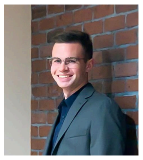
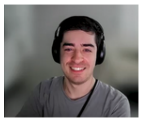
Predicting River Erosion and Path Changes with Satellite Images

Worcester Polytechnic Institute

CS 534 Artificial Intelligence

Final Report

Team Members:

Colin Streck <cjstreck@wpi.edu> (B.S. in Math, B.S./M.S. in Computer Science)	
Justin Healey <jjhealey@wpi.edu> (B.S. in Computer Science, M.S. in Cybersecurity)	
Adam Veilleux <adveilleux@wpi.edu> (B.S. in Computer Science & Engineering, M.S. in Data Science)	
Sam Veilleux <seveilleux@wpi.edu> (B.S. in Mechanical Engineering, M.S. in Computer Science)	

Github Repository

1 August 2024

Predicting River Erosion and Path Changes Using Satellite Images
Authors: C. Streck, J. Healey, S. Veilleux, A. Veilleux
Date: 1 August 2024

Abstract:

Predicting river erosion and path changes is crucial for managing ecological balance, agriculture, and human settlements. This report leverages satellite imagery and state-of-the-art AI methodologies to forecast these changes, aiming to mitigate the devastating effects of riverbank erosion, such as agricultural land loss, infrastructure destruction, water pollution, and community displacement. Current methods, including Convolutional Neural Networks (CNNs), Long Short-Term Memory networks (LSTMs), Vision Transformers (VITs), and Superpixel Segmentation, each offer unique advantages and challenges. This project combines VITs and LSTMs and CNNs to predict river erosion using Google Earth Timelapse Engine videos from 1984 to 2022. Preprocessing involves extracting regions of interest and chronological sorting of images. Superpixel segmentation simplifies the classification problem, while LSTMs handle the temporal aspects of river morphology. The proposed model's performance will be evaluated using accuracy, precision, recall, F1 score, and Matthews correlation coefficient (MCC). This approach aims to provide valuable insights for environmental management and disaster prevention, highlighting the potential of AI in predicting and mitigating the impacts of natural disasters. The proposed model has offered several key results: the ability to predict slight changes in aerial river images and semantic segmentation, classifying rivers from all other features within an image. However, the diversity of our image sets proved to be unable to generalize to all regions of the world, due to drastically different climates and erosion rates of rivers.

KeyWords:

River erosion, river path changes, satellite imagery, AI methodologies, Convolutional Neural Networks (CNNs), Long Short-Term Memory networks (LSTMs), Vision Transformers (VITs), Google Earth Timelapse Engine, environmental management, disaster prevention, Matthews Correlation Coefficient (MCC).

Table of Contents

Table of Figures

Section 1: Introduction

1.1 Motivation & Background

Rivers are dynamic natural systems that are crucial for ecological balance, agriculture, and human settlements. However, riverbank erosion (physical changes to riverbeds and banks via the removal of sediment or engineered materials over time), changes in river paths, and high amplitude flooding pose significant long term challenges. The consequences can be devastating, leading to a loss of agricultural land, destruction of infrastructure, water pollution, displacement of communities, and loss of life. According to a recent project by the Asian Development Bank, a 200 million dollar loan was provided to the Indian government "to strengthen flood and riverbank erosion risk management along the Brahmaputra River in Assam, India" [1]. Additionally, in areas such as the UK, costs of erosion and its associated impacts such as water pollution cost “£336 million a year and are a considerable source of water pollutants, costing £238 million a year to remediate” [2]. One such example of rivers changing and leading to damages are near Rapidan Dam in Minnesota where a house collapsed into a river due erosion of the river bank, forcing the family to evacuate and search for a new home [12]. This is just one small scale example; the larger, arguably more devastating effects of river path changes can be seen in the millions of people in southeast asia being affected by extreme flooding during monsoon season, [13] causing several hundred deaths of people and wildlife, and millions of destroyed homes.

The ability to predict river path changes before they occur has the potential to improve planning in areas where these disasters occur and inform timely interventions. This project aims to leverage state-of-the-art AI methodologies and satellite imagery to predict river erosion and path changes with the goal of providing valuable insights for environmental management and disaster prevention/mitigation.

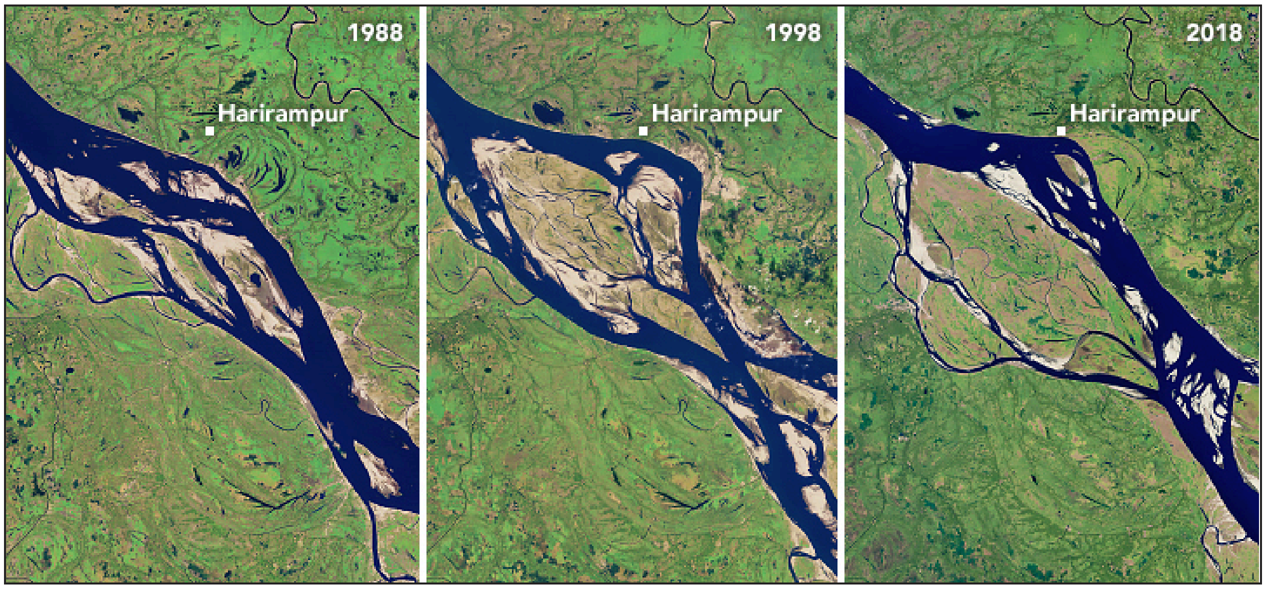


Figure 1: Harirampur Region River Changes over time

1.2 State-of-the-Art Methods

Several state-of-the-art (SOTA) methods have been developed to address a range of problems regarding image classification and segmentation. These methods can be applied to river erosion and path prediction using AI and satellite imagery, including convolutional neural networks (CNNs), long short-term memory recurrent neural networks (LSTMs), vision transformers (VITs), and super pixel transformers. Each has their own advantages and disadvantages, which are discussed in the following sections.

1.2.1 Convolutional Neural Network (CNN)

Description: Convolutional Neural Networks use a kernel, or filter, to reduce the input size across layers. The resulting feature map accounts for the correlation among neighboring pixels in an image, which is then pooled and passed through a neural network (NN) to classify the image. This reduces the input size to the neural network, which is part of what makes CNNs efficient compared to traditional NNs. In 2023, a new variation of a CNN, dubbed ForCNN, was developed to expand traditional CNNs to perform time series predictions on images [11].

Advantages: CNNs are fast and computationally efficient with modern hardware. Moreover, they fuse the feature extraction and classification tasks into a single process, which optimizes features directly from the raw input, and they are immune to small transformations in input data [3].

Disadvantages: Prone to overfitting; CNNs rely on the assumption that the relationships among closer pixels are more relevant than further pixels, which may not be true in all scenarios.

