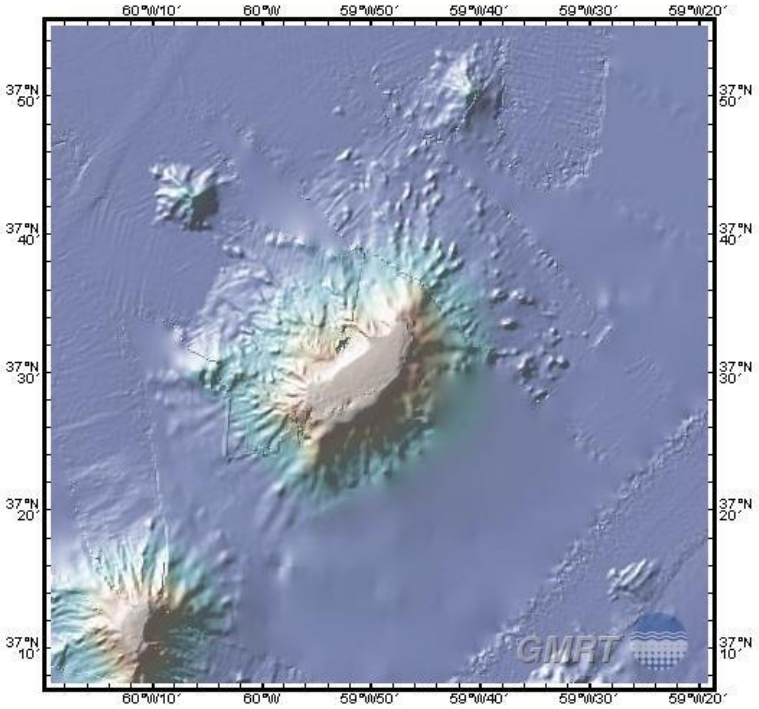
Seamount detection on Bathymetric images

Sebastian Krüger & Team

16.01.2025

Project Goal: Classification of bathymetric data images into “with seamount” or “without seamount”

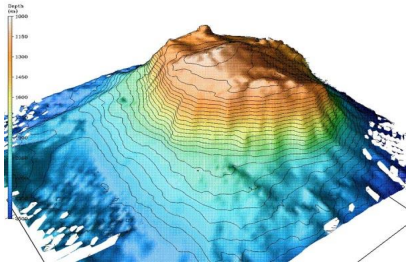
without seamounts

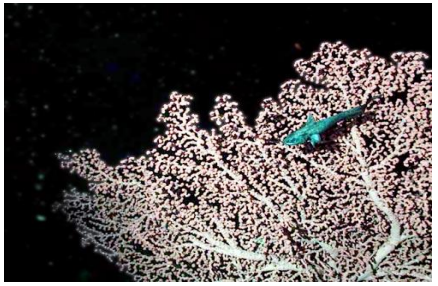


with seamounts

Introduction

- Less than 30 % of the seafloor has been mapped with high-resolution technology.
- Seamounts are of particular ecological and economic interest due to their unique ecosystems and mineral resources potential. They can also pose hazards to marine navigation and contribute to tsunami formation.
- Data from surveys comes piece by piece and is mostly processed manually and subjectively. Machine Learning could help to automate and speed up this process.





Literature Review

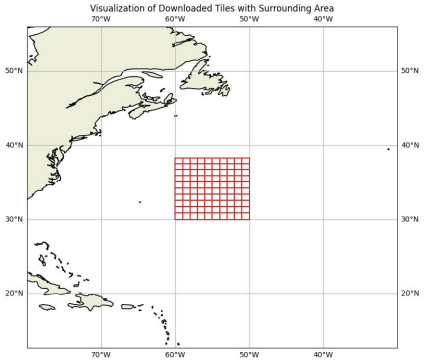
Approaches or solutions that have been tried before on similar projects.

