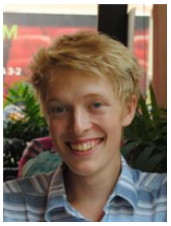
DTU Space
Institut for Rumforskning og -teknologi

Forecasting catch of Sprattus Sprattus in the North Sea using deep learning

30110 - ***Introductory project - Earth and Space Physics and Engineering***



Carl Emil Elling, s164032

Course: 30110 - Fagprojekt, Spring 2020
Technical University of Denmark (DTU), Kgs. Lyngby

Abstract

Predicting catch of the European sprat, Sprattus Sprattus, could optimize catch routes, in turn benefiting both climate and economic interests for fisheries, as well as providing a basis for sustainable fishery. This report explores the use of Neural Networks and Deep Learning as a forecasting mechanism for the European sprat, Sprattus Sprattus in the North Sea. Using data collected by commercial fisheries as well as environmental and hydrographic data, different models based on the Convolutional Long Short-Term Memory network architecture are build, captitalizing on recent advances in weather forecasting. The best model built in the report yielded a Zero-Normalized Cross Correlation of 0.1488. Here, 1 is perfect correlation and 0 is none, demonstrating that the model does indeed learn some correlation with the data. However, it is not nearly enough to accurately forecast, and more work is needed. Results indicate that some of this correlation can be explained by the model partially copying the last data in its input, making it even more unreliable. It was found that the data available was so sparse that overfitting was a big issue, especially with complex networks. This, combined with biased data, makes using state-of-the-art deep learning forecasting algorithms unfeasible on the given dataset.

Table of contents

List of abbreviations

ANN	**A**rtificial **N**eural **N**etwork
ARIMA	**A**uto**R**egressive **I**ntegrated **M**oving **A**verage
CNN	**C**onvolutional **N**eural **N**etwork
CPUE	**C**atch-**P**er-**U**nit-**E**ffort
DBN	**D**eep **B**elief **N**etwork
LSTM	**L**ong-**S**hort **T**erm **M**emory
MAE	**M**ean **A**bsolute **E**rror
MAPE	**M**ean **A**bsolute **P**ercentage Error
MSE	**M**ean **S**quared **E**rror
ReLU	**R**ectified **L**inear **U**nit
RMSE	**R**oot **M**ean **S**quared **E**rror
RNN	**R**ecurrent **N**eural **N**etwork
ZNCC	**Z**ero **N**ormalized **C**ross **C**orrelation
DMI	**D**anish **M**eterological **I**nstitute
VMS	**V**essel **M**onitoring **S**ystem

1 Introduction

A field in rapid development, fishery has not yet been modernized and digitalized to the extent many other commercial areas have. It is an old-fashioned profession, with fishing spots being explored by force of habit, experience and gut feel [1]. Due to mandatory collection and report of fishing data, fishery already has some infrastructure for the use of big data, and especially machine learning is of interest to examine latent distributions behind the collected data. The 2020's being the UN Decade of Ocean Science for Sustainable Development, there is a need for understanding the marine world better in order to learn how to fish sustainably and thus better protect ocean environments. As is stands now, the entire field has much to gain in use of big data, prompting the European Marine Board to recommend it as a possible solution to these issues [2].

This is also true of the European Sprat, a species of fish commercially used as fishmeal in livestock applications. There is a great potential for optimizing catch routes, both economically and environmentally. Shorter search times are associated with lower fuel usage, and information about population fluctuations could help avoid overfishing.

This leads to the research question of this project:

Can the catch of Sprattus Sprattus in the North Sea be predicted using deep learning?

In the pursuit of this, the following questions will be attempted answered:

Q1: *Are any environmental factors predictors for the occurence of sprat in a given place, and can they be used in a forecasting model?*

Q2: *Which biases exist in the data, and how can this be accounted for?*

2 Theory

2.1 Sprattus Sprattus

The research question leads to investigating the European Sprat itself and the current methodology for forecasting catch areas.

The European Sprat or Sprattus Sprattus is a common fish in the North Sea. It is a small fish, around 16cm in maximal length, and often mistaken for a young herring. Especially in adolescent stages, herring occur at the same areas as sprat, making bycatches a large problem in commercial sprat fisheries.

In the past few decades, commercial interest for the fish has risen, as it is primarily used for fish meal. This is used as protein supplement in fish and livestock feed, and due to the ever increasing demand for fish and livestock, the demand for sprat rises.[3] Denmark is a key contributor to this supply, as most of the sprat fishing in the North Sea is done by Danish vessels, accounting for around 25-40% of the total sprat catches [4].

When fishing sprat, fishermen usually use different environmental factors and hydrographical fronts to plan fishing trips, along with personal experience [1]. In recent years, larger, more efficient fishing vessels have been used in sprat fishery, but data sharing and data usage is mostly neglected. Even though larger vessels catch more fish, it is a field regulated by quotas, and not in immediate danger of overfishing. [4]

Historically however, it is a fish that has suffered large intra-annual variations. There is debate about the cause of this, with some evidence that overfishing of cod and the subsequent rise in feed and decrease in natural predators are the largest predictors for sprat occurence, at least in the Baltic sea. This left sprat and herring competing for zooplankton, a major food source, making the density of either a predictor for the other. A small percentage of the sprat variation can also be explained by salinity and salinity fronts, possibly due to zooplankton being highly affected by this [5].

2.1.1 Data

As much marine data is, the sprat catch data used in this project is arranged in a C-squares grid. C-squares is a spatial data format designed to be computationally inexpensive. As a 1D index of rastered data, it subdivides already established meteorological divisions of the world into smaller degrees of longitude and latitude. [6]

When catching small fish such as the sprat, economics of scale come into play. Since the price per kilo is rather low, fishing vessels only fish for them when there is likelihood of a large catch-per-unit-effort (CPUE). This is a unit of catch, given by the catches per hour for a vessel in a given C-square. This value has then been normalized to a 200 gross-tonnage vessel [1] The catches are mapped to a c-squares spatial grid based on one hour intervals of location data from the vessel monitoring system (VMS). It is important to note that CPUE does *not*

necessarily reflect the actual population exactly, but vary with a large number of parameters. In theory, CPUE is proportional to the actual population [7]:

$$CPUE = q \cdot N_t \tag{1}$$

where N_t is the abundance at time t and q is a catchability variable. However, the catchability rate varies with efficiency of fishing vessel, environment and a host of other factors. In the given dataset, it gives rise to an uncertainty in the final model. [1]

Due to the nature of CPUE data, it is only sampled in the area where the fishing vessel is, making the catch data spotty. That imparts a heavy bias on certain spots favored by fishermen. Along with a variety of other factors this makes conventional modelling and forecasting difficult.[3]

The complexity of the data-bias on the catch data is even greater as, until 2017, the so-called "Sprat box" (translated from danish "Brislingekasse") was in place. A precaution against bycatches of adolescent herring, the box was in place for 30 years, making it necessary for sprat catches to be made $> 30km$ off the coast of Jutland till that point [4]. While the primary purpose of the box was protection of herring, it also benefitted the sprat. This means that there is an artificial hole in the data, which a given model will learn to replicate.

Besides the catch data, environmental variables are based on the ERGOM model made available from the Danish Meterological Institute (DMI).[1]

This includes parameters such as salinity, water temperature, phytoplancton (as a chlorophyl layer) etc., but also hydrographical fronts. These are stratification fronts where a given parameter has the highest gradient, and it has been shown in many cases that these are favorable for a number of fish, including the sprat [8, 5].

2.2 Deep Learning

Trying to predict fish occurrences is a regressive problem, perfectly suited for machine learning. The underlying distribution of fish is not random, at least not on a short timescale, with movements of the fish being dictated by environmental and biological parameters.

Deep learning is a subset of machine learning loosely based on how the brain learns. It uses Artificial Neural Networks (ANNs) as a model in order to learn patterns and behaviours that classic statistical methods might not be able to describe. This makes it a very powerful tool for modelling and forecasting underlying distributions of non-random data.

While ANNs come in many shapes and sizes, they are mostly modelled after the concept of a *perceptron*. This is essentially a building block, consisting of a node with a number of weighted inputs and an output. Signals are attenuated through the input weights and passed to the node. In essence, each node can be seen as a linear regression applied to the the inputs and then mapped to an activation function. The node then outputs the result of the activation function.[9]

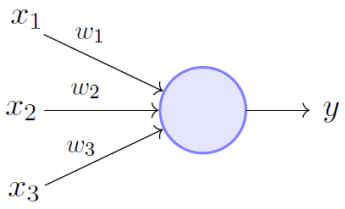


Figure 1: A perceptron. From towardsdatascience.com

Originally, these were developed as a mathematical basis for describing the neurons of the brain. If the input signals, attenuated by the different weights add up to more than zero, the neuron was activated. The output of this system is computed with the activation function:

$$\phi(v_i) = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & otherwise \end{cases} \tag{2}$$

Or as previously mentioned, a linear regression, the output being being binary thresholded by the regression. Now, most data isn't linear, making this type of network unsuited to The clever thing about activation functions is the fact that they can be non-linear, capturing non-linear correlations in data. While the perceptron is highly linear, other activation functions can be applied to the input data. Popular choices include the sigmoid logistic (eq. 3), tanh (eq.4) and rectified linear unit (ReLU, eq.5) functions:

$$\phi(v_i) = \frac{1}{1 + \exp^{-v_i}} \tag{3}$$

$$\phi(v_i) = tanh(v_i) \tag{4}$$

$$\phi(v_i) = max(0, v_i) \tag{5}$$

One neuron, however, is not enough to recognize complex distributions of data. This is where the "network" in Artificial Neural Network comes into play. Typically, the neurons can be distributed into columns, in deep learning lingo known as "layers". Each column connects to its immediate neighbouring columns via the weighted edges, with the first layer being the inputs to the network, and the last layer being the output. Sandwiched between these, the complex interaction of the neurons output signals, weights and activations are done in the *hidden layers*. If every neuron in a layer is connected to all neurons in the next layer, it is said to be a *dense* layer, and if all layers are dense, the NN is fully connected.

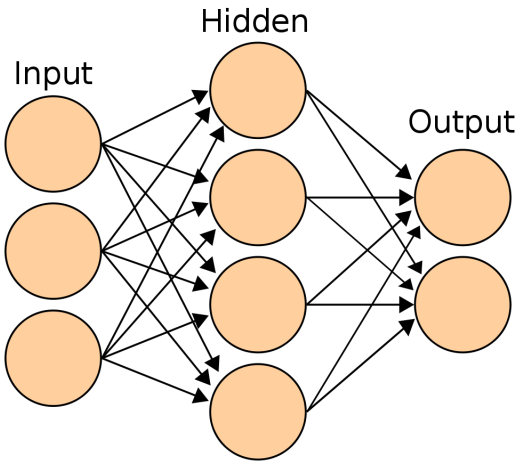


Figure 2: A neural network with an input-, hidden- and output layer. From Wikipedia

This structure is called a feed-forward network, since it feeds the input forward through the different layers, activating learned pathways in the process [9]. The output is a parametric function $\mathbf{f}(\mathbf{x}, \mathbf{w})$, where x are inputs and w are weights.

2.2.1 Optimizers & Gradient descent

In order for the network to actually predict anything, it needs to know what a correct prediction is. We thus define a loss function, a function that gives the deviation of the prediction from the actual wanted value. This might be a simple mean squared error between the values, but often in deep learning, types of cross entropy is used. Given two probability distributions, this function is a measure of the entropy between two probability functions [10].

After this, training steps can be made. Classically, this is supervised learning problem. That is, we are given a set of observations and corresponding labels - the actual values to be predicted.

Although the network can be described parametrically, we cannot simply solve for the weights that minimize the error. However, for a single time step, you can find whether an increase or decrease of weights would decrease the error. That is, you can find the gradient of the parametric network with respect to the error. That leads to an iterative process of modifying the weights to obtain a better prediction, effectively learning. This method, called *gradient descent* is the backbone of modern deep learning. [9, 10]

Different variations and implementations of this exist, collectively called optimizers. However, they all seek to minimize the loss function.

A popular choice is the Adaptive Moment (Adam) optimizer, often used in problems with any sort of noise [11, 12]

2.2.2 Modern networks

A simple feedforward network might be able to predict the next letter in a word, but in a longer sequence, the sequential nature of the network is difficult to recognize. The feedforward network simply has to learn all rules and underlying distributions for *each input*, and that is not always doable. Thus, the network does not have good retention abilities nor does it have any way to do shape recognition - rather important parts for this project.

RNNs and LSTM

A solution to the temporal retention issue are Recurrent (**OBS:** *NOT* Recursive!) Neural Networks - RNNs. These are basically a way to send the output of one prediction step as an input parameter to the next. There are many types of RNN, so only the most acclaimed subtype will be discussed here.

A basic RNN can be described as in eq. 6:

$$H_{i+1} = A(H_i, x_i) \tag{6}$$

That is, the output H_{i+1} of a node depends on both the input to the network x_i but also the previous output of the network H_i - also known as a *hidden state*.

This recursion has a problem though. Values get passed through a network that, at each training step might attenuate or increase the output. This means that the output have a high risk of either completely disappearing or exploding. This is one of the key problems of modern deep learning, often called the **vanishing or exploding gradient problem**, since the gradient used for training is directly dependent on the output of the network.