1.2.2 Long Short - Term Memory Recurrent Neural Networks

Description: Long Short-Term Memory (LSTM) networks can be used to predict future changes in sequential data, which is useful for forecasting river morphology alterations due to erosion and sediment deposition. Utilizing a sequence of satellite images that capture the river's historical behavior, LSTMs can predict its future shape changes. As demonstrated in the study by Descovi et al.[9], LSTMs have successfully used daily precipitation data to predict streamflow in the Brazilian Pantanal Basin. This methodology can be extended to predict river shape changes by training the LSTM network on satellite imagery. The LSTM's cell structure with input, forget, and output gates allows it to retain information about past river shapes and discard unnecessary information. By learning from the progression of images, the LSTM can forecast the river's future morphology.

Advantages: LSTM-RNN is perfect for handling sequential data such as time series of satellite images. "LSTM networks provide nodes with dynamic states, acting as memory, to enhance overall performance" (Xu et al., 2020). The LSTM has the ability to learn long-term relationships from data. "The LSTM network achieves good performance in terms of three evaluation criteria … demonstrating its powerful capacity in learning non-linear and complex processes." [10].

Disadvantages: LSTM can be difficult to train as it requires careful selection of parameters. "The batch size and the number of LSTM cells is a sensitive parameter and should be carefully tuned" [10]. Moreover, LSTMs can be computationally expensive with large data sets.

1.2.3 Vision Transformation for Semantic Image Segmentation

Description: Semantic segmentation is the process of assigning a class label to each pixel of an image and locally identifying the different classes in the image within spatial location. Vision transformers (VIT), where a transformer is a method that uses an attention mechanism to "draw global dependencies between input and output" [7], are an alternative to CNNs that provide the architecture required to perform semantic image segmentation. VITs assemble tokens by "extracting non-overlapping patches followed by a linear projection of their flattened representation or by applying a feature extractor," then "augment[ing] [the image] with a positional embedding and [adding] a patch-independent readout token" [5]. These tokens are pushed through several transformer stages and then "fusion modules fuse and upsample the representations to generate a fine-grained prediction" [5]. In other words, the VIT is used as the pre-processing to reform the image into an assembly of pieces and transformed back into image representations at various sizes and fused into features to predict using a convolutional decoder. This method provides a great performance increase of upwards of 28% when compared to top convolutional methods [5] and can be used to identify rivers changing over time by feeding a specific model, in our case, satellite images of specific areas containing rivers. This approach would require the model to first identify the river and then separately calculate the changes over time using a different sequential temporal model. VIT is a great choice for this task as there are millions of satellite images which in benchmarks can produce accuracies of up to 88.55% on ImageNet, 90.72% on ImageNet-ReaL, 94.55% on CIFAR-100, and 77.63% on the VTAB suite of 19 tasks [4].

Advantages:

- Great parallelization [6]
- High resolution of final output segmentation image [6]
- Computationally efficient [4]

Disadvantages:

- Requires very large datasets [6]
- Struggles with rich intra-class variation, context variation, occlusion ambiguities, and low resolution [6]
- Requires high end equipment, especially with large datasets [4]

1.2.4 Superpixel Transformers for Semantic Segmentation

Description: Superpixel Transformers can be used as part of segmentation approach in which pixels in the image are broken up into groups called superpixels that can be assigned semantic meaning [8]. Superpixel groupings are partitioned based on their similarity to each other, that is, if two pixels are in the same superpixel, it is assumed that they have some relationship to each other. Techniques for generating superpixels include watershed-based, density-based, graph-based, contour-evolution, path-based, clustering-based, energy-optimization, and wave-pathlet methods [8]. Superpixel approaches would be well-suited to this task because there is a lot of data to be stitched together from the satellite-images and superpixel grouping would provide a way of simplifying images to relevant features.

Advantages:

- Shrinks classification problem from pixel-level to superpixel-level [8]
- Enables more complex algorithms to be run [8]
- Improves computational complexity [8]

Disadvantages:

- Requires additional preprocessing [8]
- Superpixel sizes vary in effectiveness depending on the level of noise in the image [8]

1.3 Proposed Solution Process

Our proposed solution uses a combination of SOTA deep learning methods, specifically superpixel and LSTM, to predict riverbank erosion/change using Google Earth Timelapse Engine videos from 1984 to 2022. This solution is outlined in the following sections.

1.3.1 Preprocessing and Data Collection

The data for this process was collected using Google Earth's timelapse feature, which allows users to view timelapses of regions around the globe from 1984 to 2022. After collecting the time lapses for a number of rivers, the first preprocessing step is to break the time lapse videos into frames. Next, we extract regions of interest (ROIs) to identify the timestamp and the river region, and lastly, we sort the frames for each river into chronological order. The year timestamps are recognized and validated using optical character recognition (OCR). This preprocessing workflow results in a dataset containing a number of subsets of chronologically ordered timelapse images for rivers, where each subset reflects the temporal changes of an individual river over time.

1.3.2 Model Training Superpixel

We first use the SOTA semantic segmentation method Superpixel to segment our river images into byte masks containing the predicted rivers. Pixels in the satellite image are grouped into superpixels, which are then assigned semantic meaning; in this case, the binary values are "river" and "not river."

1.3.3 Model Training and Predictions with LSTM

The LSTM for using time series river masks curated in the preprocessing step is implemented in python. After organizing preprocessed river mask images into sequences, we use the open-source TensorFlow and Keras python modules to train the model. The model's architecture includes ConvLSTM2D layers for spatiotemporal data, BatchNormalization layers for stability, and a Conv3D layer for predicting future river masks and is compiled with an optimizer and binary cross-entropy loss function.

1.4 Evaluation of SOTA methods

To evaluate the SOTA methods, we will measure performance using accuracy, precision, recall, F1 score, MCC, and temporal consistency. The experiments will use the preprocessed dataset, training models on a training set and validating on a testing set, with cross-validation for robustness. Quantitative metrics will be complemented by qualitative visual inspections for a systematic comparison of SOTA methods to select the best-performing model for predicting riverbank erosion.

Section 2. SOTA Literature Review

2.1 Convolutional Neural Network (CNN):

CNNs are computationally efficient algorithms for image classification tasks. The core of a CNN is the convolution phase, which occurs during feature extraction when the kernel passes over sections of the image. As the kernel moves across the image, each section is convoluted and pooled, allowing the neural network (NN) to process smaller sections of the image at a time while retaining the important relationships and feature patterns in the pixels. This is what allows CNNs to be very efficient compared to other NN-based algorithms.

As a result of the convolution and pooling steps, CNNs take advantage of the fact that pixels that are closer together are likely more relevant to each other than pixels that are far apart. This is true in many feature extraction problems, albeit not all, and this principle can be an advantage or disadvantage depending on the specific problem. For river detection, it is true that closer pixels are more relevant to each other than further pixels and this primary principle in CNNs is a key advantage.

2.2 Convolutional Long Short - Term Memory Recurrent Neural Networks:

Convolutional Long Short-Term Memory (ConvLSTM) networks are a variant of the LSTM network "that capture spatial features in multi-dimensional data by convolution process" [14]. Unlike a simple LSTM, which is limited to one dimensional sequences, ConvLSTM incorporates convolutional structures within its gates, making it suitable for tasks involving multi-dimensional data, such as videos and imagery. ConvLSTM utilizes gates, an input, forget, and output, which take an input, choose to forget the last step, or output the layer to the next step. Figure 2 shows the workflow of a ConvLSTM where the ConvLSTM layers process the input data over time. Each layer passes its output to the next, capturing both spatial and temporal dependencies.

