Shoreline Segmentation and Classification: An AI-Based Approach Conducted on Aerial Images

Vegard Dale

Artificial Intelligence, 5-year master programme, University of Agder.

Contributing authors: vegard17@uia.no;

Abstract

This paper investigates the effects of using Artifical Intelligence(AI) for shoreline classification and segmentation. By using aerial images captured in the south of Norway, a dataset is generated by mapping an existing dataset of geometries to the images. A performance comparison is made using a set of models on different representations of the data. Despite the challenges related to the limited amount of data as well as the noise present in the data, the results are promising. The results shows that the models are highly capable of distinguishing between the different classes. Reaching a maximum mean interesction of union(mIoU) of 49.7% the UperNet[1] model with the Swin Transfomer[2] backbone achieved the best performance. The research conducted in this paper contributes valuable insights into the effectiveness of AI in shoreline classification, showcasing data representations, pre-processing and AI model comparisons.

1 Introduction

Sudden oil spills pose a significant threat to aquatic ecosystems, endangering fish, marine mammals, seabirds, and coastal areas at large [3]. The adverse effects of such incidents are far-reaching, necessitating prompt and effective response strategies. However, the successful containment and cleanup of oil spills are dependent on understanding the unique characteristics of the affected geological environments. Addressing oil spills requires a sophisticated approach that considers the diverse nature of coastal landscapes to effectively address and mitigate the impact of the oil spill. The method and extent of cleanup are essential considerations, depending on factors such as wave exposure, shoreline type, impacted natural resources, and area usage.[4].
The availability of such data is crucial to automate such procedures. However, acquiring and maintaining an accurate shoreline dataset poses significant challenges. Coastal environments are dynamic and subject to constant change due to natural processes, weather events, and human activities. Remote sensing technologies which has shown promising results, face limitations in terms of computation and cost[5].
Recent advancements in deep learning may enable Artificial Intelligence (AI) to potentially replace current methods, offering a more efficient and accurate approach to detect and classify shorelines. Currently, there is a lack of research regarding the use of deep learning for shoreline classification, and considering the huge impact this could have on environmental monitoring and resource management it is important to advance the research on this topic. This paper seeks to investigate the following issues:

1. How can we represent the problem of shoreline classification suitable for an AI implementation
2. Can we use AI to effectively and accurately classify shorelines

2 Related Work

Currently there is a lack of research in the field of shoreline segmentation and classification using deep learning. Similar research often involves land-sea segmentation for shoreline extraction to analyse erosion and sea level changes. [6] propose and compare a modified Unet variant to traditional Machine Learning(ML) methods. The study is conducted using high resolution aerial images in the "ANWR region on the eastern North Slope of Alaska"[6]. The study showed that traditional ML methods were able to produce similar performance to the modified Unet model. The authors argue that due to only using two classes(land and water), the simplistic nature of the task allows the ML based methods to achieve similar performance. [7] studies lake area and shoreline extraction using remote sensing. The study introduces a novel convolutional neural network architecture for segmentation and edge extraction. The proposed LaeNet architecture extracts features using no-downsampling convolutional layers, followed by a single channel no-downsampling layer to predict the land/water segmentation. The shoreline edges are then extracted using Canny edge detection.

In the field of shoreline classification, [8] propose a CNN-LSTM which combines the VGG16[9] architecture with a Long Short-Term Memory(LSTM) network. The LSTM network updates a mask matrix to enhance the features extracted from the images by VGG16. The principle of the mask matrix is to highlight the most relevant regions of the images. The experiments are performed on images captured in different areas of Thailand through different time periods. The images were classified into "aquaculture shoreline, biological shoreline, dam shoreline, estuary shoreline, othermanmade shoreline, sandy shoreline, silt shoreline and wharf shoreline"[8]. The results showed that the proposed network classified shorelines with a 94.57% accuracy, outperforming the VGG16 without the LSTM.

