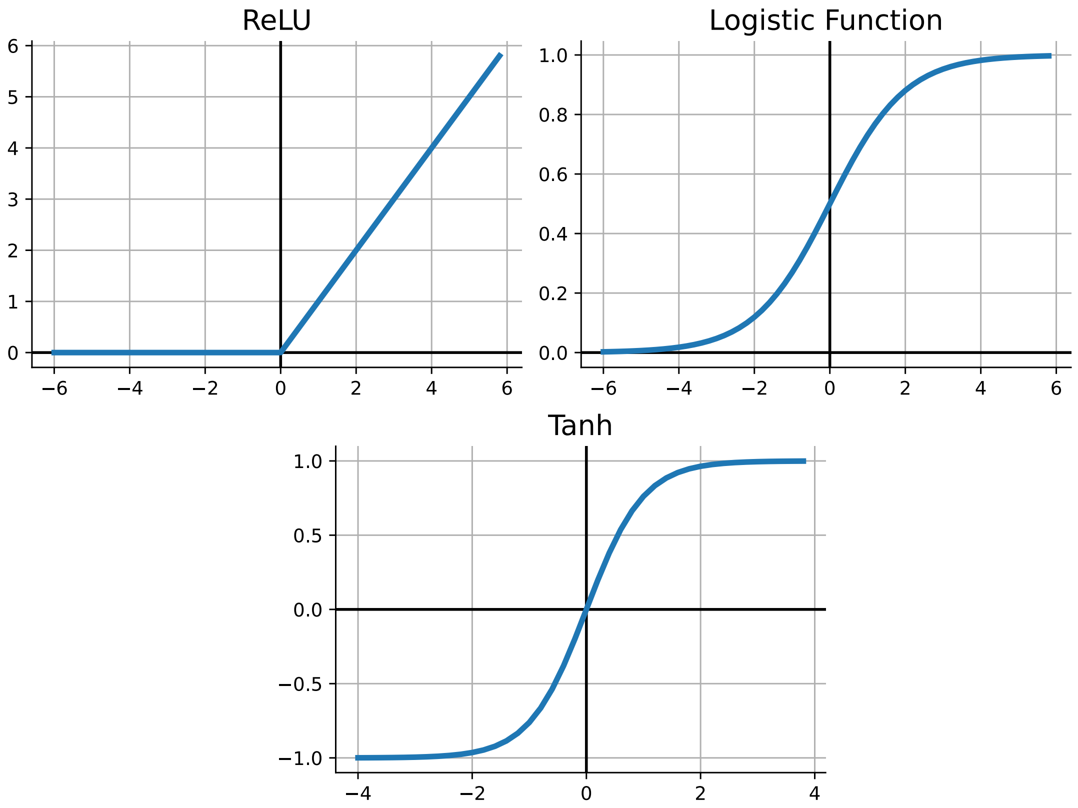


Figure 2.4: Visualization of activation functions.

While activation functions in the intermediate layers control the model's ability to learn patterns from the training data, the activation in the output layer determines what form of outputs the network can produce. An output activation is not necessary for regression tasks, but in classification, it is required to enforce probability outputs for each class. The **softmax** activation is used for multiclass classification tasks. Softmax is a soft version of the argmax[1] function, returning a probability distribution over a discrete variable with multiple possible values [17, p. 184]:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.2}$$

where $\text{softmax} : \mathbb{R} \to (0, 1)$ and $\sum_i \text{softmax}(z_i) = 1$.

On the other hand, the logistic function is most applied as output activation for binary- and multilabel classification, where the output units are binary.

2.2.2 Optimizing a Neural Network

Deep learning methods aim to compute $\hat{y} = f(x, w)$, where $\hat{y}$ is an output given input x, represented by the set of tunable parameters w, which refers to the tunable weights and biases. For simplicity, we may further ignore the biases and refer to the parameters just as the weights. Input x corresponds to a set of features, and in classification tasks, $\hat{y}$ will be class probabilities, while for regression tasks, $\hat{y}$ is a continuous variable. Neural networks are initially assigned with random parameters. To enhance predictions, they have to be optimized to minimize a continuous function $L(\hat{y}, w)$, referred to as the **loss function**. $L(\hat{y}, w)$ measures the discrepancy between predictions and actual values for the set of parameters p [31], known as the loss. Through the training process, p will move towards the set of values minimizing $L(\hat{y})$ for the training data. A low training loss does not necessarily imply a low generalization error. Since the parameters are optimized with respect to the training data, the model may need refinement concerning the bias-variance trade-off (section 2.1.1) to improve generalization. The two most commonly used loss functions are Mean Squared

[1] Argmax is a mathematical operation that retrieves the argument returning the maximum value of a target function.

Error (MSE) (equation 2.3), and Cross-Entropy (CE) (equation 2.4), respectively used for regression, and classification tasks:

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 \tag{2.3}$$

$$CE = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{C} y_{ij} \cdot \log(\hat{y}_{ij}) \tag{2.4}$$

where N is the number of data points, C is the number of classes, $\hat{y}_i$ and y_i is predicted and ground-truth values for sample i, while $\hat{y}_{ij}$ and y_{ij} is predicted and ground truth values of sample i of class j.

Deep learning models can contain millions of parameters that need adjustments for the model to close in towards an optimum. Parameter alterations are achieved by computing its gradients with respect to the loss function. The negative gradient of a parameter states which direction to alter the parameter to minimize the loss [33]. The gradients are defined as the partial derivative of the loss function L with respect to the parameters w, $\frac{\partial L(w)}{\partial w}$.

The procedure of iteratively updating the parameters according to the gradients towards a minimum is called **gradient descent** (equation 2.5). For high-magnitude gradients, there will be larger training steps. Usually, the training steps tend to get smaller when getting closer to a minimum. The update step of parameters w at time k is defined as:

$$w_k = w_{k-1} - \epsilon\frac{\partial L(w)}{\partial w} \tag{2.5}$$

where ϵ is a training constant called **learning rate** determining the step size moving toward the global minimum of the loss function. The choice of learning rate is essential for a smooth training process, where a too large learning rate can lead to unstable training and convergence to suboptimal solutions. Nevertheless, a too small learning rate can lead to slow training and stagnation at local minimums. Due to inefficiency, it is usually not preferable to compute the gradients after every successive data point is processed. Instead, the most common approach is to sample a random mini batch from the training set and compute the mean gradient, followed by a corresponding weight update. This process is called **mini-batch gradient descent** and will give a noisy estimate of gradient values [33]. The batch size is a tunable hyperparameter that can affect performance and training time [26]. When batch size equals one, it is usually referred to as **Stochastic Gradient Descent (SGD)**.

There are usually applied additional algorithms extending gradient descent like **momentum** and **Root Mean Square Prop (RMSProp)** to accelerate training. Momentum aggregates an exponential moving average of current and previous gradients and is used to determine the update step's direction. RMSProp also exploits the exponential moving average, but rather than adjusting direction, it applies an adaptive learning rate for each parameter to increase convergence abilities. Both momentum and RMSProp are methods to help reduce oscillations towards an optimum [17, p. 296].

2.2.3 Back-propagation

The process of propagating an input $\mathbf{x}$ through a neural network to produce output $\hat{y}$ is called **forward-propagation**. However, to compute the gradients, the information from the loss needs to flow backward through all previous layers. By changing the parameters in one layer, the effect on the loss is only dependent on how it affects the next layer and how changes in this next layer affect the loss. Because of this, we can apply the chain rule from calculus to compute the gradients of each layer efficiently. This neat application of the chain rule to calculate gradients is called **back-propagation**. Back-propagation is essential for efficiently optimizing deep learning models [17, p. 204].

The efficiency of back-propagation can be illustrated by computing $\frac{\partial y}{\partial x}$ from the computational graph in figure 2.5. Equation 2.8 shows the approach taken by back-propagation, where $f(x)$ is computed once and stored in subsequent variables. Compared to the naive approach in equation 2.9, $f(x)$ is recomputed multiple times, which is viable but tedious.

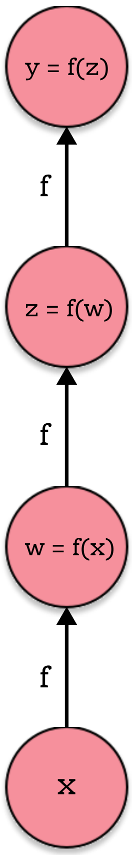


Figure 2.5: A computational graph where the same function f is applied at every computation step. For simplicity, this is not a neural net with activations and weights but just a graph to illustrate the efficiency of the chain rule.

$$
\begin{aligned}
&\frac{\partial y}{\partial x} && (2.6)\\
=&\frac{\partial y}{\partial z}\frac{\partial z}{\partial w}\frac{\partial w}{\partial x} && (2.7)\\
=&f'(z)f'(w)f'(x) && (2.8)\\
=&f'(f(f(x)))f'(f(x))f'(x) && (2.9)
\end{aligned}
$$

This back-propagation example from Goodfellow et al. [17, p. 211] is a little simplified compared to how it is done in MLPs, where the gradient of the loss function is computed with respect to the parameters recursively backward from the output to the first layer via intermediate hidden layers containing activation functions.

2.2.4 Vanishing and Exploding Gradients

Due to the nature of the back-propagation algorithm, the use of chain rule by computing derivatives of layers deep into the network leads to numerous matrix multiplications. Consequently, if products of derivatives get elevated, the gradients may exponentially raise and lead to what is called **exploding gradients**. This may lead to a highly unstable model or even numerical overflow and invalid weights. Similarly, the problem of **vanishing gradients** occurs if the derivatives become very small and exponentially decrease, which may lead to insignificant values close to zero [41].

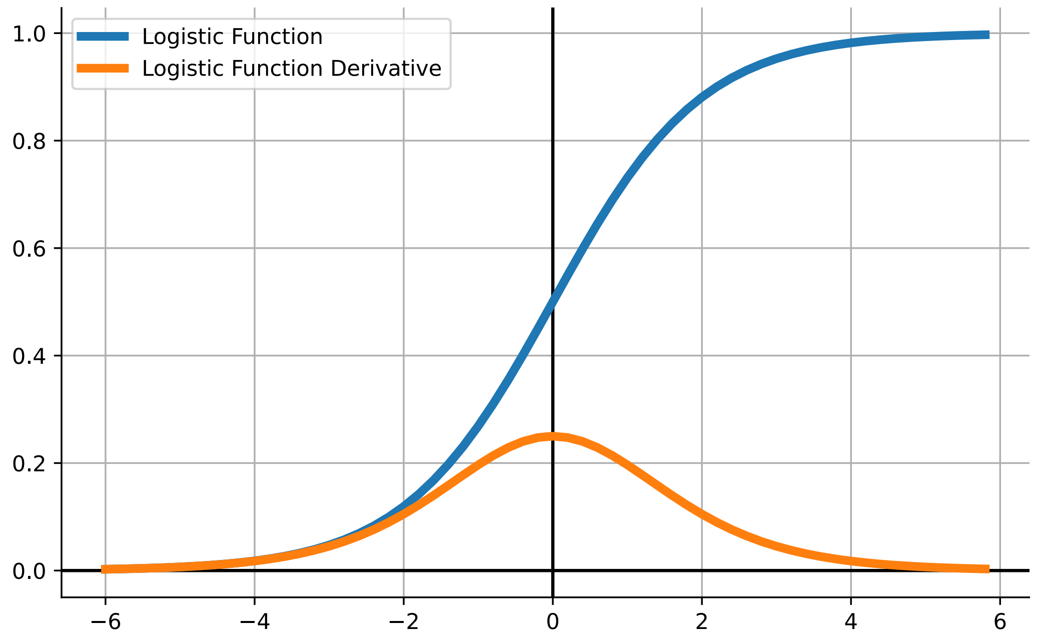


Figure 2.6: Logistic Function derived.

A choice of non-saturating activation functions is a simple step to reduce the risk of vanishing gradients. To illustrate this, figure 2.6 shows that sigmoid suffers from saturation towards zero and are prone to vanishing gradients. For this reason, ReLU is more used due to a derivative of one for all positive values. There are several methods to address vanishing and exploding gradients, for example, gradient clipping and careful weight initialization described further by Goodfellow et al. [17, p. 301], in addition to batch normalization (section 2.2.5) and skip-connections (section 2.3.2) described later in the thesis.

2.2.5 Regularization Methods

While there are several definitions of the term **regularization**, it is most widely known as a technique doing modifications to a learning algorithm to enhance generalization [17, p. 118]. Regularization helps avoid overfitting due to reduced variance and is more likely to perform better on unseen data.

Two of the most common regularization techniques are **L1** and **L2**, which modifies the loss function by adding a term independent of the target values, penalizing the parameters by their norm. A regularized loss function $L'(w, x, y)$ is denoted as:

$$L'(w, x, y) = L(w, x, y) + \lambda R(w) \tag{2.10}$$

where w is trainable parameters, x is the input, and y is the target values. λ is a tunable parameter weighing the significance of the regularization term.

The regularization term for L2 is given by:

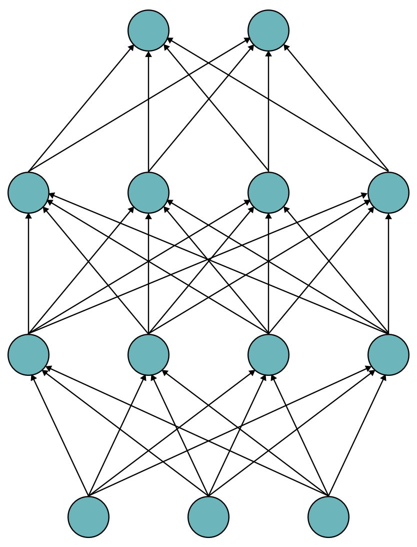
$$R(w) = \sum_{i=1}^{n} w_i^2 \tag{2.11}$$

and L1:

$$R(w) = \sum_{i=1}^{n} |w_i| \tag{2.12}$$

The main difference between the two is that L2 penalizes larger weights more and does not affect smaller weights as much. L1 regularization coefficients are more likely to be sparse compared to L2. Hence, L1 is handy when performing feature selection [17, p. 236], but L2 is generally favorable otherwise.

Dropout is another widely used regularization method introduced in 2014 by Srivastava et al. [53]. Noise is introduced to the training process by omitting a number of units temporarily for a given layer during training. A different set of units will be dropped for every iteration given by a probability, referred to as the dropout rate. Thus, neurons will have different influences on the output for each iteration. By random sub-sampling of layer outputs, the model's capacity will diminish, thus also reducing overfitting. At test time, all units are kept to provide accurate and consistent outputs.


(a) Regular Neural Network.

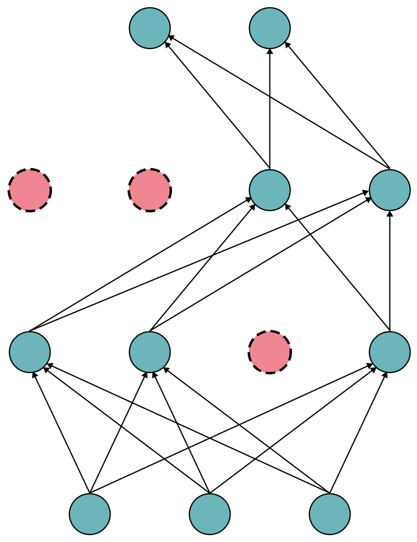

(b) Neural Network with dropout.

Figure 2.7: Dropout example.

In addition to the regularization methods mentioned above, **batch normalization** was introduced in 2015 by Ioffe and Szegedy [23]. Batch normalization is not directly a regularization method but has some regularizing effects. By standardizing the distribution of the network's internal activations, the internal covariate shift of the network will be reduced. By limiting the range of hidden units, deeper layers in particular will become more stable since there will be limitations on how changes in earlier layers can affect the later ones. This leads to a more stable and faster training and might reduce the importance of other regularization techniques like dropout and L2. Normalized outputs for the internal activations z, can be defined as:

$$z^{\text{Normalized}} = \left(\frac{z - mean_z}{sd_z}\right) \tag{2.13}$$

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a specific type of neural network primarily used to handle image data. The main difference from regular feed-forward neural networks is the introduction of **convolutional layers**, which extracts high-dimensional features from the input while maintaining spatial integrity [31]. As the name implies, convolutional layers exploit the mathematical operation **convolution** rather than general matrix multiplication like regular fully connected layers. Convolutions multiply a learned set of weights with local spatial patches of the input to obtain visual features like edges, corners, or colors [31]. A convolutional layer contains one or multiple **filters** containing the learned weights that encapsulate what features the network is looking for. The output from convolutional layers is called **feature maps** and is obtained from the learned filters sweeping over the input and computing local weighted sums. All convolutional layers have a unique set of filters, and extracted feature maps are further fed through subsequent layers to extract even higher-order features. CNN input starts as raw pixel values, and the learned features from the first layer could typically represent edges, while the next layer would spot arrangements of the edges. Hence, the subsequent layers could arrange the combinations of edges into familiar objects and arrange them into meaningful features for the specific task [33].

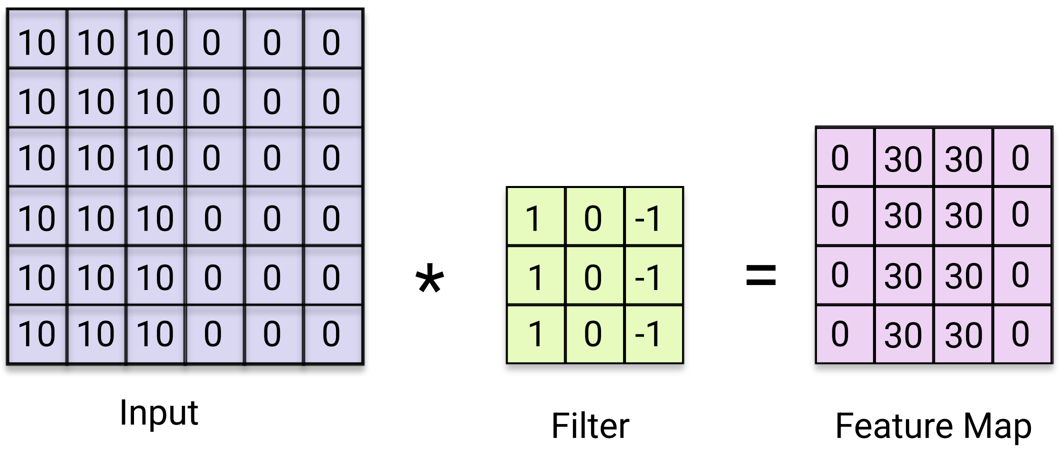


Figure 2.8: Example of convolution (denoted as $*$) with an edge detection filter.