- Source 1: “New global seamount census from altimetry-derived gravity data” (2011)
  - Objective: Global seamount census using vertical gravity gradient (VGG) data and ML approaches.
  - Outcomes: Identified 24,643 seamounts taller than 100m, with refined spatial identification.
  - Relation: Demonstrates feature detection methods and introduces a usable database.
- Source 2: “Discovery and analysis of topographic features using learning algorithms: A seamount case study” (2013)
  - Objective: Automated neural network for detecting seamounts from bathymetric data.
  - Outcomes: Achieved low false positives in complex terrains but struggled with small features.
  - Relation: Similar project from 11 years ago, so modern technology could provide better results/approaches.
- Source 3: “Seamount Detection Using SWOT-Derived Vertical Gravity Gradient” (2024)
  - Objective: Assessed SWOT satellite efficiency in detecting seamounts using simulated VGG data.
  - Outcomes: Improved accuracy over previous altimeters; challenges remained for elliptical features.
  - Relation: Could possibly hint to a future huge data inflow (big area with satellite imagery instead of ship’s surveys) . ML is only a side topic here.

Dataset Characteristics, Evolution of dataset

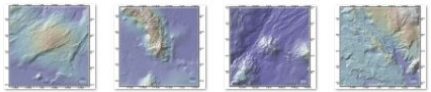
initial — different approach — manual selection — increased and refined

Equal tiles of a sea area

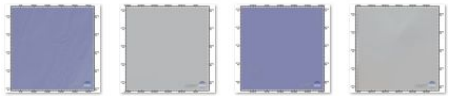


Balanced sets of pos/neg samples

positive:
automatic from seamount database
(centered on the charted seamounts)

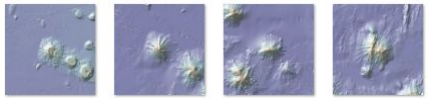


negative:
automatized programmatically
(centered on random points far away from charted seamounts)

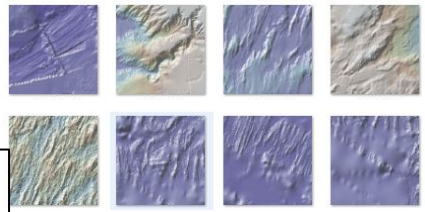


Manually selected images deemed good for training

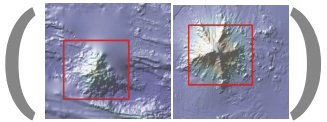
Selecting examples of mostly isolated mounts.
And negative samples with some structure



Samples in each class
100 → 300 → 500



Increased the amount of images to 500, deleted arbitrary



plus special set with bounding boxes (automated)

POSITIONS, DIMENSIONS:

Yesson C, Clark MR, Taylor M, Rogers AD (2011). The global distribution of seamounts [...] Data URL: http://data.unep-wcmc.org/datasets/41

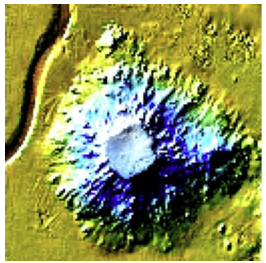
IMAGES:

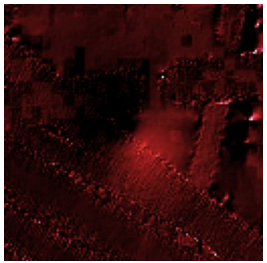
https://www.gmrt.org/
details see last page

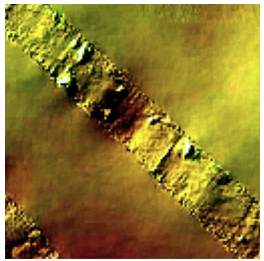
Dataset Characteristics

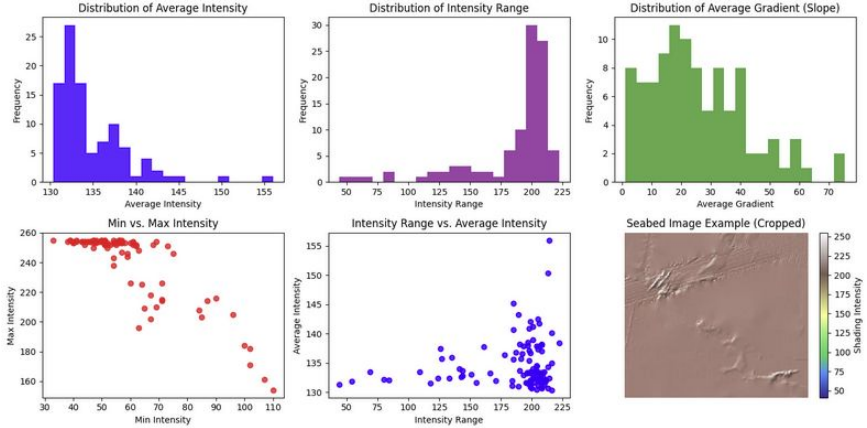
- Missing Values:
  - No explicit missing data, but low-res areas may lack sufficient detail for analysis.