Since the debut of AlexNet[10] in 2012, CNNs have emerged as the predominant approach in computer vision[11]. The increased interest in CNNs sparked a surge in advancing new models and techniques within the field. In recent years more advanced techniques and deeper and more complex models have been introduced. [12] introduced Unet, an encoder-decoder architecture for Biomedical image segmentation. The idea behind the encoder-decoder architecture was to add successive layers to the contractive part of the network where downsampling operations were replaced by upsampling operations. By assembling the downsampled output with the upsampled output, a successive CNN layer can use the combined information to obtain a more precise output. The paper also argues for the use of image augmentations to simulate deformations typically seen in this type of data to increase the performance on a limited amount of data.
Since the release of Unet, a number of variations and improvements have been introduced. [13], [14], and [15] replaced the backbone of the original Unet architecture with the ResNet[16], DenseNet[17] and vgg[9] architecture respectively. In addition to replacing the backbone, other variations such as Unet++[18] introduced "a series of nested, dense skip pathways"[18]. The skip connections is used to combine the output from multiple sub networks. This ensures more cohesive and accurate relationship

between the features captured by the encoder and those reconstructed by the decoder.

[19] introduced the Vision Transformer. Transformers were widely adopted in the field of natural language processing(NLP) and the Vision Transformer introduced a new approach of applying transformers in computer vision. By interpreting an image as a sequence of patches the standard transformer from NLP was applied. The vision transformer were able to match or exceed state of the art results on multiple image classification datasets. [2] introduced the Swin Transformer. There was a challenge in using the original transformer due to the semantic gap between text and image data. To handle this issue a hierarchical transformer whose representation was computed using shifted windows was introduced. The regular transformer from NLP computes the relationship between a token to all other tokens using self attention. This is insufficient for computer vision due to the large amount of tokens and the quadratic complexity. Instead of computing global relationships, relationships are computed within the local windows which results in a linear complexity. This windows based self attention is more scalable for computer vision tasks.

The experiments conducted in this paper seeks to compare and evaluate the performance of the different models and techniques mentioned in this chapter for shoreline semantic segmentation. Due to the lack of research on this topic, this paper is meant to provide the foundation for future research. The goal is to localize the optimal data representation, pre-processing, and model implementations which can be analyzed for future improvements.

3 Method

This section highlights the steps taken from acquiring the data to implementing and running the models.

3.1 Data

The data consists of aerial images and shoreline geometries. The aerial images are collected from Norge i Bilder[20] and covers some of the coastal area in the south of norway. The aerial images are delivered in .tif format and contains the meta information shown in figure 1.


```
▼<NorgeIBilderExport>
   <mdOwner>Geovekst</mdOwner>
   <mdProducer>Norge i bilder</mdProducer>
   <mdUnit>.01</mdUnit>
   <mdCoordsys>23</mdCoordsys>
   <mdResolution>0.1</mdResolution>
   <mdFormat>tif</mdFormat>
   <eastBP>144800</eastBP>
   <westBP>144000</westBP>
   <northBP>6498600</northBP>
   <southBP>6498000</southBP>
   <Dato>2023-10-17</Dato>
   <mdFilename>33-1-444-103-36.tif</mdFilename>
 </NorgeIBilderExport>
```


Fig. 1: Aerial images meta information

The *eastBP*, *westBP*, *northBP* and *southBP* features contains the bounding box coordinates of the aerial image. The *mdCoordsys* feature represents the coordinate system used for the bounding box coordinates.
The shoreline dataset consists of labeled linestring geometries. The label represents the type of shoreline and consists of the classes provided in table 1.

Shoreline Types
Menneskeskapt struktur
Strandberg
Blokkstrand
Leirstrand
Steinstrand
Sandstrand

Table 1: Shoreline Types

- **Menneskespat struktur** represents structures like seawalls and wharfs observable along the shoreline[4].
- **Strandberg** Represents rounded rocks or solid rocks along the shoreline[4].
- **Blokkstrand** Represents rocky beaches where the diameter of the rocks are larger than 60 centimeters[4].
- **Leirstrand** Represents muddy beaches where the diameter of the grains or particles are less than 0.63 millimeters[4].
- **Steinstrand** Represents rocky beaches where the diameter of the rocks are between 6 - 60 centimeters[4].
- **Sandstrand** Represents sandy beaches where the diameter of the grains or particles are between 0.063 - 2 millimeters[4].

Although this data set is considered "labeled", the dataset was created in the 90s. This means the dataset is not updated to account for changes that may have happened in recent years. The geometries in the dataset are also very inaccurate and deviates from the actual shoreline in the images.