Long Short-Term Memory (LSTM) networks are a subset of RNNs that aim to solve this issue.

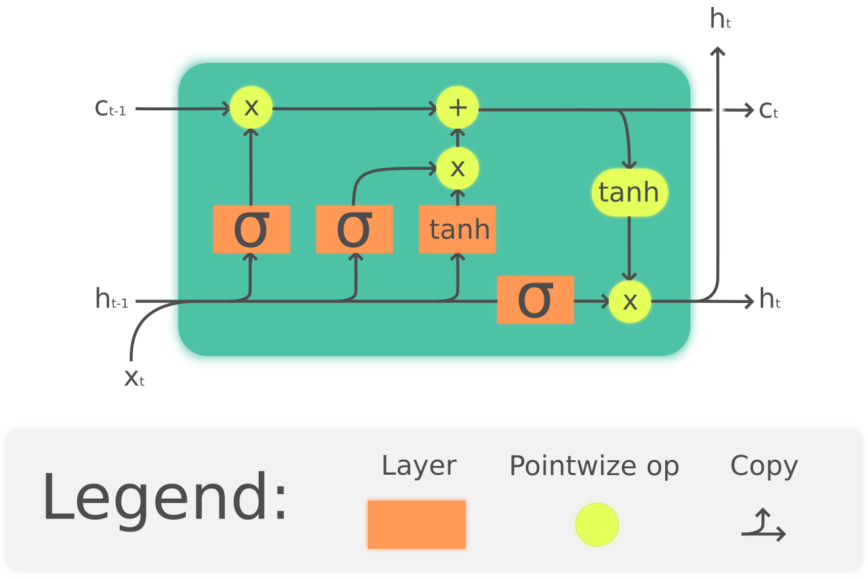


Figure 3: LSTM architecture. In this case, 3 gates have a sigmoidal activation function, while the last is a tanh. From Wikipedia

Here, an input x_t is passed to the LSTM cell along with the previous hidden state h_{t-1} of the network. This is the recurrent part of the network. After this, the signal is passed through a series of gates. As with the perceptron and activation functions, the gates are limiting switches, letting only part of the signal through. The main difference from the standard RNN is the fact that this cell has a memory, c that is also passed to the next cell. This means that it can "choose" to retain weights for a longer time if they are advantageous in minimizing the error, essentially "remembering" past events .

The output, memory and state of the network are passed through to the next iteration. It is a sort of leak of information to the next step, which is exactly why it is so useful in forecasting [10]

Convolutional Neural Networks

In recent years, computervision has been the main driver for deep learning, and have been used in applications such as facial recognition, medical analysis and even self driving cars [13]. The main driver for this is the Convolutional Neural Network (CNN), originally developed by LeCun et al. in 1989 [14].

Although called a *Convolutional* Neural Network, the network is based on the Cross-Correlation operator. This is a mathematical operation in which a kernel is slid across a dataset, multiplying each corresponding element before summation [10]

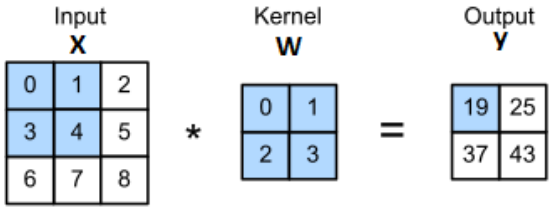


Figure 4: Example of kernel cross-correlation. From d2l.ai

Each pass of the kernel w over the input x outputs one cell in the output y as follows

$$y[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} x(i+u, j+v)w(u,v) \tag{7}$$

So for the example in fig. 5, output $y[1,1]$ would be

$$y[1,1] = 0 \cdot 0 + 1 \cdot 1 + 3 \cdot 2 + 4 \cdot 3 = 19$$

The kernel decides the output, and can thus be set so that it is higher or lower given certain circumstances. This can extract patterns such as edges and corners from the image, known in computer vision as features.

The convolutional neural network uses these kernels as arrays of individual weights, connected to arrays of nodes. In a way, it is a multidimensional extension of the common feed-forward network. It has the added benefit of congregating relevant spatial data into a sort of feature space. Usually, other CNN layers are applied afterwards in order to extract higher level features.

For instance going from corners and edges to circles, then eyes and at last faces. This feature map might then be used to classify whether a picture contains a face.

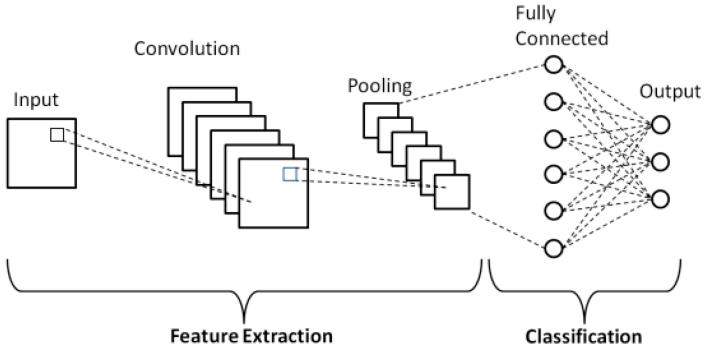


Figure 5: Example of a CNN architecture. The "Pooling" layers are simply a type of data aggregation. From mdpi.com

2.2.3 Pitfalls of deep learning

Tuning a neural network is an art as much as it is a science. Countless layer types, parameters and training options are available, and there is not necessarily a black white answer to the optimal setup. A few pitfalls are present in most learning problems though, as well as some solutions to these.

In deep learning, two key challenges, over- and underfitting, are opposite issues that both occur when suboptimally learning the given dataset. Intuitively, one can see that not training the model enough on the data means it will not solve the learning problem optimally. This is underfitting, and might be adjusted for using a model with a larger capacity or more training epochs. A larger problem comes in overfitting, where a model learns the training data too well, and gives an unnaturally high accuracy in the training stage. In both cases, the model doesn't *generalize* well to new data, but overfitting is especially difficult with a too sparse dataset. Here, stopping training early is key, as is having enough data and a model with suitable complexity. It might also be negated using regularization techniques, essentially introducing noise to add variance in some data dimensions [10].

The accuracy metrics used in training the network is what determines the end model. Choosing a loss function unfit for the data will therefore yield bad results - e.g. if data has large outliers, certain loss functions might weigh these too low or high.

A problem specific to forecasting is that, for a time series, if you tell a model to predict t+1, it might just find that t+1=t for a given point is good enough. No good solutions to this are readily available, so tuning the network, regularizing the reccurent parts of the network and modifying the loss function are tools to avoid this.

3 State-of-the-art

Today, forecasting is a diverse field with applications in many fields. When the physics behind a phenomenon is well understood, such as the weather or ice coverage of the poles, physical models can be used to describe the phenomenon and predict future dynamics [11, 15]. However, some dynamics are not well described, making it difficult to model analytically with these methods.

In the case of the sprat, growth and drivers behind population fluctuations are poorly understood, and few attempts at forecasting have been made [3].

Traditionally, statistical methods such as ARIMA can be used when modelling time-series data to learn the underlying patterns[16]. This works for data, where the latent relationships hidden in the data are linear, but when a non-linearity is introduced, these models are more unstable. In this case, neural networks have an advantage. Modelling complex biological systems, means introducing a highly non-linear system, which is easily done in deep learning (see chp. 2) [15, 10]. An added benefit of NNs is the fact that they function even with irregular data, which is commonly seen in fishing data [17].

Few, if any, forecasts have been made on catch rates of the sprat, let alone ones using deep learning. However, other seasonal maritime phenomena have been spatially forecasted.

Grasso et al. (2019) [11] describes forecasting of harmful algae blooms called red tides. Here, as in the case of the sprat, traditional methods of forecasting fail. This is primarily due to statistical models only capturing seasonality and interannual variations, when weekly data is needed as to avoid poisonous shellfish. To circumvent this, the success of deep learning in image recognition is used. Grasso et al. collects data for each sampling location in a batch of 5 weeks classified according to toxicity the subsequent week. Data from one year was used to train a densely connected neural network. The network does not consider spatial variations between sampling locations. It was optimized using the Adam optimizer and used dropout to avoid overfitting. On 4 different classes of toxicity, the accuracy ranged from 95-99% up to 2 weeks in advance.

Qin et al. (2017) [18] also tackles forecasting red tides. However, while Grasso et al. purely uses deep learning, Mengjiao combines ARIMA with a type of deep learning called a Deep Belief Network (DBN), a somewhat obscure type of NN. This essentially combines the advantages of the linear ARIMA with the non-linearity of the NN. DBN uses unsupervised learning, as opposed to the densely connected feedforward NN used by Grasso. Mean accuracy=17.2%. This method uses 12 types of environmental data to forecast.

A similar method is adapted by Aquino et al. (2016) [17]. It is interesting due to very few data points, similar to the dataset of this paper. Using an ARIMA combined with a simple feedforward network, it predicts the total quarterly catch of big-eyed scad, a 1D time series. It does so with a RMSE of 0.006 and MAE of 0.034. Unfortunately, the paper only tests on the training data, making validation results dubious at best.

While maritime forecasting with deep learning is not well developed, weather forecasting is.

Certain types of neural networks have been developed specifically for meterology. Seeing that weather radar data is spatio-temporal data, just like maritime catch data, the networks used here are of interest.

One such network is the Convolutional LSTM or ConvLSTM for short, developed by Shi et al, (2015) [19]. Specifically for nowcasting of precipitation it combines recent advances in convolutional neural networks with state-of-the-art time-series forecasting with the LSTM architecture.

Another area of interest is video generation. Solving essentially the same issue of predicting the next frame of a time series given as input. Some models even have multiple channels for RGB data, analogous to the environmental parameters of marine data. Villegas et al. (2019) combines the ConvLSTM layers of Shi et al. (2015) with an encoder-decoder network, essentially learning the latent distributions of both motion and content of a video [20].

A comparison of the different methods and results is shown in table 1.

Paper	Pre-processing	Network layers/ type	Optimizer & loss function	Sample Size	Purpose	Results
Shi et al. 2015[19]		ConvLSTM	Optimizer:N/A & Loss:Cross-entropy	8148 training samples. 2037 validation samples.	Precipitation nowcasting	Correlation of 0.908 where conventional models ranged from 0.843-0.849.
Grasso et al. 2019[19]	Data is packed such that it can be classified in 4 categories.	Dense w. dropout	Optimizer:Adam & Loss:Categorical Cross-entropy	3198 samples with 60:20:20% as training:validation:testing	Forecasting red tides	95-99% accuracy - however, using regular samples from same areas
Qin et al. 2017 [18]	Uses 12 environmental parameters as input data	ARIMA with Deep Belief Network	Optimizer: Particle Swarm Optimization Loss: N/A	6200 samples with 60:20:20% as training:validation: testing	Forecasting red tides	Mean Absolute Percentage Error=17.2% Average RMSE=0.048
Aquino et al. 2017	1D catch data as input	ARIMA and simple dense feedforward network	Optimizer: N/A Loss: RMSE & MAE	61 samples.	Forecasting big-eyed scad catches	RMSE=0.006 MAE=0.034
Villegas et al. 2019 [21]	Image sequence as input	ConvLSTM Encoder-decoder as adversarial GAN	Optimizer: N/A Loss: Predicition loss	61 samples.	Video generation	On UCF-101 dataset: S/N ration around 20 with pure ConvLSTM as 15

Table 1: table of relevant projects and related parameters

4 Method

The method used is designed for a system with following specifications. Many other systems will work, but have not been tested:

1. Python 3.7
2. Nvidia GTX 1050Ti GPU running the CUDA parallel computing platform[1]
3. Windows 10
4. Tensorflow 2.1, build: gpu_py37h55f5790_0
5. A list of Python-libraries imported can be found in the source code. See app. A

First, data was imported. Since it was compiled as C-squares data, the Python package netCDF4 was used to unpack. In the data-compilation, every C-square had coordinates, CPUE for the sprat and different maritime parameters – the latter being normalized while unpacking. The catch data was log-transformed in order to have negative values when catch-per-unit-effort was below 1.

Due to hardware limitations, data had to be resized. This was done using the OpenCV Python package. Keeping the lon-lat scale of the data constant, a linear area interpolation was applied. [22]

A training set then had to be created. In forecasting with deep learning, this can be done in many ways, depending on the amount of data. Most commonly, one of 2 methods are applied: Either an n-sized array of input values can be given with an array with data shifted one value left as target. This allows the network to easier learn the sequential nature of the data, since n-1 of the values are equal to the input array, only time-shifted. Unfortunately, this method can easily overfit and learn the data, since the accuracy is n-1/n % if the network just copies the input. Otherwise, a set of data is given as input, with the next step as target. As opposed to the first method, it does not as easily overfit, seeing that it cannot just copy the input. However, it might not be as robust, since it does not have an inherit sequential dimension. Both methods were tried, the latter having the most success.

The data resolution and timeframe limit the output resolution and forecasting timeframe. For this project, it means that sprat catches can only be forecast a whole number of weeks forward, and due to the low sample size, only a very short time ahead.

Next, different models were compiled to examine differences in hyperparameters, optimizers and layers. As previously mentioned, the convLSTM architecture was developed for forecasting and has had great success in that area. This meant that, while some other networks, such as ResNet were considered, all the tested models were ultimately convLSTM-based. The models were first trained with 3 years of the training set, the other part serving as validation and test, split evenly.