Essentially, the purpose CNNs is to transform the input to a more processable shape and similarly maintain essential features. Hence, the number of spatial dimensions will decrease deeper in the network while the number of channels increases. **Pooling layers** are introduced to reduce the parameters of the feature maps further. As a standard component of CNNs, pooling layers summarize features produced by the convolutional layers. This produces feature maps with reduced parameters, which are also more robust to variations of spatial location in previous layers [32]. In contrast to convolutional operations, pooling is a fixed function and not learned. Two of the most commonly used pooling operations are **average pooling** and **max pooling**, which correspondingly computes the average or maximum values of all patches in the given feature map.

After the input is processed through all convolutional- and pooling layers, we remain with a set of feature maps that are further processed into one or multiple fully connected layers to produce the desired output. A factor making convolutions advantageous for image data is that local properties in an image are invariant to location. Thus, a vertical edge can be detected in different image parts utilizing the same set of parameters. Another advantage is that each convolution output value only depends on a local patch of the input compared to ANNs, where the output will be connected to all input values. Both these factors lead to a significant reduction of model parameters and make it efficient for image processing [31].

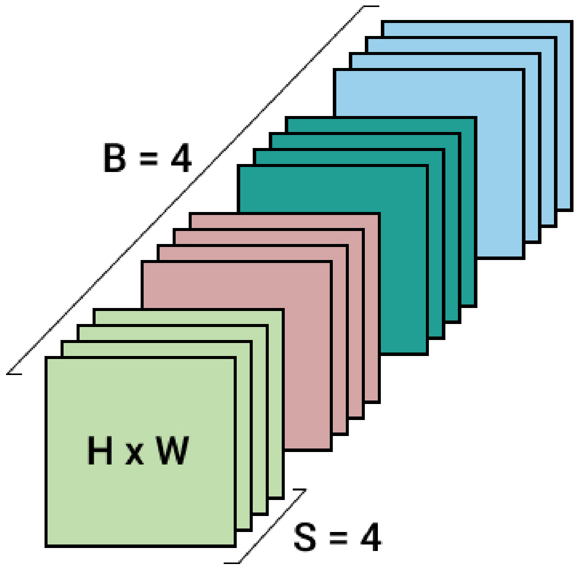


Figure 2.9: Video input to a convolutional neural network. The colors represent batches.

A convolution input is usually composed of a batch of images. In this work, we will work with videos; thus, the input will be slightly different. Instead, the input is made from a batch of image sequences. The shape of the input can be defined as $I = (B, S, C, H, W)$, where B represents the batch size, S the sequence length, C is the number of channels in the image, which would be three for an RGB image, H and W is the height and width of the frames. An example of a video input is illustrated in figure 2.9.

2.3.1 3D Convolutions

The convolution operation explained in section 2.3 is a 2D convolution only computing features on the two spatial dimensions. The 2D convolutions can be extended to 3D convolutions, where a three-dimensional filter is applied in the same way as for 2D convolutions but sliding across three dimensions of the input to produce a volume of output features (figure 2.10). 3D convolutions can be effective when dealing with multidimensional data, for example videos where 3D convolutions will compute both spatial and temporal features [24].

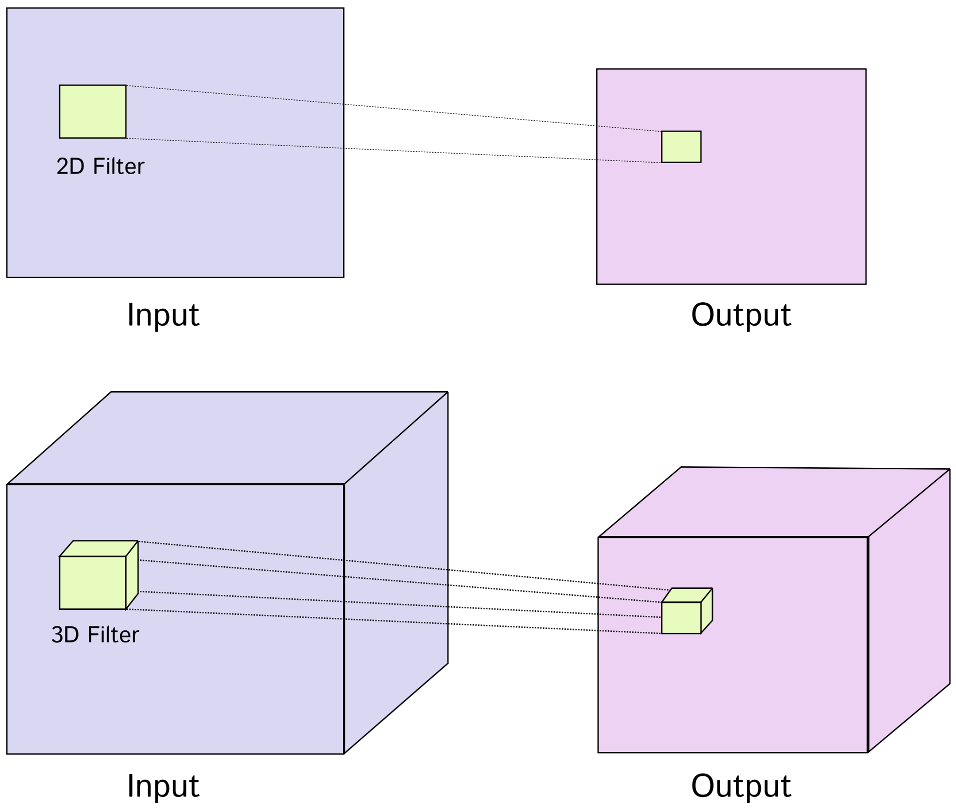


Figure 2.10: 2D (top) and 3D convolution (bottom).

2.3.2 Residual Networks

The number of stacked layers defines the depth of a neural network. Deeper networks lead to the extraction of higher-level features and can solve more complex tasks. However, it is harder to make a deeper network converge due to the problems of vanishing and exploding gradients (section 2.2.4). For too deep networks, the performance tends to saturate and degrade at some point, making them perform worse than shallower ones [21].

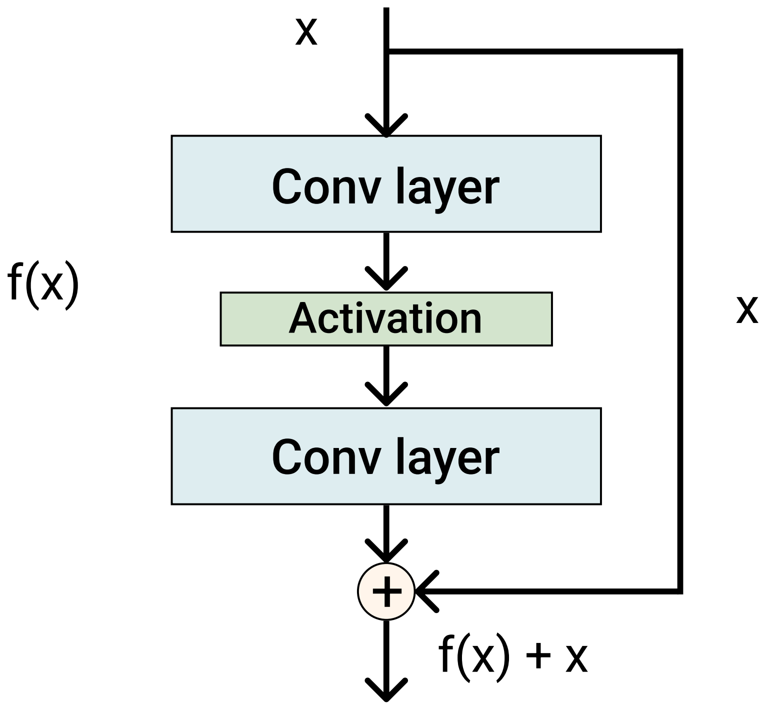


Figure 2.11: Residual block.

To overcome the challenge of training deeper CNNs, He et al. [21] proposed Residual Networks (ResNet), utilizing **skip-connections**. Skip-connections are a direct path from the input of a block of layers directly to the output, forming a residual block, as shown in figure 2.11. Skip-connections allow gradients to flow easier through the network by adding the opportunity to skip blocks of layers and avoid non-linear activation functions if they weaken performance [21]. The skip-connections are an identity mapping of the input x. Thus, if the desired mapping of output is x, the parameters of $f(x)$ are pushed towards zero, and the identity mapping will propagate x to the output and preserve information from previous layers [20]. Consequently, the choice of depth is less important to configure for ResNets since skip connections allow the network to pass over irrelevant layers if beneficial. The extension of skip connections has been vital and has entitled training deeper networks.

2.4 Recurrent Neural Networks

Another branch of deep learning is Recurrent Neural Networks (RNN), which provides a neat method for handling temporal dynamics in sequential inputs, like speech recognition or video classification. The main benefit of RNNs over regular feed-forward networks is that RNN models can store contextual information about previous sequence elements and compress it to a low dimensional space [39]. In this way, the RNN will always have an internal state represented as neurons that store information about the history which will be fed into the network as an input accompanied by all subsequent sequence elements. The current state at time step t can be stated as a function $h_t = f(h_{t-1}, x_t)$ of the input x_t and the previous state h_{t-1}.

Similar to CNNs, RNNs also practice parameter sharing by using equivalent weight matrices independent of the time step. This drastically reduces the number of parameters compared to using MLPs with their own set of weights for each time step. There are several variations of RNN structures depending on the number of inputs processed and outputs generated. Figure 2.12 illustrates a so-called many-to-many RNN, where an input sequence is to produce a sequential output.

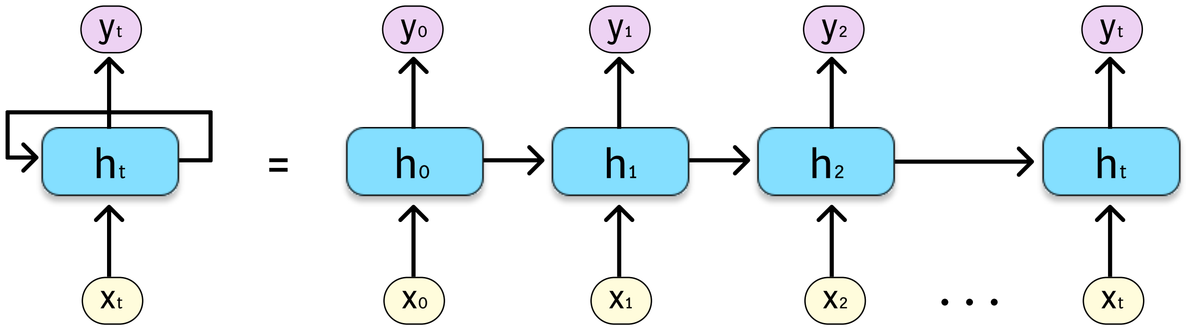


Figure 2.12: Unfolding of a Many-to-many Recurrent Neural Network.

The loss of an RNN output is simply computed by accumulating the loss at all time steps. Moreover, since all computations depend on the previous time steps, we cannot simply apply the standard back-propagation algorithm (section 2.2.3) right away. Firstly, the computational graph has to be unfolded for all time steps, then the unrolled graph is back-propagated, and the weights are updated on the up-rolled network [17, p. 384]. This approach to calculating gradients for sequential models is referred to as Backpropagation Through Time (BPTT).

In many tasks such as translation and speech recognition, the output is dependent on the entire input sequence to make predictions on an adequate basis. For instance, in translation tasks where the ordering of words varies for the different languages, a model would need the whole sentence context to get the correct interpretation necessary to translate it correctly. So far, we have only looked at RNNs that restrict their hidden state to information about the past and up to the current time step but not further. Schuster and Paliwal [45] introduced the **Bidirectional Recurrent Neural Networks (BRNN)** to utilize forward sequence information in addition to the past. The principle of BRNNs is that by splitting the states in both a positive and negative time direction, the positive directed state is computed by feeding the sequence as a regular RNN, and the negative directed state is computed by feeding the sequence in reverse [45]. This enables BRNN models to exploit both previous and future context to make predictions. The information acquired from each direction is usually combined into one state, summarizing the contextual information from both directions. One of the drawbacks of BRNN is that the entire sequence of data is required before any processing can be done, which can be challenging when employing it in real-time applications.

Despite RNNs ability to learn contextual information, difficulties arise when long-term sequential dependencies occur. The computations of gradients in BPTT involve multiplying weight matrices with itself for every time step since every previous weight calculation will affect the current output [54]. This makes RNNs prone to the vanishing and exploding gradients problem (section 2.2.4). Thus, when gradients become small, information longer in the past will be more insignificant. Hence, we can say RNNs has a sound short-term memory, but the long-term memory does not satisfy the demands for tasks containing dependencies ranging longer timespans.

2.4.1 Long Short-Term Memory

With RNNs limitations of learning long-term dynamics in mind, **Long Short-Term Memory (LSTM)** was introduced by Hochreiter and Schmidhuber [22] in 1997 and has proven a great success for complex tasks in domains like machine translation and speech recognition, which RNNs struggle to solve. LSTM solves the vanishing and exploding gradient problem by introducing memory cells and gate units, ensuring a more sustainable error flow for longer distances. The advantage of different gated units is controlling the information flow, striving to constrict the stored information to be relevant, and avoiding ambiguity.

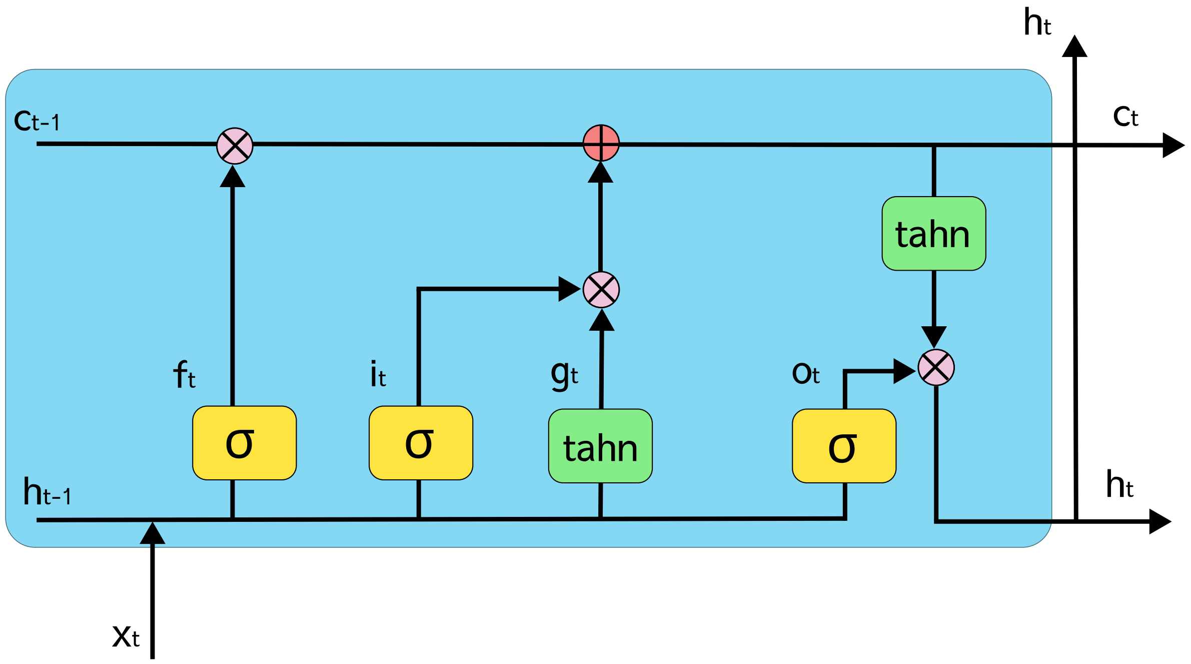


Figure 2.13: LSTM Cell.

There are used several names and notations for the different gate units, but we will refer to them as the following, input-, forget-, output-, and gate gate notated as i_t, f_t, o_t, and g_t. The gate processing are computed by the following functions:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.14}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.15}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.16}$$

$$g_t = tahn(W_g x_t + U_g h_{t-1} + b_g) \tag{2.17}$$

where W and U learned weights for the different gates, b is the gate biases, x_t is the input at time step t, and σ refers to the logistic function (2.2.1). Then the cell state c_t and hidden state h_t are computed by the functions:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2.18}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.19}$$

The cell state is not to be confused with the hidden state, as the cell state works as the flowing memory of the network, while the hidden state is the output activation of each LSTM unit and is concatenated with the next LSTM unit's input. All the gates learn a set of parameters suited for their corresponding task. The forget gate determines what information is relevant in the current cell state. The input gate computes what new input information should be stored in memory, then elementwise multiplied with the gate gate activations that operate like candidate values of what to add to the cell state. The tanh activation ensures that the candidate values can be either negative or positive, allowing the model to increase or decrease the values in the cell state. Negative values would not be possible exclusively with a logistic activation function. The role of the output gate is to calculate the next hidden state bypassing the input and previous hidden state into a logistic function and extracting information from the updated cell state.

By employing gate units and memory cells that seamlessly work together, LSTMs can retain long-term dependencies and lessens the problem of vanishing and exploding gradients compared to regular RNNs.

2.5 Transfer Learning

Solving computer vision and Natural Language Processing (NLP) tasks necessitates a high standard of data and computational power. For this reason, transfer learning can be employed, which works by transferring information from similar solved problems to another domain [60]. This can be illustrated in object detection tasks, where CNN-based models extract relevant features to recognize entities. In most object-detection models, the first few convolutional layers will search for edges and low-level features pertinent across multiple domains. This can be exploited by transferring pre-learned filters from trained object detectors to new tasks. Usually, the earlier layers, in particular, benefit from pretrained weights, and the subsequent layers can be tuned to fit the relevant domain. Transfer learning can lead to better performance despite the lack of large data quantities and faster training compared to models trained end to end. However, transfer learning will only work if the initial problem is sufficiently similar to the target problem.

2.6 t-SNE

Due to the growing amount of data with potentially thousands of features like in image data, unsupervised dimensionality reduction methods to preserve important information are essential to enable visualization and exploration of high-dimensional data. A widely used example of such a method is t-Distributed Stochastic Neighbor embedding (t-SNE), introduced by Van der Maaten and Hinton in 2008 [58]. Traditionally, linear dimensionality-reduction techniques like Principal Component Analysis (PCA) [48] have been used to preserve longer distances between dissimilar data points. t-SNE on the other hand is a non-linear method that aims to keep similar data points closer to each other and works better where the relationship in the data tends to be non-linear. We can say t-SNE focuses on preserving local similarities in the data, while PCA preserves the global ones. The process of t-SNE maps the pairwise similarities of all points to distributions both in high- and low-dimensional space and further uses the KL-divergence[2] to optimize the distributions to be as similar as possible [58].

The first step is to convert high dimensional data points to approximate Gaussian distribution centered on each point. The conditional probability $P(j|i)$ states the probability that data point x_i would pick data point x_j as neighbor according to a Gaussian distribution centered at x_i, where data points closer to x_i will have a higher value. For every data point, a set of probabilities will be calculated corresponding to similarities to every other point [58].