3.2 Data Preparation

The shoreline dataset was manually labeled to account for the large amount of errors that was present in the existing dataset. The data was labeled using Qgis[21] which is a graphical interface to visualize maps and geometries. Qgis sped up the labeling process as it allowed for selection and labeling of large sections of shorelines simultaneously. Using Qgis also made it possible to visulize the bounding boxes of the aerial images to highlight what geometries to should be labeled. This process was time consuming and approximately 120 aerial images were labeled.
A single aerial image covers an area of 60000 m^2 and the RGB images consists 8000x6000 pixels. to reduce the size of the images and the amount of labels for each image, the area covered by the image was divided into a 2x4 grid where the image was cropped into 8 new images.

The aerial images and shoreline dataset were not connected, meaning the geometries in the shoreline dataset needs to be transformed to represent the shoreline in the images. To map the geometries to the images, the geometries were grouped by using the intersection of the area covered by the aerial image. By then using the coordinates of the image, the geometry coordinates were mapped to pixel coordinates. Then depending on the model to be used, the bounding box labels and segmentation maps were generated.

The bounding box labels should represent the entire segment covered by each shoreline class. This means that there should not be two adjacent bounding boxes with the same class as it could lead to confusion for the model. Therefore, the geometries needs to be sorted in the correct order to combine them into a single geometry. In order to do this the smallest distance from the first point in the geometry to the last point in the other geometries was calculated. This calculation was done by generating a Balltree[22] from the sklearn[23] library which is an efficient nearest neighbor

search algorithm. For closed geometries(geometries where all start and end points are connected to another geometry inside the bounds of the image) a random starting point can be selected and the geometries can be sorted based on closest points. For geometries that move outside the bounds of the image, the geometries at the edge of the image needs to be selected as starting point. To locate these geometries, geometries where the endpoint distance is greater than 0 were selected as starting points as it means the endpoint of the geometry is not connected to another geometry. The bounding boxes were then generated by using the minimum and maximum x,y pixel coordinates of each shoreline segment.

When generating the segmentation maps, a decision had to be made on what parts of the shoreline should be used to represent the shoreline segmentations. Shoreline semantic segmentation is unique compared to other segmentation tasks. Normally it is trivial to tell what parts of an image belongs to a specific class, but for shorelines there are multiple possibilities for representing the segmentation maps. The shoreline segmentation maps were represented as a bold line covering parts of the water and land in alignment of the shorelines. This decision was based upon that the color of the water following the shoreline could be an important factor in deciding what class the shoreline represents. For example when comparing sandy beaches to rounded rocks, the color in the water a small distance from the shoreline would be brighter for the sandy beach than the rounded rocks. To generate the segmentations, the original shoreline geometries were mapped to pixel coordinates in the image and a buffer of 5 meters was generated from the geometries. This results in a bold line following the original shoreline with a width of 10 meters. The class id is then used to encode the segmentation maps.

3.3 Data pre-processing

The manual labeling was a time consuming process which resulted in not reaching a desirable amount of labeled images. This resulted in some of the classes being highly underrepresented. Specifically, the *Blokkstrand*, *Leirstrand*, *Sandstrand* and *Steinstrand* classes were rarely observed in the selected area. Therefore, these classes were combined to form a single class named *Strand*. The final dataset consists of 366 cropped aerial images, with the class distribution shown in table 2

Shoreline Type	count
Strandberg	25393
Strand	2767
Menneskeskapt struktur	3932
Background	1581

Table 2: Shoreline class distribution

The background class was annotated in areas where it was hard to tell the exact shoreline class due to shadows or trees. As we can observe from the table, there is still imbalance in the class distributions which has to be taken into consideration throughout the pre-processing steps.
Certain aerial images appeared entirely black or included significant black sections. Including these images in the dataset introduces noise as some of the segmentation maps are covering the black areas. To filter images that contain a large portion of black areas, the images were converted to grayscale and the ratio of black was calculated for each image. Images which contained a significant portion of black areas were removed based on a threshold.