- Image Preparation:
  - cropped and resized to 128x128 pixels (binary class.) 256x256 (obj. det.)
- Feature Distributions:
  - Visualization: intensity, gradient, scatter plots

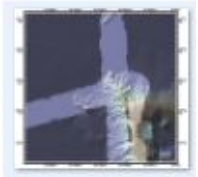








- Challenges:
  - Grey areas in non-seamount images.
  - High-res areas not rectangular.



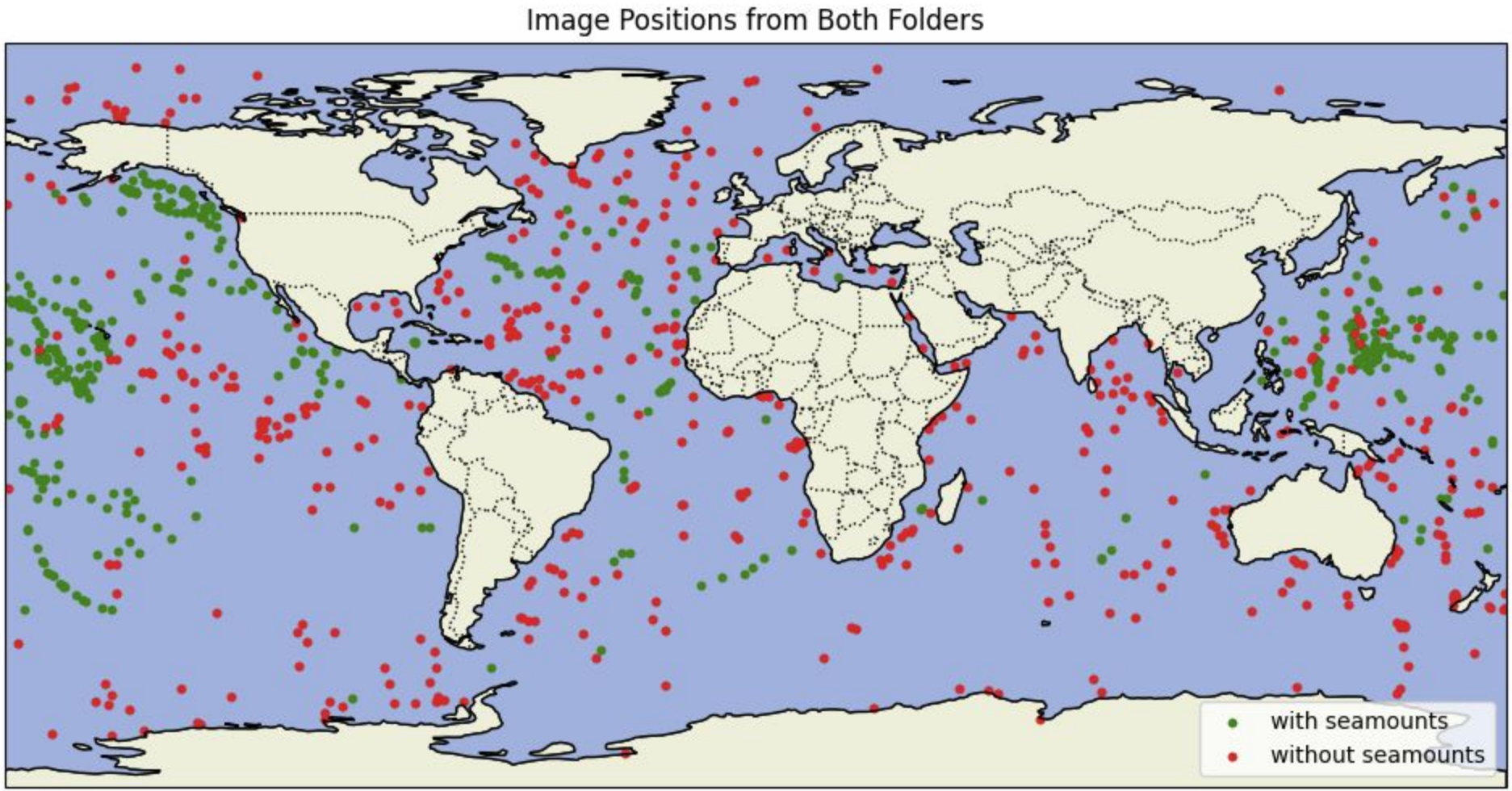


Image Positions from Both Folders
with seamounts
without seamounts

Baseline Model: Logistic Regression

Started with LR Model from sklearn framework

Source code with key Concepts to apply LR model to image data

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

image = cv2.imread(file_path)
image.flatten()  # Flatten to vector for regression applied to every image
...
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
...
logistic_model = LogisticRegression(random_state=42, max_iter=1000)
logistic_model.fit(X_train, y_train)
```

Logistic Regression (LR): Validation accuracy 83%, Test accuracy 85%.

Baseline Model: Random Forest

As we were not satisfied with the results of the LR model, we used an alternative baseline model

Source code with key Concepts to apply RF model to image data

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

image.flatten()  # Flatten to vector for regression applied to every image
...
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