[1] While simple networks can be run on CPU architectures, it is not feasible for forecasting purposes. GPU not strictly required to be Nvidia.

Early on, it became apparent that the network could achieve decent accuracy by predicting zeros everywhere, due to the sparse nature of the data. To negate this, a custom loss function was created, that weighted true non-zero predictions higher than true zero predictions.

Then, a simple model was used to tune the weights of this function, as was the2 different optimizers compared here, to use in more complex network architectures.

While model performance was measured in MSE, MAE and the custom weighted loss during training, some sort of test parameter was also needed. Here, the sum of the zero-normalized cross correlation (ZNCC) betwen the ground truth and the prediction was used [23]. Due to concerns of the network copying the past timestep, the correlation between the ground truth at this step and the prediction was also calculated. It does not make sense to correlate when one image is all zeroes, which is why the ZNCC only correlates how well the non-zero data fits. This is intentional, otherwise predicting zeros for all frames would yield a very high accuracy. The difference between ZNCC and shifted ZNCC was calculated. The larger this metric, the less likely the network is to copy the previous timestep as prediction. Due to wanting both this difference and the total ZNCC to be high, a final metric was calculated by

$$Difference\_ZNCC + Total\_ZNCC = 2 \cdot Total\_ZNCC - shifted\_ZNCC$$

An animation of the predictions of the trained models was used for qualitative inspection of the forecasting abilities of the network.

All model performances were evaluated on the test set.

5 Results

Results for the ZNCC of the model predictions evaluated on the test set are given in table 2. The different models used can be found in the appendix.

Model	**Total correlation**	**Shifted correlation**	**Difference**	**Total+Diff**
Model 1e: *Adam, lr : 0.003*	0,148800613	0,255973731	-0,10717	0,041627
Model 6	0,140047386	0,24931477	-0,10927	0,03078
Model 1c *weight:0.4*	0,134609796	0,241050127	-0,10644	0,028169
Model 1e *Adadelta*	0,025585213	0,027724262	-0,00214	0,023446
Model 1c *weight:0.1*	0,078427567	0,133574835	-0,05515	0,02328
Model 1d	0,132850721	0,246607758	-0,11376	0,019094
Model 1c *weight:0.2*	0,123087675	0,229920409	-0,10683	0,016255
Model 1c *weight:0.8*	0,01863178	0,022715686	-0,00408	0,014548
Model 5	0,118430504	0,22720915	-0,10878	0,009652
Model 1b	0,019730977	0,030978398	-0,01125	0,008484
Model 1a	0,126303922	0,245962663	-0,11966	0,006645
Model 3	0,001403529	0,000936908	0,000467	0,00187
Model 2	0,068323723	0,137819455	-0,0695	-0,00117
Model 1e: *Adam, lr : 0.01*	0,129198457	0,260169303	-0,13097	-0,00177
Model 1c *weight:0.6*	0,153882634	0,313752533	-0,15987	-0,00599
Model 4	0,148629177	0,361049772	-0,21242	-0,06379
Baseline	0,202275360	1	-0.79772	-0.59544

Table 2: ZNCC for all created models, evaluated on test set

Baseline is if the network just copies the last frame.

MSE and MAE are plotted in figure 6 along with the ZNCCs.

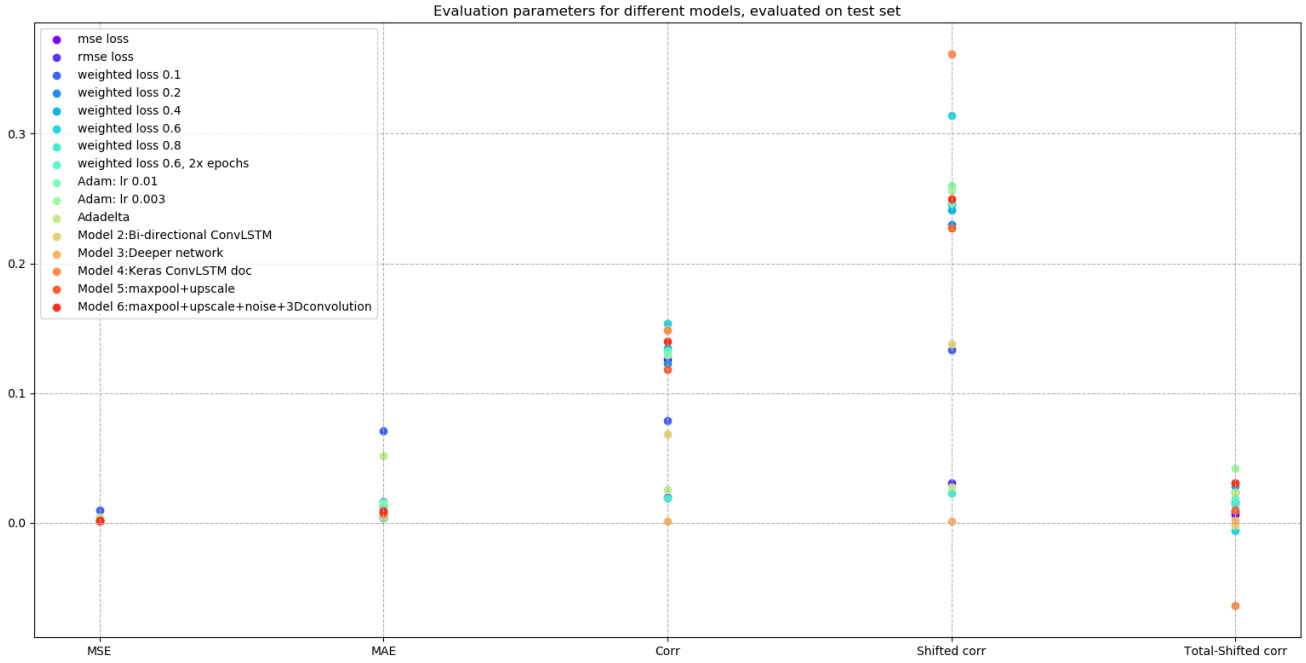


Figure 6: Evaluations of the different models plotted.

The two best models, Model 6 and Model 1e: *Adam, lr : 0.003* were used in creating qualitative animations, showcasing predictions vs ground truth. These, along with the data are sensitive information, and with proper permission from DTU Aqua, can be found at the private GitHub repository for the project[2].

An example of model predictions is shown in figure 7

[2]For access to the private GitHub repo, write to carlemilelling@hotmail.com

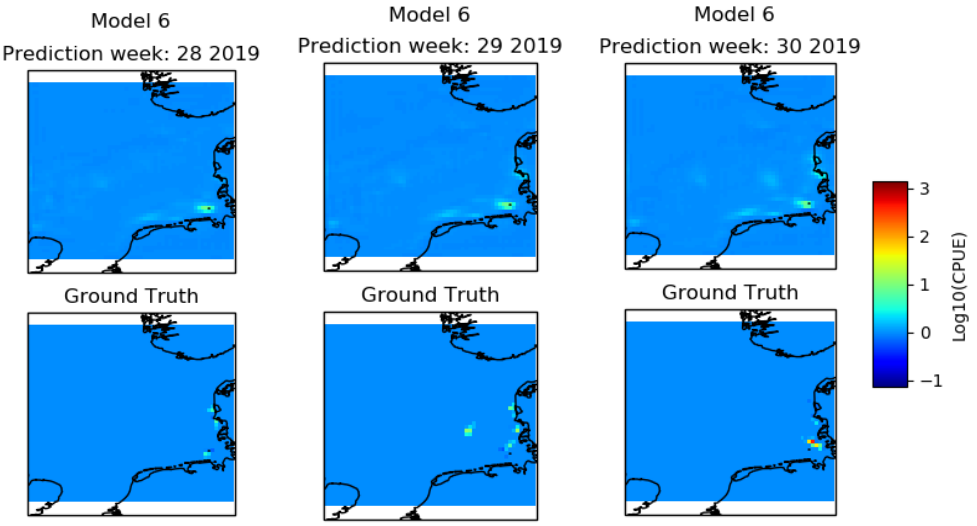


Figure 7: Example of model predictions. Model 6, predicted on test set

6 Discussion

The results of this report are far from stellar. The best models were mostly also the simplest, only achieving up to 0.14 in correlation. Results indicate a partial copy of the previous timestep as output of the models. While the baseline set has the highest correlation of any model, none of the models have an enormous shifted correlation. It can be argued that the model only copies a small part of the previous frame that will be in the next frame, giving an actual prediction.

Other papers achieve higher accuracy using ConvLSTM. This is likely due do much, much (!) larger sample sizes making it easier to increase model complexity without overfitting and having the model actually learn the input data. To circumvent this, the neural networks created in this paper apply dropout and regularization techniques heavily, making achieving a high accuracy mathematically impossible. It also explains the simple models having the best results. In that regard, Model 6 is an outlier, and possibly the most promising model for further work.

If all $154 \cdot 170$ spatial points are looked at as a single, independent data source with 12 channels and 241 time points, one might say that we have ≈ 5 million data points available for training. However, a list of data-reducing factors makes this but a dream:

- Irregularity in CPUE - not captured at the same points multiple times.
- Not all 12 channels of environmental parameters are predictors for sprat.
- Reduction of image resolution due to large (!) computational demand of CNNs.
- When trying to capture temporal *and* spatial correlations, the correlations themselves become the data to train on. Thus, the millions of individual data points become far fewer spatio-temporal correlations.
- Interannual periods where sprat are not fished, leaving multiple $154 \cdot 170 \cdot 12$ frames as non-usable data.

In general ANNs need large amounts of data to generalize, CNNs being notorious for needing even more [10]. Thus, with the relatively small sample size of the sprat catch data, it might not be feasible to forecast with a complex model using current deep learning technology.

To circumvent this, it might be worthwhile exploring exotic network architectures, such as the hybrid models used in chp. 3.

Even if the model had predicted perfectly, many data biases exist. CPUE is only registered where there is sprat fishing, falsely giving the impression that there are no sprat anywhere else. Where there is catch data is also a bias; fishermen usually fish from force of habit, meaning a system might learn the underlying force of habit instead of the fish distribution. The ground truth is not really ground truth, then. Even taking all this into account, CPUE is not even a perfect normalization of catches, with larger ships generally having higher CPUE!

The biases should be negated in data collection - sampling each fishing location regularly, along with locations that are not fished. This is to give a density map that might be used in debiasing.

I would have liked to do an analysis on which parameters are most significant predictors for the occurrence of sprat, but that would be a project in itself. Instead, a framework was made, such that any environmental variables can be used in training the networks. It was observed, that by including all parameters instead of just CPUE in training, all models became more robust, indicating that at least some of the provided parameters can be used in forecasting sprat.

7 Conclusion & future perspectives

Due to the very small sample size, the network architectures developed here are average at best. Reaching a maximum correlation of 0.1488 with model 1c *Adam, learning rate:0.003* , it can only be concluded that, given the data available for this project and a desire for a very complex forecast, is difficult and not practical using ConvLSTM-based networks alone. This also partially answers research question **Q1**, in that, using state-of-the-art deep learning forecasting methods, environmental factors are not especially good predictors for CPUE.

However, evidence suggest that, given a good enough model, parameters such as salinity fronts or zooplankton might give an edge to the final predictions. As discussed in chp 2, much of sprat abundance might be explained by observations of fish that either compete with or prey on the sprat itself. In the future, building a model incorporating these might be able to achieve better results than this report did.

The answer to **Q2** also plays a part in the results achieved, or rather, lack thereof. Since the given fishing data is highly localized, highly irregular and highly biased, the network learns habits of fishermen more than it learns the actual latent density function behind sprat occurrences. This would best be negated by having multiple temporal samples at the same spatial points and correcting with help from these.

To develop the forecasting further, it is worth exploring exotic networks and hybrid models. These hybrid models might be a pre-processing step, like the ARIMAs described in chp. 3. It could also be a reduction of data redundancy.

An idea could be to reduce forecast complexity, in that a less complex output drastically reduces the number of weights needed in the system, making it less likely to overfit.

References

[1] Asbjørn Christensen et al. *Dynamic user driven development of ocean maps to increase the value of the Danish industrial fishery*. Tech. rep. DTU Aqua, 2019.

[2] Lionel Guidi and Antonio Fernandez Guerra. "Big Data in Marine Science". In: *European Marine Board IVZW Future Science Brief* 6.April (2020), 52.

[3] Claire Moore et al. "Age verification of north Atlantic sprat". In: *Fisheries Research* 213 (2019), 144–150. ISSN: 01657836. DOI: 10.1016/j.fishres.2019.01.018.

[4] Erik Hoffmann and Henrik Carl. "Atlas over danske saltvandsfisk - Brisling". In: *Statens Naturhistoriske Museum* September (2019).

[5] Michele Casini, Massimiliano Cardinale, and Joakim Hjelm. *Inter-annual variation in herring, Clupea harengus, and sprat, Sprattus sprattus, condition in the central Baltic Sea: what gives the tune?* Tech. rep. 3. 2006, 638–650.

[6] Tony Rees. ""C-Squares", a new spatial indexing system and its applicability to the description of Oceanographic Datasets". In: *Oceanography* 16.1 (2003), 11–19. ISSN: 10428275. DOI: 10.5670/oceanog.2003.52.