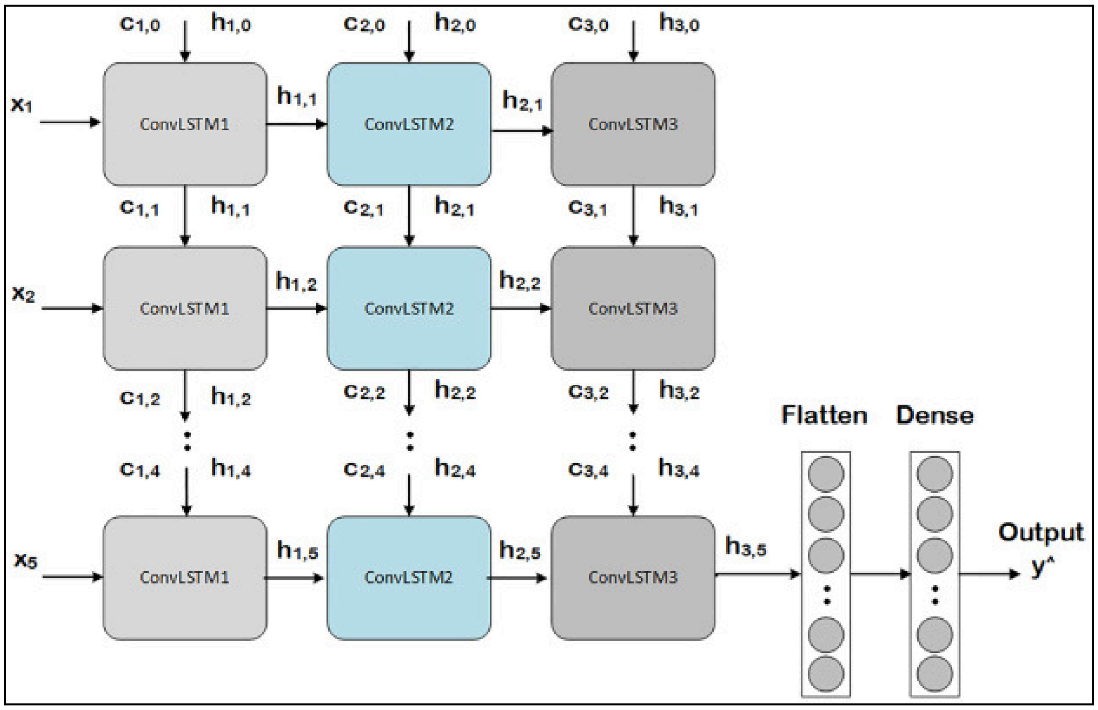

$c_{1,0}$ $h_{1,0}$ $c_{2,0}$ $h_{2,0}$ $c_{3,0}$ $h_{3,0}$
x_1 ConvLSTM1 $h_{1,1}$ ConvLSTM2 $h_{2,1}$ ConvLSTM3
$c_{1,1}$ $h_{1,1}$ $c_{2,1}$ $h_{2,1}$ $c_{3,1}$ $h_{3,1}$
x_2 ConvLSTM1 $h_{1,2}$ ConvLSTM2 $h_{2,2}$ ConvLSTM3
$c_{1,2}$ $h_{1,2}$ $c_{2,2}$ $h_{2,2}$ $c_{3,2}$ $h_{3,2}$
Flatten
Dense
$c_{1,4}$ $h_{1,4}$ $c_{2,4}$ $h_{2,4}$ $c_{3,4}$ $h_{3,4}$
x_5 ConvLSTM1 $h_{1,5}$ ConvLSTM2 $h_{2,5}$ ConvLSTM3 $h_{3,5}$
Output $y^{\wedge}$

Figure 2: Convolutional Long Short - Term Memory Recurrent Neural Networks

In 2023, The National Library of Medicine conducted a study to address the challenges of adjusting the hyperparameters when tuning deep learning models, as different parameter settings can alter performance of the model. The researchers used LSTM and ConvLSTM models and evaluated their performance of different sets of hyper-parameters, including batch size, number of layers in the model, and the number of filters in each layer of the model, and came to the following conclusion:

> "As the batch size increases within a layer, the performance of the ConvLSTM model improves in terms of generalization and speed, although the maximum training accuracy is achieved when the batch size is 32. As the number of layers increases from one to two, there is a slight increase in training accuracy of the model" [14]

The researchers also concluded that increasing from two to three layers does not significantly affect accuracy, and in fact, "increasing the number of filters negatively impacts the generalization of the classifier" [14]. The disadvantages of using the ConvLSTM are computational time, generalization, and batch size sensitivity; moreover, increasing the number of filters negatively impacts the generalization of the classifier, and the maximum training accuracy occurs with a batch size of 32, however, 32 is not great for validation accuracy.

2.3 Vision Transformation for Semantic Image Segmentation:

Vision transformation is a SOTA method for semantic image segmentation. Vision transformers take an input image and tokenize it (shown in orange in Figure 3) by extracting non-overlapping patches. Next, the vision transformer performs a linear projection of the tokens flattened representation (e.g. DPT-Base and DPT-Large both Dense Prediction Transformers) or by applying a ResNet-50 CNN feature extractor (i.e. DPT-Hybrid). The image embedding is then modified with a positional embedding and a patch-independent readout token is added (shown in red in Figure 3). These tokens are passed through multiple transformer stages, typically between 12 and 24 layers. Next, the tokens are reassembled from different stages into an image-like representation at multiple resolutions (shown in green in Figure 3). Fusion modules progressively fuse and upsample the representations to generate a fine-grained prediction (shown in purple in Figure 3).

VITs are advantageous to use over other methods, such as CNNs, because they are able to learn global features of images, have high resolution of final output segmentation image, can be parallelized very well, and are comparatively computationally efficient. In the context of river detection and path changes prediction, VITs can be used to identify rivers, a global feature, in aerial images; additionally, since the models available are trained on millions of images, VIT may be faster to train and process predictions compared to other SOTA methods.

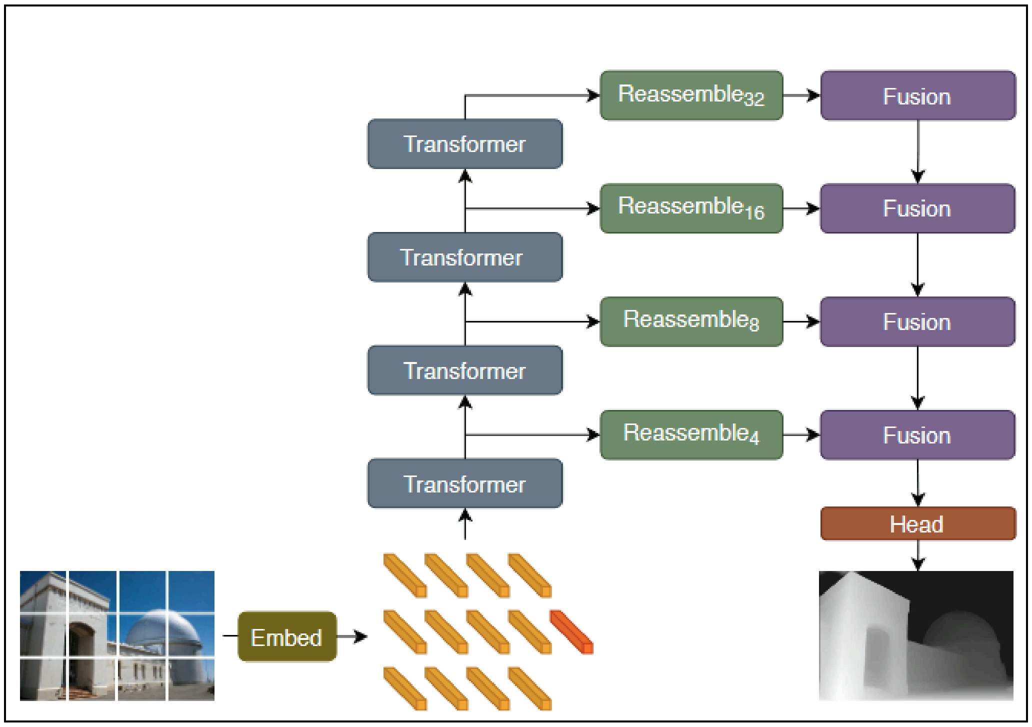


Figure 3: Vision Transformation for Semantic Image Segmentation

2.4 Superpixel Transformers for Semantic Segmentation:

Superpixel transformers are a SOTA alternative to the use of iterative clustering as used in the classical Simple Linear Iterative Clustering mode (SLIC) which, due to the structure of its workflow, can be difficult to implement as a part of a deep learning workflow [8]. In superpixel pipelines, a CNN is used as the encoder and the data is then fed through multiple layers of multilayer perceptrons and bilinear resizing, resulting in data at various scales that can be amalgamated into a structure called a hypercolumn, similar to the use of scaled pyramids in the Scale Invariant Feature Transform for feature detection. Pixel feature data can then be sent into a tokenization, after which the classes of individual superpixels are predicted by multi-head self-attention modules which determine context between superpixels, finally feeding into a softmax function that converts the findings back into pixel-space, rather than superpixel space.