In the same way, as we computed a distribution P for the high dimensional data, we also want a distribution Q for the corresponding low dimensional space. But compared to the

[2]Kullback-Leibler divergence is a statistical similarity measure between two probability distributions.

high-dimensional counterparts, a Student's t-distribution[3] rather than Gaussian is used for measuring similarities. The Student t-distribution's heavier tails will cause distant points to be penalized less, resulting in improved long-distance dependencies measures [58].

After both distributions are computed, gradient descent reduces the Kullback-Leibler(KL) divergence of distributions P and Q. A lower divergence makes the lower-dimensional mappings more correspondent to the high-dimensional ones. Further details on t-SNE can be found in Van der Maaten and Hinton [58].

2.7 Model Evaluation

2.7.1 Metrics

Both during and after model training, methods to get feedback on performance and stability are required. Using suitable evaluation metrics can provide insight into the models' capabilities and demands. However, utilizing unfitting metrics can be misleading and lead to poor configuration choices in the future.

To employ evaluation metrics for classification tasks, we need to address which classes are treated as the positive and negative ones. The positive classes are usually the classes cared most about; like in event detection, the positive classes will be samples containing the actual events, and the negative class will be samples where no event occurs. There are four general outcomes possible when working with classification predictions:

True Positive (TP) is when the ground-truth value is positive, and the model correctly predicts the positive class.

False Positive (FP) is when the ground-truth value is negative, and the model incorrectly predicts it to be positive.

True Negative (TN) is when the ground-truth value is negative, and the model correctly predicts it to be negative.

False Negative (FN) is when the ground-truth value is positive, and the model incorrectly predicts it to be negative.

[3]Student's t-distribution is a symmetrical, bell-shaped probability distribution with heavy tails.

Accuracy

The classification accuracy of a model simply computes the percentage of correct predictions. For tasks with a balanced number of classes, accuracy can be fitting.

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.20}$$

Precision

Precision explains the models' ability to be accurate when predicting positive cases. It is defined as the fraction of true positive examples out of all positive predictions.

$$\text{precision} = \frac{TP}{TP + FP} \tag{2.21}$$

Recall

Recall answers what proportion of positive cases the model discovers, defined as.

$$\text{recall} = \frac{TP}{TP + FN} \tag{2.22}$$

There is a trade-off between good recall and precision, and hence both metrics should be provided to give sufficient insight. This can be illustrated if a model predicts the positive class for all samples, it would have a perfect recall score, but the precision would likely be poor.

F1-Score

F1-score is a single metric combining both precision and recall, by their harmonic mean, defined as:

$$\text{F1} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{2.23}$$

Thus, a decrease in precision or recall will also lead to an F1-score reduction, making it a good measure for the general performance of a model. F1-score can be computed as an overall value, or it can be calculated as a value per class, which is often applied in multiclass problems. However, if good precision is more desirable than recall, or vice versa, a weighted F1-score may be suitable. Rather than taking the harmonic mean of precision and recall, a weighted F1-score calculates the average score for each class. This can be handy when some classes are emphasized more than others.

2.7.2 Class Imbalance

In many classification problems, the majority of samples will belong to the negative class, like in spam- or disease detection, where the minority class is the one of importance. To train a model on such imbalanced datasets is challenging because models will naturally have a high bias towards the imbalance experienced during training. The model will typically have a higher volume of predictions on the over-represented class, and the minority class can be wrongly classified since the majority class is prioritized to maximize correct predictions. When dealing with class imbalance, it is significant to choose evaluation metrics carefully. For instance, accuracy could give unreliable results for skewed data. Despite reaching high accuracy scores, the results might be trivial. This can be illustrated in a classification task where 9/10 samples are the negative class, and the model can achieve 90 percent just by exclusively predicting the negative class. Thus, F1-score would be a more reliable measure.

There are a variety of methods to handle skewed data distributions. When addressing the imbalance on the data-level, the most straightforward approaches are **random over-sampling** and **undersamping** [7]. Under-sampling refers to simply deleting data points for the majority class to even the class distribution, while over-sampling duplicates data points from the minority class. The main drawback of these approaches is that under-sampling gives away information that might be useful, and oversampling might lead to overfitting due to multiple equal examples from the minority class. Another method is to create synthetic data of the minority class by duplicating samples and slightly modifying them, referred to as **data augmentation**. We can extract more information from the dataset by augmenting the minority class samples; for example, we can apply affine transformations like flipping, distortion, or color shifting to images. Then both the synthesized and original samples are utilized during training [43].

2.7.3 Cross-Validation

Models with different settings are trained and evaluated during hyperparameter search to seek suitable configurations to improve performance. The models are evaluated on the validation set held out from the training data to estimate how the models would do in practice. However, it is not certain that our validation data is representative of real-world data. Especially when the size of the validation set is limited, there could be a high variance in the results for different partitions of validation data. To reduce the variance in our evaluation, we can do **cross-validation**. The essence of cross-validation is to split the data into folds, where each model is evaluated iteratively. A distinct fold is used as validation data for each iteration and the remaining folds for training. Hence, each configuration of hyperparameters will be trained and evaluated for different data subsets. This reduces the randomness in the generalization estimates and can reduce bias during model selection.

3 | Related Works

With the expanding datasets and processing power of the latter years, deep learning methods for video analysis and action recognition have bloomed, considering their ability to extract useful spatio-temporal features [29]. The aim of most traditional deep learning methods related to video analysis is to extract informative spatial features with convolutional operations and model the temporal characteristics of video's sequential attributes. Karpathy et al. [25] introduced a multi-stream spatial-temporal 2D CNN model that aggregates frame features at various time steps. Their solution proved to perform particularly well in combination with transfer learning. Another popular approach is 3D convolutions, which computes spatio-temporal features by convolving a 3D kernel on a volume formed by a sequence of frames [24, 56]. However, a downside of 3D convolutions is the substantially increased number of parameters compared to models relying on 2D convolutions [29]. Moreover, a prevailing method is to aggregate all temporal information from the visual features represented by a CNN by employing an RNN structure on top of it, usually represented as an LSTM [12, 8, 47, 52]. One of the earlier successful such architectures was the long-term recurrent convolutional networks (LRCN) influential for tasks involving time-dependent visual data [12].

Due to the rapidly increasing quality and volumes of ecological data [37], AI solutions for computer vision tasks have become more applicable in analysis and automation in ecological studies. Although the growth of dataset sizes, manual labeling and interpretation of data are still the bottlenecks for such applications due to the required quantities of high-quality annotated data [37]. In marine science, deep learning-based computer vision algorithms have shown promising results for practical applications like species classification, individual detection, and tracking [50, 5, 11, 38, 28, 34], impending future automation of monitoring marine ecosystems. Despite the increased attention to deep learning and computer vision in marine science, the utilization of video classification is still relatively untouched [8].

Despite recent studies showing promising results on action recognition for humans, the challenges are not directly transferable to the domain of fish, mainly due to less-defined poses and increased environmental noise [40]. The lack of well-defined poses makes it intuitively harder to distinguish actions from single frames and makes some previously successful methods for human action recognition less suited, for instance, skeleton-based architectures [14, 13]. Moreover, studies on behavioral classification in marine science substantiate the importance of exploiting the temporal context video data provides [8, 40]. Conway et al. [8] extended the use of CNN for regular image classification by combining it with a recurrent neural network (RNN) structure to classify seal and penguin behavior from video streams. The use of temporal context proved substantial improvements compared to image classification for the same task, benefitting from introducing temporal information. Compared to this thesis, Conway et al. classified behaviors frame-by-frame from dynamic cameras mounted to the animals, while in this work, the data origins from static underwater cameras with behavioral events spanning shorter time intervals and can therefore be perceived as more subtle. Måløy et al. [40] proposed a model recognizing the feeding behavior of salmon in underwater videos, with promising results. Their so-called Dual-Stream Recurrent Network (DSRN) architecture is based on a two-stream recurrent network with a 2D convolution spatial stream and a 3D convolution motion stream with substantial depths. The aggregated output features are fed to an LSTM. Contrarily to our study, the task of salmon action recognition was a binary classification problem with considerably more data samples for the positive class (43K frames), which allows utilization of deeper models in order to reduce the variance and enhance predictive power.

4 | Material and Methods

4.1 Study Species

The benthic species, corkwing wrasse, have males who build and guard nests in shallow water where females lay their eggs (figure 4.2). In this species, territorial males build and defend nests made of seaweed in rocky environments, primarily from May to July. Most of the males develop as dominant colorful nesting males, while some develop as sneaker males that grow slower and mimic females, to sneak fertilize eggs in dominant males' nests [44]. The sneaker males and females are not distinguishable by their visual characteristics (figure 4.1). Because the nesting corkwing males defend territories and nests, they regularly perform chases of intruders or competitors. Other typical behaviors around nests would be courting and spawning, with trajectories containing frequent and unpredictable movement changes [15].

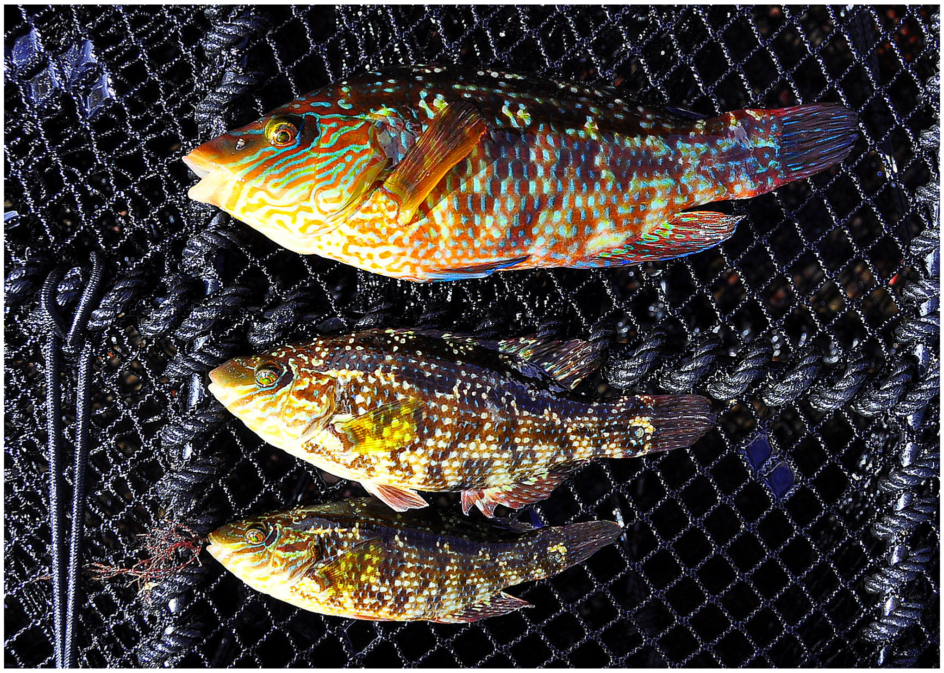


Figure 4.1: A nesting corkwing wrasse male on top, a female in the middle, and a sneaker male at the bottom.

Photo by Tonje K. Sørdalen

Of the six wrasse species occurring in Norwegian waters, corkwing wrasse is one of four species harvested to be used in the aquaculture as cleaner fish to fight lice manifestation on salmon [9].

4.2 Study System

In a marine protected area (MPA) outside Austevoll in Norway, commercial fishing of wrasses is prohibited, and the area is used as a field site of studies on corkwing wrasse. This site provides a source of real-world ecological data to study population dynamics and mating behavior by utilizing modern camera technologies and RFID systems[1]. This system has acquired substantial amounts of underwater video data of corkwing wrasse, like the image in figure 4.2. Besides being biologically relevant in itself, it can also further be used to develop machine learning solutions for video analysis. The video data used in this thesis was

[1]Radio Frequency Identification System (RFID) allows for non-contact data transferring by the use of RFID readers, PIT tags (passive identification tags), and antennas.

collected by small video rigs at 1-3 meters of depth during spring and summer 2018-2020, focusing on 12 different corkwing wrasse nests. The video catalog from Austevoll includes over 100 hours of GoPro videos (GoPro HERO6 and HERO7) obtained from nesting sites.

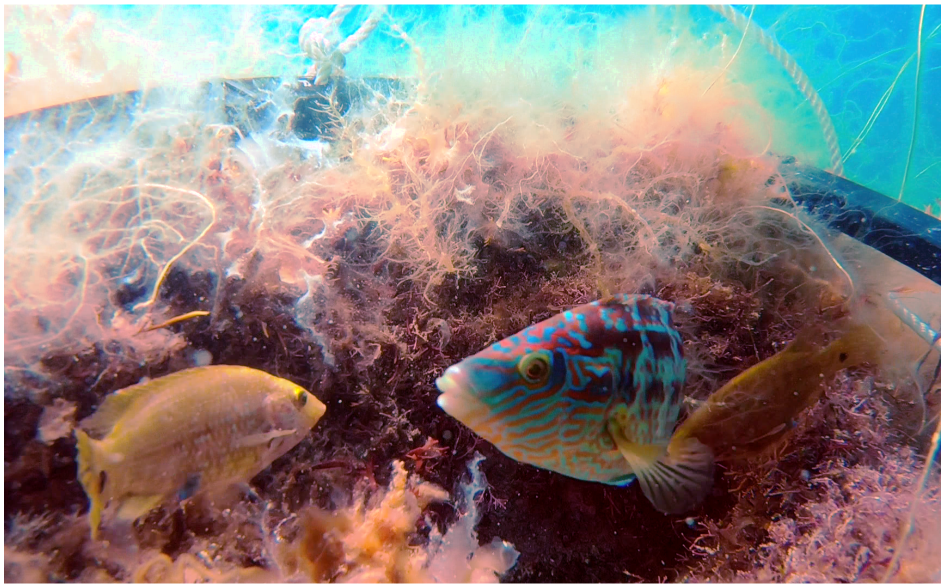


Figure 4.2: A colorful nesting male surrounded by two females. The image is captured from the camera RFID system, and the antenna can be seen in the background.

Photo by Tonje K. Sørdalen/Kim Halvorsen

4.3 Video Data

The video lengths vary from around 60 to 100 minutes which are split into 4-5 parts not longer than 1058 seconds (approx. 17,5 min) each. The footage is recorded at 60 frames per second with a resolution of 1920×1080. The video catalog is structured on the recording date, followed by the ID number of the videos on the given date. Then, each split of a video is stored in MP4 file format, commonly used for storing multimedia data. Figure 4.3 displays examples of frames from different nests, all including the presence of a nesting male.

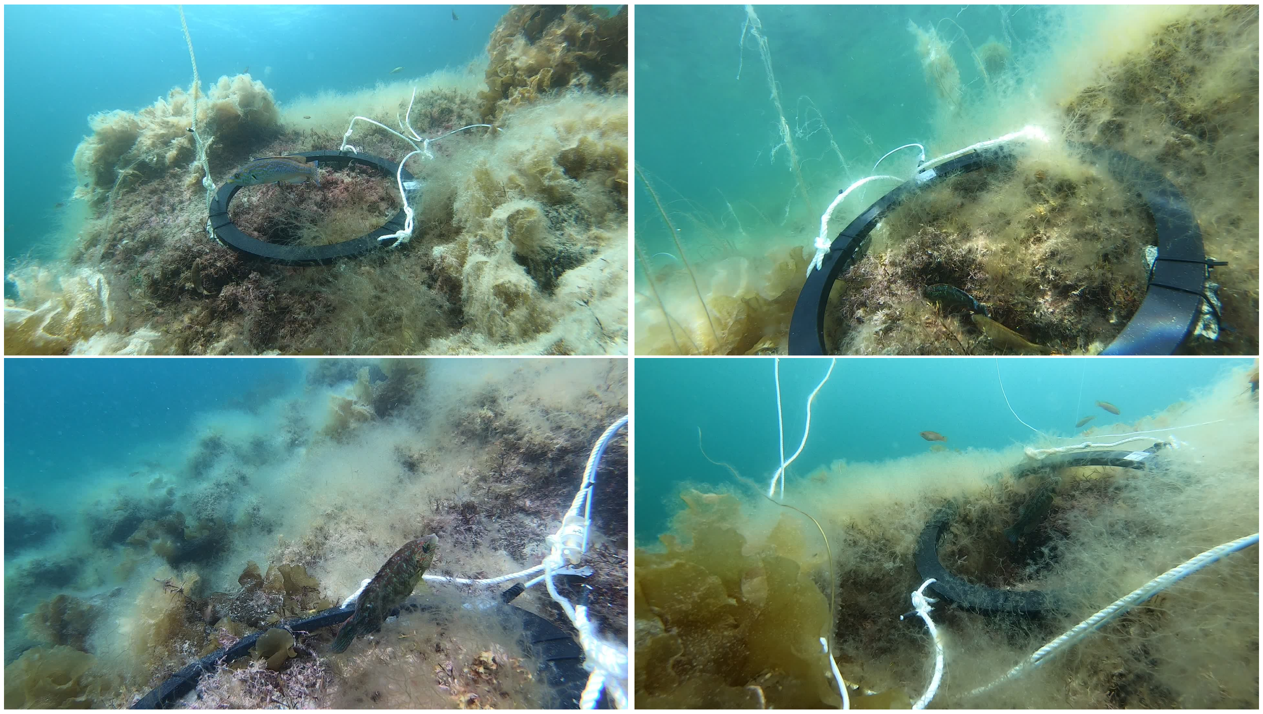
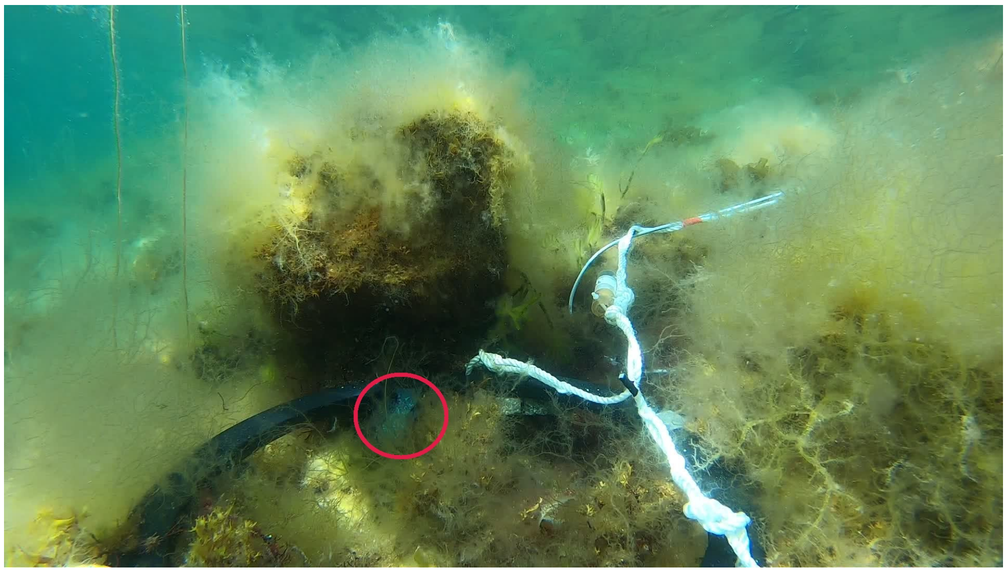


Figure 4.3: Example frames collected from various nests.