[7] Mark N. Maunder et al. "Interpreting catch per unit effort data to assess the status of individual stocks and communities". In: *ICES Journal of Marine Science* 63.8 (2006), 1373–1385. ISSN: 10543139. DOI: 10.1016/j.icesjms.2006.05.008.

[8] Hannes Baumann et al. "The German Bight (North Sea) is a nursery area for both locally and externally produced sprat juveniles". In: *Journal of Sea Research* 61.4 (2009), 234–243. ISSN: 13851101. DOI: 10.1016/j.seares.2009.01.004. URL: http://dx.doi.org/10.1016/j.seares.2009.01.004.

[9] Tue Herlau, Mikkel N. Schmidt, and Morten Mørup. *Introduction to Machine Learning and Data Mining*. 1.1. DTU, 2018.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016. DOI: 10.1016/b978-0-12-391420-0.09987-x. URL: http://www.deeplearningbook.org.

[11] Isabella Grasso et al. "The hunt for red tides: Deep learning algorithm forecasts shellfish toxicity at site scales in coastal Maine". In: *Ecosphere* 10.12 (2019). ISSN: 21508925. DOI: 10.1002/ecs2.2960.

[12] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), 1–15.

[13] Mariusz Bojarski et al. "End to End Learning for Self-Driving Cars". In: 63 (2016), 1–9. URL: http://arxiv.org/abs/1604.07316.

[14] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1 (1989), 541–551.

[15] Young Jun Kim et al. "Prediction of monthly Arctic sea ice concentrations using satellite and reanalysis data based on convolutional neural networks". In: *Cryosphere* 14.3 (2020), 1083–1104. ISSN: 19940424. DOI: 10.5194/tc-14-1083-2020.

[16] Yitian Chen et al. "Probabilistic forecasting with temporal convolutional neural network". In: *Neurocomputing* xxxx (2020), 31–34. ISSN: 18728286. DOI: 10.1016/j.neucom.2020.03.011. URL: https://doi.org/10.1016/j.neucom.2020.03.011.

[17] Ronald L Aquino et al. "A hybrid ARIMA and neural network model applied to forecast catch volumes of Selar crumenophthalmus A Hybrid ARIMA and Neural Network Model Applied to Forecast Catch Volumes of Selar crumenophthalmus". In: *AIP Conference Proceedings 1905* 050006.November (2017).

[18] Mengjiao Qin, Zhihang Li, and Zhenhong Du. "Red tide time series forecasting by combining ARIMA and deep belief network". In: *Knowledge-Based Systems* 125 (2017), 39–52. ISSN: 09507051. DOI: 10.1016/j.knosys.2017.03.027. URL: http://dx.doi.org/10.1016/j.knosys.2017.03.027.

[19] Xingjian Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *Advances in Neural Information Processing Systems 28* 28 (2015). ISSN: 16877268. DOI: 10.1155/2018/6184713.

[20] Ruben Villegas et al. "Decomposing motion and content for natural video sequence prediction". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2019), 1–22.

[21] Shengnan Guo et al. "Deep Spatial – Temporal 3D Convolutional Neural Networks for Traffic Data Forecasting". In: 20.10 (2019), 3913–3926.

[22] *GitHub - opencv/opencv: Open Source Computer Vision Library*. URL: https://github.com/opencv/opencv.

[23] *Zero Normalized Cross-correlation - Wikipedia*. URL: https://en.wikipedia.org/wiki/Cross-correlation#Zero-normalized_cross-correlation_(ZNCC).

Appendices

A Code

A.1 Imports

```
# -*- coding: utf-8 -*-
"""
Imports.


@author: Carl Emil Elling
"""

#Arrays
import numpy as np

#Neural network construction
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional,Conv3D, ConvLSTM2D,Dense,Dropout,Flatten, Time

#from tensorflow.keras.normalization import BatchNormalization
import netCDF4 as netcdf #For importing C-squares data
import cv2 #for resizing

#Figures
import matplotlib as mpl
import matplotlib.animation as animation
import matplotlib.pyplot as plt

import cartopy.crs as ccrs #map projections
import cartopy.feature as cfeature #features in maps
import seaborn as sns #Color for maps
from scipy import signal #For correlation


pause=False #For pausing animations
```

A.2 Data Initialization

```
# -*- coding: utf-8 -*-
"""
Data Initialization

@author: Carl Emil Elling
"""

def normalized_netcdf_variable(ncf, varname, accept_range):
    # masking load by netCDF4 apparently fails; directly work with var.data and manually detect
    # scaled variable to range [0,1]; project fills to 0
    # project values outside accept_range to 0, 1 respectively
    var  = ncf.variables[varname]
    ymin = max(np.amin(np.where(var[:].data==var._FillValue,  1.e20, var[:].data)), accept_range
    ymax = min(np.amax(np.where(var[:].data==var._FillValue, -1.e20, var[:].data)), accept_range
    var  = np.where(var[:].data==var._FillValue, ymin, var[:].data) # project fills to lower lim
    scaled_var = (var-ymin)/(ymax-ymin) # rescale
    scaled_var = np.where(scaled_var<0, 0, scaled_var)
    scaled_var = np.where(scaled_var>1, 1, scaled_var)
    print("normalized_netcdf_variable: %f < %s < %f" % (ymin, varname, ymax))
    return  scaled_var


def nint(x): return int(round(x))


# -----------  load coaxial signal+response data -----------
ncf_env = netcdf.Dataset("C:/Users/carle/Box/GeoRum - Bachelor 8. semester + juni/30100 Fagproje
cpue = ncf_env.variables['cpue'][:] #Catch per unit effort
nobs = ncf_env.variables['nobs'][:] #number of observations?
time = ncf_env.variables["time"][:] #Tid
lon  = ncf_env.variables["lon"][:] #Longitude
lat  = ncf_env.variables["lat"][:] #Latitude

msl  = normalized_netcdf_variable(ncf_env, 'msl',     (0,     1e10)) #Lufttryk
u10  = normalized_netcdf_variable(ncf_env, 'u10',     (-1000, 1000)) #10m øst vind
v10  = normalized_netcdf_variable(ncf_env, 'v10',     (-1000, 1000)) #10m vest vind
attn = normalized_netcdf_variable(ncf_env, 'attn',    (0,     1e10)) #lysdæmpning i vand
CHL  = normalized_netcdf_variable(ncf_env, 'CHL',     (0,     1e10)) # klorofyl - mg eller mu_g/
mld  = normalized_netcdf_variable(ncf_env, 'karamld', (0,     12e3)) #Dybde af springlag
salt = normalized_netcdf_variable(ncf_env, 'vosaline',  (0,     100)) #salinitet
```

```
sst  = normalized_netcdf_variable(ncf_env, 'analysed_sst', (0, 500)) #overfladetemperatur
swh  = normalized_netcdf_variable(ncf_env, 'swh',      (0,     1000))#bølgehøjde
sst_front = normalized_netcdf_variable(ncf_env,  'sst_front',  (0,     1000)) #temperaturfrontin
salt_front = normalized_netcdf_variable(ncf_env, 'salt_front', (0,     1000)) #salinitetsfrontin
ncf_env.close()
#variable fra: HBM-ERGOM og GUDP-VIND

cpue = np.where(cpue>0, np.log(cpue,where=cpue>0), 0) # log transform

data=np.stack((cpue,msl,u10,v10,attn,CHL,mld,salt,sst,swh,sst_front,salt_front),axis=0) #Data to
data_input_shape = np.shape(data)
#%% Resize by interpolation
#Resizing data - remember to resize back after prediction with network.
#Resizing is done so aspect ratio between lat and lon are preserved - finding whole ints that ma
#These are: (29,25),(58,50),(87,75),(116,100),(145,125)
resize_lat=50
resize_lon=58

#Interpolation type.
interpolation = cv2.INTER_AREA #Linear area interpolation. Works as linear 1D interpolation when
lat_resized = cv2.resize(lat.reshape(np.size(lat),1),(1,resize_lat),interpolation=interpolation)
lon_resized = cv2.resize(lon.reshape(np.size(lon),1),(1,resize_lon),interpolation=interpolation)

#Resizing parameters
cpue_resized = np.transpose(cv2.resize(np.transpose(cpue,(1,2,0)),(resize_lon,resize_lat),interp
msl_resized = np.transpose(cv2.resize(np.transpose(msl,(1,2,0)),(resize_lon,resize_lat),interpol
u10_resized = np.transpose(cv2.resize(np.transpose(u10,(1,2,0)),(resize_lon,resize_lat),interpol
v10_resized = np.transpose(cv2.resize(np.transpose(v10,(1,2,0)),(resize_lon,resize_lat),interpol
attn_resized = np.transpose(cv2.resize(np.transpose(attn,(1,2,0)),(resize_lon,resize_lat),interp
CHL_resized = np.transpose(cv2.resize(np.transpose(CHL,(1,2,0)),(resize_lon,resize_lat),interpol
mld_resized = np.transpose(cv2.resize(np.transpose(mld,(1,2,0)),(resize_lon,resize_lat),interpol
salt_resized = np.transpose(cv2.resize(np.transpose(salt,(1,2,0)),(resize_lon,resize_lat),interp
sst_resized = np.transpose(cv2.resize(np.transpose(sst,(1,2,0)),(resize_lon,resize_lat),interpol
swh_resized = np.transpose(cv2.resize(np.transpose(swh,(1,2,0)),(resize_lon,resize_lat),interpol
sst_front_resized = np.transpose(cv2.resize(np.transpose(sst_front,(1,2,0)),(resize_lon,resize_l
salt_front_resized = np.transpose(cv2.resize(np.transpose(salt_front,(1,2,0)),(resize_lon,resize

#Have to re-stack all resized parameters, since OpenCV resize only takes up to 3D arrays
data_resized=np.stack((cpue_resized,msl_resized,u10_resized,v10_resized,attn_resized,CHL_resized
data_resized=np.transpose(data_resized,[1,2,3,0]) #Transposing the channels last
```

```
#for doing multi-channel network
split = [0.6,0.2,0.2] #the split of the data: training:validation:test
split = [int(len(time)*split[0]),int(len(time)*(split[0]+split[1]))]

#chosing years instead
#split_years = [2017,2018]
#split = [np.searchsorted(time,split_years[0]),np.searchsorted(time,split_years[1])]


train_data = data_resized[:split[0],:,:,:]
val_data = data_resized[split[0]:split[1],:,:,:]
test_data=data_resized[split[1]:,:,:,:]

time_train=time[:split[0]]
time_val=time[split[0]:split[1]]
time_test=time[split[1]:]

#For doing 1ch networks
train_cpue = train_data[0,:,:,:,np.newaxis]
val_cpue = val_data[0,:,:,:,np.newaxis]
test_cpue = test_data[0,:,:,:,np.newaxis]
```