Superpixel transformers are advantageous over the SLIC alternative for their adaptability to a deep learning workflow, enabling much better use of hardware acceleration than iterative clustering algorithms due to the ordering in which the method processes information. The method trades off speed in the SLIC method for greater accuracy with more dense images, making it ideal for working with satellite imagery.

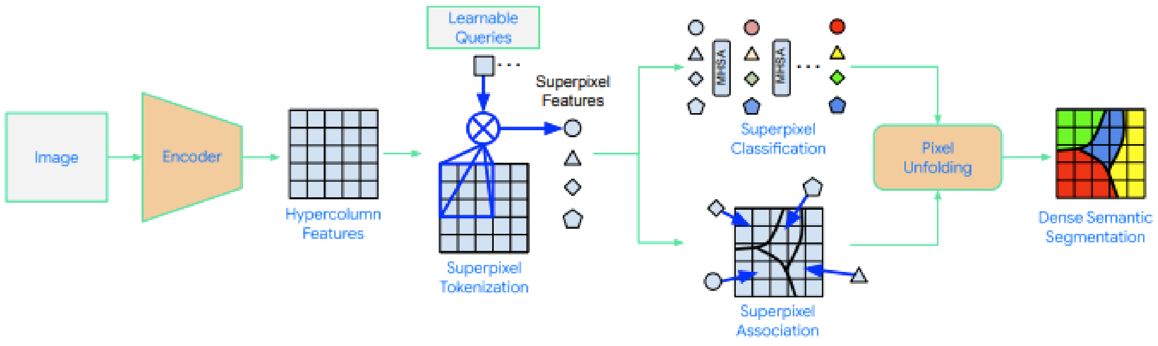


Figure 4: Superpixel Transformer for Semantic Segmentation

Section 3. Proposed High-level Solution and Process

3.1 Workflow

Data Acquisition and Preprocessing: Satellite imagery is obtained from Google Earth Timelapse Engine videos from 1984 to 2022. Next, the videos are broken into one image for each year and resized to reduce noise.

Region of Interest (ROI) Extraction: ROI extraction utilizes OCR with the Pytesseract python library to extract years from the image. From each image, semantic segmentation is used to identify and isolate river areas.

Model Training Superpixel: Superpixel transformers are used to semantically segment an image by modeling similarities between pixels and matching them, creating a segmented image.

Model Training LSTM: ConvLSTMs are used to model spatial and temporal dependencies in river morphology over time and to predict the next image in the sequence, thus predicting changes in the river's path over time.

Evaluation and Validation: Model performance is assessed using metrics such as accuracy, precision, F1 score, and Mathews Correlation Coefficient (MCC) to ensure robustness in predictions.

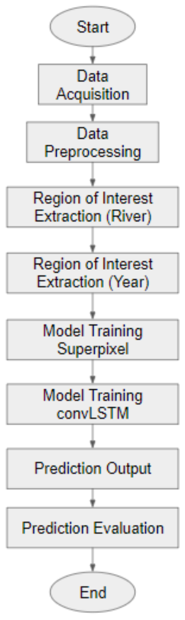
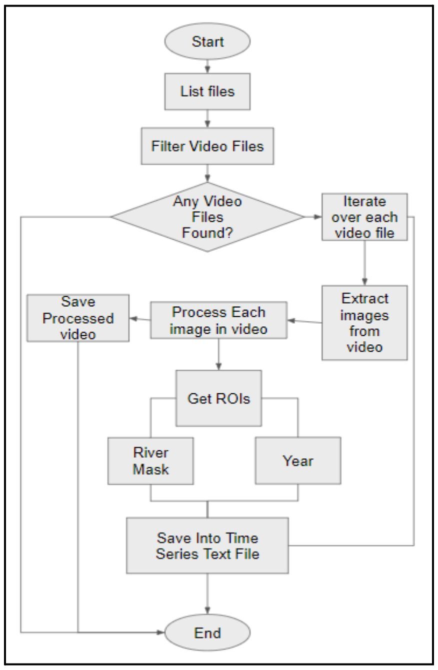


Figure 5: Workflow Diagram

The following diagrams show the process for data preprocessing and model training. The data preprocessing pipeline begins with listing and filtering the video files, breaking the videos into frames, and extracting regions of interest (ROIs) from each frame and masking the rivers. The model training pipeline starts by loading the preprocessed masks, splitting into train and test datasets, and finally training and evaluating the model.


Start
List files
Filter Video Files
Any Video Files Found?
Iterate over each video file
Extract images from video
Process Each image in video
Save Processed video
Get ROIs
River Mask
Year
Save Into Time Series Text File
End

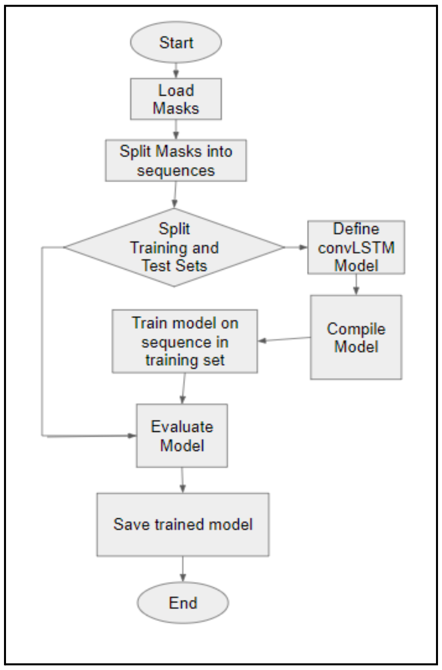

Start
Load Masks
Split Masks into sequences
Split Training and Test Sets
Define convLSTM Model
Compile Model
Train model on sequence in training set
Evaluate Model
Save trained model
End

Figure 6: Data Preprocessing Pipeline

Figure 7: Model Training Pipeline

3.2 Development and Implementation

3.2.1 Data Acquisition and Preprocessing

The data acquisition and preprocessing stage involves extracting relevant images from satellite videos and preparing them for analysis. The primary data source is satellite imagery videos spanning from 1984 to 2022 obtained from Google Earth Timelapse Engine. These videos are refined into one representative image per year and subsequently processed to reduce noise and standardize their size.

In the sample code below, the process begins by reading all the files in a specified data folder and filtering out the video files. Once the video files are identified, the next step is to extract meaningful regions of interest (ROIs) from the images. This is achieved using the get_ROIs function, which allows users to interactively select two ROIs from the first frame of each video. One ROI contains the year of the image, while the other contains the river area of interest. The extract_year_from_image function is responsible for extracting the year from the time ROI. This function enlarges the image to enhance the text recognition accuracy, converts it to grayscale, and applies a series of thresholding and blurring operations to isolate the text. Pytesseract, an OCR tool, is then used to extract the year from the processed image. The extracted text is cleaned and validated to ensure it falls within the expected range of years.

Next, the extract_time_series function processes each video frame by frame. It reads the frame, extracts the year using the previously defined function, and if the year is valid and sequential, it saves the corresponding river ROI image. This results in a series of images corresponding to different years, which are saved as individual frames in a new video file. Finally, a text file is generated to record the extracted years, providing a time series of the data.

The entire preprocessing pipeline is executed for each video file in the data folder, ensuring that the data is consistently prepared for further analysis.