There could be occurrences of different species in the streams , but the species of interest is exclusively corkwing wrasse. The videos are captured at nests belonging to dominant males, which most behavior analysis will be centered around. There can be multiple females or sneaker males present simultaneously, but mostly there will only be a single nesting male individual at each nesting site. Since the videos are collected from several camera rigs over time, there will be environmental varieties, such as background, lightning, and time of day. Compared to videos in standardized conditions, these videos contain different degrees of natural noise like occlusion and high turbidity, making it more challenging for a model to distinguish the valuable features in the data. Primarily, occlusion occurs regularly for some nests, with seaweed and tangle obscuring the visibility of individuals (figure 4.4a). There are also substantial variations in the region of interest in the videos. Since the distance from the camera to the nest differs, the spatial dimensions of the relevant regions can deviate. When the region of interest is smaller (figure 4.4b), the probability of information loss increases when images get downsampled during the CNN forward-pass. On the other hand, such natural noise might be helpful for real-world applications, and training data with a diverse set of habitats benefits models to be more robust [11].


(a) Occluding obstacles make only the tail fin visible

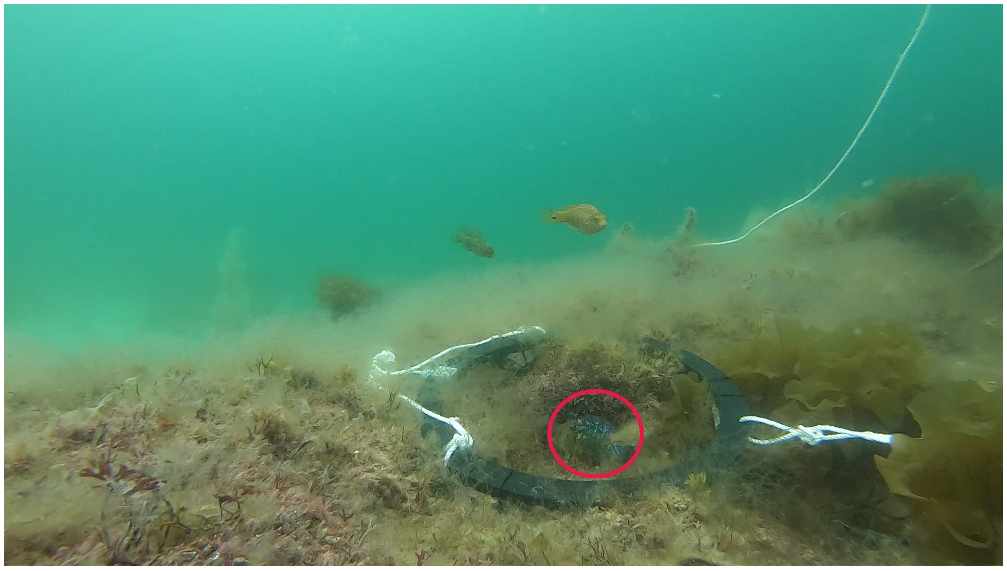

(b) Sample with a long distance from the camera to the nest.

Figure 4.4: Examples of natural noise in video data. The nesting males are marked within the red circle, and both frames are during a spawning event.

4.4 Behavior Annotations

A set of behavior annotations of events occurring in the video streams is provided to supplement the video data. The annotations are labeled manually by a taxonomic expert [15] using the event logging software BORIS [16] for video data. The annotations contain 12 different behavior events with quantities between 4 and 2900 observations for each. A behavior annotation is defined by the video file, nest, and behavior class information. Additionally, annotation provides temporal information like the point of start, point of stop, and duration of a given event.

The behavior types are separated into two categories; **state-** and **point behaviors**. State and point behaviors are distinguished by the duration of the events. While point behaviors are more or less instant (for example, "spawning" or "sneaking"), state behaviors usually span over longer periods (for example, "male present" or "nest blocking") and have to a lesser extent a fixed duration compared to point behaviors. Whereas point behaviors will be completed in a blink of an eye, state behaviors can last from one second to several minutes. In fact, most state behaviors are not truly behaviors since they mainly provide information about the presence of species and gender, like the behavior classes "male present" and "female present". However, some state events describe actual behaviors, such as the class "male with seaweed", which describes when a male is carrying a seaweed in his mouth. Thus, the state event is generally recognizable just by looking at still images and does not necessitate exploiting temporal information to the same extent as point events, which are considerably more subtle and intuitively harder to distinguish from one another. Even for humans, there can be hard to pinpoint when certain point events occur without adequate domain experience. Since point events have a short and relatively constant duration, there is only provided a starting point of the events and no defined stop point. Hence, there will be uncertainty about the event's exact duration. For this reason, the sequences used for training cannot be too short to prevent any information loss of actual events. Furthermore, some frames before the defined event start should also reside in the sequences as a margin of error to the actual exact starting point. Besides, a model may find some valuable prior contextual information for the classification task that may not be obvious to human eyes.

There are more samples of state events than point events, with an average of 965 samples per state behavior, to 135 per point behavior. Additionally, state events span a longer time and will therefore emerge substantially more frequently in the frames than the point events. This led to significantly larger quantities of data for state behavior samples. Although, point behaviors are the event type mainly focused upon in this thesis. This is because the classification of the point behaviors will likely demand extraction of spatio-temporal features to a more significant extent than the state behaviors, in addition to being more biologically relevant. The state behaviors mostly rely on the spatial context since they generally denote instances rather than behaviors. The classes used in our proposed behavior classifier are “spawning” and “chase”, both point behaviors.

Spawning description The female swims into the nest entrance and flicks her abdomen into the nest, releasing eggs. The dominant male immediately performs the same behavior with multiple flicks, releasing sperm. The release of eggs and sperm is not visible, therefore the behavior is inferred [15].

Chase description The dominant male chases a threat away from the nest. The threat is not always visible and is defined as the male orientating himself in a specific direction before accelerating rapidly towards his target [15].

Spawning and chase are the two classes of point behaviors containing the most samples, correspondingly 344 and 319. Spawning was observed in six of the twelve nests, where some of the nests contained large proportions of the total spawning samples. On the other hand, chase events occurred in all nests with a generally lower density each.

State events have not been used in any of the experiments herein, but merely for sanity checks and partly in data distribution phases to ensure most training samples contain relevant content, like nesting males.

4.5 Data Preparation

This section describes how our raw video data are prepared to be applicable in CNN architectures. In addition, we describe the choices when preparing the annotations, as careful data preparation is a vital step for reliable performance. A model will never be better than the data it is trained on [30].

Video data

All videos are extracted from video files to a set of image files with FFMPEG[2]. Due to time and computational power available during training, there was infeasible to extract all 60

[2]FFMPEG is a free open-source software for handling multimedia files [55].

frames per second, which would result in more than 60 000 frames for each video file where a substantial amount of frames would be nearly identical. Despite not extracting all frames, we would need a sufficient number of frames to gain enough information about the events. We extracted ten frames per second, where the priority was to utilize the computational power on longer-time sequences rather than have sequences with more frames per second. An increased fps may result in not being able to process sequences with a sufficient real-time length efficiently.

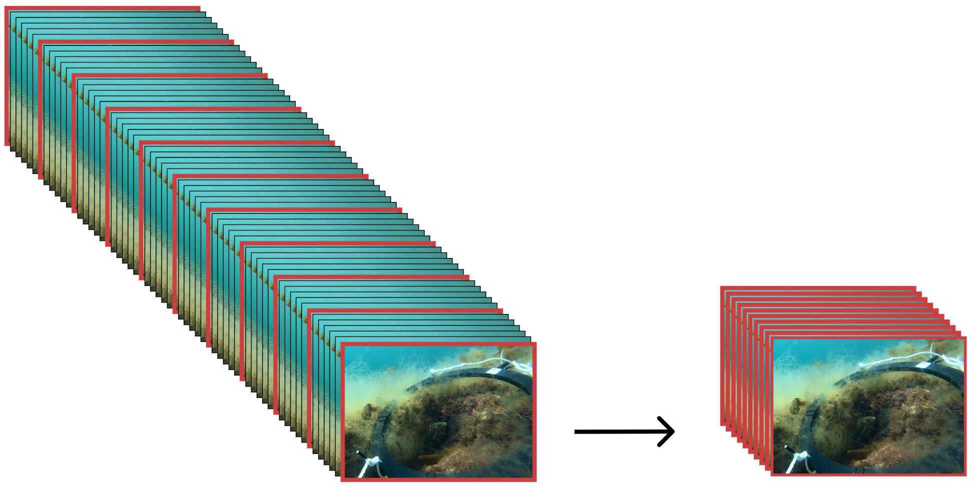


Figure 4.5: Ten frames extracted from one second of video data at 60 fps.

4.5.1 Annotations

In the raw annotations, events are provided with a corresponding video file, time stamp, and behavior class which makes all events unique, as shown in table 4.1. Since the video files are converted to a set of image files, we want an event to correspond to an image file rather than a timestamp. A script was made to process the video-based annotations as input and process them into annotations where each row corresponds to a single frame rather than an event, as showed in table 4.2. The frame annotations produced are organized such that each frame has a corresponding one-hot label vector, where each class has a coherent column, either one or zero, defining the presence. There will also be an additional column stating if there is no behavior event present in a frame. The user defines the classes the frame annotations should contain, which in this case was spawning and chase. The frames are arranged in ascending order, initiated at zero for each individual video file. The timestamps of the raw annotations did not consider that the full videos are stored as equal-sized parts, which needed to be handled during transformation.

Video path	**Behavior**	**Start (s)**
/20200603/1/GH010063.MP4	Chasing	278.919
/20200603/7/GH10080.MP4	Spawning	718.686

Table 4.1: Raw Video Annotations.

Video path	**Frame number**	**Spawning**	**Chasing**	**None**
/20200603/1/GH010063	2788	0	1	0
/20200603/1/GH010063	2789	0	1	0
/20200603/1/GH010063	2790	0	1	0
....	...	...	...	...
/20200603/7/GH10080	7186	1	0	0
/20200603/7/GH10080	7187	1	0	0
/20200603/7/GH10080	7188	1	0	0

Table 4.2: Extracted Frame Annotations from video annotations in table 4.1 at ten fps.

Working with ten fps at data initial annotated for 60 fps videos entails that the annotation timestamps will not precisely correspond to a specific frame. For this reason, the frame closest to an annotation time stamp is categorized with the given behavior, and the two neighbor frames are categorized with the same behavior. This results in a single point behavior event in the raw annotations, producing three frames with the same behavior. This was mainly done for experimental purposes in the early phases of the work and will not impact the final dataset when frames are assembled into sequences.

There is currently a 1:1 mapping between frames and labels, but for the data to be applicable for temporal dynamic models like RNN and LSTM, the data need to be of sequential nature. Thus, sequence-based annotations were constructed from the frame annotations (table 4.2) by a script that iterates sequentially through the frames and forms **sequence labels**, as shown in table 4.3. The desired sequence length can be chosen as a parameter. Sequence labels will be produced such that the frame which is actually labeled with a behavior class is centered in the sequence with the same amount of contextual information both prior and after to the actual labeled event. For example, for a sequence with a length of seven, the fourth frame will consist of the frame closest to the actual video timestamp of the event, and there will be three frames both prior and after. In contrast to frame labels, sequence labels do not have a one-hot vector but a single label of the behavior event occurring in the sequence.

Video path	**Start frame**	**Stop frame**	**Behavior**	
/20200603/1/GH010063	2782	2798	C	
....	...	...	...	...
/20200603/7/GH10080	7180	7194	S	

Table 4.3: Extracted Sequence labels from the frame labels in 4.2 with a sequence length of fifteen.

When only working with the spawning and chasing behaviors, there will never be overlapping events in the same sequence if they are no longer than 30 frames since the events do not occur that frequently. Still, overlapping events can occur in real-life cases within 30 frames. Nevertheless, we seek to keep the task a regular multi-class classification problem to reduce complexity with a limited amount of data. Therefore, the sequence length will range between 7 and 30 frames during training and evaluation.

4.5.2 Preprocessing and Sampling

A sample of data will consist of a sequence of frames with a single corresponding sequence label. Before training, the dataset is initialized where the ratio of negative to positive sample is defined as a parameter. Naturally, our positive samples are the sequences containing spawning and chase events, and the negative samples are the rest. Additionally, the desired videos the dataset should contain can be provided as an argument when constructing it, which is practical when partitioning into training and evaluation data.

Training models with high-resolution images are time-consuming and demand more computational capacity [49]. Due to computational constraints, it is infeasible to train CNN models employing sequences of 1920×1080 images with our resources. For this reason, downsampling is performed at training time, halving the spatial dimensions to 960×540.

This reduces the number of input pixels from approximately two million to five hundred thousand. Naturally, a pixel reduction leads to loss of information, but hopefully, sufficient spatial dimensions are retained to seek discriminative features. All input pixel values are normalized as an additional transformation at training time. Normalizing refers to the process of constraining all pixel values to a given range, which has proven to make computations more efficient [46].

With over 50 hours of total video where only a few hundred events of relevant behavior, the data suffer from the problem of class imbalance explained in section 2.7.2. We apply both under-sampling and data augmentation to our data to mitigate the imbalance. Under-sampling is done on data samples containing none of our relevant behavior events. We experiment with constraining the under-sampling to include a sufficient amount of negative samples where a male individual occurs in the sequences by utilizing the "Male present" class in the annotations. The thought behind constraining the negative samples is to avoid the model simply classifying behaviors from the fact that male individuals occur or not. Even though it is not likely, there could be a risk that most negative samples of a dataset contain non-relevant samples, like footage from camera placement or sequences without male individuals. Potentially, it could lead to a model learning to classify behaviors just by detecting the presence of a nesting male.

The fact that CNNs are invariant to position allows us to perform data augmentation. Several augmentations methods are mentioned in 2.7.2; however, all are not equally suitable for our task. Since the region of interest varies depending on the videos, the random crop is excluded since we risk withdrawing important information. The colors for the objects of interest are mainly similar regardless of the environment, hence there is no color augmentation applied either. On the other hand, a horizontal flip is applied to increase relevant data points. Flipped images can correspond to fishes with different orientations and noisy representations of previously seen nesting sites.

4.6 Models

Mainly, there are two model architectures applied in this thesis. The first architecture is 3D CNN, based on 3D convolutions described in 2.3.1. The second architecture is a composition of a CNN (section 2.3) and RNN (section 2.4) method to extract spatial and temporal features.

4.6.1 3D CNN

Our 3D CNN architecture is based on Tran et al. [57] collection of ResNet models utilising spatio-temporal 3D convolutions (section 2.3.1). We utilized the ResNet 3D architecture, with slight modification in the classification layer to fit our task. The ResNet 3D employs 2D and 3D convolutions, residual links, ReLU activations, max pooling, and batch normalization. More details about the model can be found in appendix B.2. Experiments were performed both with pre-trained weights and end-to-end trained models.

The 3D CNNs were substantially more time-consuming to train than Recurrent Convolutional Neural Network (RCNN). Thus, we have not been able to experiment with numerous configurations.

4.6.2 Recurrent Convolutional Neural Network (RCNN)

The RCNN model constructed in this study is separated into two main components; CNN and RNN. The CNN component is referred to as the image encoder. The image encoder produces lower-dimensional vector representations of the input images. These image representations are proceeded to the RNN component, referred to as the sequence encoder. The sequence encoder should capture the temporal dynamics in the sequence of image representations. The final spatio-temporal representations of the image sequences are then fed to a classification layer and a softmax function to produce class probabilities. An overview of the RCNN architecture is illustrated in figure 4.6.

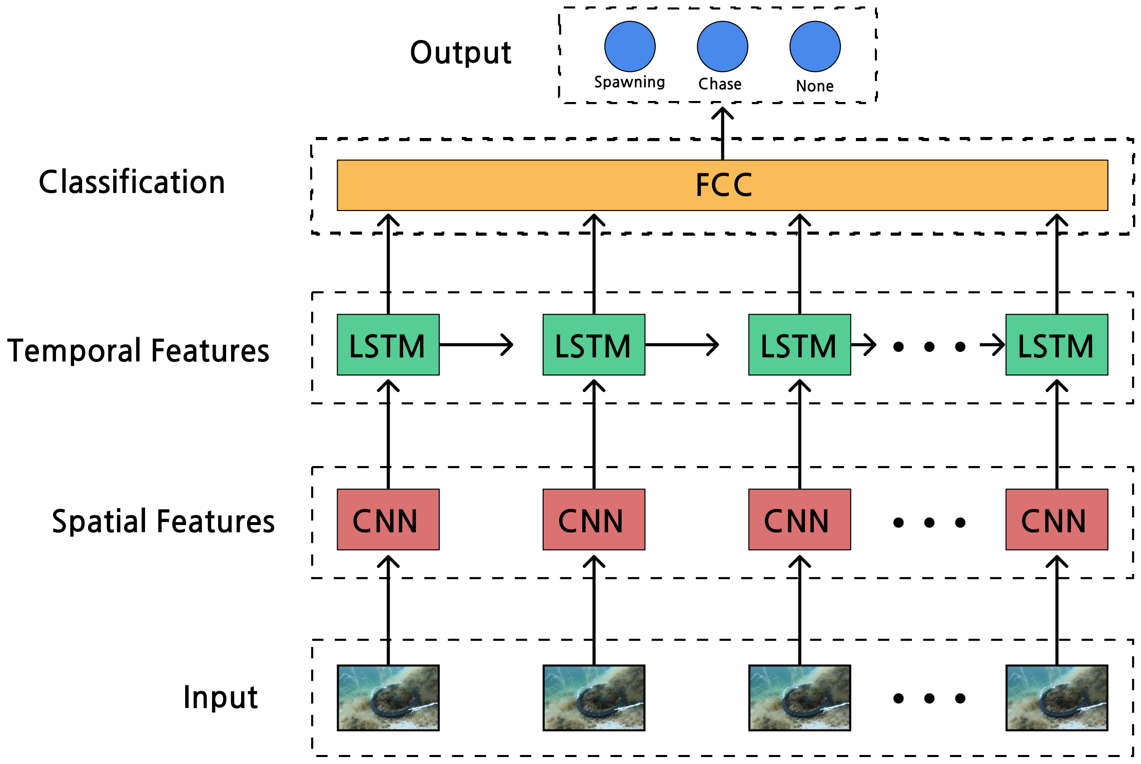


Figure 4.6: RCNN architecture.

Image Encoder

The image encoder is based on the PyTorch[3] implementations of **ResNet** or **VGGNet**. ResNets utilizes residual links (section 2.3.2), ReLU(figure 2.4) activations, max pooling, and batch normalization(section 2.2.5). VGG is a group of deep convolutional neural networks invented by the Simonyan and Zisserman Visual Geometry Group at the University of Oxford in [51], effective for visual recognition. Compared to ResNet, VGGNet generally has a significant larger number of parameters and utilizes ReLU activations and max pooling.