A.3 Models

```
# -*- coding: utf-8 -*-
"""
Model library.

All models saved as classes to be called in main script. Also training loop

@author: Carl Emil Elling
08-07-2020
"""
import tensorflow as tf
import os
import matplotlib as mpl
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional,Conv3D, ConvLSTM2D,Dense,Dropout,Flatten, Time
```

```
def fn_run_model(model,X_train,Y_train,xval,yval,batch_size,nb_epochs,model_name,working_path):
    #Function for fitting model to training and validation data
    history= tf.keras.callbacks.History()

    save_path= os.path.join(working_path,"model_weights/"+model_name+"/1/model.ckpt")
    cp_callback = tf.keras.callbacks.ModelCheckpoint(
            #Callback for saving weights
            filepath=save_path,
            verbose=1,
            monitor="val_loss", #Monitoring validation loss and saving only the best weights in
            save_weights_only=True,
            save_freq="epoch",
            save_best_only=True,
            mode="min")

    #Fitting the model
    history=model.fit(
            X_train,
            Y_train,
            batch_size=10,
            epochs=nb_epochs,
            verbose=1,
            callbacks=[cp_callback],
            validation_data=(xval,yval)
            )
    fig, ax1 = plt.subplots(1,1)
    ax1.set_title(model_name)
    ax1.plot(history.history["val_loss"],label='val_loss')
    ax1.plot(history.history["loss"],label='*loss*')
    ax1.legend()

#%% Custom Loss functions

def custom_rmse(true,pred):
    #given a target Y and prediction Y_hat, compute RMSE.
    #RMSE weighs large errors much higher than small errors, compared to MAE
    error=tf.sqrt(tf.reduce_mean(tf.square(true - pred)))
    return error

def weightedRMSELoss(w):
    #Weighs true zero-predictions lower than true nonzero predictions
    #From https://stackoverflow.com/questions/57618482/higher-loss-penalty-for-true-non-zero-pre
```

```
    #Weights between 0 and 1. Lower weight means lower weight for true zero-predictions - essent
    #weight above 1 means penalizing true nonzero predictions - that is making extra certain pre
    def loss(true, pred):

        #error has to be done in multiple steps, since tf.reduce_mean otherwise divides by zero
        error = tf.square(true - pred)
        error = tf.keras.backend.switch(tf.math.equal(true, 0), w**2 * error , error) #penalize
        error = tf.sqrt(tf.reduce_mean(error))
        return error

    return loss


"""
Loss function: Perceptual distance from
https://www.youtube.com/watch?v=fmgaOiOMXuU
"""
#def perceptual_distance(true, pred):
#    true *= 255.
#    pred *= 255.
#    rmean = (true[:, :, :, 0] + pred[:, :, :, 0]) / 2
#    r = true[:, :, :, 0] - pred[:, :, :, 0]
#    g = true[:, :, :, 1] - pred[:, :, :, 1]
#    b = true[:, :, :, 2] - pred[:, :, :, 2]
#
#    return K.mean(K.sqrt((((512+rmean)*r*r)/256) + 4*g*g + (((767-rmean)*b*b)/256)))
#%%
"""
Model 1
Low complexity architechture. Less deep network might help with smaller sample size
https://stackoverflow.com/questions/49540320/cuda-out-of-memory-when-training-convlstmd2d-model
"""

class model_1:
    def __init__(self,input_shape,output_shape=None,loss_type="mean_squared_error",optimizer="Ad
        #creates model.
        #kernel size essentially means capturing speed. To capture fast motions, a large kernel
        self.model=tf.keras.Sequential([
                tf.keras.Input(
                        #shape=(None, 150, 174,1)
                        shape=input_shape #small data sample for testing purposes
                        ),# Variable-length sequence of 1x150x174 frames
```

```
                tf.keras.layers.GaussianNoise(noise),
                ConvLSTM2D(
                        filters=16, kernel_size=(kernel,kernel),
                        padding='same', return_sequences=True,
                        activation='tanh', recurrent_activation='tanh',
                        kernel_initializer='glorot_uniform', unit_forget_bias=True,
                        data_format='channels_last', #Specifies which dimension of input data ar
                        dropout=0.6, recurrent_dropout=0.5, go_backwards=True,
                        stateful=False, #If true, the first frame in next batch will be last fra
                        strides=(1, 1)
                        ),
                BatchNormalization(),


                ConvLSTM2D(
                        filters=1, kernel_size=(kernel,kernel),
                        padding='same', return_sequences=False, #Return sequences attribute tell
                        activation='sigmoid', recurrent_activation='tanh',
                        kernel_initializer='glorot_uniform', unit_forget_bias=True,
                        data_format='channels_last',
                        dropout=0.1, recurrent_dropout=0.1, go_backwards=True,
                        stateful=False,
                        strides=(1, 1)
                        )
                ])
        if optimizer=="Adam":
            opt = tf.keras.optimizers.Adam(learning_rate=lr)
        elif optimizer=="Adadelta":
            opt = "Adadelta"
        self.model.compile(loss=loss_type, optimizer=opt,metrics=['mse','mae'])
        self.model.summary()
    def runmodel(self,X,Y,X_hat,Y_hat,batch_size,nb_epochs,working_path,model_name="Model1"):
        fn_run_model(self.model,X,Y,X_hat,Y_hat,batch_size,nb_epochs,model_name,working_path)
        self.model.summary()
        #saving weights
        #name_weights = "final_model_fold" + str(j) + "_weights.h5"
    def predict(self,x):
        y=self.model.predict(x)
        return y
    def save_weights(self,path):
        self.model.save_weights(path)
    def evaluate(self,X_eval,Y_eval):
```

```
        evaluation = self.model.evaluate(X_eval,Y_eval)
        return evaluation
    def load_weights(self,checkpoint_path):
        self.model.load_weights(checkpoint_path)
        print("Weights loaded")


#%%
"""
Model 2
Bi-directional ConvLSTMs. More exposure to data might help train the network
https://datascience.stackexchange.com/questions/27628/sliding-window-leads-to-overfitting-in-lst
"""

class model_2:
    def __init__(self,input_shape,output_shape=None,loss_type="mean_squared_error",kernel=4):
        #creates model.
        #kernel size essentially means capturing speed. To capture fast motions, a large kernel
        self.model=tf.keras.Sequential([
                tf.keras.Input(
                        #shape=(None, 150, 174,1)
                        shape=input_shape #small data sample for testing purposes
                        ),# Variable-length sequence of 1x150x174 frames
                tf.keras.layers.GaussianNoise(0.4),
                Bidirectional(ConvLSTM2D(
                        filters=8, kernel_size=(kernel,kernel),
                        padding='same', return_sequences=True,
                        activation='tanh', recurrent_activation='tanh',
                        kernel_initializer='glorot_uniform', unit_forget_bias=True,
                        data_format='channels_last', #Specifies which dimension of input data ar
                        dropout=0.1, recurrent_dropout=0.1, go_backwards=True,
                        stateful=False, #If true, the first frame in next batch will be last fra
                        strides=(1, 1)
                        )),
                BatchNormalization(),
                ConvLSTM2D(
                        filters=16, kernel_size=(kernel,kernel),
                        padding='same', return_sequences=True,
                        activation='tanh', recurrent_activation='tanh',
                        kernel_initializer='glorot_uniform', unit_forget_bias=True,
                        data_format='channels_last', #Specifies which dimension of input data ar
                        dropout=0.2, recurrent_dropout=0.2, go_backwards=True,
```

```
                        stateful=False, #If true, the first frame in next batch will be last fra
                        strides=(1, 1)
                        ),
                BatchNormalization(),

                ConvLSTM2D(
                        filters=1, kernel_size=(kernel,kernel),
                        padding='same', return_sequences=False, #Return sequences attribute tell
                        activation='sigmoid', recurrent_activation='tanh',
                        kernel_initializer='glorot_uniform', unit_forget_bias=True,
                        data_format='channels_last',
                        dropout=0.1, recurrent_dropout=0.1, go_backwards=True,
                        stateful=False,
                        strides=(1, 1)
                        )
                ])
        opt_adam = tf.keras.optimizers.Adam(learning_rate=0.005)
        self.model.compile(loss=loss_type, optimizer=opt_adam,metrics=['mse','mae'])
        self.model.summary()
    def runmodel(self,X,Y,X_hat,Y_hat,batch_size,nb_epochs,working_path,model_name="Model2"):
        fn_run_model(self.model,X,Y,X_hat,Y_hat,batch_size,nb_epochs,model_name,working_path)
        self.model.summary()
        #saving weights
        #name_weights = "final_model_fold" + str(j) + "_weights.h5"
    def predict(self,x):
        y=self.model.predict(x)
        return y
    def save_weights(self,path):
        self.model.save_weights(path)
    def load_weights(self,checkpoint_path):
        self.model.load_weights(checkpoint_path)
        print("Weights loaded")
    def evaluate(self,X_eval,Y_eval):
        evaluation = self.model.evaluate(X_eval,Y_eval)
        return evaluation

#%%
"""
MODEL3:

Using deeper conv-LSTM network
```

```
Simplified version of:
https://www.youtube.com/watch?v=MjFpgyWH-pk

"""
class model_3:
    def __init__(self,input_shape,output_shape=None,loss_type="mean_squared_error",kernel_size=3
        self.n_filters=16

        self.model = tf.keras.Sequential([
        tf.keras.Input(input_shape,name='Input'),
        ConvLSTM2D(filters=16, kernel_size=(kernel_size,kernel_size),
                   padding='same', return_sequences=True,
                   activation='tanh', recurrent_activation='tanh',
                   kernel_initializer='glorot_uniform', unit_forget_bias=True,
                   data_format='channels_last', #Specifies which dimension of input data are the
                   dropout=0.5, recurrent_dropout=0.5, go_backwards=True,
                   stateful=False, #If true, the first frame in next batch will be last frame fro
                   strides=(1, 1)
                   ),
        ConvLSTM2D(filters=self.n_filters,kernel_size=(kernel_size,kernel_size),
                   padding='same',return_sequences=True,
                   activation='tanh', recurrent_activation='tanh',
                   kernel_initializer='glorot_uniform', unit_forget_bias=True,
                   data_format='channels_last', #Specifies which dimension of input data are the
                   dropout=0.5, recurrent_dropout=0.5, go_backwards=True,
                   stateful=False, #If true, the first frame in next batch will be last frame fro
                   strides=(1, 1)
                   ),
#        ConvLSTM2D(filters=self.n_filters,kernel_size=(3,3),padding='same',return_sequences=Tru

        TimeDistributed(MaxPool2D(pool_size=(2,2))),

        ConvLSTM2D(filters=self.n_filters,kernel_size=(kernel_size,kernel_size),
                   padding='same',return_sequences=True,
                   activation='tanh', recurrent_activation='tanh',
                   kernel_initializer='glorot_uniform', unit_forget_bias=True,
                   data_format='channels_last', #Specifies which dimension of input data are the
                   dropout=0.5, recurrent_dropout=0.5, go_backwards=True,
                   stateful=False, #If true, the first frame in next batch will be last frame fro
                   strides=(1, 1)
                   ),
        ConvLSTM2D(filters=self.n_filters,kernel_size=(kernel_size,kernel_size),
```

```
                padding='same',return_sequences=True,
                activation='tanh', recurrent_activation='tanh',
                kernel_initializer='glorot_uniform', unit_forget_bias=True,
                data_format='channels_last', #Specifies which dimension of input data are the
                dropout=0.5, recurrent_dropout=0.5, go_backwards=True,
                stateful=False, #If true, the first frame in next batch will be last frame fro
                strides=(1, 1)
                ),
#         ConvLSTM2D(filters=self.n_filters,kernel_size=(3,3),padding='same',return_sequences=Tru

        TimeDistributed(UpSampling2D((2,2))),


        Conv3D(
            filters=1, kernel_size=(kernel_size,kernel_size,kernel_size),
            activation="sigmoid", padding="same"
        ),
        ConvLSTM2D(
            filters=1, kernel_size=(kernel_size, kernel_size),
            padding='same', return_sequences=False, #Return sequences attribute tells whether ou
            activation='sigmoid', recurrent_activation='tanh',
            kernel_initializer='glorot_uniform', unit_forget_bias=True,
            dropout=0.1, recurrent_dropout=0.1, go_backwards=True,
            stateful=False,
            strides=(1, 1)
        )

        ])
        opt = tf.keras.optimizers.Adam(learning_rate=0.005)
        self.model.compile(loss=loss_type,optimizer=opt,metrics= ['mse','mae'])
        self.model.summary()
    def predict(self,x):
        x = self.model.predict(x)

        return x

    def runmodel(self,X,Y,X_hat,Y_hat,batch_size,nb_epochs,working_path,model_name="model3"):
        fn_run_model(self.model,X,Y,X_hat,Y_hat,batch_size,nb_epochs,model_name,working_path)
        self.model.summary()
    def summary(self):
        self.model.summary()
    def save_weights(self,path):
```

```
        self.model.save_weights(path)
    def evaluate(self,X_eval,Y_eval):
        evaluation = self.model.evaluate(X_eval,Y_eval)
        return evaluation
    def load_weights(self,checkpoint_path):
        self.model.load_weights(checkpoint_path)
        print("Weights loaded")
#%%
"""
MODEL 4:
From Keras documentation
https://keras.io/examples/vision/conv_lstm/
https://medium.com/@rajin250/precipitation-prediction-using-convlstm-deep-neural-network-b9e9b61

Maybe add gaussian noise for robustness?

"""
class model_4():
    def __init__(self,input_shape,output_shape=None,loss_type='mean_squared_error',kernel_size=3
        self.model = Sequential(
            [
                tf.keras.Input(
                    shape=input_shape #Input shape for network needs to have same dimensionality
                ),
                ConvLSTM2D(
                    filters=8, kernel_size=(kernel_size, kernel_size),
                    padding='same', return_sequences=True,
                    activation='tanh', recurrent_activation='hard_sigmoid',
                    kernel_initializer='glorot_uniform', unit_forget_bias=True,
                    dropout=0.2, recurrent_dropout=0.5, go_backwards=True,
                    stateful=False, #If true, the first frame in next batch will be last frame f
                    strides=(1, 1)
                ),
                BatchNormalization(),
                TimeDistributed(MaxPool2D(pool_size=(2,2),strides=(2,2),padding="same")),
                ConvLSTM2D(
                    filters=10, kernel_size=(kernel_size, kernel_size),
                    padding='same', return_sequences=True,
                    activation='tanh', recurrent_activation='tanh',
                    kernel_initializer='glorot_uniform', unit_forget_bias=True,
                    dropout=0.2, recurrent_dropout=0.5, go_backwards=True,
                    stateful=False,
```

```
                strides=(1, 1)
            ),
            BatchNormalization(),
            TimeDistributed(UpSampling2D(size=(2,2))),
            ConvLSTM2D(
                filters=10, kernel_size=(kernel_size, kernel_size),
                padding='same', return_sequences=True, #Return sequences attribute tells whe
                activation='tanh', recurrent_activation='tanh',
                kernel_initializer='glorot_uniform', unit_forget_bias=True,
                dropout=0.2, recurrent_dropout=0.5, go_backwards=True,
                stateful=False,
                strides=(1, 1)
            ),
            BatchNormalization(),

            Conv3D(
                filters=1, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
            ),

            ConvLSTM2D(
                filters=1, kernel_size=(kernel_size, kernel_size),
                padding='same', return_sequences=False, #Return sequences attribute tells wh
                activation='sigmoid', recurrent_activation='tanh',
                kernel_initializer='glorot_uniform', unit_forget_bias=True,
                dropout=0.1, recurrent_dropout=0.1, go_backwards=True,
                stateful=False,
                strides=(1, 1)
            )
        ])


    opt = tf.keras.optimizers.Adam(learning_rate=0.005)
    self.model.compile(optimizer=opt,loss=loss_type,metrics=['mse','mae'])
    self.model.summary()
def runmodel(self,X,Y,X_hat,Y_hat,batch_size,nb_epochs,working_path,model_name="Model4"):
    fn_run_model(self.model,X,Y,X_hat,Y_hat,batch_size,nb_epochs,model_name,working_path)
    self.model.summary()
def predict(self,x):
    y=self.model.predict(x)
    return y
def summary(self):
    self.model.summary()
```

```
    def save_weights(self,path):
        self.model.save_weights(path)
    def load_weights(self,checkpoint_path):
        self.model.load_weights(checkpoint_path)
        print("Weights loaded")
    def evaluate(self,X_eval,Y_eval):
        evaluation = self.model.evaluate(X_eval,Y_eval)
        return evaluation
#TODO: add the prediction as ground truth to get predictions further in the future?

#%%
"""
MODEL 5:
Network inspired by:
https://www.youtube.com/watch?v=MjFpgyWH-pk
Input dimensions need to be even for this architechture to work
"""
class model_5():
    def __init__(self,input_shape,output_shape=None,loss_type='mean_squared_error',kernel_size=3
        self.n_filters=8
        self.output_filters = 3
        self.n_downsamples = 1
        if input_shape[2]%2**self.n_downsamples!=0 or input_shape[3]%2**self.n_downsamples!=0:
            #Test whether input shape is still an int after downsampling with n (2x2) maxpooling
            print("Input shape is no longer even after downsampling. Upsampling to same shape wi
            #If it is the case, zero pad the input shape?
            #First, finding the incompatible input dimension
#           dim1 = input_shape[2]%2**n_downsamples
#           dim2 = input_shape[3]%2**n_downsamples
    #       if dim1!=0:
    #           c1 = tf.keras.layers.ZeroPadding3D(padding=((dim1,0),(0,0),(0,0)))(c1)
    #       elif dim2!=0:
    #           c1 = tf.keras.layers.ZeroPadding3D(padding=((0,0),(dim2,0),(0,0)))(c1)
    #       else:
    #           #if both are incompatible
    #           input_img = tf.keras.layers.ZeroPadding3D(padding=((dim1,0),(dim2,0),(0,0)))(in

        self.input_img = tf.keras.Input(input_shape,name='Input')

            #or reshape entire
        self.x = ConvLSTM2D(filters=self.n_filters,kernel_size=(3,3),padding='same',return_seque
        self.x = ConvLSTM2D(filters=self.n_filters,kernel_size=(3,3),padding='same',return_seque
```

```
        self.c1 = ConvLSTM2D(filters=self.n_filters,kernel_size=(3,3),padding='same',return_sequ

        self.x = TimeDistributed(MaxPool2D(pool_size=(2,2),strides=(2,2),padding="same"))(self.c

        self.x = ConvLSTM2D(filters=2*self.n_filters,kernel_size=(3,3),padding='same',return_seq
        self.x = ConvLSTM2D(filters=2*self.n_filters,kernel_size=(3,3),padding='same',return_seq
        self.c2 = ConvLSTM2D(filters=2*self.n_filters,kernel_size=(3,3),padding='same',return_se

        self.x = TimeDistributed(UpSampling2D(size=(2,2)))(self.c2)
        self.x = tf.keras.layers.Concatenate()([self.c1,self.x])

        self.output = ConvLSTM2D(filters=1,kernel_size=(3,3),padding='same',return_sequences=Fal
        self.model = tf.keras.Model(self.input_img,self.output)

        opt_adam = tf.keras.optimizers.Adam(learning_rate=0.005)
        self.model.compile(optimizer=opt_adam,loss=loss_type,metrics=['mse','mae'])
        self.model.summary()
    def runmodel(self,X,Y,X_hat,Y_hat,batch_size,nb_epochs,working_path,model_name="Model5"):
        fn_run_model(self.model,X,Y,X_hat,Y_hat,batch_size,nb_epochs,model_name,working_path)
        self.model.summary()
        #saving weights
        #name_weights = "final_model_fold" + str(j) + "_weights.h5"
    def predict(self,x):
        y=self.model.predict(x)
        return y
    def summary(self):
        self.model.summary()
    def save_weights(self,path):
        self.model.save_weights(path)
    def load_weights(self,checkpoint_path):
        self.model.load_weights(checkpoint_path)
        print("Weights loaded")
    def evaluate(self,X_eval,Y_eval):
        evaluation = self.model.evaluate(X_eval,Y_eval)
        return evaluation

        #%% Model 6
"""
MODEL 6:
Network inspired by model 5 with noise and conv3D
Input dimensions need to be even for this architechture to work. for is_int(input_dim/2**n), the
"""
```

```
class model_6():
    def __init__(self,input_shape,output_shape=None,loss_type='mean_squared_error',kernel_size=3
        self.n_filters=10
        self.output_filters=3
        self.n_downsamples = 1
        self.kernel_size = 3
        if input_shape[2]%2**self.n_downsamples!=0 or input_shape[3]%2**self.n_downsamples!=0:
            #Test whether input shape is still an int after downsampling with n (2x2) maxpooling
            print("Input shape is no longer even after downsampling. Upsampling to same shape wi

        self.input_img = tf.keras.Input(input_shape,name='Input')
        self.x=tf.keras.layers.GaussianNoise(0.2)(self.input_img)
            #or reshape entire
        self.x = ConvLSTM2D(filters=self.n_filters,kernel_size=(self.kernel_size,self.kernel_siz
                            padding='same',return_sequences=True,
                            data_format='channels_last',
                            recurrent_dropout=0.3)(self.x)
        self.x = ConvLSTM2D(filters=self.n_filters,kernel_size=(self.kernel_size,self.kernel_siz
                            padding='same',return_sequences=True,
                            data_format='channels_last',
                            recurrent_dropout=0.3)(self.x)
        self.c1 = ConvLSTM2D(filters=self.n_filters,kernel_size=(self.kernel_size,self.kernel_si
                            padding='same',return_sequences=True,
                            data_format='channels_last',
                            recurrent_dropout=0.3)(self.x)

        self.x = TimeDistributed(MaxPool2D(pool_size=(2,2),strides=(2,2),padding="same"))(self.c

        self.x = ConvLSTM2D(filters=2*self.n_filters,kernel_size=(self.kernel_size,self.kernel_s
                            padding='same',return_sequences=True,
                            data_format='channels_last',
                            recurrent_dropout=0.3)(self.x)
        self.x = ConvLSTM2D(filters=2*self.n_filters,kernel_size=(self.kernel_size,self.kernel_s
                            padding='same',return_sequences=True,
                            data_format='channels_last',
                            recurrent_dropout=0.3)(self.x)
        self.c2 = ConvLSTM2D(filters=2*self.n_filters,kernel_size=(self.kernel_size,self.kernel_
                            padding='same',return_sequences=True,
                            data_format='channels_last',
                            recurrent_dropout=0.3)(self.x)

        self.x = TimeDistributed(UpSampling2D(size=(2,2)))(self.c2)
```

```
        self.x = tf.keras.layers.Concatenate()([self.c1,self.x])

        self.x = Conv3D(filters=self.output_filters,kernel_size=(self.kernel_size,self.kernel_si
                        padding="same",data_format='channels_last')(self.x)

        self.output = ConvLSTM2D(filters=1,kernel_size=(self.kernel_size,self.kernel_size),
                                 padding='same',return_sequences=False,
                                 data_format='channels_last')(self.x)
        self.model = tf.keras.Model(self.input_img,self.output)

        opt_adam = tf.keras.optimizers.Adam(learning_rate=0.005)
        self.model.compile(optimizer=opt_adam,loss=loss_type,metrics=['mse','mae'])
        self.model.summary()
    def runmodel(self,X,Y,X_hat,Y_hat,batch_size,nb_epochs,working_path,model_name="Model6"):
        fn_run_model(self.model,X,Y,X_hat,Y_hat,batch_size,nb_epochs,model_name,working_path)
        self.model.summary()
        #saving weights
        #name_weights = "final_model_fold" + str(j) + "_weights.h5"
    def predict(self,x):
        y=self.model.predict(x)
        return y
    def summary(self):
        self.model.summary()
    def save_weights(self,path):
        self.model.save_weights(path)
    def load_weights(self,checkpoint_path):
        self.model.load_weights(checkpoint_path)
        print("Weights loaded")
    def evaluate(self,X_eval,Y_eval):
        evaluation = self.model.evaluate(X_eval,Y_eval)
        return evaluation
```