Sample Python Implementation:

```python
import os
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import RectangleSelector
import pytesseract

folder = "Data"
target_folder = "CleanData"
all_files = [f for f in os.listdir(folder) if '.' in f]
is_video = lambda f: f.endswith('mp4') or f.endswith('webm') or f.endswith('mkv')
video_files = [f for f in all_files if is_video(f)]

def get_ROIs(image) -> [np.s_, np.s_]:
    rois = []
    def onselect(eclick, erelease):
        x1, y1 = int(eclick.xdata), int(eclick.ydata)
        x2, y2 = int(erelease.xdata), int(erelease.ydata)
        rois.append(np.s_[y1:y2, x1:x2])
        plt.gca().add_patch(plt.Rectangle((x1, y1), x2-x1, y2-y1, edgecolor='red',
facecolor='none'))
        plt.draw()
    fig, ax = plt.subplots()
    ax.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
    rs = RectangleSelector(ax, onselect, interactive=True)
    print("Select two ROIs and then press any key to continue...")
    plt.show(block=True)
    plt.close(fig)
    return rois

def extract_year_from_image(time_image) -> int:
    width = int(time_image.shape[1] * 4)
    height = int(time_image.shape[0] * 4)
    dim = (width, height)
    resized_image = cv.resize(time_image, dim, interpolation=cv.INTER_LANCZOS4)
    gray_image = cv.cvtColor(resized_image, cv.COLOR_BGR2GRAY)
    _, threshold = cv.threshold(gray_image, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
    blurred_image = cv.GaussianBlur(threshold, (5, 5), 0)
    _, threshold = cv.threshold(blurred_image, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
    top_right_pixel_value = threshold[0, -1]
    if top_right_pixel_value < 127:
        threshold = cv.bitwise_not(threshold)
    config = '--oem 3 --psm 6 outputbase digits'
    text = pytesseract.image_to_string(threshold, config=config)
    text = "".join([i for i in text if i in '0123456789'])
    if text == "" or text == None:
        return 0
    text = int(text)
    if text < 1970 or text > 2024:
        return 0
    return text

def spaced_by_one(x) -> bool:
    current = x[0]
    for item in x[1:]:
        current += 1
        if current != item:
            return False
    return True

def extract_time_series(video_path, output_path) -> bool:
    cap = cv.VideoCapture(video_path)
    if not cap.isOpened():
        print(f'Error: Could not open video at {video_path}')
        return False

    ret, frame = cap.read()
    if not ret:
        print(f'Error: Failed to read capture')
        return False

    rois = get_ROIs(frame)
    if len(rois) != 2:
        print("Error: Please select at least two ROIs.")
        return False

    time_roi, river_roi = rois

    last_year = 0
    time_series_data = []
    output_images = []
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        new_year = extract_year_from_image(frame[time_roi])
        if new_year <= last_year:
            continue

        print(f'Read frame from {new_year}')
        time_series_data.append(new_year)
        last_year = new_year
        output_images.append(frame[river_roi])
    cap.release()

    if time_series_data is not []:
    #if spaced_by_one(time_series_data):
        with open(output_path.split('.')[0] + '_timeseries.txt', 'w') as f:
            for year in time_series_data:
                f.write(f"{year}\n")

        out = cv.VideoWriter(output_path, cv.VideoWriter_fourcc(*'XVID'), 2.0,
(output_images[0].shape[1], output_images[0].shape[0]))
        for image in output_images:
            out.write(image)

        out.release()
        observed = len(time_series_data)
        expected = time_series_data[-1]-time_series_data[0]+1
        print(f'Approximate frames dropped: {expected}-{observed}={expected-observed}')
        return True
    else:
        return False

for f in video_files:
    video_path = os.path.join(folder, f)
    output_path = os.path.join(target_folder, os.path.splitext(f)[0] + '.avi')
    res = extract_time_series(video_path, output_path)
    while not res:
        res = extract_time_series(video_path, output_path)
```

3.2.2 Semantic Segmentation

The most successful attempt at semantic segmentation with a deep learning model was the use of a encoder-decoder architecture in which the encoder was a SwinV2 transformer and the decoder was a multi-layer convolutional neural network using ReLu as an activation function periodically throughout it. The logic behind it was that the transformer's multi-head attention model would perform feature extraction and focus on the important regions in the image rather than applying uniform focus to everything in the image. Unfortunately, the model kept getting stuck at an accuracy of roughly 70% as calculated by a binary cross entropy function. While the model isn't exactly an explainable model, the going theory as to the issue is that the issue was with the decoder, whose attempts created a monochromatic mask which was more effective than usual because the task is binary segmentation rather than segmentation of many classes. Use of multiple optimizers (Adam and SGD being the most successful) and various pre-trained and random weights being used as different starting positions did not resolve this, nor did experimentation with various learning rates, which is the usual solution to get caught in an extrema in a gradient-based optimizer. Experimentation also considered whether to include the encoder in the gradient calculation, but including or excluding it did not affect the model's ability to escape the trivial solution. Gradient calculation was also why initial attempts at extraction of hypercolumn features from an image for superpixel segmentation were unsuccessful, as the assignment operation does not have a gradient value, which makes sense from a calculus perspective.

The most successful segmentation results were achieved by a classical method based on superpixel calculation. Superpixels are groupings of pixels based on similarity in position, color, and intensity. This reduces the decision-space considerably, allowing for more complex methods to be layered on top. The SLIC approach was applied to extract the superpixels, which were visually determined to be sufficiently accurate, especially regarding edge detection of the rivers. The superpixels still needed to be semantically identified, so a heuristic approach was created. Essentially, due to the textural homogeneity of the surface of water in satellite imagery, k-means segmentation with a k value of two for binary results is an effective classifier, but overly sensitive when identifying. Once the image was loosely segmented with the more present color being labeled black and the other being labeled white (as typically the river area in an image is less than the land area) the contours were calculated, and the largest contour, which would be the main body of the river, was kept. After, due to some strange textures in the outputs, a morphological close operation was applied to the image. The results of this process looked river-like but did not accurately match the edges of the true river, so an intersection algorithm determined if the heuristic mask had an intersection of greater than 50% of a given superpixel by summing the masked image arrays and comparing them in order to include the SLIC superpixels that belonged to the river. The Python code used to do so appears below, and is also linked in the Colab notebook below. The notebook also contains the code for the most effective deep learning method, but it is not recommended to use this when segmenting the dataset.

Link to segmentation code implementation.

Python implementation:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from skimage.segmentation import slic

def mask_segmented_image(binary_image):
    gray = cv.cvtColor(binary_image, cv.COLOR_BGR2GRAY)
    unique_colors, counts = np.unique(gray, return_counts=True)
    color1, color2 = unique_colors
    count1, count2 = counts
    more_common_color = color1 if count1 > count2 else color2
    result_image = np.where(gray == more_common_color, 0, 1).astype(np.uint8) * 255
    result_image = cv.cvtColor(result_image, cv.COLOR_GRAY2BGR)
    color_ratio = max(count1/count2, count1/count2)
    return result_image, color_ratio

def kmeans_segmentation(image):
    # initial k-means segmentation
    twoDimage = image.reshape((-1,3))
    twoDimage = np.float32(twoDimage)
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    ret,label,center=cv.kmeans(twoDimage,2,None,criteria,10,cv.KMEANS_PP_CENTERS)
    center = np.uint8(center)
    res = center[label.flatten()]
    segmented_image = res.reshape((image.shape))
    masked, color_ratio = mask_segmented_image(segmented_image)

    # needs to be grayscale for contour and morphology
    masked = cv.cvtColor(masked, cv.COLOR_BGR2GRAY)

    # Avoid excessive filtering when necessary
    if color_ratio > 2.0:
        return masked


    # Remove small contours
    contours = cv.findContours(masked, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
    contours = contours[0] if len(contours) == 2 else contours[1] # unpack if tuple
    largest_area = max(map(cv.contourArea, contours), default=0)
    for c in contours:
        area = cv.contourArea(c)
        if area < largest_area:
            cv.drawContours(masked, [c], -1, (0,0,0), -1)