PyTorch provides various sizes of ResNet and VGG nets, all available with weights pre-trained on the ImageNet dataset[4]. We modified the models to produce the desired encoding

[3]PyTorch is an open-source Python-based framework used for ML [42]

[4]ImageNet is a database containing more than 14 million annotated images for 20 000 different classes [10]. The data set is regularly used for developing and benchmarking visual recognition algorithms.

size. We experimented with different degrees of transfer learning (section 2.5). For complete transfer learning, all parameters were imported, and the last dense layer had to be modified to the desired output dimension, thus also tuned. We experimented with an image encoder trained end-to-end and partly trained by freezing most imported parameters and solely tuning the last few layers.

We experimented with a few variations of the nets described above, respectively ResNet18, ResNet50, and VGG13. The numbers refer to the depth of the models. Further details about the models can be found in appendix B.1.

Sequence Encoder

For all reported results in the study, the sequence encoder component will consist of an LSTM. We experimented with various hyperparameters of the sequence encoder, such as hidden state size, depth, and bidirectional LSTM. The temporal dynamics of the input sequence will be summarized in the last hidden state of the sequence encoder and proceeded into a fully connected layer to produce a prediction.

4.7 Experimental Setup

This section will describe how our models are configured, optimized, and evaluated.

4.7.1 Online Evaluation

The data are manually split into train-, validation-, and test videos, ensuring that all contain a sufficient number of behavior events from different nests. During training, there is a persistent tracking of loss for every epoch, while the model is evaluated on the holdout evaluation videos for every fifth epoch. While the training set is stratified to have a more even class distribution, that is unnecessary for the validation set since a skewed class distribution will not affect the model during validation as long fitting evaluation metrics are applied. Nevertheless, the validation set is also stratified on the negative samples to reduce the computation time of the evaluation steps. The sampling of the validation data proceeds with the same sequence length and preprocessing steps as done for the training set. The metrics employed to evaluate validation data are precision, recall, F1-score, accuracy, and loss.

In order to find the best configurations for different model architectures, a random hyperparameter search was performed. The search space was initially large but was gradually narrowed later in the process when discovering indications of how values affected performance. After some time, these trends were evident with the random search and some exhaustive search for individual variables. A reason to narrow the search space was to allow cross-validation to make the training expenses more feasible. The narrowed hyperparameter space can be seen in table 4.4. Note that the negative:positive ratio hyperparameter was only applied with other values than 1:1 if Weighted Cross Entropy=True, since the training will suffer from data imbalance otherwise.

Hyperparameter	**Values**
Initial Learning Rate	10^x where x $\in$ $\boldsymbol{U}_{[-5,-1]}$
Image Representation Size	512, 1024
Batch Size	1, 2, 4
CNN Model	ResNet18, ResNet50, VGG13
LSTM Layers	1, 2
Momentum	.5, .9, .99
LSTM Layer Size	512, 1024
Bidirectional LSTM	True, False
Optimizer	Stochastic Gradient Descent (SDG), Adam
Weighted Cross-Entropy	True, False
Negative:Positive Labels Ratio	1:1, 2:1, 3:1, 5:1

Table 4.4: Narrowed Hyperparameter Search Space.

A four-fold cross-validation was performed for every configuration, and a fold containing 30-50 samples for every positive class. The four different folds of validation videos were manually composed to secure the diversity of the nesting sites and properly distributed behavior events to be statistically representative. Cross-validation was suited due to the limited number of samples in the initial validation set. The purpose is to reduce the variance of evaluation results and provide more reliable estimates of generalization abilities. Cross-validation was also handy to observe the stability of the model for different sets of unobserved samples.

4.7.2 Training Scheme

Initially, the models were configured from the hyperparameters retrieved in a random search. Separate models were trained for every cross-validation fold, with identical configurations. The training lasted 100-150 epochs for the RCNN and up to 250 for the 3D CNN. Tracking performance was done continuously during training, logging results of the online evaluation, as described in section 4.7.1. There were both experimented with a constant and a dynamic learning rate. For the dynamic one, a linear scheduler gradually decreased the initial learning rate to seek improved convergence properties.

The training was finalized with a last evaluation of the training- and validation data, logged for further analysis, and saved the trained model weights.

4.7.3 Experiments

The trained models will be quantified on the test data, in the same manner as online evaluation, reported as class-wise precision, recall, and F1-scores. We will also compare how the models do for different sequence lengths.

Candidate models from the cross-validation were manually selected regarding how the model configurations performed for the validation folds, focusing on the performance of the positive classes. These candidate models were further inspected on the test data. We experimented on how the sequence length of the input affects performance. When adjusting sequence length, it is worth noting that there will be some randomness when constructing the datasets since negative samples will not always be the same for different sequence lengths. This may make results for different sequence lengths deviate a little.

There were carried out considerably more analyses and experiments for the RCNN models than for 3D CNN models, mainly due to lesser computational expenses.

4.7.4 Image Representations Evaluation

Since our RCNN models are split into two major components, it is insightful to analyze their performance separately. Evaluating different system components gives better insight and can save time from exhaustive experiments on a component if the bottleneck occurs elsewhere in the system. Moreover, if our image representation component does not produce any meaningful representations, the relevance of the sequence encoder will diminish since its input data would mainly be noise. For this reason, we did assessments of the image representations produced by the image encoder of an RCNN candidate model.

Extrinsically, we get an insight into the quality of image representations by applying them to our sequence model and quantifying the performance, as described previously. However, we would also like to intrinsically evaluate them to perceive the strengths and limitations of the image encoder. Therefore, we would like to visualize the representations, allowing us to analyze them qualitatively.

We only visualized representation from one model since the analysis's purpose was a general insight into the model's ability to extract meaningful spatial features rather than applying the analysis to various configurations to compare results.

t-SNE

To visualize the embedding data, we will use the dimensionality-reduction method t-SNE, as described in 2.6. Our high-dimensional data will be embedded into two-dimensional data, enabling visualization. By scatter plots, we can see the relations and patterns in the data. For instance, meaningful representations concerning the behavior classes would be presented as distinct clusters, where data for similar behaviors would be densely populated.

The data were visualized both with respect to the classes and the videos. Sequences from the same video contain the same environment and background; thus, visualizing with respect to the videos may indicate if the representations are mainly extracted from environmental- and background features. There are also provided visualizations for different time steps to analyze if the embedding relations between the classes were less distinguishable further away from the start of the behavior event. In addition, to examine the differences between frames prior to the behavior event compared to subsequent ones.

4.7.5 Tools

The open-source multimedia handling tool FFMPEG [55] was used for extracting frames from the video data. The construction and training of the models were done using the python based deep learning framework PyTorch [42], version 1.9.0. All models were built with components from PyTorch's torch.nn module, simplifying building, customizing, and training the neural networks. Data preprocessing and iterating were employed with the PyTorch modules transforms, dataset, and dataloader, combined with the Python Image Library (Pillow) [6]. All online evaluations were logged in Tensorboard [2], providing a neat overview and visualizations of training progress and the results.

5 | Results

This section will report our findings and how the trained models perform on unseen data.

5.1 Experiments

All models have been trained with different configurations of hyperparameters and varying model architectures, but our primary methods evaluated are categorized as 3D CNN (section 2.3.1) and RCNN (section 4.6.2). The results reported in this chapter are computed from candidate models selected based on performance on the validation data during the cross-validation. We report on how the selected candidate models generalized to the test set.

Subset	**# spawning samples**	**# chase samples**
Training	257	234
Validation	43	46
Test	44	39
Total	344	319

Table 5.1: Number of positive samples for data subsets. The number of validation samples is for the first fold of cross-validation.

Since the quantities of evaluation data are moderate (table 5.1), the results may fluctuate for tiny alterations in the models or data. Therefore, the results will be presented for single models and statistical patterns by aggregating results for multiple candidate models (appendix A.2). This reduces the variance and may provide a more stable estimate of what capabilities to expect of such models.

5.1.1 General Findings

The optimizer settings were generally better with SDG than with Adam due to faster and more frequent convergence. Models using Adam tended to get stuck at predicting the same class for all samples, which might be due to insufficient tuning of the optimizer parameters. In general, there was no dominant learning rate but rather dependent on the corresponding architecture and hyperparameter settings. However, a general pattern was that models tended to struggle with convergence for learning rates higher than 0.001. There was no correlation between model performance and transfer learning settings throughout experiments, but the pre-trained ones processed data slightly faster during training. There was neither any significance if the sequence encoders in the RCNN were bidirectional or not.

5.1.2 Convergence properties

We tracked all models during the training phase by their training and validation loss to see how they progressed over time. Generally, numerous of our models converged during the random parameter search within 120 epochs of training; however, about two-thirds did not converge and got stuck in suboptimal solutions. In figure 5.1, we illustrate five models' progress during training. These models were picked selectively to represent the different typical convergence patterns.

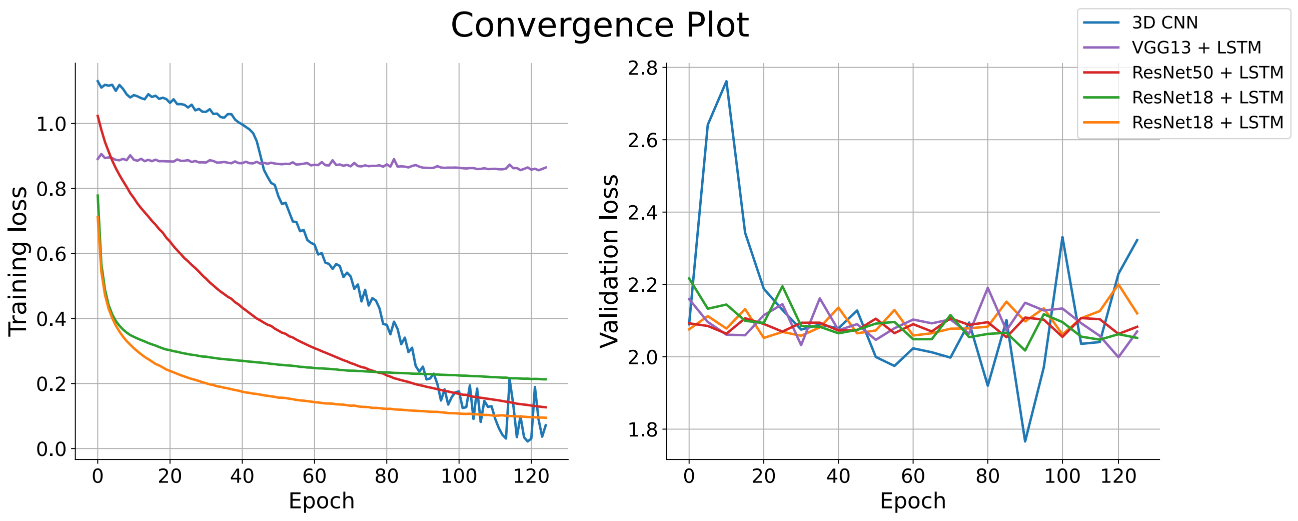


Figure 5.1: Figure comparing convergence properties on both training and validation data.

From figure 5.1 we can see that no model does manage to reduce the validation loss throughout training, despite the convergence properties of the training data, suggesting deficient generalization abilities. The 3D CNN model is yet to fully converge after 120 epochs; thus, 3D CNNs have trained longer, up to 250 epochs. The unstable validation loss is a consequence of the small number of samples in the validation data.

5.1.3 3D CNN

We did not manage to produce numerous converging 3D CNN models. We report the test set performance to one of the converging ones for different sequence lengths.

Sequence length	**7**	**15**	**23**	**29**
Precision (Test)	0.22	0.15	0.26	0.27
Recall (Test)	0.27	0.29	0.27	0.30
F1-Score (Test)	0.24	0.22	0.26	0.27
Accuracy (Test)	0.25	0.24	0.27	0.31
F1-Score (Validation)	0.27	0.24	0.34	0.29
Accuracy (Validation)	0.25	0.36	0.41	0.32

Table 5.2: Test data performance of a 3D CNN model for various sequence lengths input. The metrics are calculated as macro-averages.

Table 5.2 displays the generalization ability of the best-performing model on the validation data.

The 3D CNN models were tedious to train and challenging to converge. Therefore, we did not devote much time to further experimenting with 3D convolution models but instead shifted focus towards RCNN models.

5.1.4 RCNN

A great number of converging RCNN models were trained. Yet, most of the converging ones were models with a ResNet18 architecture as the image encoder, while there was a lower density of converging models with deeper ResNets and VGG models. Thus, we narrowed the documented results of this section only to models consisting of ResNet18 image encoders.

Table 5.3 displays one of the better-performing candidate models' test data performance for different input sequence lengths.

Sequence length	**7**	**15**	**23**	**29**
Precision (Test)	0.27	0.33	0.30	0.27
Recall (Test)	0.32	0.36	0.36	0.35
F1-Score (Test)	0.28	0.34	0.32	0.30
Accuracy (Test)	0.34	0.38	0.38	0.37
F1-Score (Validation)	0.35	0.41	0.37	0.40
Accuracy (Validation)	0.40	0.44	0.38	0.41

Table 5.3: Results of an RCNN candidate model. The metrics are calculated as macro-averages.

As seen in figure 5.3, the test scores deviate from the validation score. For further insight, figure 5.2 illustrates that the model does substantially better on the positive classes than the negative, which mainly seems to be ignored.

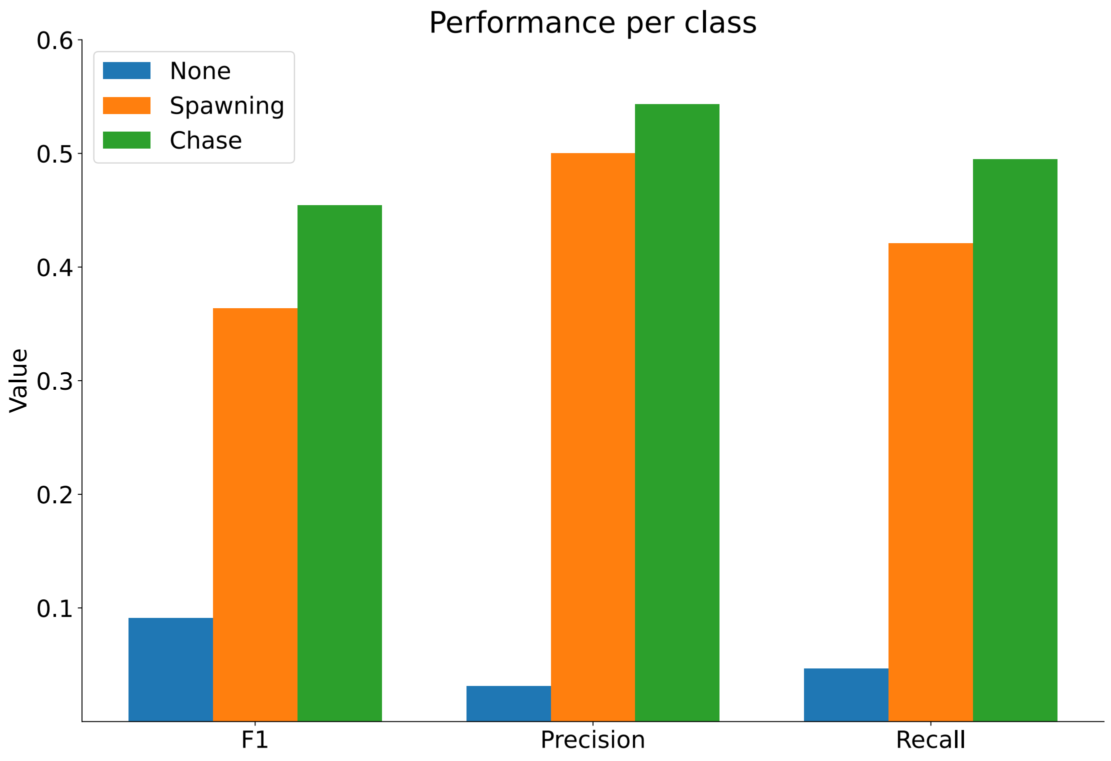


Figure 5.2: Performance per class with sequence length=29.

However, figure 5.2 only displays one single model on a small number of samples. Figure 5.3 and 5.4 visualize some general performance patterns across twenty candidate models.

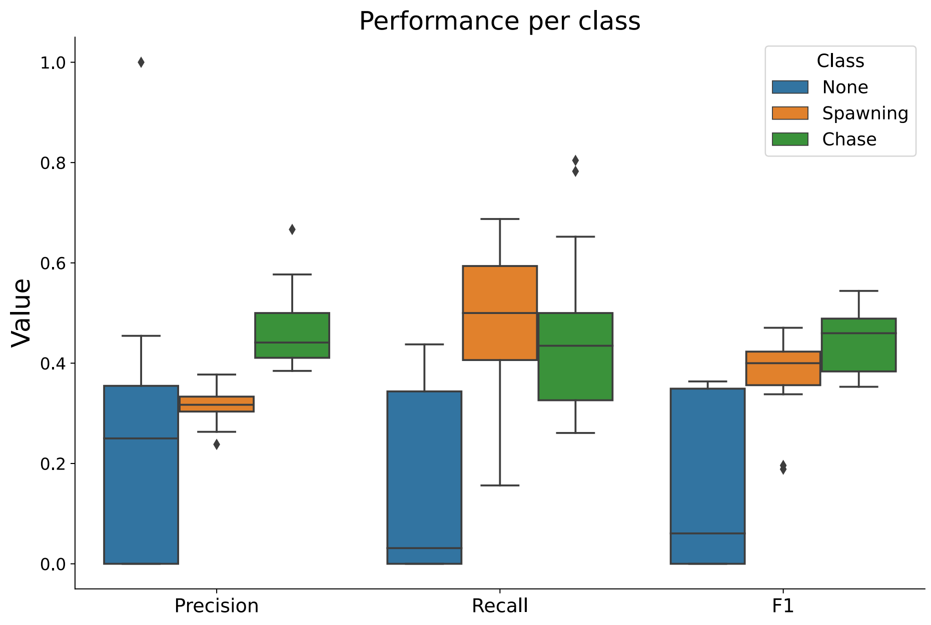


Figure 5.3: Box plot of performance per class for 20 candidate models, with sequence length = 29.

Figure 5.3 shows that the models broadly ignore the negative class. In terms of F1-score, spawning and chase are nearly similar, but there tends to be higher precision for chase and slightly higher recall for spawning.