...

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

Random Forest (RF): Achieved 85% accuracy with matching precision, recall, and F1-scores.

- **Results/Learnings:**
  - LR struggled with non-seamount detection, while RF performed worse on the positive class.
  - Good results questioned CNN necessity. Especially a prior RF model, that had 99% accuracy. **But:**
  - Dataset challenges: Unicoloured tiles in non-seamount images were probably responsible for wrong training.

Model Definition overview

two types of CNN

binary classification and image augmentation

Layer (Type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dropout (Dropout)	(None, 128)	0

binary classification and object detection

Layer (Type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 256, 256, 3)]	0
conv2d (Conv2D)	(None, 256, 256, 32)	896
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
conv2d_1 (Conv2D)	(None, 128, 128, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_3 (Conv2D)	(None, 32, 32, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 256)	0
conv2d_4 (Conv2D)	(None, 16, 16, 512)	1,180,160
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 512)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 1024)	33,555,456
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524,800
dropout_1 (Dropout)	(None, 512)	0
class_output (Dense)	(None, 1)	513
bbox_output (Dense)	(None, 4)	2,052

Model Definition

CNN + Image Augmentation

- 128x128 + 3 Colors Channel Input Layer
- 3 Layer of 2D Convolution + Pooling layer
- Flatten + 1 Dense layer with 128 Neurons followed by one (0.5) dropout layer
- One Neuron with Sigmoid activation as Output layer
- On top applied image augmentation

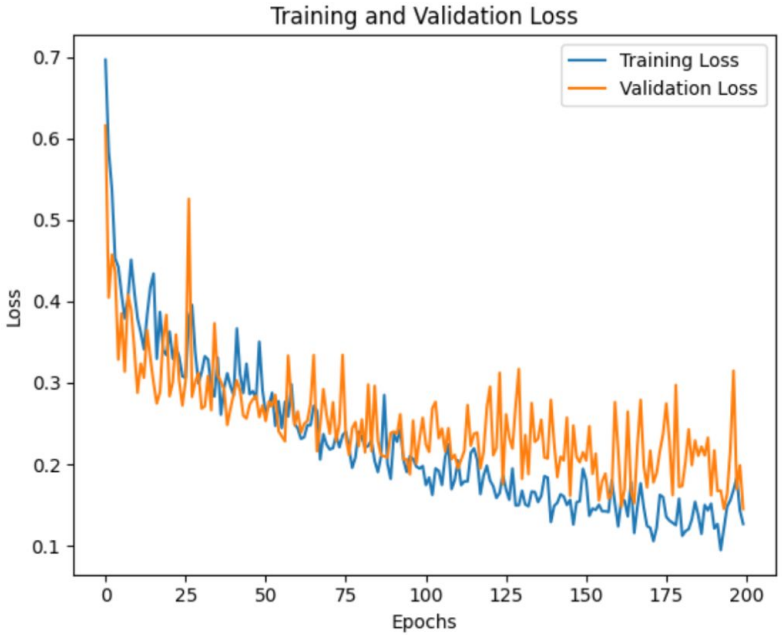
```
# Define ImageDataGenerator for augmentation
train_datagen = ImageDataGenerator(
    rotation_range=30,  # Random rotations
    width_shift_range=0.2,  # Random horizontal shifts
    height_shift_range=0.2,  # Random vertical shifts
    shear_range=0.2,  # Random shearing
    zoom_range=0.2,  # Random zoom
    horizontal_flip=True,  # Random horizontal flip
    fill_mode='nearest'  # Fill mode for new pixels
)
```

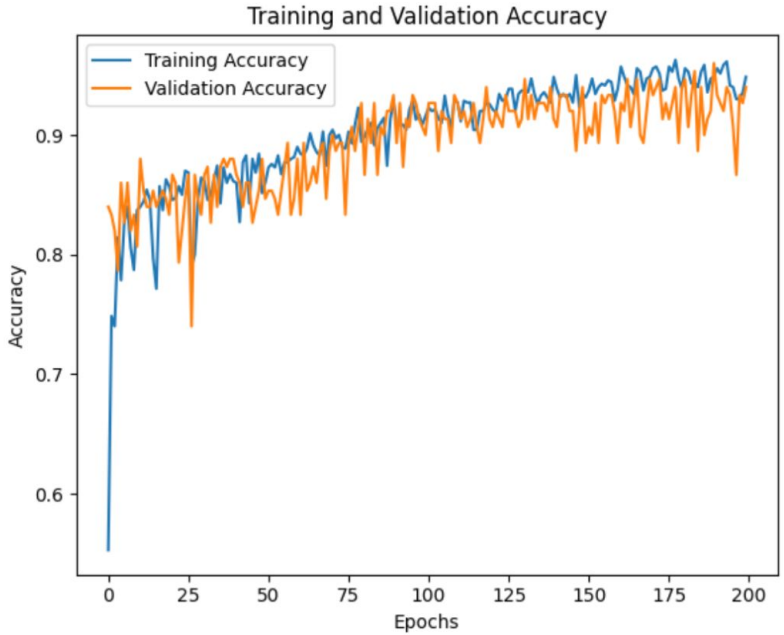
Hyperparameter:

- adam optimizer with LR 0.001
- binary crossentropy loss function
- 200 epoch
- batchsize of 16

Model Learning process

CNN + Image Augmentation





Evaluation Metrics 1/2

CNN + Image Augmentation

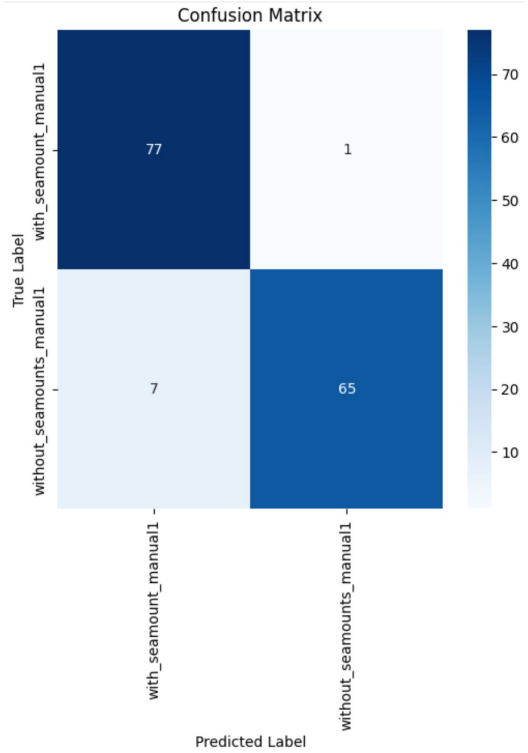
Performance on Test Set

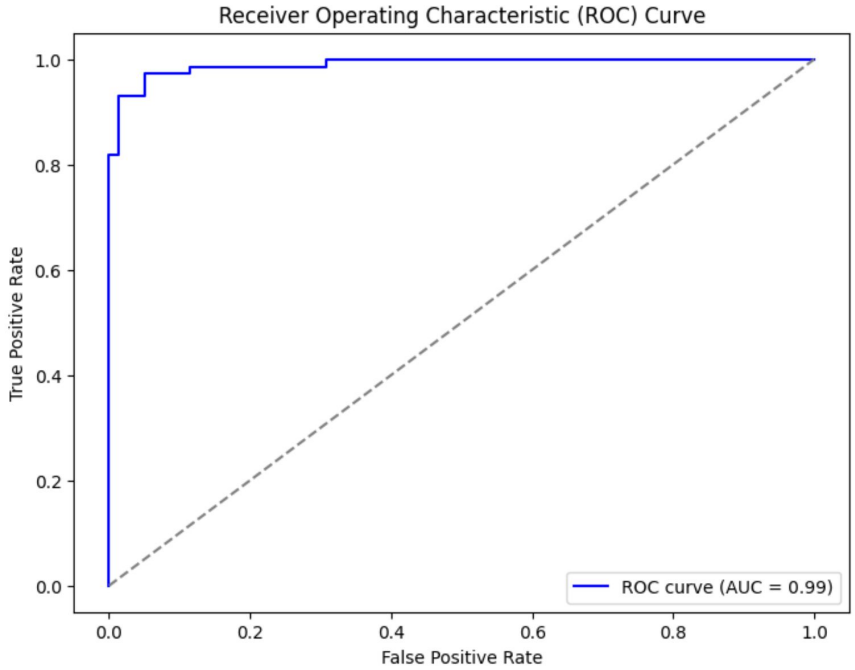
- Accuracy: 0.947
- Loss: 0.146

	precision	recall	f1-score	support
with_seamount_manual1	0.92	0.99	0.95	78