Yolo Dataset

Three datasets were generated using the cropped aerial images and labels. The first model which was tested and meant to serve as a baseline was YOLO[24]. The dataset generated for the YOLO implementation is refered to as *d_yolo*. The YOLO dataset was generated following the Ultralytics[25] YOLO format.

Resized Image Dataset

The next set of models used for testing outputs segmentation maps. Therefore, a different dataset was generated. As the output of the models either can be represented as logits or a probability distribution for each pixel, two datasets were generated. In the first version of this dataset the images were resized from 4000x1500 to 800x300. This dataset is referred to as *d_resized*. When predicting a probability distribution, the labels are one-hot encoded along the third dimension. Due to the large size of the images, using the full size images with the one-hot encoded labels resulted in memory

issues. For this version of the dataset two models were compared namely Unet[12] and Unet++[18]. To handle the issue of class imbalances, the class weights shown in table 3 was used.

Shoreline Type	class weight
Strandberg	0.20
Strand	0.45
Menneskeskapt struktur	0.30
Background	0.05

Table 3: class weights

Different image transformations and filters were applied and tested to increase the model performance. To possibly allow the model to better detect the shoreline edges, the Scharr filter was applied. The Scharr filter detects edges independent of direction[26] which suggests it is a good fit for shoreline edge detection.
To enhance the textures of the individual shoreline classes, the Laplacian filter was applied. The Laplacian filter highlights edges where the color intensity rapidly changes[27] which may improve edge detection on smaller details and textures. Figure 2 highlights the difference between applying the Scharr and Laplacian filters.

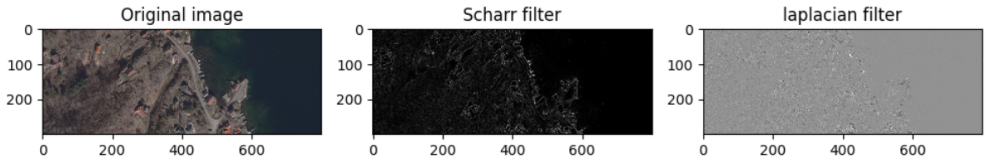


Fig. 2: Filter comparisons

Full Sized Image Dataset

For the second semantic segmentation dataset, another approach of handling the memory issues was used. For this dataset the full 4000x1500 images were used and the model outputs logits instead of a probability distribution. This dataset is referred to as *d_full*. To limit the size of the images as well as handle the class imbalance issues, the images were randomly cropped. For each iteration an image was randomly cropped into a new 500x1500 image. The cropping was also constrained to including a maximum of 50 percent of a single class which helps with the issue of class imbalance. The random cropping also helps dynamically increasing the size of the dataset and prevent overfitting as in each iteration the model trains on different views of the images. Random flips with a probability of 0.5 was also introduced to dynamically increase the size of the dataset. For better generalization, the images were randomly resized in the range 0.3 to 1.05. The pre-processing steps were performed for each iteration up to 25000 iterations.

To improve generalization, a number of photometric distortions were tested. With a probability of 0.5 the images were distorted using random brightness, contrast, saturation and hue[28]. This could improve the models generalization capabilities by randomly introducing noise that is common due to differences in environmental conditions.

3.4 Models and implementation

Due to the lack of research on shoreline classification and segmentation, numerous models which applies different techniques were tested and compared. Model selection relied on both experience and the performance demonstrated by the models in comparable image recognition tasks, such as those involving satellite images.

YOLO was the first model that was tested and meant to serve as a baseline. The YOLO implementation was sourced from the Ultralytics[25] python library. YOLOv8 is the most recent YOLO version and was used due to the elevated performance compared to earlier versions.The YOLO implementation was run on the *d_yolo* dataset.

For the *d_resized* dataset, Unet and Unet++ was compared. The Unet and Unet++ implementation was sourced from[29] which is a repository of different Unet implementations used for satellite images. Both Unet implementations were compared using the same amount of filters and similar parameters. The Unet implementation was modified to use the correct inputs in the 2D convolutional block, which was an error in the existing implementation. The upsampling convolutions were modified to get the correct dimensions. Specifically, the stride was changed from (3,3) to (3,2) and the padding was changed from same to valid in the first and second sub-networks in the expansive part of the network.
For the Unet++ implementation, some intermediate outputs had to be resized to get the correct dimensions. To increase the complexity of the model the number of filters were doubled from $32, 64, 128, 256, 512$ to $64, 128, 256, 512, 1024$. For the Unet++ implementation, deep supervision was used. By using deep supervision The loss for sub-network outputs are calculated for intermediate outputs and back propagated.
The pre-processing steps involved for the Unet and Unet++ implementation consists of the steps mentioned in section 3.3 which involved applying the Scharr and Laplacian filters which were added as a new channel to the third dimension. The class weights shown in table 3 was also used for both models.