    # Morph close
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (5,5))
    close = cv.morphologyEx(masked, cv.MORPH_CLOSE, kernel, iterations=2)
    return close

def binary_slic_segmentation(image):
    segments = slic(image, n_segments=300, compactness=10, sigma=1)
    return segments

def classical_segmentation(image):
    kmeans_mask = kmeans_segmentation(image)
    superpixels = binary_slic_segmentation(image)
    binary_mask = np.zeros_like(kmeans_mask)
    for superpixel in np.unique(superpixels):
        mask = np.zeros_like(kmeans_mask)
        mask[superpixels == superpixel] = 255
        overlap = np.logical_and(kmeans_mask, mask)*255
        if np.sum(overlap) / np.sum(mask) > 0.5:
            binary_mask[superpixels == superpixel] = 255
    return binary_mask
```

3.2.3 Model Training

To predict river erosion over time, the model employs convolutional LSTM (ConvLSTM) networks. This approach leverages the ConvLSTM's ability to capture both spatial and temporal dependencies within the data, making it well-suited for the task of analyzing time series of images. The training process begins with loading river mask images, which represent areas of interest in the satellite imagery where changes in the river are observed. These images are organized into sequences, serving as input to the ConvLSTM model, and the dataset is then split into training and testing sets to ensure that the model is evaluated on unseen data.

The model is constructed using the Sequential API from TensorFlow's Keras library and incorporates several ConvLSTM2D layers. These layers perform both convolutional and LSTM operations, allowing the model to learn spatial features through convolution while capturing temporal dependencies via the LSTM mechanism. Specifically, each ConvLSTM2D layer is followed by a BatchNormalization to help normalize the activations and improve the training stability and speed by ensuring that the inputs to each layer have a consistent distribution.

Each ConvLSTM2D layer processes the input sequences, learning to recognize patterns until the final ConvLSTM2D layer outputs a sequence that encapsulates the learned spatio-temporal features. The model is compiled using the Adaptive Moment Estimation (ADAM) optimizer and a binary cross-entropy loss function, which is suitable for binary classification tasks.

During the training phase, the model iteratively adjusts its weights to minimize the loss function. This process involves evaluating the model's performance on both the training and validation data. Key metrics are monitored throughout the training process, such as loss and accuracy, to provide insights into the model's learning progress and effectiveness.

Python implementation:

```python
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import ConvLSTM2D, BatchNormalization, Conv3D
from sklearn.model_selection import train_test_split

from utils import load_river_masks, create_sequences


# -- Config vars -- #
# Directory containing the masks and the extension for the mask images
MASK_DIR:str = 'path/to/masks/'
MASK_EXT:str = 'png'

# Number of images in each sequence
SEQUENCE_LEN:int = 10

# Proportion of dataset to use for test data
TEST_SET_SIZE:float = 0.2

# Random state to pass to train_test_split; define as a value to generate a reproducable
output, or None for randomness
RANDOM_STATE:int | None = None

# -- Loading the data -- #
# Load all the masks
all_masks:np.ndarray = load_river_masks(MASK_DIR, MASK_EXT)

# Split the masks into X and Y data
X_data, Y_data = create_sequences(all_masks, SEQUENCE_LEN)

# Split into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=TEST_SET_SIZE,
random_state=RANDOM_STATE)


# -- Creating the model -- #
# Define the model
model = Sequential()

# Add layers
model.add(ConvLSTM2D(filters=64, kernel_size=(1, 1),
                     input_shape=(None, X_train.shape[2], X_train.shape[3],
X_train.shape[4]),
                     padding='same', return_sequences=True))
model.add(BatchNormalization())

model.add(ConvLSTM2D(filters=64, kernel_size=(1, 1),
                     padding='same', return_sequences=True))
model.add(BatchNormalization())

model.add(ConvLSTM2D(filters=64, kernel_size=(1, 1),
                     padding='same', return_sequences=False))
model.add(BatchNormalization())

model.add(Conv3D(filters=1, kernel_size=(3, 3, 3),
                 activation='sigmoid',
                 padding='same', data_format='channels_last'))

model.compile(optimizer='adam', loss='binary_crossentropy')
model.summary()
```

3.2.4 Evaluation and prediction

Once the model is trained, it can be evaluated using the test data. The model's performance metrics, i.e. loss and accuracy, are analyzed to ensure the model’s effectiveness in predicting river morphology changes, this is done per frame predicted. The trained model can then be used to predict changes in river erosion based on new input sequences.

Python Implementation:

```
# Evaluate the model
evaluation = model.evaluate(X_test, Y_test)
print(f'Test Loss: {evaluation}')

# Predict future changes
predicted_images = model.predict(X_test)
```

```
# Visualize the results
def plot_predictions(actual, predicted):
    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    axes[0].imshow(actual, cmap='gray')
    axes[0].set_title('Actual Image')
    axes[1].imshow(predicted, cmap='gray')
    axes[1].set_title('Predicted Image')
    plt.show()

for i in range(5):
    plot_predictions(Y_test[i], predicted_images[i])


# Flatten the labels
        y_true = ground_truth.values.flatten()
        y_pred = predicted.values.flatten()

        # Calculate Precision and Recall
        precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
        recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)