without_seamounts_manual1	0.98	0.90	0.94	72
accuracy			0.95	150
macro avg	0.95	0.94	0.95	150
weighted avg	0.95	0.95	0.95	150

Evaluation Metrics 2/2

CNN + Image Augmentation



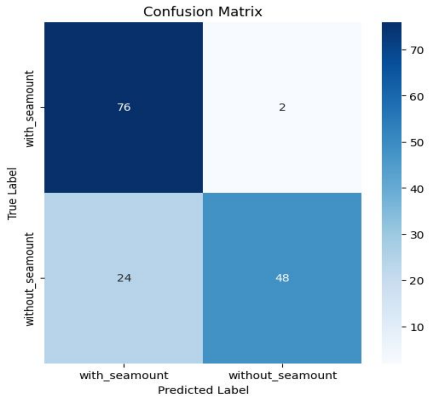


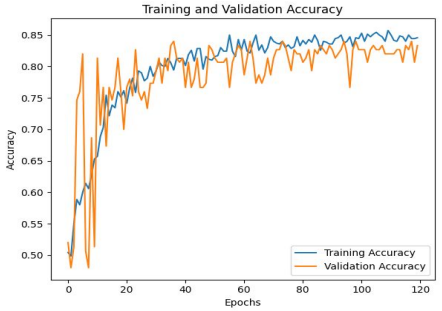
Tryout CNN + Transferlearning

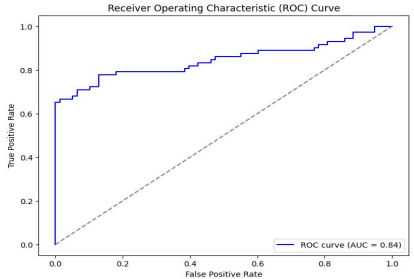
Use RES-NET50 and finetuning

- remove default classification toplayer and add new sigmoid toplayer
- train with 120 epochs
- batchsize 32 / learningrate 0.001 / Adam Optimizer / binary_crossentropy loss function
- no improvement to CNN with augmentation and very fast overfitting
- **Most interesting found, that model performs well on detecting seamounts with very few “false negative (2)” but bad on a lot of false positive (24)**

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 309ms/step - accuracy: 0.8485 - loss: 0.3725
Test Loss: 0.40781211853027344, Test Accuracy: 0.8266666531562805
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 505ms/step
                precision    recall  f1-score   support

   with_seamount       0.76      0.97      0.85        78
without_seamount       0.96      0.67      0.79        72

        accuracy                           0.83       150
       macro avg       0.86      0.82      0.82       150
    weighted avg       0.86      0.83      0.82       150
```