For the *d_full* dataset a new set of models were used. This new set of models were sourced using the mmseg[30] repository. The MMSegmentation repository is an open source pytorch based toolbox used for semantic segmentation[30]. The repository consists of numerous base models and backbone implementations as well as preprocessing pipelines for easy integration and modification. The UperNet[1] with Swin Transformer[2] backbone, PsPNet[31] with ResNet[16] backbone and DeeplabV3[32] with ResNet[16] backbone were tested and compared on the *d_full* dataset. The Adam, AdamW and Stochastic Gradient Descent(SGD) was compared using different weight decay configurations. The pre-processing tests and comparisons involved for these models are those mentioned in 3.3. The mmseg model implementations are constrained

to 1D and 3D inputs which made it difficult to apply the Scharr and Laplacian filters. As some of the models did not have pre-trained weights available, the models were compared training from scratch. The code used in this project is publicly available at https://github.com/vegarddale/ikt464-project. **Note**, only the relevant parts of the MMsegmentation imlpementation is added.

4 Results

The performance of the different implementations are validated on the validation datasets. The results are validated using both the individual class Intersection of Union(IoU) and the mean IoU(mIoU) for all classes. The dataset was split into training and validation where 80% of the data was used for training and 20% for testing. The models were trained on the Tesla V100-SXM3-32GB GPU.

d_yolo results

The experiments from the YOLO implementation were conducted on the *d_yolo* dataset. The default YOLO parameters were used with pre-trained weights. Unfortunately, the YOLO implementation showed no results. The model did not make any predictions on the validation dataset. This could possibly be due to the existing noise present in the data or the limited amount of data. Because the YOLO implementation was meant to serve as a baseline and due to limited time, no further experiments were performed to improve the performance.

d_resized results

This section highlights the results obtained using the Unet and Unet++ implementations. Both models were trained using the Adam optimizer using a learning rate of *0.001* and weight decay of *0.00005*. Similar complexity was used on both models with a maximum 1024 number of filters. The class weights shown in 3 were used for both models. The models were trained for 100 epochs. Early stopping was implemented to prevent overfitting and the best weights were restored. Table 4 presents a comparison of the mean IoU for all classes over 5 individual runs between both models, considering raw images and applying the Scharr and Laplacian filters.

Name	Raw	Laplacian	Scharr
Unet(mIoU)	0.424	0.411	**0.430**
Unet++(mIoU)	0.409	0.415	0.410

Table 4: mIoU Unet/Unet++

The results showed that the original Unet architecture on average slightly outperformed the modified Unet++ architecture except when applying the Laplacian filter. This was surprising as the Unet++ architecture is supposedly more capable at reducing the semantic gap between the features captured in the downsampling and upsampling layers. Given the limited amount of training data and the presence of noise in the dataset, it is possible that the simplicity of the Unet architecture compared to Unet++, makes it less prone to the existing noise in the data.

Table 5 shows the mIoU for the individual classes averaged over 5 runs for the Unet implementation. This comparison is made to highlight whether the Laplacian and Scharr filters specifically improves the performance on individual classes.

