Levee Seepage Identification from Aerial Images using Machine Learning

Sofiane Benkara
sbenkara@uno.edu

Follow this and additional works at: https://scholarworks.uno.edu/td

Recommended Citation
https://scholarworks.uno.edu/td/3001

This Thesis-Restricted is protected by copyright and/or related rights. It has been brought to you by ScholarWorks@UNO with permission from the rights-holder(s). You are free to use this Thesis-Restricted in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis-Restricted has been accepted for inclusion in University of New Orleans Theses and Dissertations by an authorized administrator of ScholarWorks@UNO. For more information, please contact scholarworks@uno.edu.
Levee Seepage Identification from Aerial Images using Machine Learning

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

By

Sofiane Benkara

B.A. University of Algiers, 2007

June, 2011
Acknowledgments

All praises to Allah for the completion of this thesis. I feel blessed and guided that finally, I can finish this thesis.

First and foremost, I would like to sincerely express my special thanks and gratitude to my master’s thesis advisors and mentors, Dr. Md. Tamjidul Hoque. Dr. Hoque is a great mentor, and I appreciate him giving me the learning opportunity with his research group. I could not have imagined having a better supervisor and mentor. This thesis would have been a lot harder to undertake without your support.

The Department Chair, Dr. Mahdi Abdelguerfi, for providing positive encouragement and guidance. It has been a great pleasure and honor to have you as my supervisor; This would not have been possible without your support. My sincere thanks also go to you.

Dr. Shaikh Arifuzzman from the Department of Computer Science, for his insightful suggestions and for taking the time to serve as my committee member and judge my defense.

Graduate Coordinator Dr. Summa, for the endless motivation and support since I joined UNO until I finish this thesis.

Dr. Maik Flanagan from the New Orleans District, U.S. Army Corps of Engineers, for his expertise and assistance. And for providing us with the data needed to begin this research.

My fellow Research students, for the friendly environment and helpful discussions about different issues during these months and many fruitful discussions on machine learning.

My UNO professors for their support, intuitive comments, and helpful criticisms.

Finally, with great affection and fondness, I’d like to recognize my family and friends, without whose support this journey would not have been possible without their support.
Table of Content

List of Tables iv
List of Figures v
Abstract vi
Chapter 1 Introduction 1
Chapter 2 Background 2
Chapter 3 Related work 5
Chapter 4 Dataset 7
  Data Preprocessing 8
  Image augmentation 8
  Data Analysis 9
  Plotting of Kernel (Learned Weights) in Sample Domain 11
Chapter 5 Experimental setup 12
Chapter 6 Proposed method 14
  Discrete Cosine Transform (DCT) 15
  Thepade’s Sorted n-ary Block Truncation Coding (TSBTC n-ary) 17
  One-dimensional Convolutional Neural Network 18
  Ensemble meta-estimator 20
Chapter 7 Results 22
  Base model 22
  Base model with batch normalization 24
  Base model with batch normalization and regularization 25
  Base model with batch normalization on CNN layers 26
  Ensemble Learning 28
Chapter 8 Segmentation 30
  U-Net 30
  Attention U-Net 30
  Metrics 31
  Conclusions 34
  Limitations and future work 35
References 36
List of Tables

Table 1. - Survey on recent Convolutional Neural Networks methods. 6
Table 2. - Data augmentation parameters applied on the training set. 9
Table 3. - Performance evaluation of baseline models. 13
Table 4. - The architecture of the 1D deep convolutional neural network. 19
Table 5. - Different models composition. 20
Table 6. - Correlation table between the six different models. 21
Table 7. - Models’ performance showing accuracy and false-negative rate. 29
Table 8. - Models’ performance using augmentation and Ensemble learning. 29
Table 9. - Attention U-net and U-net accuracy and dice score comparison. 31
List of Figures