Object Detection Model

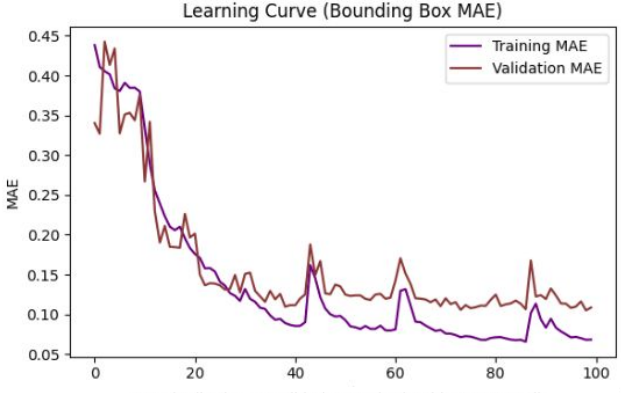
- Training data w. bounding boxes (a .csv defines boxes, no object is [-1,-1,-1,-1])
- Image augmentation: essential. Made via download script, not in model.
- Special attributes of the model construction:

- 2 different loss functions and 2 different output layers

```python
# Custom combined loss function
def custom_loss(y_true, y_pred):
    class_true, bbox_true = y_true
    class_pred, bbox_pred = y_pred

    # Binary cross-entropy for classification
    class_loss = tf.keras.losses.binary_crossentropy(class_true, class_pred)

    # Huber loss for bounding box regression (only for positive class)
    bbox_loss = huber_loss(bbox_true, bbox_pred)
    bbox_loss = tf.where(class_true == 1, bbox_loss, 0.0)

    return class_loss + bbox_loss
```

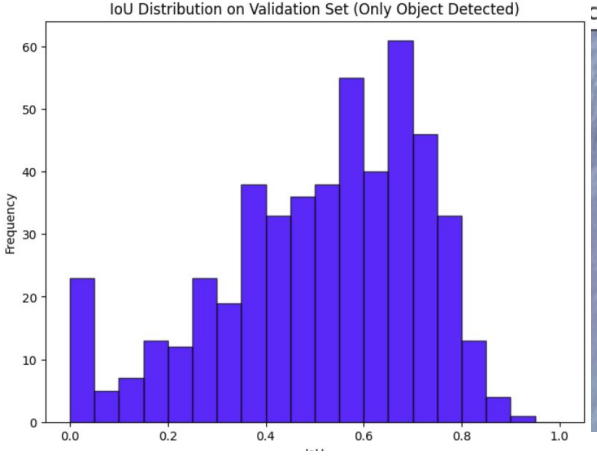
```python
# Classification branch
class_output = layers.Dense(1, activation='sigmoid', name='class_output')(x)

# Bounding box regression branch
bbox_output = layers.Dense(4, activation='linear', name='bbox_output')(x)

model = models.Model(inputs, [class_output, bbox_output])
return model
```