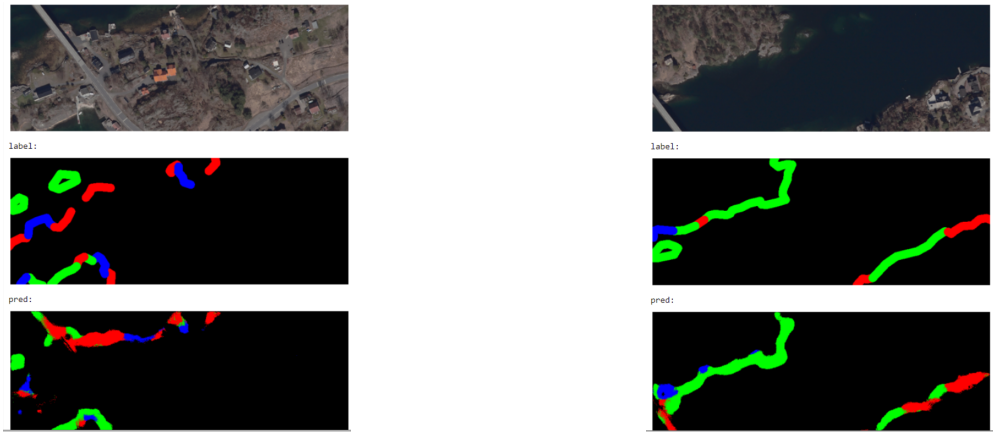
Name	Raw	Laplacian	Scharr
Background(Unet)	0.957	**0.962**	0.956
Strandberg(Unet)	0.412	0.410	**0.423**
strand(Unet)	0.103	0.088	**0.112**
Menneskapt struktur(Unet)	0.224	0.183	**0.231**

Table 5: mIoU all classes

The results showed that for the Unet implementation, adding the Scharr output to the input increased the performance on all classes except for the background. *Menneskeskapt struktur* had the largest performance increase when applying the Scharr filter. This is explained by the fact that the features belonging to this class exhibit a significant dissimilarity compared to all other classes, making them easily distinguishable and separable. The application of the Scharr filter highlights this dissimilarity by effectively capturing the edges.
A comparison was also made adding both Scharr and Laplacian to the input. These results have been omitted as they showed no performance gain.

When inspecting the predictions from the models, it is clear why it is difficult to achieve a high IoU on some of the classes using the current dataset. Figure 3 highlights some of the predictions made by the Unet implementation in comparison to the input image and the labels. The green segmentations are *Strandberg*, red is *Menneskeskapt struktur* and blue is *Strand*.

The first image is an example of how much noise is present in some of the images. Because the segmentation maps are generated from the linestrings, some of the linestrings have been removed due to deviating too much from the actual shoreline. In this example the model performs better than the actual labels, but does not achieve a high IoU due to the noise. The second image also highlights how the model



(a) Unet prediction 1 (b) Unet prediction 2

Fig. 3: prediction vs label

could outperform the labels. In this example the model is able to segment the bridge to the correct class, but the label does not include the segmentation.

d_full results

For the *d_full* dataset, the models were run for 25000 iterations using the AdamW optimizer with an initial learning rate(LR) of 0.001. For the first 12000 iterations a linear LR decay was used, from 12-24000 a polynomial decay was used and for the remaining 1000 iterations a constant LR was used. Table 6 shows the top 1 results obtained throughout the 25000 iterations.

Class	**IoU (%)**
Background(UperNet-Swin)	96.54
Strandberg(UperNet-Swin)	50.95
Strand(UperNet-Swin)	16.87
Menneskapt Struktur(UperNet-Swin)	34.63
Mean	**49.7**
Background(PSP-ResNet)	96.35
Strandberg(PSP-ResNet)	51.96
Strand(PSP-ResNet)	17.38
Menneskapt Struktur(PSP-ResNet)	31.95
Mean	49.4
Background(DeepLabV3-ResNet)	96.79
Strandberg(DeepLabV3-ResNet)	52.96
Strand(DeepLabV3-ResNet)	14.57
Menneskapt Struktur(DeepLabV3-ResNet)	29.07
Mean	48.3

Table 6: Semantic Segmentation Metrics

Compared to the Unet and Unet++ implementations, the results from the *d_full* dataset showed a significant performance increase. Reaching a maximum mIoU of 49.7 %, the UperNet-Swin model slightly outperformed PSP-ResNet and DeepLabV3-ResNet. Determining whether the improvement in performance stems from the models or the data is challenging, as the various implementations were not assessed across the different datasets. However, since the images have not been resized, no information is lost which is an important factor. Also, the pre-processing is more optimized to enhance generalization and accommodate the limited amount of data which could explain the performance increase.
Individually, the *Strand* class achieves the worst performance across all models. This is to be expected as it is highly underrepresented. In some cases, it could also be difficult to distinguish from the *Strandberg* class.