A.4 Model evaluation

```
# -*- coding: utf-8 -*-
"""
Model Evaluation.

Library for evaluation functions.
@author: Carl Emil Elling
08-07-2020
"""
```

```
import numpy as np
import tensorflow as tf

import matplotlib as mpl
import matplotlib.animation as animation
import matplotlib.pyplot as plt

import Model_lib
#%% Load model:

def model_load(model,input_shape,checkpoint_path,loss_type='mean_squared_error'):
    model=Model_lib.model(input_shape,loss_type)
    model.load_weights(checkpoint_path)
    return model

#%%MODEL EVALUATION
"""
Evaluation of model. Based on Zero Normalized Cross Correlation. A perfect correlation is 1, a p
seeing the correlation over time - to check for actual prediction or just learning prev step
"""
t=1
def eval_ZNCC(model,X,Y,t=1):
    #Evaluates a model based on zero-normalized cross correlation between target and prediction.
    #Also returns the ZNCC for prediction and target time-shifted t spots.
    #This to check whether model accuracy is only a figment of it copying input
    totcor = []
    shiftcor = []
    for i in range(np.size(Y,axis=0)):
        #loop over all targets
        pred = model.predict(X[i:i+1,:,:,:,:])


        #zero-normalized cross correlation
        im1 = Y[i,:,:,0]
        im2 = pred[0,:,:,0]
        if np.std(im1)!=0 and np.std(im2)!=0:
            im1 = (im1-np.mean(im1))/np.std(im1)
            im2 = (im2-np.mean(im2))/np.std(im2)
            cor  = np.sum(im1*im2)/np.size(im1) #ZNCC
            totcor.append(cor) #appended to a list to be able to plot it
        # elif np.std(im1)=0 and np.std(im2)=0:
        #     #If both images are 0, total correlation
```

```
            #     totcor.append(1)
                #If one image is only zeroes, it doesn't make sense to correlate with another. Thus,

            #Correlation with 1-shifted input
        for i in range(np.size(Y,axis=0)-t):
            pred = model.predict(X[i:i+1,:,:,:,:])
            im1 = Y[i-t,:,:,0] #shifting target 1 timestep forward to check whether model only learn
            im2 = pred[0,:,:,0]
            if np.std(im1)!=0 and np.std(im2)!=0:
                im1 = (im1-np.mean(im1))/np.std(im1)
                im2 = (im2-np.mean(im2))/np.std(im2)
                cor  = np.sum(im1*im2)/np.size(im1)
                shiftcor.append(cor)
            # elif np.std(im1)=0 and np.std(im2)=0:
            #     #If both images are 0, total correlation
            #     totcor.append(1)

        totcor_sum=sum(totcor)/len(totcor)
        shiftcor_sum=sum(shiftcor)/len(shiftcor)
        return totcor,shiftcor,totcor_sum,shiftcor_sum


    #%% Baseline set
    def createBaseline(X,Y,t=1):
        #creates a baseline if the network copies the last frame.
        totcor = []
        shiftcor = []
        for i in range(np.size(Y,axis=0)):
            #last frame of
            pred = X[i:i+1,9,:,:,:]


            #zero-normalized cross correlation
            im1 = Y[i,:,:,0]
            im2 = pred[0,:,:,0]
            if np.std(im1)!=0 and np.std(im2)!=0:
                im1 = (im1-np.mean(im1))/np.std(im1)
                im2 = (im2-np.mean(im2))/np.std(im2)
                cor  = np.sum(im1*im2)/np.size(im1) #ZNCC
                totcor.append(cor) #appended to a list to be able to plot it
            # elif np.std(im1)=0 and np.std(im2)=0:
            #     #If both images are 0, total correlation
```

```
        #     totcor.append(1)
            #If one image is only zeroes, it doesn't make sense to correlate with another. Thus,

        #Correlation with 1-shifted input
    for i in range(np.size(Y,axis=0)-t):
        pred = pred = X[i:i+1,9,:,:,:]
        im1 = Y[i-t,:,:,0] #shifting target 1 timestep forward to check whether model only learn
        im2 = pred[0,:,:,0]
        if np.std(im1)!=0 and np.std(im2)!=0:
            im1 = (im1-np.mean(im1))/np.std(im1)
            im2 = (im2-np.mean(im2))/np.std(im2)
            cor  = np.sum(im1*im2)/np.size(im1)
            shiftcor.append(cor)
        # elif np.std(im1)=0 and np.std(im2)=0:
        #     #If both images are 0, total correlation
        #     totcor.append(1)

    totcor_sum=sum(totcor)/len(totcor)
    shiftcor_sum=sum(shiftcor)/len(shiftcor)
    return totcor,shiftcor,totcor_sum,shiftcor_sum

#%%
def evaluate(model,model_name,x_test,target_test):
    #Creates evaluation array
    #Inputs model, and test set (X,Y)

    #First, standard evaluation on loss (rather useless comparison since different loss function
    evaluation=model.evaluate(x_test,target_test)
    #Then, ZNCC. Also differ ence between ZNCC and time shifted ZNCC. If positive, ZNCC is bette
    totcor,shiftcor,totcor_sum,shiftcor_sum=eval_ZNCC(model,x_test,target_test)
    evaluation=np.append(evaluation,totcor_sum) #ZNCC
    evaluation=np.append(evaluation,shiftcor_sum) #shifted ZNCC
    evaluation=np.append(evaluation,totcor_sum-shiftcor_sum) #ZNCC difference
    evaluation=np.append(evaluation,model_name) #appending model name
    return evaluation
```