Figure 1. - Sketch for through-seepage failure analysis of a river levee. 2
Figure 2. - Close-up image of Seepage near New Orleans levee. 3
Figure 3. - Close-up image of Seepage near New Orleans levee. 3
Figure 4. - Aerial image of Seepage near New Orleans levee. 4
Figure 5. - Aerial image of Seepage near New Orleans levee. 4
Figure 6. - Synthetically generated images of levee without Seepage. 7
Figure 7. - Sample of high-resolution aerial images from FloodNet Dataset. 8
Figure 8. - Image containing Seepage before and after augmentation. 9
Figure 9. - Graph visualizing the dataset balance between the two classes. 10
Figure 10. - Graph visualizing the class spread. 10
Figure 11. - Kernel weight coefficients of the RGB vector. 11
Figure 12. - Block diagram of the proposed method. 15
Figure 13. - Pixels to DCT coefficients on a (64 × 64) image slice. 16
Figure 14. - Pixels to the frequency domain on a (64 × 64) image slice. 16
Figure 15. - TSBTC processing input into a flattened and sorted array vector. 18
Figure 16. - Models’ training and validation accuracy results after 50 epoches. 21
Figure 17. - Base model training and test accuracy over epochs. 22
Figure 18. - Base model training and test loss over 50 epochs. 23
Figure 19. - Base model confusion matrix results. 23
Figure 20. - Base model with batch normalization training and test accuracy. 24
Figure 21. - Base model with batch normalization training and test loss. 24
Figure 22. - Base model with batch normalization confusion matrix results. 25
Figure 23. - Base model with Batch Normalization and Regularization accuracy. 25
Figure 24. - Base model with Batch Normalization and Regularization loss. 26
Figure 25. - Base model with BN and Regularization Confusion Matrix results. 26
Figure 26. - Base model with BN on CNN layers training and test accuracy. 27
Figure 27. - Base model with batch normalization on CNN layers training and test loss. 27
Figure 28. - Base model with batch normalization on CNN layers Confusion Matrix results. 28
Figure 29. - U-Net model training and validation accuracy. 30
Figure 30. - U-Net training and validation loss. 30
Figure 31. - Comparison between ground truth right and U-Net prediction on the left. 32
**Abstract**

Levees protect from natural disasters that can threaten human health, infrastructure, and biological systems by protecting low-lying lands near or below sea level from flooding. However, Seepage in those levees undermines their structural integrity, leading to failures. Today the United States has approximately over a hundred thousand miles of levee, many of which are reaching or have surpassed their initial design life.

Given the concern, there is a need to develop reliable, rapid, and non-intrusive levee monitoring systems to detect the presence of Seepage. This study explores the use of Deep Convolutional Neural Network (DCNN) integrated with Discrete Cosine Transform (DCT) and Thepade’s Sorted Block Truncation Coding (TSBTC) to detect Seepage in aerial images. It also compares existing models that achieved good results for the classification of aerial images using decision trees, Support Vector Machines, and k-means clustering. Our model detected Seepage with great accuracy, with fewer resources, and faster speed when compared.

*Keywords – Seepage, Aerial Image classification, Neural Network, Discrete Cosine Transform, Thepade’s Sorted Block Truncation Coding n-ary, Image segmentation*
Chapter 1 Introduction

Seepage monitoring continues to be a concern for levee engineers. Seepage occurs when water seeps through the levee body and reaches protected areas, usually due to erosion along the sides of an opening whenever the hydraulic shear stress is greater than the critical shear stress of the soil. Therefore, levee seepage is one of the most frequent causes of initial levee failure.

Researchers have made several attempts to detect Seepage in dams by using time series data such as water level, rainfall, pressure, temperature, permeability coefficient, and water viscosity to detect and analyze the trend of Seepage. However, most of those attempts were limited to specific dams’ topologies. In addition, it is common for dams to have equipment like piezometers installed in the dam’s foundation, which makes collecting data much more accessible. Unfortunately, that is not the case for levees, and therefore data is lacking. This gap in research led us to explore aerial images to learn visual features of Seepage and use them for detection.
Chapter 2 Background

This section contains the background of levee seepage, its significance concerning levee health, and object detection research fields.

Seepage is among the most frequent hydraulic causes that can initiate the breaching process of a levee [1]. In addition, Seepage of water, particularly during a flood, cause concern for the safety of a levee when internal erosion mechanisms occur [2]. Although the primary function is to retain water to protect surrounding lands from flooding, a levee can fail if water reaches protected lands by seeping over, through, or under the levee body.

The sketch in Figure 1 depicts the schematization of the levee with Seepage along the sides of an opening. Seepage initiates if hydraulic shear stress is greater than critical shear stress of the soil. In Figure 1, Seepage is going directly through the levee embankment. We can see the phreatic line in blue, which is known as the seepage line or saturation line. This line separate the saturated soil mass from the unsaturated one. On the right side of the image we can see the emergence of the seepage at the landside slope toe. The imergence of the seepage is usually either at the levee toe or on at a higher landside elevation depending on the levee geometry.

However, it can take multiple forms, i.e., through the foundation, such as piping, or as a form of erosion when finer particles are carried out of the soil mass.

By design, the levee system in New Orleans keeps the city safe from Lake Pontchartrain and the Mississippi River. A breach of one or more levees would cause massive flooding in the entire polder where the breaches occurred, making the entire area subject to the impact of this hazard. Additionally, much of New Orleans is below sea level, and once a levee breach occurs, it becomes challenging, time-consuming, and costly to remove the water from the affected area. Affected structures and infrastructure may remain submerged in floodwaters for lengthy periods until the reparation of the breach. A breach in any levee is possible from either natural or artificial causes, and the resulting flooding would be catastrophic [3]. A levee failure has the potential to flood an entire city. In 2005, Hurricane Katrina flooded 80% of the city with areas under 6 to 20 feet of water [5].