Figure 4 highlights some of the predictions made by the UperNet-Swin model. Visually, the model shows promising results in it's capability at distinguishing between the classes. Although, in some cases the model predicts objects that are not following

the shoreline which could be the result of the limited amount of data and the noise present in the data.

It is also worth mentioning that the background covers a large portion of the images. Naturally, the models will learn to predict the background class for a majority of the pixels which results in a high IoU. This leads to a misleading performance assessment in calculating the mIoU which is part of the reason the individual IoU is provided.

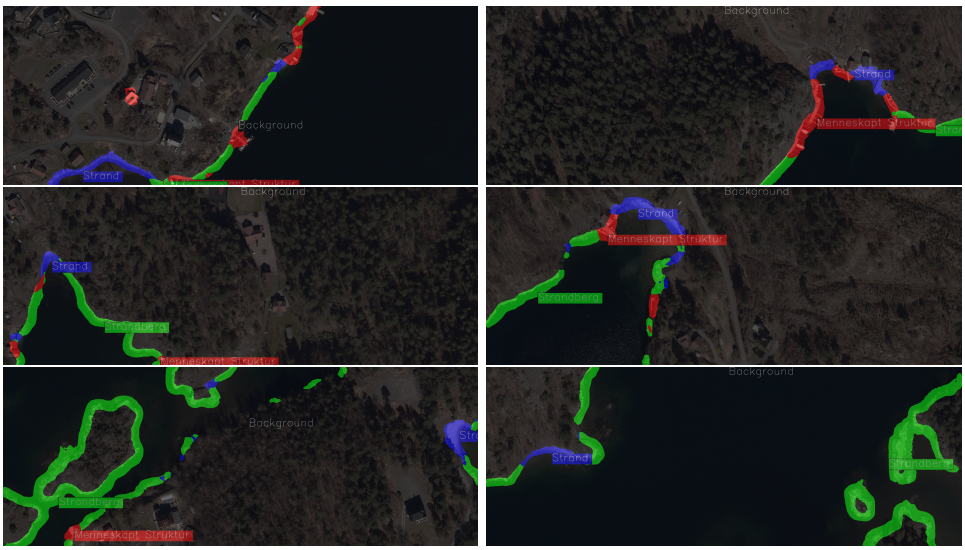


Fig. 4: UperNet-Swin predictions

5 Conclusion

This paper explores how to represent the problem of shoreline semantic segmentation and the data preparation steps involved. By generating segmentation maps covering a width of 10 meters following the shoreline, a variety of pre-processing steps and model implementations were tested and compared. Based on the results, the data representation sufficiently captures the problem at hand. As the study demonstrated promising outcomes, leveraging the full-sizes cropped images and employing the pre-processing steps outlined in 3.3 emerged as a an effective approach. Given the limited dataset and inherent noise in the data, the findings suggest that the techniques and models explored in this paper hold significant potential for further advancements. This research lays a solid foundation for future investigations and applications in the field, offering valuable insights into addressing challenges related to data representation, pre-processing and model performance in shoreline analysis.

6 Future Work

There are still some challenges in regards to shoreline semantic segmentation, specifically concerning the data. The data needs to be cleaned and the dataset should be expanded. Due to the insufficient amount of data some classes were underrepresented and combined to a form a single class. Therefore, it is still unknown how well the models would perform on the *Strand* subtypes. In addition more classes should be added as there are multiple subtypes not mentioned in this paper that depends on the slope of the shoreline. In this case depth data could also be added to the input.

References

[1] Xiao, T., Liu, Y., Zhou, B., Jiang, Y., Sun, J.: Unified perceptual parsing for scene understanding. CoRR **abs/1807.10221** (2018) 1807.10221

[2] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. CoRR **abs/2103.14030** (2021) 2103.14030

[3] Directorate, N.P.: ACUTE POLLUTION AND OIL SPILL PREPAREDNESS AND RESPONSE. https://www.norskpetroleum.no/en/environment-and-technology/oil-spill-preparedness-and-response/