A.5 Main script

```
# -*- coding: utf-8 -*-
"""
Inspiration from MIT Deep learning & https://www.youtube.com/watch?v=fmgaOiOMXuU
```

```
@author: Carl Emil Elling
08-07-2020
Resized data from the start. Single frame as target. Multiple channels of data as input

Remember to set this script as working directory
"""



#%% Visualize an area of a data type to use in training
data_format = cpue_resized
vararea=data_format[:,:,:]
#max and min values of the dataset
cpuemax = np.amax(cpue)
cpuemin = np.amin(cpue)
varmax = np.amax(vararea)
varmin = np.amin(vararea)
#colormap and normalization
cmap = mpl.cm.jet
norm = mpl.colors.Normalize(vmin=cpuemin,vmax=cpuemax)
varnorm = mpl.colors.Normalize(vmin=varmin,vmax=varmax)
#margins for the map projection
ymargin = 0
xmargin = 0

#Setting up the figure
fig=plt.figure()
spec = mpl.gridspec.GridSpec(ncols=2, nrows=1, figure=fig,width_ratios=[15,1])
#Axis with coastlines
ax2 = fig.add_subplot(spec[0],projection=ccrs.PlateCarree())
ax2.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y
#colorbar
cax = fig.add_subplot(spec[1])
#cb1 = mpl.colorbar.ColorbarBase(cax, cmap=cmap,
#                                norm=norm,
#                                orientation='vertical')

varcb = mpl.colorbar.ColorbarBase(cax, cmap=cmap,
                                  norm=varnorm,
                                  orientation='vertical')
cax.set_ylabel('Log10(CPUE)')
```

```
def onClick(event):
    #For pausing animation
    #From: https://stackoverflow.com/questions/16732379/stop-start-pause-in-python-matplotlib-an
    global pause
    pause ^= True
fig.canvas.mpl_connect('button_press_event', onClick)

def animatepred(i):
    #Function to be repeated in animation

    ax2.clear()

    #visualize as contour
#    ax2.contourf(lon_resized, lat_resized, vararea[i,:,:], 10,
#                 transform=ccrs.PlateCarree(),norm=varnorm,cmap=cmap)
    #visualize raster
    ax2.pcolormesh(lon_resized, lat_resized, vararea[i,:,:],
                   transform=ccrs.PlateCarree(),norm=varnorm,cmap=cmap)

    ax2.coastlines(resolution='10m', color='black', linewidth=1)

    #ax2.add_feature(cfeature.NaturalEarthFeature('physical', 'land', '50m', edgecolor='black',l

    ax2.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(la
    ax2.set_title('CPUE predicted iter: %d'%(i))
    return ax2

anim1 = animation.FuncAnimation(fig,animatepred,frames=241,blit=False,interval=200)
plt.show()

#%% ------ create learning set ------
Seq_size=10
def to_sequence(seq_size,obs,dims=False,sample=False):
    #Here, observations are the 2D spatial data for each time point
    #Returns a seq_size length sequence of the observations, with the immediate next observation
    #Sample is a tuple containing (latmin,latmax),(lonmin,lonmax)
    x = []
    xnew=[]
    y = []
    #obs.reshape((obs.shape[0],obs.shape[1]*obs.shape[2])) #reshaping into 2D observations. Not
    for i in range(np.size(obs,0)-seq_size): #Taking the total abount of observation points and
```

```
            x.append(obs[i:i+seq_size]) #training set
            y.append(obs[i+seq_size]) #value after training set.
        x=np.array(x)
        y=np.array(y)
        y=y[:,:,:,0,np.newaxis] #Chose only the CPUE as target. To have same dimensionality as the t
        if isinstance(dims,tuple):
            #If dims is specified, only select these dims
            dimdict = {
                    "cpue":0,
                    "msl":1,
                    "u10":2,
                    "v10":3,
                    "attn":4,
                    "CHL":5,
                    "mld":6,
                    "salt":7,
                    "sst":8,
                    "swh":9,
                    "sst_front":10,
                    "salt_front":11,
                    "all":[0,1,2,3,4,5,6,7,8,9,10,11] #edit this for doing custom, easier selections
                    }

            for channel in dims:
                #loop over all chosen dimensions
                if channel in dimdict:
                    #check if channel name is in dictionary
                    print(dimdict[channel])
                    xnew.append(x[:,:,:,:,dimdict[channel]])
                else:
                    print("channel:'%s' is not a valid parameter"%channel)
            x=np.array(xnew)
            x=np.transpose(x,[1,2,3,4,0])
        if isinstance(sample,tuple):
            #if there is a sample area specified, extract only sampled area
            x=x[:,:,sample[0][0]:sample[0][1],sample[1][0]:sample[1][1],:] #small part of data for t
            y=y[:,:,sample[0][0]:sample[0][1],sample[1][0]:sample[1][1],:]

        #TODO: Make sequences randomly picked from observations. This in order to learn more robustl
        return x,y
    #dims=("cpue","msl","salt_front")
    dims=("all")
```

```
x_train,target_train = to_sequence(Seq_size,train_data,dims=dims)
x_val,target_val = to_sequence(Seq_size,val_data,dims=dims)
x_test,target_test = to_sequence(Seq_size,test_data,dims=dims)
input_shape_simple=np.shape(x_train)[1:] #Shape is (batch size,lat,lon,channels). The number of
output_shape_simple=np.shape(target_train)[1:]


#%% Another learning set - with target as ground truth shifted by 1 timestep
def to_sequence_shift(seq_size,obs,dims=False,sample=False):
    #Here, observations are the 2D spatial data for each time point
    #Returns a sequence of length seq_size and target data.
    #Target data is a sequence of the same length, shifted by 1 in a positive timestep
    #Sample is a tuple containing (latmin,latmax),(lonmin,lonmax) used to sample a specific area
    x = []
    xnew=[]
    y = []
    for i in range(np.size(obs,0)-seq_size): #Taking the total abount of observation points and
        x.append(obs[i:i+seq_size]) #training set
        y.append(obs[(i+1):(i+seq_size+1)]) #values shifted by 1 in the number of frames
    x=np.array(x)
    y=np.array(y)
    y=y[:,:,:,:,0,np.newaxis] #Chose only the CPUE as target. To have same dimensionality as the
    if isinstance(dims,tuple):


        #If dims is specified, only select these dims
        dimdict = {
                "cpue":0,
                "msl":1,
                "u10":2,
                "v10":3,
                "attn":4,
                "CHL":5,
                "mld":6,
                "salt":7,
                "sst":8,
                "swh":9,
                "sst_front":10,
                "salt_front":11,
                "all":[0,1,2,3,4,5,6,7,8,9,10,11] #edit this for doing custom, easier selections
                }
```

```
        for channel in dims:
            #loop over all chosen dimensions
            if channel in dimdict:
                #check if channel name is in dictionary
                print(dimdict[channel])
                xnew.append(x[:,:,:,:,dimdict[channel]])
            else:
                print("channel:'%s' is not a valid parameter"%channel)
        x=np.array(xnew)
        x=np.transpose(x,[1,2,3,4,0])
    if isinstance(sample,tuple):
        #if there is a sample area specified, extract only sampled area
        x=x[:,:,sample[0][0]:sample[0][1],sample[1][0]:sample[1][1],:] #small part of data for t
        y=y[:,:,sample[0][0]:sample[0][1],sample[1][0]:sample[1][1],:]

    #TODO: Make sequences randomly picked from observations. This in order to learn more robustl
    return x,y
#dims=("cpue","msl","salt_front")
dims=("all")
shifted_x_train,shifted_target_train = to_sequence_shift(Seq_size,train_data,dims=dims)
shifted_x_val,shifted_target_val = to_sequence_shift(Seq_size,val_data,dims=dims)
shifted_x_test,shifted_target_test = to_sequence_shift(Seq_size,val_data,dims=dims)

input_shape_simple=np.shape(shifted_x_train)[1:] #Shape is (batch size,lat,lon,channels). The nu
output_shape_simple=np.shape(shifted_target_train)[1:]

#%% Make sure targets fit data - for debugging purposes
fig = plt.figure()
cmap = mpl.cm.jet
cmax = np.amax(cpue_resized)
cmin = np.amin(cpue_resized)
norm = mpl.colors.Normalize(vmin=cmin,vmax=cmax)
xmargin=0
ymargin=0

spec = mpl.gridspec.GridSpec(ncols=3,nrows=1,figure=fig,width_ratios=[15,15,1])
ax1 = fig.add_subplot(spec[0],projection=ccrs.PlateCarree())
ax2 = fig.add_subplot(spec[1],projection=ccrs.PlateCarree())
cax = fig.add_subplot(spec[2])
cb1 = mpl.colorbar.ColorbarBase(cax, cmap=cmap,
                                norm=norm,
                                orientation='vertical')
```

```
ax1.coastlines(resolution='10m', color='black', linewidth=1)
ax1.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y
ax2.coastlines(resolution='10m', color='black', linewidth=1)
ax2.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y
ax1.set_title("Last observation in next training")
ax2.set_title("Target")
cax.set_ylabel("log10(CPUE)")

i=29
ax1.contourf(lon_resized,lat_resized,x_val[1+i,9,:,:,0],100,transform=ccrs.PlateCarree(),norm=no
ax2.contourf(lon_resized,lat_resized,target_val[0+i,:,:,0],100,transform=ccrs.PlateCarree(),norm


#%% Fitting function for running all models

#%%
"""### Initializing neural nets ###"""


#%% Fitting model 1
"""
Model 1
Same low complexity network trained on different cost functions, with different optimizers and d
To determine good parameters for simple ConvLSTM, to be applied to more complex architechtures.
"""
epochs_model1=30

##
#Simplest model - mse loss
model1a=Model_lib.model_1(input_shape_simple,output_shape_simple)
model1a.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mode

# using custom rmse loss
model1b=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.custom_rmse
model1b.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mode

## using weighted loss on zero predictions - 0.1*error for zero-predictions
model1c01=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c01.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mo
## using weighted loss on zero predictions - 0.2*error for zero-predictions
model1c02=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c02.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mo
```

```
# using weighted loss on zero predictions - 0.4*error for zero-predictions
model1c04=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c04.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mo
# using weighted loss on zero predictions - 0.6*error for zero-predictions
model1c06=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c06.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mo
# using weighted loss on zero predictions - 0.8*error for zero-predictions
model1c08=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c08.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mo
#%% Loading already saved model weights
model1a=Model_lib.model_1(input_shape_simple,output_shape_simple)
model1a.load_weights(str(Path(working_path) / "model_weights/model1a/1/model.ckpt"))

model1b=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.custom_rmse
model1b.load_weights(str(Path(working_path) / "model_weights/model1b/1/model.ckpt"))

model1c01=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c01.load_weights(str(Path(working_path) / "model_weights/model1c01/1/model.ckpt"))

model1c02=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c02.load_weights(str(Path(working_path) / "model_weights/model1c02/1/model.ckpt"))

model1c04=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c04.load_weights(str(Path(working_path) / "model_weights/model1c04/1/model.ckpt"))

model1c06=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c06.load_weights(str(Path(working_path) / "model_weights/model1c06/1/model.ckpt"))

model1c08=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedR
model1c08.load_weights(str(Path(working_path) / "model_weights/model1c08/1/model.ckpt"))

#%% Evaluation of different weights
eval1a=Model_evaluate.evaluate(model1a,"mse loss",x_test,target_test)
eval1b=Model_evaluate.evaluate(model1b,"rmse loss",x_test,target_test)
eval1c01=Model_evaluate.evaluate(model1c01,"weighted loss 0.1",x_test,target_test)
eval1c02=Model_evaluate.evaluate(model1c02,"weighted loss 0.2",x_test,target_test)
eval1c04=Model_evaluate.evaluate(model1c04,"weighted loss 0.4",x_test,target_test)
eval1c06=Model_evaluate.evaluate(model1c06,"weighted loss 0.6",x_test,target_test)
eval1c08=Model_evaluate.evaluate(model1c08,"weighted loss 0.8",x_test,target_test)


#%% Plot values
```

```
#each eval array contains:loss,mse,mae,corr,shiftcor,diffcor,model name