Figure 1. The figure depicts the schematization of the levee with Seepage along the sides of an opening. Seepage is going directly through the levee embankment in this example. This causes the landside of the levee to excrete muddy water, which is the visual signature of a Seepage. Effectively, levee Seepage is when water moves away from the river channel, either below or through the levee and surrounding land surface [3].
Figure 2 and Figure 3 are images near a levee in New Orleans. One visual clue of Seepage is the water being muddy or discolored.

**Figure 2.** A close-up image of Seepage near a levee in the New Orleans, Muddy water is one of the visual clues for Seepage, usually caused by water flow through soil, detaching soil grains and carrying them away.

**Figure 3.** A close-up image of Seepage near a levee in New Orleans displays cloudy water. Particles transported by the Seepage, such as clay or silt and sediments, usually cause the water to look cloudy.
Figure 4 and Figure 5 are aerial images near the levee in New Orleans. We can see how the seepage characteristics are more apparent from a bird’s eye view.

**Figure 4.** Aerial image near New Orleans levee with Seepage indicated by a red rectangle

**Figure 5.** Aerial image near New Orleans levee, A Seepage area with algae on the surface is highlighted in red rectangle.
Chapter 3 Related work

In the literature review, we discuss similarities and differences in the work done in similar contexts in the past compared to what we have done. Classification of images according to their contents is considered an open research problem.

Methods used for seepage detection

In research [6], experts reviewed many current predictive models, mainly dealing with one specific output variable of a given dam typology. They used an autoregressive model and a boosted regression trees (BRTs) model to detect different anomalies in dams, such as Seepage.

In work done by Ahmed Belmokre et al., they aimed to predict Seepage in dams by analyzing a feature set containing a wide range of environmental variables to determine the presence of Seepage. The approach makes use of a Random Forest Regression (RFR) [7] and Support Vector Regression machine (SVR) models to predict Seepage at different points.

In another research [8], they used a model that combines back-propagation neural networks (BPNN) architecture combined with a genetic algorithm (GA) to determine Seepage in dams. Due to its flexibility in handling non-linear distributions, this approach leads to a high accuracy model compared to statistical linear regression.

Another work used daily water levels to predict Seepage in concrete dams. Among traditional approaches, the proposed model HGWO–XGBoost adopted extreme gradient boosting (XGBoost) to predict the dam seepage. Hyperparameters of XGBoost were optimized using hybridizing grey wolf optimization (HGWO) and fivefold cross-validation [9].

Methods using 1DCNN, DCT, or TSBTC

1D CNNs have achieved excellent results in several classification applications such as personalized biomedical data, early diagnosis, and health monitoring. 1D CNNs have become popular with state-of-the-art performance in various signal processing applications such as early arrhythmia detection in electrocardiogram (ECG) [10-12], plant diseases detection [13], structural damage detection [14-18], and deterioration detection in bearings [19-23]. In a later study completed by Zhang et al. [23], both single and an ensemble of deep 1D CNNs were created to detect, localize, and quantify bearing faults. Although 1D CNNs have shown excellent results, they can be taken a step further to increase speed and accuracy by reducing the dimensionality of the extracted features. Below are a few papers which used Discrete Cosine Transform (DCT) or (TSBTC) to solve a classification problem.

In one paper, BinDCTNet used Discrete Cosine Transform (DCT) to reduce the dimensionality of the extracted features. By using Discrete Cosine Transform (DCT), this paper has achieved accuracy in less time and less memory than when using regular CNN on the same classification task [24].

Authors in [25]. Discrete Cosine Transform (DCT) with MSRBF and FROLS algorithm to classify X-ray mammograms for cancer detection. The incorporation of DTC made it capable of
contending with well-known previous CAD systems based on system identification with 93% accuracy.

A new brain tumor classification work done with dimensionality reduction using Discrete Cosine Transform (DCT) and k-means clustering combined showed that performance increases linearly with increased cluster features. However, a better performance arrived by considering more DCT coefficients for classification. The research also showed that the use of DCT significantly reduced memory utilization and increased the speed of computations.

In another study [26], authors used Thepade’s sorted N-ary block truncation coding with fourteen different machine learning classifiers for face feature extraction to identify a face as male or female. They explored various combinations of the TSBTC from two to six arrays with encouraging results.

In a novel work [27], an average of 97% and a maximum of 100% classification accuracies have been achieved using DCT with the K-means algorithm.