[4] NOFO, K..: STRANDRENSING ETTER OLJEFORURENSNING. https://www.kystverket.no/globalassets/oljevern-og-miljoberedskap/opplaringsmateriell/strandrensning-etter-oljeforurensning.pdf/download

[5] Chi, M., Plaza, A., Benediktsson, J.A., Sun, Z., Shen, J., Zhu, Y.: Big data for remote sensing: Challenges and opportunities. Proceedings of the IEEE **104**(11), 2207–2219 (2016)

[6] Aryal, B., Escarzaga, S.M., Vargas Zesati, S.A., Velez-Reyes, M., Fuentes, O., Tweedie, C.: Semi-automated semantic segmentation of arctic shorelines using very high-resolution airborne imagery, spectral indices and weakly supervised machine learning approaches. Remote Sensing **13**(22), 4572 (2021) https://doi.org/10.3390/rs13224572

[7] Liu, W., Chen, X., Ran, J., Liu, L., Wang, Q., Xin, L., Li, G.: Laenet: A novel lightweight multitask cnn for automatically extracting lake area and shoreline from remote sensing images. Remote Sensing **13**(1), 56 (2020) https://doi.org/10.3390/rs13010056

[8] Ma, Z., Liu, Z., Huang, J., Wu, K.: Coastline classification and extraction based on deep learning. In: Proceedings of 2022 10th China Conference on Command and Control, pp. 858–871. Springer, Singapore (2022)

[9] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[10] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012)

[11] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin Transformer: Hierarchical Vision Transformer using Shifted Windows (2021)

[12] Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for

Biomedical Image Segmentation (2015)

[13] Diakogiannis, F.I., Waldner, F., Caccetta, P., Wu, C.: Resunet-a: a deep learning framework for semantic segmentation of remotely sensed data. CoRR **abs/1904.00592** (2019) 1904.00592

[14] Cao, Y., Liu, S., Peng, Y., Li, J.: Denseunet: densely connected unet for electron microscopy image segmentation. IET Image Processing **14**(12), 2682–2689 (2020) https://doi.org/10.1049/iet-ipr.2019.1527 https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-ipr.2019.1527

[15] Ghosh, S., Chaki, A., Santosh, K.: Improved u-net architecture with vgg-16 for brain tumor segmentation. Physical and Engineering Sciences in Medicine **44** (2021) https://doi.org/10.1007/s13246-021-01019-w

[16] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015) 1512.03385

[17] Huang, G., Liu, Z., Weinberger, K.Q.: Densely connected convolutional networks. CoRR **abs/1608.06993** (2016) 1608.06993

[18] Zhou, Z., Siddiquee, M.M.R., Tajbakhsh, N., Liang, J.: Unet++: A nested u-net architecture for medical image segmentation. CoRR **abs/1807.10165** (2018) 1807.10165

[19] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. CoRR **abs/2010.11929** (2020) 2010.11929

[20] Bilder, N.: Norge i Bilder. https://www.norgeibilder.no/

[21] Qgis: Qgis. https://qgis.org/en/site/

[22] developers: sklearn.neighbors.BallTree. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html

[23] scikit-learn: Scikit-learn Machine Learning in Python. https://scikit-learn.org/stable/

[24] Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. CoRR **abs/1506.02640** (2015) 1506.02640

[25] Inc, U.: Turn Images Into AI to Get Useful Insights with No Code. https://www.ultralytics.com/

[26] System, B.I.: Edge Detection Filters. https://infosys.beckhoff.com/english.php?content=../content/1033/tf7xxx_tc3_vision/7866092939.html&id=%

[27] R. Fisher, A.W. S. Perkins, Wolfart., E.: Laplacian/Laplacian of Gaussian. https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

[28] OpenMMLab: DATA TRANSFORMS. https://mmsegmentation.readthedocs.io/en/latest/advanced_guides/transforms.html

[29] ashishpatel26: Satellight Image Segmentation with UNet and Its Variants. https://github.com/ashishpatel26/satellite-Image-Semantic-Segmentation-Unet-Tensorflow-keras

[30] Contributors, M.: MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark. https://github.com/open-mmlab/mmsegmentation (2020)

[31] Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. CoRR **abs/1612.01105** (2016) 1612.01105

[32] Chen, L., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation. CoRR **abs/1706.05587** (2017) 1706.05587