        # Calculate F1 Score
        f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)

        # Calculate MCC
        mcc = matthews_corrcoef(y_true, y_pred)

        # Append the results
        precisions.append(precision)
        recalls.append(recall)
        f1_scores.append(f1)
        mcc_scores.append(mcc)
```

Section 4. Final Experimental Results and Discussion

4.1 Performance Evaluation

To assess the performance of the SOTA methods in predicting river erosion, several methodologies will be used. Following the preprocessing of the data as outlined in section 3 and performing semantic segmentation on that data, which were visually inspected, the model training method of ConvLSTM will be evaluated on its accuracy, precision and Recall, F1 Score, and Matthews correlation coefficient (MCC). Accuracy will gauge the overall correctness of the predictions, while Precision and Recall, provides a balanced assessment. F1-Score is the harmonic mean of precision and recall. MCC specifically evaluates binary classification models like distinguishing between river erosion and no river erosion scenarios.

4.2 Experimental Results and Discussion

Figure 8 is an excerpt from the segmented data that underwent thresholding, contour detection, and had a grayscale binary mask applied. The binary mask isolates the river area in white against a black background to analyze changes in river morphology over time, where white dictates "river" and black dictates "not river." The segmented images are organized into sequences that each cover multiple years and serve as the input to the ConvLSTM model.

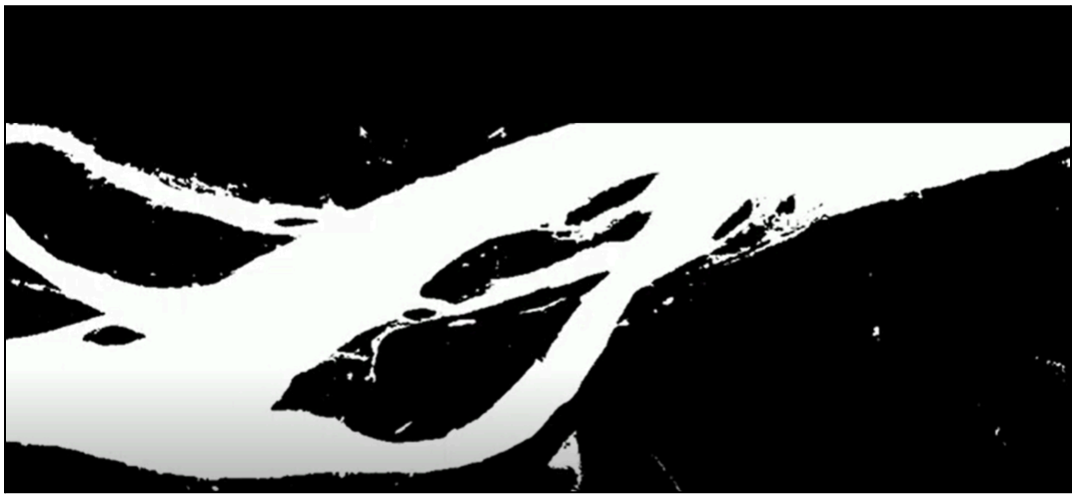


Figure 8: Clear Preprocessed Image

One limitation using segmented river photos for the training models is the variability in image quality and clarity in satellite images. Despite efforts to standardize and preprocess the images, not all segmented river masks show consistent clarity and distinguishable features, as shown in Figure 9. This variability can pose a challenge to the training model as unclear images may lead to inaccuracies in predicting river erosion. Figure 9 is an example of a less clear preprocessed image of a river that underwent the same preprocessing steps as the image in Figure 8.

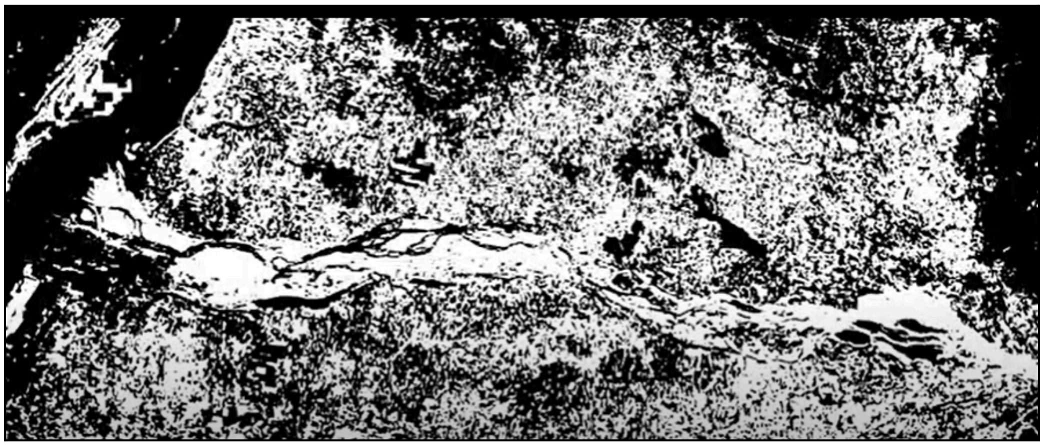


Figure 9: Unclear Preprocessed Image

The results of the ConvLSTM model highlight the challenges of achieving effective performance, primarily due to the low quality of the images used for training. The average precision and recall are mediocre, with values of 0.659 and 0.713 respectively, showing difficulty in correctly identifying river shape change. The average F1 score, which balances precision and recall, is also notably moderate at 0.682, suggesting below average overall predictive capability. Additionally, the Matthews correlation coefficient (MCC), a measure of classification quality, is low at 0.331, implying weak correlation between the predicted and actual classes.

These poor performance metrics can be attributed mainly to the necessity of compressing the images by downsampling, to a resolution of 128x128 due to hardware limitations, which significantly reduced the quality and the amount of detail in each of the training images. Downsampling caused loss of features (pieces of rivers able to be identified), precise localization of objects (ability to accurately identify boundaries of the rivers), and overfitting (due to noise). Despite these challenges, the model's training efficiency shows potential, with training times of 40 minutes on CPU and 4 minutes on GPU, indicating that computational optimization is feasible. These results suggest that while the current model configuration struggles with accuracy, there may be room for improvement through increased hardware capability, hyperparameter tuning, and increasing the amount of data collected.

Measure	**Value**
Average Precision	0.659
Average Recall	0.713
Average F1 Score	0.682
Average MCC	0.331
LSTM Model Training Time	40 minutes CPU, 4 minutes GPU
LSTM Inference Speed	1s per frame prediction, 500ms per step

Table 1: ConvLSTM Performance

Figure 9 shows a clear difference between the test image and the predicted output. The test image shows distinct features and patterns that are slightly different (areas of the river have grown, such as width) than the predicted output, highlighting the model's limitations in processing low-quality, compressed images.

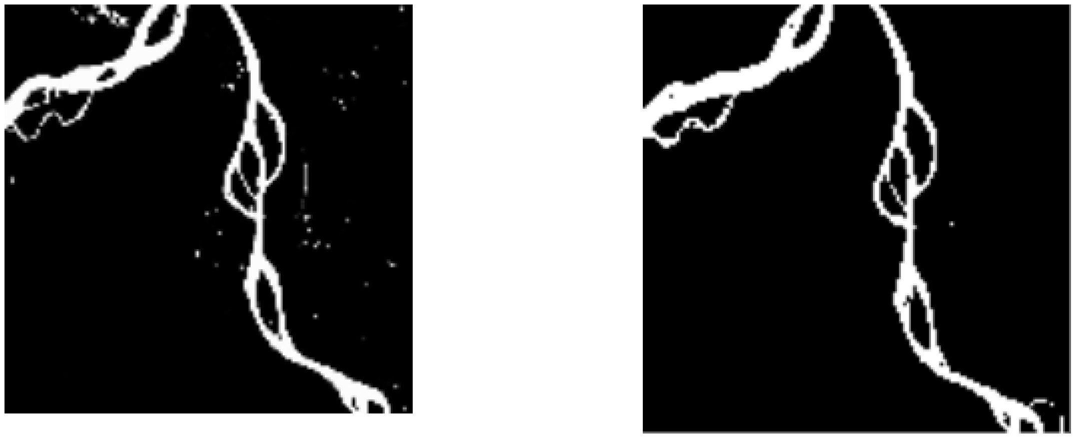


Figure 10: Test Data (Left), Predicted Output (Right)

Section 5. Lessons Learned

Throughout the project, the group gained significant skills and experiences, including a better understanding of river morphology and erosion processes, which could be applicable for environmental fields like environmental and coastal engineering. This project also increased the group's proficiency in utilizing tools such as OpenCV for improving image quality, segmenting image features, and extracting information.

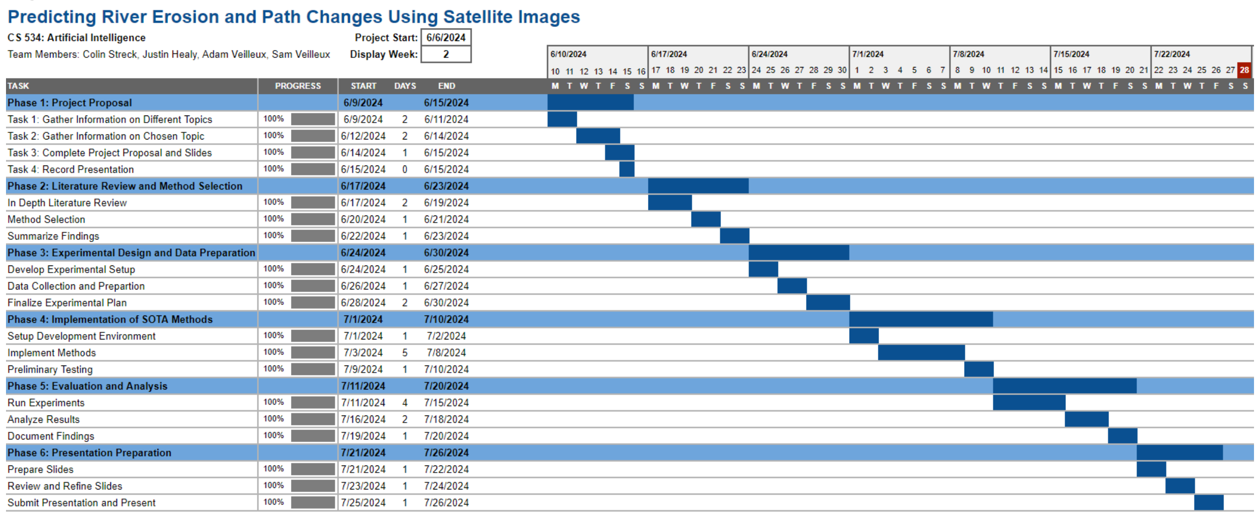
The project also increased the group's knowledge in deep learning, particularly in time series analysis and evaluation methodologies such as accuracy, precision, recall, F1 score, and Matthews Correlation Coefficient.The group also became familiar with TensorFlow's Keras API for implementing the ConvLSTM model and optimizing its performance using GPU acceleration.

A primary lesson learned from the data collection portion of the project is to focus the images collected specifically on the areas of interest to reduce noise and other distractions. Additionally, the group learned the importance of computational power and the significant limitations that can arise when training models due to insufficient hardware. Insufficient RAM can cause failures during training, necessitating image compression that can impact the accuracy and effectiveness of the model. Specifically, in this project, hardware limitations required the images to be reduced to 128x128 pixels, which significantly degraded the model's ability to capture all the nuances in river shapes.

Section 6. Conclusions and Future Work