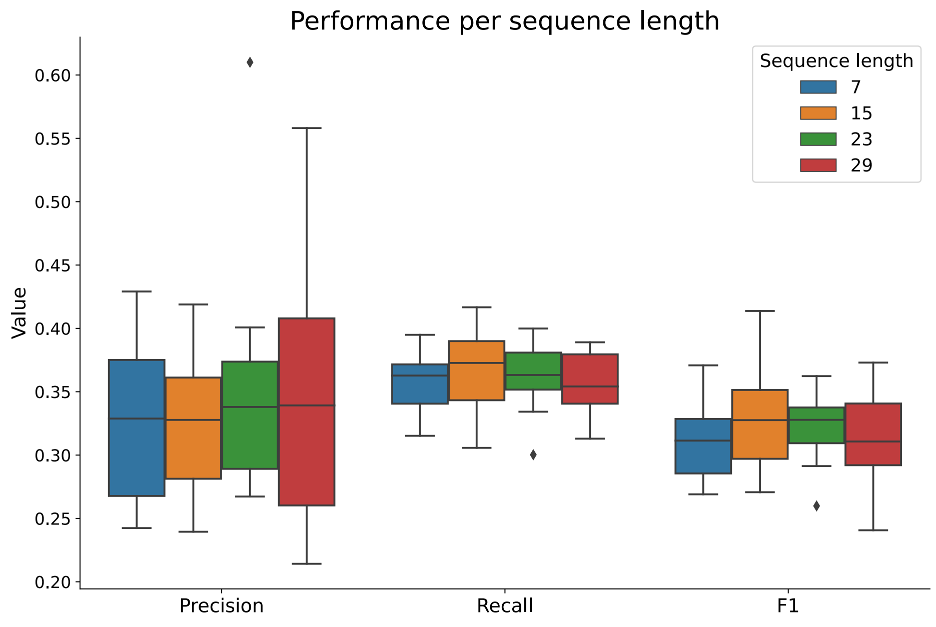


Figure 5.4: Box plot of performance for different input sequence lengths.

Figure 5.4 suggests there is hardly any variation in performance despite increasing the sequence length of the inputs. F1-score of all models in figure 5.4 can be found in appendix A.2.

Image Encoder Analysis

Despite insufficient generalization abilities of the RCNN architecture, as shown in 5.3, we exclusively analyzed if the model could capture any meaningful spatial features by visualization. The dimensions of the representations were reduced by applying t-SNE (Section 2.6), and visualized for different time steps of the sequences. The t-SNE analysis was set up for the same candidate model as in table 5.3, where the image encoder component consisted of a ResNet18.

Figures 5.5 and 5.6 visualize the representations correspondingly for the test and training data for different time steps of the input sequences. The value of t denotes the time step relative to the center of a sequence. Naturally, this corresponds to the defined starting point of an event for positive classes.

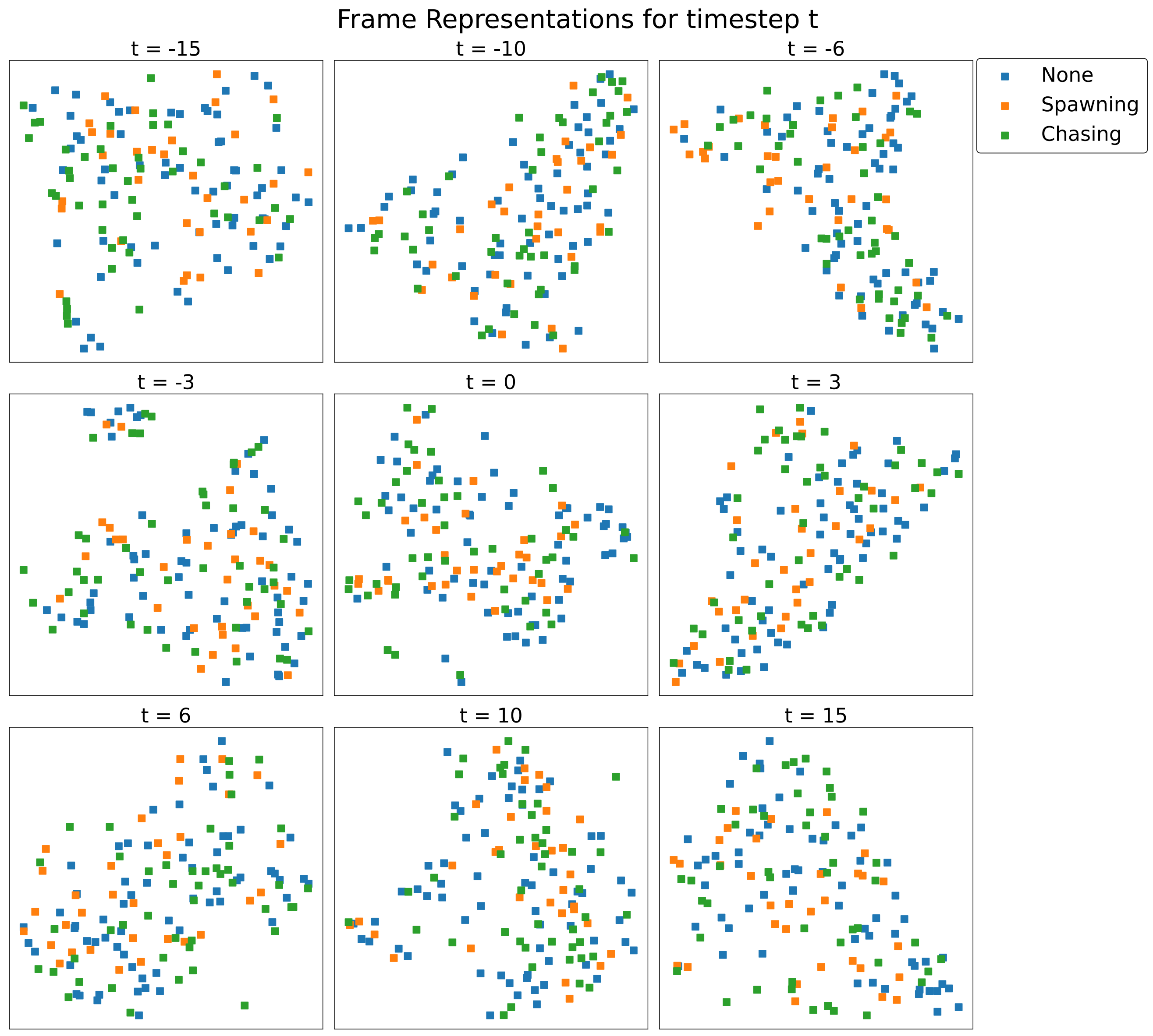


Figure 5.5: Test data visualized as two-dimensional image representations. Dimensionality reduced from 512 to 2 with t-SNE.

From figure 5.5, it is elusive to observe clusters from the test data. In figure 5.6, we visualize the training data representations for comparison. Here we can observe more prominent clusters, especially for the two positive classes. However, there tends to be more chaotic the longer distance from t=0. The model seems to separate the two positive classes well but struggles with the negative. This emphasizes the previous results (figure 5.2), where the model appears to neglect the negative class.

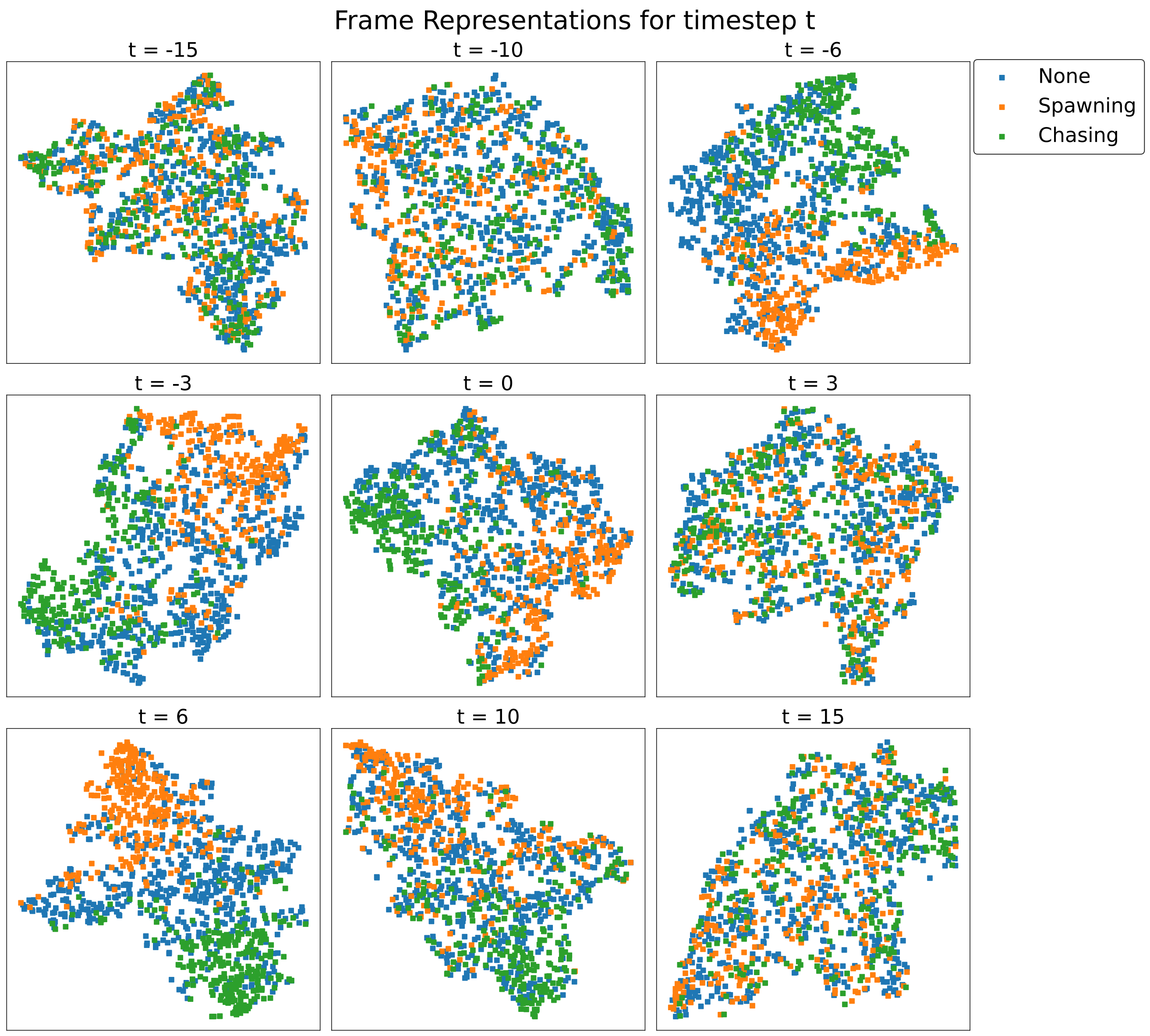


Figure 5.6: Training data visualized as two-dimensional image representations. Dimensionality reduced from 512 to 2 with t-SNE.

We also investigate if the environment and background may be significant when computing the image representations by visualizing the test data again, but now the colors represent videos the data points belong to.

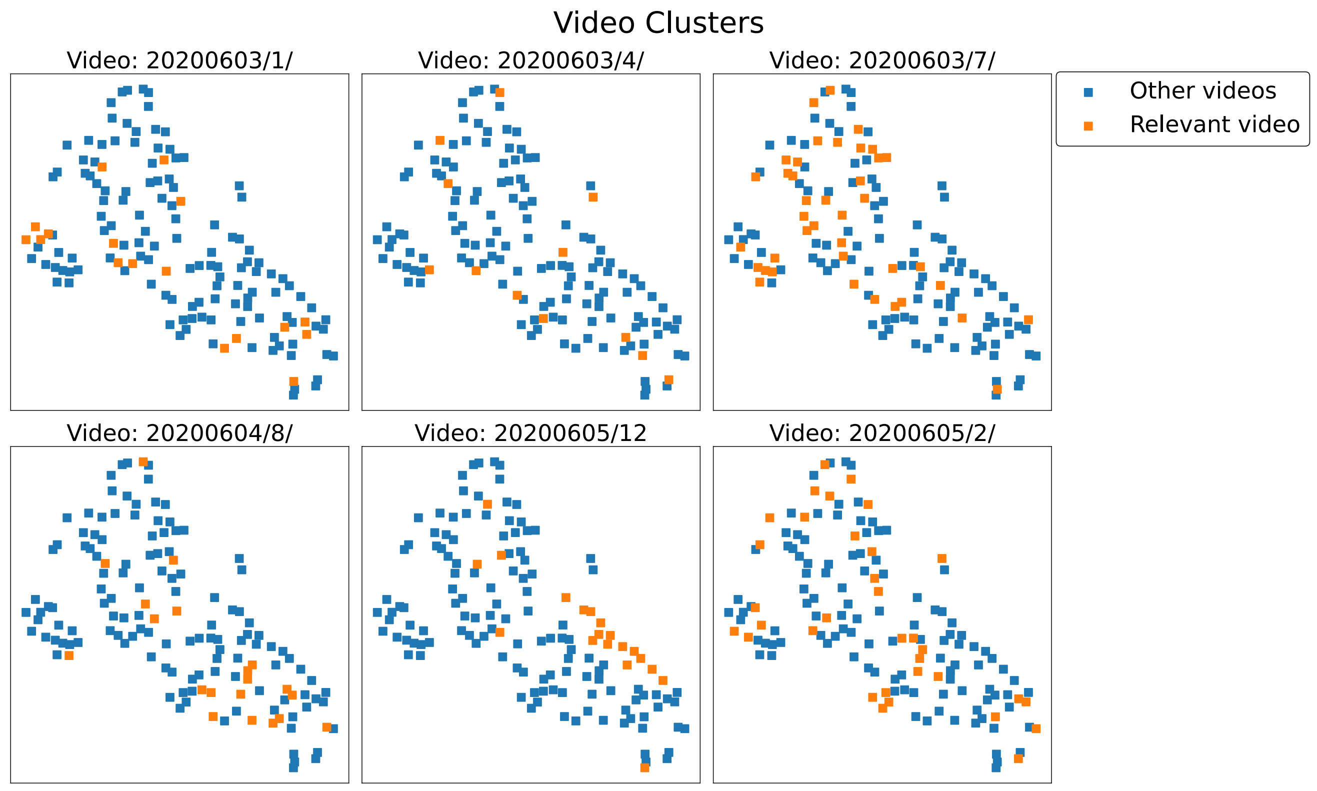


Figure 5.7: Frame representations clustered for videos. Each video has a corresponding plot where colored one vs. all, and hence all plots have similar data points. All samples are captured at t=0 in the sequences.

Figure 5.7 illustrates no well-defined clusters, despite in video 20200605/12. This is the same video illustrated in figure 4.4b, as an example of long-distance from camera to nest.

6 | Discussion

The purpose of this study was to explore deep learning methods for recognizing behavior events of corkwing wrasse in video data from a shallow-water ecosystem on the Norwegian west coast. This analysis supports the expectation that classification by recognizing distinct behaviors from shifts of elusive poses is challenging, especially with a modest amount of data. The 3D CNN model managed to achieve an F1-score of 0.27 (table 5.2), while the RCNN performed slightly better with 0.34 (table 5.3). The performance was not considerably amended despite altering the sequence lengths (figure 5.4). The insufficient generalization abilities are emphasized further by that none of the models could reduce the validation loss (figure 5.1) to any considerable extent. This demonstrates the difficulties in classifying short-spanning behaviors due to the inability to extract discriminative spatial features needed to distinguish infrequent events from natural behavior. The bottleneck of our system seems to be the extraction of useful spatial features, illustrated in figure 5.5. Hence, the analysis of temporal characteristics will be less profound without solid spatial input features. The results suggest that overfitting is hard to elude for our limited dataset, and deeper models struggle to converge.

Contrary to our ambitions, the performance of our models was not sufficient for use in real-life applications due to inferior generalization abilities. Neither the 3D CNN (5.2) nor the RCNN (5.3) models showed satisfactory predictive power for unseen data. Usually, the low-quality F1-scores in our results (figure 5.2, 5.3) relate to low precision and recall for the negative classes. This is because the models cannot detect discriminative features and instead often ignores a class and seemingly treats the problem as a binary classification task. Presumably, the models mostly ignored the negative classes because the candidate models were selected considering their validation performance for the positive classes.

Table 5.3 and figure 5.4 exhibit that increasing the sequence length does not affect the performance. The plausible explanation is the lack of sufficient visual features for the sequence encoder to utilize, although similar studies have shown that introducing temporal dynamics increases performance [8]. This is further substantiated by visualizing the image representations in figure 5.5, indicating that no distinct spatial features are perceived for the behaviors. Thus the impact of extracting temporal dynamics will diminish. Even when visualizing the low-dimensional image representations of the training data (5.6), it appears that even the overfitted model struggles to discriminate the negative class from the two positive ones, underlining the challenge of recognizing relevant visual characteristics in this task.

The reason for insufficient generalization abilities can be multifactorial, with many moving objects like architecture, configurations, and data preprocessing steps. The results could suggest that we have used unsuitable methods for the task. However, methods used with a CNN + LSTM and 3D convolutions for action recognition have previously shown promising results in similar studies [8, 56, 40, 12]. For this reason, it is plausible to assume our data is insubstantial to solve the proposed task for now. Foremost, the quantity of data used in this study is moderate compared to Måløy et al. [40] and Conway et al. [8], with around 250 data points per positive class. Additionally, the videos are captured from 12 different nests, with a skewed distribution of positive classes between them. Hence, the models are less robust in differentiating background noise and relevant data. Nevertheless, figure 5.7b does not display distinct clusters, implying that visual features are not computed exclusively from background features. However, it is worth noting that representations from one of the videos (20200605/12) were closely populated in the visualizations. This may indicate that the model computed visual features from the surroundings since the relevant details were harder to capture.

These results provide insight into the challenges and limitations of automated video analysis of marine ecological processes with data from natural habitats. In line with our concerns about natural noise and elusive poses in the introduction, this study further builds our understanding of the difficulties of employing deep learning methods for action recognition in marine science. These findings can be considered when assessing methods for collecting and labeling such data in the future to be applicable for developing automation systems.

The reliability of the model performance is inherently impacted by limited quantities of data in the validation and test set. Naturally, the results are presented assuming that the validation- and test data are a reasonable estimate of unseen real-life data, but this assumption does not necessarily hold. Especially for such few data points, the data will be prone to external biases. The convergence plot in 5.1 illustrates the fluctuating validation performance during training; hence it will impact the quantitative results due to a high standard error and perhaps not represent accurate measures. However, deviating measures are less impactful as the models are not sufficiently good to be applied in practice but rather provide insight into the general challenges of such systems.

The choice of model architectures in this project was constrained by limited time and GPU capacity; thus, we experimented with the ones shown promise in similar applications [8]. The methods used in this work are some of the most common building blocks in action recognition tasks today. Still, there exist other intriguing approaches involving multi-stream networks [25] and optical flow [51], and more recent methods like attention-based models [3, 4], which could be applied in future research. Ideally, we would like to investigate the temporal aspects of the systems further, but the incomprehensive visual features made temporal analysis less engaging. We also assumed that the methodical choice of extracting ten fps of the videos was sufficient, but we did not investigate increasing the fps.