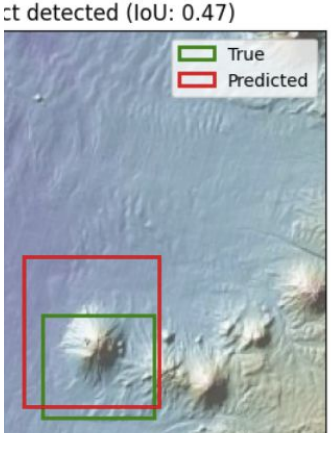
Object Detection Evaluation

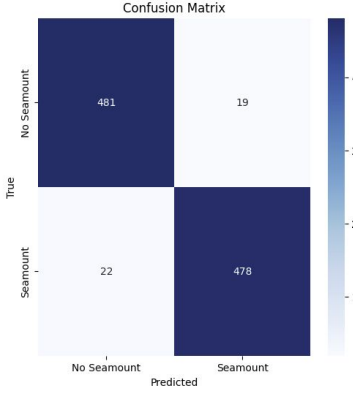


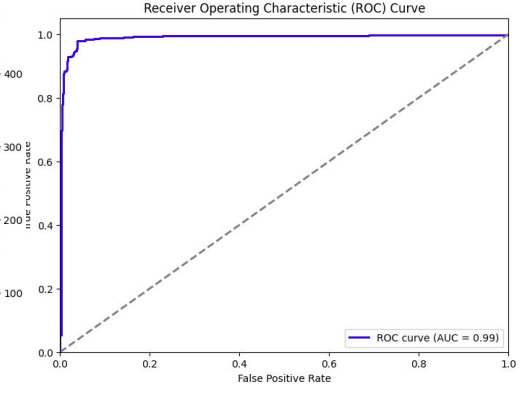
- Binary classification: same metrics as before
- Box placement: Mean Absolute Error (MAE) and Intersection over Union (IoU)

```
32/32 [==============================] - 15s 72ms/step
Classification Accuracy: 0.9590
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       500
           1       0.96      0.96      0.96       500

    accuracy                           0.96      1000
   macro avg       0.96      0.96      0.96      1000
weighted avg       0.96      0.96      0.96      1000
```







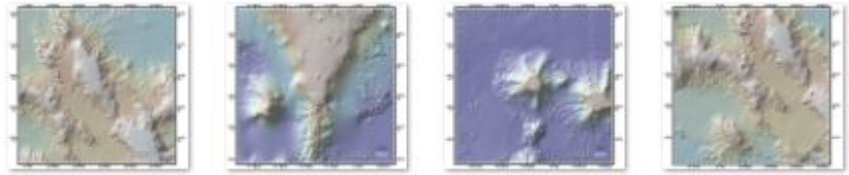


Summary of Results

- Task can be done efficiently with a CNN
- Object detection also possible but still a way to go
- Image augmentation essential in both branches
- Probably a niche where Transfer learning is not so effective

Challenges and Errors

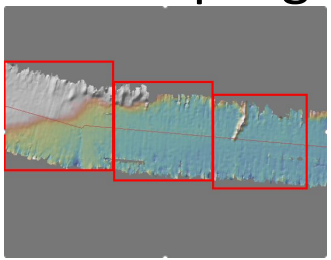
- Complex shapes of seamounts
- Creating a good dataset.

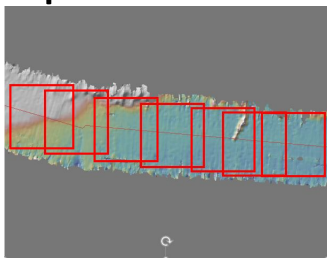


- Transforming geographical coordinates to pixel coordinates.
- Running the model (and/or tuning approaches) without ressource problems
- Tuning in general (frustrating process, takes very long for minimal optimization, while randomness in output had almost the same impact.
- Transfer Learning did not work (maybe because topographical images are significantly different to everyday images (“cats and dogs”)?
- How to know if the model is good (how to interpret the metrics)

Conclusion and Future Work

- Conclusion: Nice model for our “curated” dataset. If it can perform well in doing the real world task remains to be seen.
- Future work:
  - Use a larger dataset
  - going for complex seamount structures instead of focussing on isolated round ones
  - A solid object detection model with multiple box output (more than one seamount in image)
  - Integrating into software and develop a good process to feed images into network





Discussion

- Compared to the Random forest baseline model with 85% accuracy a CNN with ~95% accuracy is a huge improvement
- Since it is already difficult for humans to reliably determine seamounts, we expect the CNN to perform better on larger amounts of data due to decreasing concentration in humans
- It is unclear what is the best result that can be achieved on a large “real” data set

Images

Bathygraphic map tiles:

Global Multi-Resolution Topography Synthesis (GMRT)
https://www.gmrt.org/, Ryan, W. B. F., S.M. Carbotte, J. Coplan, S. O'Hara, A. Melkonian, R. Arko, R.A. Weissel, V. Ferrini, A. Goodwillie, F. Nitsche, J. Bonczkowski, and R. Zemsky (2009), Global Multi-Resolution Topography (GMRT) synthesis data set, *Geochem. Geophys. Geosyst.*, 10, Q03014, doi:10.1029/2008GC002332. Data doi: 10.1594/IEDA.100001

Introduction slide:

https://en.wikipedia.org/wiki/Seamount , all public domain

Title slide:

Image generated with DALL-E 3 (based on a prompt requesting the similarity to Caspar David Friedrich: "Der Wanderer über dem Nebelmeer")

Q&A

- Thank you! Questions?