plotdata = np.array([eval1a,eval1b,eval1c01,eval1c02,eval1c04,eval1c06,eval1c08])
plotdata = np.insert(plotdata,0,np.array(["Loss","MSE","MAE","Corrr","Shiftcorr","Diffcorr","Mod
colors = mpl.cm.rainbow(np.linspace(0, 1, np.shape(plotdata)[0]))
plt.title("Evaluation parameters for same model, different cost functions")
# plt.tick_params(
#     axis='x',          # changes apply to the x-axis
#     which='both',      # both major and minor ticks are affected
#     bottom=False,      # ticks along the bottom edge are off
#     top=False,         # ticks along the top edge are off
#     labelbottom=False) # labels along the bottom edge are off
for model,c in zip(plotdata[1:],colors):
      # print(model[:-1])
      #print(c)
      plt.scatter(np.linspace(0,5,np.shape(model)[0]-1),model[:-1].astype(np.float),color=c,labe
plt.legend()
plt.xticks(np.arange(6),plotdata[0,:-1])
plt.grid(ls="--")

#%% further model 1 parameter tuning
#Using 0.6 and more epochs
model1d=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMS
model1d.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_mode

#Optimizers:
#Adam high learning rate
model1eAdam01=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weigh
model1eAdam01.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epoch

#Adam lower learning rate, more epochs
model1eAdam003=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.weig
model1eAdam003.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epoc


#Adelta
model1eAdadelta=Model_lib.model_1(input_shape_simple,output_shape_simple,loss_type=Model_lib.wei
model1eAdadelta.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epo
#%% Load model weights
model1d=Model_lib.model_1(input_shape_simple,output_shape_simple)
model1d.load_weights(str(Path(working_path) / "model_weights/model1d/1/model.ckpt"))
model1eAdam01=Model_lib.model_1(input_shape_simple,output_shape_simple)
```

```
model1eAdam01.load_weights(str(Path(working_path) / "model_weights/model1eAdam0_01/1/model.ckpt"
model1eAdam003=Model_lib.model_1(input_shape_simple,output_shape_simple)
model1eAdam003.load_weights(str(Path(working_path) / "model_weights/model1eAdam0_003/1/model.ckp
model1eAdadelta=Model_lib.model_1(input_shape_simple,output_shape_simple)
model1eAdadelta.load_weights(str(Path(working_path) / "model_weights/model1eAdadelta/1/model.ckp
#%%
#Evaluate
eval1d=Model_evaluate.evaluate(model1d,"weighted loss 0.6, 2x epochs",x_test,target_test)
eval1eAdam01=Model_evaluate.evaluate(model1eAdam01,"Adam: lr 0.01",x_test,target_test)
eval1eAdam003=Model_evaluate.evaluate(model1eAdam003,"Adam: lr 0.003",x_test,target_test)
eval1eAdadelta=Model_evaluate.evaluate(model1eAdadelta,"Adadelta",x_test,target_test)

#%%
plotdata = np.array([eval1a,eval1b,eval1c01,eval1c02,eval1c04,eval1c06,eval1c08,eval1d])
plotdata = np.insert(plotdata,0,np.array(["Loss","MSE","MAE","Corrr","Shiftcorr","Diffcorr","Mod
colors = mpl.cm.rainbow(np.linspace(0, 1, np.shape(plotdata)[0]))
plt.title("Evaluation parameters for same model, different cost functions")
# plt.tick_params(
#     axis='x',          # changes apply to the x-axis
#     which='both',      # both major and minor ticks are affected
#     bottom=False,      # ticks along the bottom edge are off
#     top=False,         # ticks along the top edge are off
#     labelbottom=False) # labels along the bottom edge are off
for model,c in zip(plotdata[1:],colors):
      # print(model[:-1])
      #print(c)
      plt.scatter(np.linspace(0,5,np.shape(model)[0]-1),model[:-1].astype(np.float),color=c,labe
plt.legend()
plt.xticks(np.arange(6),plotdata[0,:-1])
plt.grid(ls="--")

#%% Optimizer comparison
plotdata = np.array([eval1eAdam01,eval1eAdam003,eval1eAdadelta])
plotdata = np.insert(plotdata,0,np.array(["Loss","MSE","MAE","Corrr","Shiftcorr","Diffcorr","Mod
colors = mpl.cm.rainbow(np.linspace(0, 1, np.shape(plotdata)[0]))
plt.title("Evaluation parameters for same model, different optimizers")
# plt.tick_params(
#     axis='x',          # changes apply to the x-axis
#     which='both',      # both major and minor ticks are affected
#     bottom=False,      # ticks along the bottom edge are off
#     top=False,         # ticks along the top edge are off
#     labelbottom=False) # labels along the bottom edge are off
```

```
for model,c in zip(plotdata[1:],colors):
      # print(model[:-1])
      #print(c)
      plt.scatter(np.linspace(0,5,np.shape(model)[0]-1),model[:-1].astype(np.float),color=c,labe
plt.legend()
plt.xticks(np.arange(6),plotdata[0,:-1])
plt.grid(ls="--")
#%%
"""
Model 2
"""
epochs_model2 = 30
model2=Model_lib.model_2(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMSE
model2.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_model

#%%
"""
MODEL3:
"""
epochs_model3 = 30
model3 =Model_lib.model_3(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMS
model3.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_model


#%% Model 4
"""
MODEL 4:
Modification of keras ConvLSTM documentation
"""
epochs_model4 = 60
model4 =Model_lib.model_4(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMS
model4.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_model



#%% Model 5 - as functional style


"""
MODEL 5:
Network inspired by:
https://www.youtube.com/watch?v=MjFpgyWH-pk
```

```
Input dimensions need to be even for this architechture to work
"""

model5 = Model_lib.model_5(input_shape_simple,loss_type=Model_lib.weightedRMSELoss(0.5))
epochs_model5=80
model5.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_model


#%% Model 6
"""
MODEL 6:
Network inspired by model 5 with noise and conv3D
Input dimensions need to be even for this architechture to work. for isloss_type=Model_lib.weigh
"""

model6 = Model_lib.model_6(input_shape_simple,loss_type=Model_lib.weightedRMSELoss(0.5))
epochs_model6=80
model6.runmodel(x_train,target_train,x_val,target_val,batch_size=Seq_size,nb_epochs=epochs_model

#%% Load already saved models
model2=Model_lib.model_2(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMSE
model2.load_weights(str(Path(working_path) / "model_weights/model2/1/model.ckpt"))

model3=Model_lib.model_3(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMSE
model3.load_weights(str(Path(working_path) / "model_weights/model3/1/model.ckpt"))

model4=Model_lib.model_4(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMSE
model4.load_weights(str(Path(working_path) / "model_weights/model4/1/model.ckpt"))

model5=Model_lib.model_5(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMSE
model5.load_weights(str(Path(working_path) / "model_weights/model5/1/model.ckpt"))

model6=Model_lib.model_6(input_shape_simple,output_shape_simple,loss_type=Model_lib.weightedRMSE
model6.load_weights(str(Path(working_path) / "model_weights/model6/1/model.ckpt"))


#%% Prediction tests

""" Single image prediction """
i = 20 #frame no
model = model2 #model to use
```

```
pred = model2.predict(x_test[i:i+1,:,:,:,:])
predtest=pred[0,:,:,0]
target = target_test[i,:,:,0]

cpuemax = np.amax(cpue_resized)
cpuemin = np.amin(cpue_resized)
cmap = mpl.cm.jet
norm = mpl.colors.Normalize(vmin=cpuemin,vmax=cpuemax)
ymargin = .5
xmargin = 0
fig=plt.figure()
spec = mpl.gridspec.GridSpec(ncols=2, nrows=2, figure=fig,width_ratios=[15,1])
ax2 = fig.add_subplot(spec[0],projection=ccrs.PlateCarree())
ax3 = fig.add_subplot(spec[2],projection=ccrs.PlateCarree())
ax2.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y
ax3.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y

cax = fig.add_subplot(spec[1])
cb1 = mpl.colorbar.ColorbarBase(cax, cmap=cmap,
                                norm=norm,
                                orientation='vertical')
ax2.pcolormesh(lon_resized, lat_resized, pred[0,:,:,0], 100,
             transform=ccrs.PlateCarree(),norm=norm,cmap=cmap)
ax3.pcolormesh(lon_resized, lat_resized, target, 100,
             transform=ccrs.PlateCarree(),norm=norm,cmap=cmap)
ax2.coastlines(resolution='10m', color='black', linewidth=1)
ax2.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y
ax3.coastlines(resolution='10m', color='black', linewidth=1)
ax3.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y

ax2.set_title('Prediction')
ax3.set_title('Ground Truth')
cax.set_ylabel('Log10(CPUE)')
#%%MODEL EVALUATION
"""
Evaluation of model. Based on Zero Normalized Cross Correlation. A perfect correlation is 1, a p
seeing the correlation over time - to check for actual prediction or just learning prev step
"""
eval2=Model_evaluate.evaluate(model2,"Model 2:Bi-directional ConvLSTM",x_test,target_test)
eval3=Model_evaluate.evaluate(model3,"Model 3:Deeper network",x_test,target_test)
eval4=Model_evaluate.evaluate(model4,"Model 4:Keras ConvLSTM doc",x_test,target_test)
```

```
eval5=Model_evaluate.evaluate(model5,"Model 5:maxpool+upscale",x_test,target_test)
eval6=Model_evaluate.evaluate(model6,"Model 6:maxpool+upscale+noise+3Dconvolution",x_test,target
#%%
totcorBase,shiftcorBase,totcorsumBase,shiftcorsumBase=Model_evaluate.createBaseline(x_test,targe
#%% Evaluation of complex models
# plotdata = np.array([eval2,eval3,eval4,eval5,eval6])
plotdata = np.array([eval1a,eval1b,eval1c01,eval1c02,eval1c04,eval1c06,eval1c08,eval1d,eval1eAda

plotdata = np.insert(plotdata,0,np.array(["Loss","MSE","MAE","Corr","Shifted corr","Total+diff",
colors = mpl.cm.rainbow(np.linspace(0, 1, np.shape(plotdata)[0]))
plt.title("Evaluation parameters for different models, evaluated on test set")
# plt.tick_params(
#     axis='x',          # changes apply to the x-axis
#     which='both',      # both major and minor ticks are affected
#     bottom=False,      # ticks along the bottom edge are off
#     top=False,         # ticks along the top edge are off
#     labelbottom=False) # labels along the bottom edge are off
for model,c in zip(plotdata[1:],colors):
      # print(model[:-1])
      #print(c)
      plt.scatter(np.linspace(0,4,np.shape(model)[0]-2),model[1:-1].astype(np.float),color=c,lab
plt.legend(loc='upper left')
plt.xticks(np.arange(5),plotdata[0,1:-1])
plt.grid(ls="--")


#%% getweek

#%%
"""Model animation"""
model=model1eAdam003
# model_name="Model 6:Deep NN w/ maxpool+upscale+noise+3Dconvolution"
model_name="Model 1e: Adam w/ learning rate 0.003"

cpuemax = np.amax(cpue_resized)
cpuemin = np.amin(cpue_resized)
cmap = mpl.cm.jet
norm = mpl.colors.Normalize(vmin=cpuemin,vmax=cpuemax)
ymargin = .5
xmargin = 0
fig=plt.figure()
spec = mpl.gridspec.GridSpec(ncols=2, nrows=2, figure=fig,width_ratios=[15,1])
```

```
#, [ax2,cax] = plt.subplots(1,2,gridspec_kw={"width_ratios":[15,1]})
ax2 = fig.add_subplot(spec[0],projection=ccrs.PlateCarree())
ax3 = fig.add_subplot(spec[2],projection=ccrs.PlateCarree())
ax2.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y
ax3.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(lat)+y

cax = fig.add_subplot(spec[1])
cb1 = mpl.colorbar.ColorbarBase(cax, cmap=cmap,
                                norm=norm,
                                orientation='vertical')
cax.set_ylabel('Log10(CPUE)')
fig.suptitle(model_name)



def animateboth(i):
    #xin=x_test[1:,:,:,:,:]+pred
    pred = model.predict(x_test[i:i+1,:,:,:,:])
    yr=int(time_test[i+10]) #year
    mo=int((time_test[i+10]-yr)*12)+1#month
    wk=int((time_test[i+10]-yr)*52)+1 #week
    predtest=pred[0,:,:,0]
    realtest=target_test[i,:,:,0]
    cpuemax = np.amax(cpue)
    cpuemin = np.amin(cpue)
    norm = mpl.colors.Normalize(vmin=cpuemin,vmax=cpuemax)
    ax2.clear()
    ax3.clear()
    ax2.pcolormesh(lon_resized, lat_resized, predtest,
            transform=ccrs.PlateCarree(),norm=norm,cmap=cmap)
    ax2.coastlines(resolution='10m', color='black', linewidth=1)
    ax2.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(la
    ax2.set_title('CPUE predicted, week: %d %d'%(wk,yr))
    ax3.pcolormesh(lon_resized, lat_resized, realtest,
                transform=ccrs.PlateCarree(),norm=norm,cmap=cmap)
    ax3.coastlines(resolution='10m', color='black', linewidth=1)
    ax3.set_extent([np.amin(lon)-xmargin, np.amax(lon)+xmargin, np.amin(lat)-ymargin, np.amax(la
    ax3.set_title('CPUE real, week: %d %d'%(wk,yr))
    return ax2,ax3

anim3 = animation.FuncAnimation(fig,animateboth,frames=np.size(x_test,axis=0),interval=600,blit=
plt.show()
```

```
#For writing final animation
savedir = Path(working_path,"final_render/model1eAdam0_003.gif")
writergif = animation.PillowWriter(fps=1)
anim3.save(savedir, writer=writergif)
```