7 | Further Work and Conclusion

In light of our findings, the most prominent approach to increasing such models' capabilities is to increase the magnitude of data. Consequently, more training data would make a model robust to noisy and chaotic data from numerous environments. Moreover, increased validation and test data would make the tuning and evaluation process more reliable since it will mitigate external biases and better resemble real-world data. Although, data collection is expensive and time-consuming. Therefore, a suggestion for further research is to reduce the sequences' spatial dimensions by cropping the region of interest rather than using the whole frame during training. This would require some additional preparation work, but considering cameras are static throughout the videos, the nest will be in the same position for every frame in the same video. Hence, cropping could be done manually video-by-video as a data preparation step. Nevertheless, a neater approach going forward would be to add some positional coordinates of relevant video content in future annotating work, for example, the spatial positions of the nests. Considering the videos are static, this would be an effortless but probably effective act. Long-term, such annotations could also be helpful in developing an object detection model, automating this process. Smaller frames could allow using higher resolution images during training, which may add important information when detecting subtle actions in noisy environments. Another idea for further studies could be to pre-train the model on some state behavior classes since containing considerably more samples. Then the weights could be tuned to fit the point behavior events afterward. This might enhance the models' fundament to exclude noisy features since the relevance of the samples will mainly be in the spatial location of where the fish occur, independent if state- or point behaviors.

This research aimed to investigate the applicability of deep learning methods to classify behavioral events of corkwing wrasse from underwater videos in its natural habitat. Based on qualitative and quantitative analyses of the proposed models, we can conclude that wild underwater videos present a challenge. The video data includes lots of redundant data, which complicates recognizing elusive, short-spanning poses. This research clearly illustrates that redundancy and noise in such ecological data need to be drastically reduced without big scales of training data in order to replace manual analysis.

List of Acronyms and Abbreviations

AI	Artificial Intelligence.
ANN	Artificial Neural Networks.
BPTT	Backpropagation Through Time.
BRNN	Bidirectional Recurrent Neural Networks.
CE	Cross-Entropy.
CMR	capture-mark-recapture techniques.
CNN	Convolutional Neural Networks.
FN	False Negative.
FP	False Positive.
IMR	Institute of Marine Research.
LSTM	Long Short-Term Memory.
ML	Machine Learning.
MLP	Multilayer Perceptron.
MSE	Mean Squared Error.
NLP	Natural Language Processing.
PCA	Principal Component Analysis.
RCNN	Recurrent Convolutional Neural Network.
RNN	Recurrent Neural Networks.
SDG	Stochastic Gradient Descent.
SGD	Stochastic Gradient Descent.
t-SNE	t-Distributed Stochastic Neighbor embedding.
TN	True Negative.
TP	True Positive.

Bibliography

[1] Directorate of fisheries, 2021.
URL: `https://www.fiskeridir.no/English/Aquaculture/Statistics/Cleanerfish-Lumpfish-and-Wrasse`. [Date visited: 11, May 2022].

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[3] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6836–6846, 2021.

[4] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding. *arXiv preprint arXiv:2102.05095*, 2(3):4, 2021.

[5] Cigdem Beyan, Vasiliki-Maria Katsageorgiou, and Robert Fisher. Extracting statistically significant behaviour from fish tracking data with and without large dataset cleaning. *IET Computer Vision*, 12, 11 2017. doi: 10.1049/iet-cvi.2016.0462.

[6] Alex Clark. Pillow (pil fork) documentation, 2015.
URL: `https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf`. [Date visited: 24. October 2021].

[7] Justin Class imbalance, Johnson and Taghi Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6:27, 03 2019. doi: 10.1186/s40537-019-0192-5.

[8] Alexander M Conway, Ian N Durbach, Alistair McInnes, and Robert N Harris. Frame-by-frame annotation of video recordings using deep neural networks. *Ecosphere*, 12(3): e03384, 2021.

[9] WRT Darwall, MJ Costello, R Donnelly, and S Lysaght. Implications of life-history strategies for a new wrasse fishery. *Journal of Fish Biology*, 41:111–123, 1992.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[11] Ellen M Ditria, Michael Sievers, Sebastian Lopez-Marcano, Eric L Jinks, and Rod M Connolly. Deep learning for automated analysis of fish abundance: the benefits of training across multiple habitats. *Environmental Monitoring and Assessment*, 192(11): 1–8, 2020.

[12] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[13] Yong Du, Yun Fu, and Liang Wang. Skeleton based action recognition with convolutional neural network. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 579–583, 2015. doi: 10.1109/ACPR.2015.7486569.

[14] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1110–1118, 2015. doi: 10.1109/CVPR.2015.7298714.

[15] Ben Ellis. Fisheries driven changes in the population structure of the corkwing wrasse (*Symphodus melops*) have negative repercussions for male egg care behaviour. 2021.

[16] Olivier Friard and Marco Gamba. Boris: a free, versatile open-source event-logging software for video/audio coding and live observations. *Methods in ecology and evolution*, 7(11):1325–1330, 2016.

[17] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[18] Morten Goodwin, Kim Tallaksen Halvorsen, Lei Jiao, Kristian Muri Knausgård, Angela Helen Martin, Marta Moyano, Rebekah A Oomen, Jeppe Have Rasmussen, Tonje Knutsen Sørdalen, and Susanna Huneide Thorbjørnsen. Unlocking the potential of deep learning for marine ecology: overview, applications, and outlook. *ICES Journal of Marine Science*, 79(2):319–336, 2022.

[19] Kim Tallaksen Halvorsen, Tonje Knutsen Sørdalen, Leif Asbjørn Vøllestad, Anne Berit Skiftesvik, Sigurd Heiberg Espeland, and Esben Moland Olsen. Sex-and size-selective harvesting of corkwing wrasse (symphodus melops)—a cleaner fish used in salmonid aquaculture. *ICES Journal of Marine Science*, 74(3):660–669, 2017.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[24] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013. doi: 10.1109/TPAMI.2012.59.

[25] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[26] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[27] Holly K Kindsvater, Kim Tallaksen Halvorsen, Tonje Knutsen Sørdalen, and Suzanne H Alonzo. The consequences of size-selective fishing mortality for larval production and sustainable yield in species with obligate male care. *Fish and Fisheries*, 21(6):1135–1149, 2020.

[28] Kristian Muri Knausgård, Arne Wiklund, Tonje Knutsen Sørdalen, Kim Tallaksen Halvorsen, Alf Ring Kleiven, Lei Jiao, and Morten Goodwin. Temperate fish detection and classification: A deep learning based approach. *Applied Intelligence*, pages 1–14, 2021.

[29] Yu Kong and Yun Fu. Human action recognition and prediction: A survey, 2018.
URL: `https://arxiv.org/abs/1806.11230`.

[30] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. Data preprocessing for supervised leaning. *International journal of computer science*, 1(2):111–117, 2006.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, pages 253–256. IEEE, 2010.

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[34] Sebastian Lopez-Marcano, Eric L. Jinks, Christina A. Buelow, Christopher J. Brown, Dadong Wang, Branislav Kusy, Ellen M. Ditria, and Rod M. Connolly. Automatic detection of fish and tracking of movement for ecology. *Ecology and Evolution*, 11(12): 8254–8263, 2021. doi: https://doi.org/10.1002/ece3.7656.
URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/ece3.7656`.

[35] Vanesa Lopez-Vazquez, Jose Manuel Lopez-Guede, Simone Marini, Emanuela Fanelli, Espen Johnsen, and Jacopo Aguzzi. Video image enhancement and machine learning pipeline for underwater animal detection and classification at cabled observatories. *Sensors*, 20(3):726, 2020.

[36] Mark C Mainwaring, Iain Barber, Denis C Deeming, David A Pike, Elizabeth A Roznik, and Ian R Hartley. Climate change and nesting behaviour in vertebrates: a review of the ecological threats and potential for adaptive responses. *Biological Reviews*, 92(4): 1991–2002, 2017.

[37] Ketil Malde, Nils Olav Handegard, Line Eikvil, and Arnt-Børre Salberg. Machine intelligence and the data-driven future of marine science. *ICES Journal of Marine Science*, 77(4):1274–1285, 04 2019. ISSN 1054-3139. doi: 10.1093/icesjms/fsz057.
URL: `https://doi.org/10.1093/icesjms/fsz057`.

[38] Ranju Mandal, Rod M Connolly, Thomas A Schlacher, and Bela Stantic. Assessing fish abundance from underwater video using deep neural networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2018.

[39] Tomas Mikolov. Recurrent neural network based language model. 2010.

[40] Håkon Måløy, Agnar Aamodt, and Ekrem Misimi. A spatio-temporal recurrent network for salmon feeding action recognition from underwater videos in aquaculture. *Computers and Electronics in Agriculture*, 167:105087, 2019. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2019.105087.
URL: `https://www.sciencedirect.com/science/article/pii/S0168169919313262`.

[41] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[43] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classifi-

cation using deep learning. *CoRR*, abs/1712.04621, 2017.
URL: `http://arxiv.org/abs/1712.04621`.

[44] Geoffrey W Potts. The nest structure of the corkwing wrasse, crenilabrus melops (labridae: Teleostei). *Journal of the Marine Biological Association of the United Kingdom*, 65(2):531–546, 1985.

[45] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. doi: 10.1109/78.650093.

[46] M. Shanker, M.Y. Hu, and M.S. Hung. Effect of data standardization on neural network training. *Omega*, 24(4):385–397, 1996. ISSN 0305-0483. doi: https://doi.org/10.1016/0305-0483(96)00010-2.
URL: `https://www.sciencedirect.com/science/article/pii/0305048396000102`.

[47] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

[48] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[49] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 6:60, 2019. doi: 10.1186/s40537-019-0197-0.
URL: `https://doi.org/10.1186/s40537-019-0197-0`.

[50] Shoaib Siddiqui, Ahmad Salman, Imran Malik, Faisal Shafait, Ajmal Mian, Mark Shortis, and Euan Harvey. Automatic fish species classification in underwater videos: Exploiting pretrained deep neural network models to compensate for limited labelled data. *ICES Journal of Marine Science*, 75, 05 2017. doi: 10.1093/icesjms/fsx109.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] Bharat Singh, Tim K Marks, Michael Jones, Oncel Tuzel, and Ming Shao. A multi-stream bi-directional recurrent neural network for fine-grained action detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1961–1970, 2016.

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[54] SuperDataScience. Recurrent neural networks (rnn) - the vanishing gradient problem, August 2018.
URL: `https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem`. [Online; posted 23-August-2018].

[55] Suramya Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006.

[56] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[57] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. *CoRR*, abs/1711.11248, 2017.
URL: `http://arxiv.org/abs/1711.11248`.

[58] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[59] Sébastien Villon, David Mouillot, Marc Chaumont, Emily S Darling, Gérard Subsol, Thomas Claverie, and Sébastien Villéger. A deep learning method for accurate and fast identification of coral reef fishes in underwater images. *Ecological informatics*, 48: 238–244, 2018.

[60] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

Appendix A

Supplementary results

The F1-scores for all twenty models from figure 5.4 are provided in table A.2. To fit the page, hyperparameter names had to be abbreviated in the results table. All abbreviations are described in table A.1

Hyperparameter	**Abbreviation**
Initial Learning Rate	LR
Image Representation Size	IRS
Batch Size	BS
CNN Model	CNN
LSTM Layers	LL
Momentum	M
LSTM Layer Size	LS
Bidirectional LSTM	BD
Optimizer	OPT
Negative:Positive Labels Ratio	N:P

Table A.1: Hyperparameter abbreviations.

	Sequence Length				Hyperparameters									
Model	**7**	**15**	**23**	**29**	**LR**	**IRS**	**BS**	**CNN**	**LL**	**M**	**LS**	**BD**	**Opt**	**N:P**
1	0.32	0.31	0.36	0.37	6.21E-05	1024	2	ResNet18	2	0.5	512	True	sdg	2
2	0.26	0.28	0.26	0.23	0.093222885	512	2	ResNet18	1	0.99	1024	True	sdg	2
3	0.34	0.41	0.3	0.26	0.01322793	512	1	ResNet18	1	0.5	1024	False	adam	3
4	0.29	0.24	0.32	0.31	0.0053177	1024	4	ResNet18	2	0.9	512	True	sdg	2
5	0.31	0.24	0.29	0.24	0.001777896	1024	1	ResNet18	1	0.5	1024	True	sdg	2
6	0.37	0.43	0.34	0.34	0.001365884	512	4	ResNet18	1	0.9	512	False	sdg	1
7	0.27	0.27	0.37	0.32	0.00019696	1024	1	ResNet18	1	0.99	512	True	sdg	3
8	0.29	0.26	0.33	0.29	0.093222885	512	2	ResNet18	1	0.99	1024	True	sdg	2
9	0.28	0.29	0.26	0.29	0.001365884	512	4	ResNet18	1	0.9	512	False	sdg	1
10	0.28	0.34	0.24	0.22	0.011507921	512	2	ResNet18	2	0.9	1024	False	sdg	1
11	0.29	0.23	0.25	0.39	0.000445648	512	2	ResNet18	2	0.5	1024	False	adam	1
12	0.37	0.4	0.34	0.3	0.006734113	512	2	ResNet18	2	0.99	512	True	sdg	2
13	0.34	0.39	0.39	0.39	0.015186374	1024	4	ResNet18	2	0.5	1024	False	adam	1
14	0.27	0.37	0.39	0.39	0.000274927	1024	2	ResNet18	2	0.99	512	False	sdg	1
15	0.27	0.32	0.37	0.29	1.02E-05	512	2	ResNet18	1	0.5	512	True	sdg	2
16	0.32	0.34	0.34	0.33	0.000112352	512	2	ResNet18	2	0.99	1024	True	sdg	3
17	0.32	0.33	0.36	0.38	0.00019696	1024	1	ResNet18	1	0.99	512	True	sdg	3
18	0.31	0.34	0.32	0.27	0.001574565	512	4	ResNet18	1	0.5	1024	True	sdg	2
19	0.23	0.27	0.27	0.29	0.040854785	512	2	ResNet18	1	0.99	512	False	adam	5
20	0.36	0.43	0.32	0.32	4.58E-05	1024	4	ResNet18	2	0.99	512	True	sdg	2

Table A.2: F1-Scores of various model configurations.

Appendix B

Model Architectures

B.1 Image Encoders

Summaries for the image encoder architectures can be seen below. For all image encoder architectures listed, the input have the shape (channels=3, height=540, width=960) and the output size is 1024, corresponding to the specified image representation size.