In this project, we utilized a dataset of satellite images spanning 1984 to 2022 from Google Earth Timelapse Engine to study and predict changes in a specific river over time combining Superpixel segmentation and Convolutional Long Short-Term Memory (LSTM) networks to analyze temporal changes in river morphology. The images undergo preprocessing using techniques like normalization, filtering, and edge detection to enhance river features and highlight changes. The dataset is split into training and testing sets based on a specific year, with images up to that year used for training a ConvLSTM. The trained model then predicts future changes in the river. We achieved the key milestones outlined in our Gantt Chart, by meeting often and planning the work that needed to be completed each week.

Figure 11: Gantt Chart



Predicting River Erosion and Path Changes Using Satellite Images

CS 534: Artificial Intelligence
Team Members: Colin Streck, Justin Healy, Adam Veilleux, Sam Veilleux

Project Start: 6/6/2024
Display Week: 2

TASK	PROGRESS	START	DAYS	END
Phase 1: Project Proposal		**6/9/2024**		**6/15/2024**
Task 1: Gather Information on Different Topics	100%	6/9/2024	2	6/11/2024
Task 2: Gather Information on Chosen Topic	100%	6/12/2024	2	6/14/2024
Task 3: Complete Project Proposal and Slides	100%	6/14/2024	1	6/15/2024
Task 4: Record Presentation	100%	6/15/2024	0	6/15/2024
Phase 2: Literature Review and Method Selection		**6/17/2024**		**6/23/2024**
In Depth Literature Review	100%	6/17/2024	2	6/19/2024
Method Selection	100%	6/20/2024	1	6/21/2024
Summarize Findings	100%	6/22/2024	1	6/23/2024
Phase 3: Experimental Design and Data Preparation		**6/24/2024**		**6/30/2024**
Develop Experimental Setup	100%	6/24/2024	1	6/25/2024
Data Collection and Prepartion	100%	6/26/2024	1	6/27/2024
Finalize Experimental Plan	100%	6/28/2024	2	6/30/2024
Phase 4: Implementation of SOTA Methods		**7/1/2024**		**7/10/2024**
Setup Development Environment	100%	7/1/2024	1	7/2/2024
Implement Methods	100%	7/3/2024	5	7/8/2024
Preliminary Testing	100%	7/9/2024	1	7/10/2024
Phase 5: Evaluation and Analysis		**7/11/2024**		**7/20/2024**
Run Experiments	100%	7/11/2024	4	7/15/2024
Analyze Results	100%	7/16/2024	2	7/18/2024
Document Findings	100%	7/19/2024	1	7/20/2024
Phase 6: Presentation Preparation		**7/21/2024**		**7/26/2024**
Prepare Slides	100%	7/21/2024	1	7/22/2024
Review and Refine Slides	100%	7/23/2024	1	7/24/2024
Submit Presentation and Present	100%	7/25/2024	1	7/26/2024

Future work could focus on advanced feature engineering, automated hyperparameter tuning, and will require a more powerful computer to train the model with high quality images. Additionally, refining preprocessing techniques to address variability in image quality will improve the system's usability and effectiveness. This project highlights the potential of AI methodologies in addressing environmental issues, and by predicting river erosion and path changes, society can better prepare for and mitigate the devastating impacts of river degradation.

We initially planned to compare the LSTM model with a CNN, but we decided to focus solely on the combination of the two using the ConvLSTM due to its superior capability in handling spatiotemporal data and our limited computational resources. Future work could include this comparison once we have enhanced computational resources and refined preprocessing techniques.

References

1. Asian Development Bank. (2023, October 5). ADB approves $200 million loan to strengthen flood and riverbank erosion risk management in Assam, India [ADB]. https://www.adb.org/news/adb-approves-200-million-loan-strengthen-flood-and-riverbank-erosion-risk-management-assam

2. National Aeronautics and Space Administration. (n.d.). The most erosive area on the Padma. NASA Earth Observatory. earthobservatory.nasa.gov (https://earthobservatory.nasa.gov/images/92529/the-most-erosive-area-on-the-padma)

3. Kiranyaz, Serkan; Avci, Onur; Abdeljaber, Osama; Ince, Turker; Gabbouj, Moncef; Inman, Daniel J.: *1D convolutional neural networks and applications: A survey*. Mechanical Systems and Signal Processing, Volume 151, 2021. ISSN 0888-3270, https://doi.org/10.1016/j.ymssp.2020.107398.

4. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghan, M., Minderer, M., Heigold, G., Gelly, S., Uszkorei, J., & Houlsby, N. (2021, June 3). *AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE*. arxiv. https://arxiv.org/pdf/2010.11929

5. Ranftl, R., Bochkovskiy, A., & Koltun, V. (2021, March 24). *Vision Transformers for Dense Prediction*. ARXIV. http://arxiv.org/pdf/2103.13413

6. Thisanke, H., Deshan, C., Chamith, K., Seneviratne, S., Vidanaarachchi, R., & Herath, D. (2023, May 5). *Semantic Segmentation using Vision Transformers: A survey*. arxiv. https://arxiv.org/pdf/2305.03273

7. Vaswani. (n.d.). *Papers with code - transformer explained*. Explained | Papers With Code. https://paperswithcode.com/method/transformer

8. Zhu, Alex Zihao, et al. "Superpixel Transformers for Efficient Semantic Segmentation." arXiv, 2023, arXiv:2309.16889, https://arxiv.org/pdf/2309.16889

9. Descovi, C. S., Zuffo, A. C., Mohammadizadeh, S. M., Murillo-Bermúdez, L. F., & Sierra, D. A. (2023)1. Utilizing Long Short-Term Memory (LSTM) Networks for River Flow Prediction in the Brazilian Pantanal Basin. *Holos*, 39(5), e16315 https://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/download/16315/3853/42501#:~:text=This%20article%20demonstrates%20the%20successful,future%20streamflow%20in%20the%20region..

10. Xu, W., Jiang, Y., Zhang, X., Li, Y., Zhang, R., & Fu, G. (2020). https://www.researchgate.net/publication/346125130_Using_long_short-term_memory_networks_for_river_flow_prediction Hydrology Research, 51(2).

11. Semenoglou, Artemios-Anargyros, et al. “Image-based time series forecasting: A deep convolutional neural network approach.” *Neural Networks*, vol. 157, pp. 39–53, https://doi.org/https://doi.org/10.1016/j.neunet.2022.10.006.

12. Fingerhut, H., & Collins, D. (2024b, June 27). *House on edge of Swollen River near Rapidan Dam in Minnesota collapses into the water, video shows*. ABC7 New York. https://abc7ny.com/post/house-teetering-edge-swollen-blue-earth-river-rapidan/15003815/

13. Wong, T., & Ethirajan, A. (2024, July 8). *Millions affected as floods sweep across South Asia*. BBC News. https://www.bbc.com/news/articles/cw0y6wwn9yyo

14. Durrani, A. U. R., Minallah, N., Aziz, N., Frnda, J., Khan, W., & Nedoma, J. (2023). Effect of hyper-parameters on the performance of ConvLSTM based deep neural network in crop classification. PLoS One, 18(2), e0275653. Effect of hyper-parameters on the performance of ConvLSTM based deep neural network in crop classification - PMC (nih.gov)