B.1.1 ResNet18

Listing B.1: ResNet18

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 270, 480]           9,408
       BatchNorm2d-2         [-1, 64, 270, 480]             128
              ReLU-3         [-1, 64, 270, 480]               0
         MaxPool2d-4         [-1, 64, 135, 240]               0
            Conv2d-5         [-1, 64, 135, 240]          36,864
       BatchNorm2d-6         [-1, 64, 135, 240]             128
              ReLU-7         [-1, 64, 135, 240]               0
            Conv2d-8         [-1, 64, 135, 240]          36,864
       BatchNorm2d-9         [-1, 64, 135, 240]             128
             ReLU-10         [-1, 64, 135, 240]               0
       BasicBlock-11         [-1, 64, 135, 240]               0
           Conv2d-12         [-1, 64, 135, 240]          36,864
      BatchNorm2d-13         [-1, 64, 135, 240]             128
             ReLU-14         [-1, 64, 135, 240]               0
           Conv2d-15         [-1, 64, 135, 240]          36,864
      BatchNorm2d-16         [-1, 64, 135, 240]             128
             ReLU-17         [-1, 64, 135, 240]               0
       BasicBlock-18         [-1, 64, 135, 240]               0
           Conv2d-19         [-1, 128, 68, 120]          73,728
       BatchNorm2d-20          [-1, 128, 68, 120]             256
              ReLU-21          [-1, 128, 68, 120]               0
            Conv2d-22          [-1, 128, 68, 120]         147,456
       BatchNorm2d-23          [-1, 128, 68, 120]             256
            Conv2d-24          [-1, 128, 68, 120]           8,192
       BatchNorm2d-25          [-1, 128, 68, 120]             256
              ReLU-26          [-1, 128, 68, 120]               0
        BasicBlock-27          [-1, 128, 68, 120]               0
            Conv2d-28          [-1, 128, 68, 120]         147,456
       BatchNorm2d-29          [-1, 128, 68, 120]             256
              ReLU-30          [-1, 128, 68, 120]               0
            Conv2d-31          [-1, 128, 68, 120]         147,456
       BatchNorm2d-32          [-1, 128, 68, 120]             256
              ReLU-33          [-1, 128, 68, 120]               0
        BasicBlock-34          [-1, 128, 68, 120]               0
            Conv2d-35           [-1, 256, 34, 60]         294,912
       BatchNorm2d-36           [-1, 256, 34, 60]             512
              ReLU-37           [-1, 256, 34, 60]               0
            Conv2d-38           [-1, 256, 34, 60]         589,824
       BatchNorm2d-39           [-1, 256, 34, 60]             512
            Conv2d-40           [-1, 256, 34, 60]          32,768
       BatchNorm2d-41           [-1, 256, 34, 60]             512
              ReLU-42           [-1, 256, 34, 60]               0
        BasicBlock-43           [-1, 256, 34, 60]               0
            Conv2d-44           [-1, 256, 34, 60]         589,824
       BatchNorm2d-45           [-1, 256, 34, 60]             512
              ReLU-46           [-1, 256, 34, 60]               0
            Conv2d-47           [-1, 256, 34, 60]         589,824
       BatchNorm2d-48           [-1, 256, 34, 60]             512
              ReLU-49           [-1, 256, 34, 60]               0
        BasicBlock-50           [-1, 256, 34, 60]               0
            Conv2d-51           [-1, 512, 17, 30]       1,179,648
       BatchNorm2d-52           [-1, 512, 17, 30]           1,024
              ReLU-53           [-1, 512, 17, 30]               0
            Conv2d-54           [-1, 512, 17, 30]       2,359,296
       BatchNorm2d-55           [-1, 512, 17, 30]           1,024
            Conv2d-56           [-1, 512, 17, 30]         131,072
       BatchNorm2d-57           [-1, 512, 17, 30]           1,024
              ReLU-58           [-1, 512, 17, 30]               0
        BasicBlock-59           [-1, 512, 17, 30]               0
            Conv2d-60           [-1, 512, 17, 30]       2,359,296
       BatchNorm2d-61           [-1, 512, 17, 30]           1,024
              ReLU-62           [-1, 512, 17, 30]               0
            Conv2d-63           [-1, 512, 17, 30]       2,359,296
       BatchNorm2d-64           [-1, 512, 17, 30]           1,024
              ReLU-65           [-1, 512, 17, 30]               0
        BasicBlock-66           [-1, 512, 17, 30]               0
AdaptiveAvgPool2d-67             [-1, 512, 1, 1]               0
            Linear-68                  [-1, 1024]         525,312
================================================================
Total params: 11,701,824
Trainable params: 11,701,824
Non-trainable params: 0
----------------------------------------------------------------
```

B.1.2 ResNet50

Listing B.2: ResNet50

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 270, 480]           9,408
       BatchNorm2d-2         [-1, 64, 270, 480]             128
              ReLU-3         [-1, 64, 270, 480]               0
         MaxPool2d-4         [-1, 64, 135, 240]               0
            Conv2d-5         [-1, 64, 135, 240]           4,096
       BatchNorm2d-6         [-1, 64, 135, 240]             128
              ReLU-7         [-1, 64, 135, 240]               0
            Conv2d-8         [-1, 64, 135, 240]          36,864
       BatchNorm2d-9         [-1, 64, 135, 240]             128
             ReLU-10         [-1, 64, 135, 240]               0
           Conv2d-11        [-1, 256, 135, 240]          16,384
      BatchNorm2d-12        [-1, 256, 135, 240]             512
           Conv2d-13        [-1, 256, 135, 240]          16,384
      BatchNorm2d-14        [-1, 256, 135, 240]             512
             ReLU-15        [-1, 256, 135, 240]               0
       Bottleneck-16        [-1, 256, 135, 240]               0
           Conv2d-17         [-1, 64, 135, 240]          16,384
      BatchNorm2d-18         [-1, 64, 135, 240]             128
             ReLU-19         [-1, 64, 135, 240]               0
           Conv2d-20         [-1, 64, 135, 240]          36,864
      BatchNorm2d-21         [-1, 64, 135, 240]             128
             ReLU-22         [-1, 64, 135, 240]               0
           Conv2d-23        [-1, 256, 135, 240]          16,384
      BatchNorm2d-24        [-1, 256, 135, 240]             512
             ReLU-25        [-1, 256, 135, 240]               0
       Bottleneck-26        [-1, 256, 135, 240]               0
           Conv2d-27         [-1, 64, 135, 240]          16,384
      BatchNorm2d-28         [-1, 64, 135, 240]             128
             ReLU-29         [-1, 64, 135, 240]               0
           Conv2d-30         [-1, 64, 135, 240]          36,864
      BatchNorm2d-31         [-1, 64, 135, 240]             128
             ReLU-32         [-1, 64, 135, 240]               0
           Conv2d-33        [-1, 256, 135, 240]          16,384
      BatchNorm2d-34        [-1, 256, 135, 240]             512
             ReLU-35        [-1, 256, 135, 240]               0
       Bottleneck-36        [-1, 256, 135, 240]               0
           Conv2d-37        [-1, 128, 135, 240]          32,768
      BatchNorm2d-38        [-1, 128, 135, 240]             256
             ReLU-39        [-1, 128, 135, 240]               0
           Conv2d-40         [-1, 128, 68, 120]         147,456
      BatchNorm2d-41         [-1, 128, 68, 120]             256
             ReLU-42         [-1, 128, 68, 120]               0
           Conv2d-43         [-1, 512, 68, 120]          65,536
      BatchNorm2d-44         [-1, 512, 68, 120]           1,024
           Conv2d-45         [-1, 512, 68, 120]         131,072
      BatchNorm2d-46         [-1, 512, 68, 120]           1,024
             ReLU-47         [-1, 512, 68, 120]               0
       Bottleneck-48         [-1, 512, 68, 120]               0
           Conv2d-49         [-1, 128, 68, 120]          65,536
      BatchNorm2d-50         [-1, 128, 68, 120]             256
             ReLU-51         [-1, 128, 68, 120]               0
           Conv2d-52         [-1, 128, 68, 120]         147,456
      BatchNorm2d-53         [-1, 128, 68, 120]             256
             ReLU-54         [-1, 128, 68, 120]               0
           Conv2d-55         [-1, 512, 68, 120]          65,536
      BatchNorm2d-56         [-1, 512, 68, 120]           1,024
             ReLU-57         [-1, 512, 68, 120]               0
       Bottleneck-58         [-1, 512, 68, 120]               0
```

```
           Conv2d-59          [-1, 128, 68, 120]          65,536
      BatchNorm2d-60          [-1, 128, 68, 120]             256
             ReLU-61          [-1, 128, 68, 120]               0
           Conv2d-62          [-1, 128, 68, 120]         147,456
      BatchNorm2d-63          [-1, 128, 68, 120]             256
             ReLU-64          [-1, 128, 68, 120]               0
           Conv2d-65          [-1, 512, 68, 120]          65,536
      BatchNorm2d-66          [-1, 512, 68, 120]           1,024
             ReLU-67          [-1, 512, 68, 120]               0
       Bottleneck-68          [-1, 512, 68, 120]               0
           Conv2d-69          [-1, 128, 68, 120]          65,536
      BatchNorm2d-70          [-1, 128, 68, 120]             256
             ReLU-71          [-1, 128, 68, 120]               0
           Conv2d-72          [-1, 128, 68, 120]         147,456
      BatchNorm2d-73          [-1, 128, 68, 120]             256
             ReLU-74          [-1, 128, 68, 120]               0
           Conv2d-75          [-1, 512, 68, 120]          65,536
      BatchNorm2d-76          [-1, 512, 68, 120]           1,024
             ReLU-77          [-1, 512, 68, 120]               0
       Bottleneck-78          [-1, 512, 68, 120]               0
           Conv2d-79          [-1, 256, 68, 120]         131,072
      BatchNorm2d-80          [-1, 256, 68, 120]             512
             ReLU-81          [-1, 256, 68, 120]               0
           Conv2d-82           [-1, 256, 34, 60]         589,824
      BatchNorm2d-83           [-1, 256, 34, 60]             512
             ReLU-84           [-1, 256, 34, 60]               0
           Conv2d-85          [-1, 1024, 34, 60]         262,144
      BatchNorm2d-86          [-1, 1024, 34, 60]           2,048
           Conv2d-87          [-1, 1024, 34, 60]         524,288
      BatchNorm2d-88          [-1, 1024, 34, 60]           2,048
             ReLU-89          [-1, 1024, 34, 60]               0
       Bottleneck-90          [-1, 1024, 34, 60]               0
           Conv2d-91           [-1, 256, 34, 60]         262,144
      BatchNorm2d-92           [-1, 256, 34, 60]             512
             ReLU-93           [-1, 256, 34, 60]               0
           Conv2d-94           [-1, 256, 34, 60]         589,824
      BatchNorm2d-95           [-1, 256, 34, 60]             512
             ReLU-96           [-1, 256, 34, 60]               0
           Conv2d-97          [-1, 1024, 34, 60]         262,144
             ReLU-99          [-1, 1024, 34, 60]               0
      Bottleneck-100          [-1, 1024, 34, 60]               0
          Conv2d-101           [-1, 256, 34, 60]         262,144
     BatchNorm2d-102           [-1, 256, 34, 60]             512
            ReLU-103           [-1, 256, 34, 60]               0
          Conv2d-104           [-1, 256, 34, 60]         589,824
     BatchNorm2d-105           [-1, 256, 34, 60]             512
            ReLU-106           [-1, 256, 34, 60]               0
          Conv2d-107          [-1, 1024, 34, 60]         262,144
     BatchNorm2d-108          [-1, 1024, 34, 60]           2,048
            ReLU-109          [-1, 1024, 34, 60]               0
      Bottleneck-110          [-1, 1024, 34, 60]               0
          Conv2d-111           [-1, 256, 34, 60]         262,144
     BatchNorm2d-112           [-1, 256, 34, 60]             512
            ReLU-113           [-1, 256, 34, 60]               0
          Conv2d-114           [-1, 256, 34, 60]         589,824
     BatchNorm2d-115           [-1, 256, 34, 60]             512
            ReLU-116           [-1, 256, 34, 60]               0
          Conv2d-117          [-1, 1024, 34, 60]         262,144
     BatchNorm2d-118          [-1, 1024, 34, 60]           2,048
            ReLU-119          [-1, 1024, 34, 60]               0
      Bottleneck-120          [-1, 1024, 34, 60]               0
          Conv2d-121           [-1, 256, 34, 60]         262,144
     BatchNorm2d-122          [-1, 256, 34, 60]             512
            ReLU-123          [-1, 256, 34, 60]               0
          Conv2d-124          [-1, 256, 34, 60]         589,824
     BatchNorm2d-125          [-1, 256, 34, 60]             512
            ReLU-126          [-1, 256, 34, 60]               0
          Conv2d-127         [-1, 1024, 34, 60]         262,144
     BatchNorm2d-128         [-1, 1024, 34, 60]           2,048
            ReLU-129         [-1, 1024, 34, 60]               0
      Bottleneck-130         [-1, 1024, 34, 60]               0
          Conv2d-131          [-1, 256, 34, 60]         262,144
     BatchNorm2d-132          [-1, 256, 34, 60]             512
            ReLU-133          [-1, 256, 34, 60]               0
          Conv2d-134          [-1, 256, 34, 60]         589,824
     BatchNorm2d-135          [-1, 256, 34, 60]             512
            ReLU-136          [-1, 256, 34, 60]               0
          Conv2d-137         [-1, 1024, 34, 60]         262,144
     BatchNorm2d-138         [-1, 1024, 34, 60]           2,048
            ReLU-139         [-1, 1024, 34, 60]               0
      Bottleneck-140         [-1, 1024, 34, 60]               0
          Conv2d-141          [-1, 512, 34, 60]         524,288
     BatchNorm2d-142          [-1, 512, 34, 60]           1,024
            ReLU-143          [-1, 512, 34, 60]               0
          Conv2d-144          [-1, 512, 17, 30]       2,359,296
     BatchNorm2d-145          [-1, 512, 17, 30]           1,024
            ReLU-146          [-1, 512, 17, 30]               0
          Conv2d-147         [-1, 2048, 17, 30]       1,048,576
     BatchNorm2d-148         [-1, 2048, 17, 30]           4,096
          Conv2d-149         [-1, 2048, 17, 30]       2,097,152
     BatchNorm2d-150         [-1, 2048, 17, 30]           4,096
            ReLU-151         [-1, 2048, 17, 30]               0
      Bottleneck-152         [-1, 2048, 17, 30]               0
          Conv2d-153          [-1, 512, 17, 30]       1,048,576
     BatchNorm2d-154          [-1, 512, 17, 30]           1,024
            ReLU-155          [-1, 512, 17, 30]               0
          Conv2d-156          [-1, 512, 17, 30]       2,359,296
     BatchNorm2d-157          [-1, 512, 17, 30]           1,024
            ReLU-158          [-1, 512, 17, 30]               0
          Conv2d-159         [-1, 2048, 17, 30]       1,048,576
     BatchNorm2d-160         [-1, 2048, 17, 30]           4,096
            ReLU-161         [-1, 2048, 17, 30]               0
      Bottleneck-162         [-1, 2048, 17, 30]               0
          Conv2d-163          [-1, 512, 17, 30]       1,048,576
     BatchNorm2d-164          [-1, 512, 17, 30]           1,024
            ReLU-165          [-1, 512, 17, 30]               0
          Conv2d-166          [-1, 512, 17, 30]       2,359,296
     BatchNorm2d-167          [-1, 512, 17, 30]           1,024
            ReLU-168          [-1, 512, 17, 30]               0
          Conv2d-169         [-1, 2048, 17, 30]       1,048,576
     BatchNorm2d-170         [-1, 2048, 17, 30]           4,096
            ReLU-171         [-1, 2048, 17, 30]               0
      Bottleneck-172         [-1, 2048, 17, 30]               0
AdaptiveAvgPool2d-173           [-1, 2048, 1, 1]               0
          Linear-174                 [-1, 1024]       2,098,176
================================================================
Total params: 25,606,208
Trainable params: 25,606,208
Non-trainable params: 0
----------------------------------------------------------------
```

B.1.3 VGG13

Listing B.3: VGG13

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 540, 960]           1,792
              ReLU-2         [-1, 64, 540, 960]               0
            Conv2d-3         [-1, 64, 540, 960]          36,928
              ReLU-4         [-1, 64, 540, 960]               0
         MaxPool2d-5         [-1, 64, 270, 480]               0
            Conv2d-6        [-1, 128, 270, 480]          73,856
              ReLU-7        [-1, 128, 270, 480]               0
            Conv2d-8        [-1, 128, 270, 480]         147,584
              ReLU-9        [-1, 128, 270, 480]               0
        MaxPool2d-10        [-1, 128, 135, 240]               0
           Conv2d-11        [-1, 256, 135, 240]         295,168
             ReLU-12        [-1, 256, 135, 240]               0
           Conv2d-13        [-1, 256, 135, 240]         590,080
             ReLU-14        [-1, 256, 135, 240]               0
        MaxPool2d-15         [-1, 256, 67, 120]               0
           Conv2d-16         [-1, 512, 67, 120]       1,180,160
             ReLU-17         [-1, 512, 67, 120]               0
           Conv2d-18         [-1, 512, 67, 120]       2,359,808
             ReLU-19         [-1, 512, 67, 120]               0
        MaxPool2d-20          [-1, 512, 33, 60]               0
           Conv2d-21          [-1, 512, 33, 60]       2,359,808
             ReLU-22          [-1, 512, 33, 60]               0
           Conv2d-23          [-1, 512, 33, 60]       2,359,808
             ReLU-24          [-1, 512, 33, 60]               0
        MaxPool2d-25          [-1, 512, 16, 30]               0
AdaptiveAvgPool2d-26            [-1, 512, 7, 7]               0
           Linear-27                 [-1, 4096]     102,764,544
             ReLU-28                 [-1, 4096]               0
          Dropout-29                 [-1, 4096]               0
           Linear-30                 [-1, 4096]      16,781,312
             ReLU-31                 [-1, 4096]               0
          Dropout-32                 [-1, 4096]               0
           Linear-33                 [-1, 1024]       4,195,328
================================================================
Total params: 133,146,176
Trainable params: 133,146,176
Non-trainable params: 0
----------------------------------------------------------------
```

B.2 ResNet18 3D

Listing B.4: ResNet18 3D with input shape (sequence length=25, channnels=3, height=540, width=960)

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv3d-1     [-1, 64, 25, 270, 480]          28,224
       BatchNorm3d-2     [-1, 64, 25, 270, 480]             128
```

```
              ReLU-3     [-1, 64, 25, 270, 480]               0
      Conv3DSimple-4     [-1, 64, 25, 270, 480]         110,592
       BatchNorm3d-5     [-1, 64, 25, 270, 480]             128
              ReLU-6     [-1, 64, 25, 270, 480]               0
      Conv3DSimple-7     [-1, 64, 25, 270, 480]         110,592
       BatchNorm3d-8     [-1, 64, 25, 270, 480]             128
              ReLU-9     [-1, 64, 25, 270, 480]               0
       BasicBlock-10     [-1, 64, 25, 270, 480]               0
     Conv3DSimple-11     [-1, 64, 25, 270, 480]         110,592
      BatchNorm3d-12     [-1, 64, 25, 270, 480]             128
             ReLU-13     [-1, 64, 25, 270, 480]               0
     Conv3DSimple-14     [-1, 64, 25, 270, 480]         110,592
      BatchNorm3d-15     [-1, 64, 25, 270, 480]             128
             ReLU-16     [-1, 64, 25, 270, 480]               0
       BasicBlock-17     [-1, 64, 25, 270, 480]               0
     Conv3DSimple-18    [-1, 128, 13, 135, 240]         221,184
      BatchNorm3d-19    [-1, 128, 13, 135, 240]             256
             ReLU-20    [-1, 128, 13, 135, 240]               0
     Conv3DSimple-21    [-1, 128, 13, 135, 240]         442,368
      BatchNorm3d-22    [-1, 128, 13, 135, 240]             256
           Conv3d-23    [-1, 128, 13, 135, 240]           8,192
      BatchNorm3d-24    [-1, 128, 13, 135, 240]             256
             ReLU-25    [-1, 128, 13, 135, 240]               0
       BasicBlock-26    [-1, 128, 13, 135, 240]               0
     Conv3DSimple-27    [-1, 128, 13, 135, 240]         442,368
      BatchNorm3d-28    [-1, 128, 13, 135, 240]             256
             ReLU-29    [-1, 128, 13, 135, 240]               0
     Conv3DSimple-30    [-1, 128, 13, 135, 240]         442,368
      BatchNorm3d-31    [-1, 128, 13, 135, 240]             256
             ReLU-32    [-1, 128, 13, 135, 240]               0
       BasicBlock-33    [-1, 128, 13, 135, 240]               0
     Conv3DSimple-34      [-1, 256, 7, 68, 120]         884,736
      BatchNorm3d-35      [-1, 256, 7, 68, 120]             512
             ReLU-36      [-1, 256, 7, 68, 120]               0
     Conv3DSimple-37      [-1, 256, 7, 68, 120]       1,769,472
      BatchNorm3d-38      [-1, 256, 7, 68, 120]             512
           Conv3d-39      [-1, 256, 7, 68, 120]          32,768
      BatchNorm3d-40      [-1, 256, 7, 68, 120]             512
             ReLU-41      [-1, 256, 7, 68, 120]               0
       BasicBlock-42      [-1, 256, 7, 68, 120]               0
     Conv3DSimple-43      [-1, 256, 7, 68, 120]       1,769,472
      BatchNorm3d-44      [-1, 256, 7, 68, 120]             512
             ReLU-45      [-1, 256, 7, 68, 120]               0
     Conv3DSimple-46      [-1, 256, 7, 68, 120]       1,769,472
      BatchNorm3d-47      [-1, 256, 7, 68, 120]             512
             ReLU-48      [-1, 256, 7, 68, 120]               0
       BasicBlock-49      [-1, 256, 7, 68, 120]               0
     Conv3DSimple-50       [-1, 512, 4, 34, 60]       3,538,944
      BatchNorm3d-51       [-1, 512, 4, 34, 60]           1,024
             ReLU-52       [-1, 512, 4, 34, 60]               0
     Conv3DSimple-53       [-1, 512, 4, 34, 60]       7,077,888
      BatchNorm3d-54       [-1, 512, 4, 34, 60]           1,024
           Conv3d-55       [-1, 512, 4, 34, 60]         131,072
      BatchNorm3d-56       [-1, 512, 4, 34, 60]           1,024
             ReLU-57       [-1, 512, 4, 34, 60]               0
       BasicBlock-58       [-1, 512, 4, 34, 60]               0
     Conv3DSimple-59       [-1, 512, 4, 34, 60]       7,077,888
      BatchNorm3d-60       [-1, 512, 4, 34, 60]           1,024
             ReLU-61       [-1, 512, 4, 34, 60]               0
     Conv3DSimple-62       [-1, 512, 4, 34, 60]       7,077,888
      BatchNorm3d-63       [-1, 512, 4, 34, 60]           1,024
             ReLU-64       [-1, 512, 4, 34, 60]               0
```

```
        BasicBlock-65        [-1, 512, 4, 34, 60]               0
AdaptiveAvgPool3d-66          [-1, 512, 1, 1, 1]               0
            Linear-67                     [-1, 3]           1,539
       VideoResNet-68                     [-1, 3]               0
================================================================
Total params: 33,167,811
Trainable params: 33,167,811
Non-trainable params: 0
----------------------------------------------------------------
```