Below is a survey table of methods employed on the Areal images for classification, some of which used the Discrete cosine Transform.

<table>
<thead>
<tr>
<th>Year</th>
<th>Method</th>
<th>Dataset</th>
<th>Accuracy</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019</td>
<td>CNN-C and DCT for Classification [28]</td>
<td>CIFAR-10</td>
<td>~82%</td>
<td>High performance on compressed images regardless of its information reductions.</td>
</tr>
<tr>
<td>2020</td>
<td>DCT for Forensic Filtering [31]</td>
<td>BOSS base</td>
<td>~99%</td>
<td>Strong feature learning but low accuracy on a small-size dataset.</td>
</tr>
</tbody>
</table>
Chapter 4 Dataset

The bulk of our dataset comes from a collection of low-altitude airspace images taken by the New Orleans District, U.S. Army Corps of Engineers, using low altitude drones. Geocoded images with \((7152 \times 5368)\) pixels resolution. The images are from the New Orleans levee borders. A significant portion of those images come from areas where the Seepage is confirmed and were mainly images containing Seepage on them. To balance our dataset, we create synthetic images to balance our dataset by adding images without Seepage, simulating weather conditions such as rain, and adding structures to analyze their impact on the predictions.

Figure 6. shows two synthetically created images. In (a), we added rainwater and removed Seepage. In (b), we removed Seepage from the picture to create a negative seepage image.

![Synthetically generated images of levee without Seepage: (a) Addition of rain to the image (b) removal of Seepage from the image.](image)

In addition to our dataset, we added curated images from the FloodNet Dataset. The final dataset contains about 2372 images in jpg format. Since our dataset has only two classes, a human annotator labeled each picture to one of the two classes. Additionally, our dataset was large enough so that augmentation techniques were not required, nor did they provide significant improvement when used. However, we used it in some specific cases. More on that below.
FloodNet Dataset

FloodNet is a high-resolution Aerial Imagery Dataset for Post Flood scenes. A small aerial vehicles platform, DJI Mavic Pro quadcopters, collected the images after Hurricane Harvey. The whole dataset has 2343 images, divided into training (~60%), validation (~20%), and test (~20%) sets [33]. Figure 7 displays a sample of those images.

![Figure 7. Sample of high-resolution aerial images from FloodNet Dataset.](image)

Data Preprocessing

Out of the 2372 images, We placed 272 images aside for inference. The remaining images were reshaped, normalized, split into trains, and validated at the ratio of 80:20. We have resized the images to $(512 \times 512)$ and saved them into a NumPy array. Each image from the dataset is loaded, resized, and saved into a NumPy array. Then both features, i.e., X train and labels, i.e., Y train, are protected in NumPy preprocessed file so that we do not have to perform the preprocessing when we run the subsequent training on hyper-parameter tuning. Then applied, the DCT preprocessing to reduce the dimensionality to improve the training process and results.

Image augmentation

Data augmentation is another way to reduce overfitting on models. We increase the amount of training dataset by making minor alterations to our original images by rotating or flipping the image, for example. The neural network would think these are new images. The field of data augmentation is not new and goes back to augmentation performed on the MNIST set in [34].

The Original Dataset was large enough, so there was no need to use image augmentation. However, even when used, it did not show any significant improvement. However, after the first training iteration, although accuracy was good, we aimed to reduce the number of false-negative cases. While having false-positive cases was tolerable. A false-negative case would mean that we missed a seepage, and we were more concerned by those cases. We gathered those images, which were edge cases. We then augured those specific images before concatenating them to the original dataset. This process also reduces overfitting and improves accuracy. While the accuracy improvement was negligible, performing data augmentation helped reduce the cases of false
negatives, which was a concern. Table 2 shows the parameters. Figure 8 shows the result of augmentation.

**Table 2.** Data augmentation parameters were applied to the training set.

<table>
<thead>
<tr>
<th>Data Augmentation Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width-shift</td>
<td>0.1</td>
</tr>
<tr>
<td>Height-shift</td>
<td>0.1</td>
</tr>
<tr>
<td>Zoom-range</td>
<td>0.2</td>
</tr>
<tr>
<td>Horizontal flip</td>
<td>Yes</td>
</tr>
<tr>
<td>Vertical flip</td>
<td>Yes</td>
</tr>
</tbody>
</table>

![Figure 8](image_url) **Figure 8.** An image containing Seepage before and after augmentation: (a) Image before augmentation (b) Image after augmentation.

**Data Analysis**

Data interpretation is critical to understanding the insights of the dataset. So, for example, an imbalanced dataset will lead to a biased model. On the other hand, we can quickly visualize the number of labels per class on a bar chart or histogram to understand if the dataset is balanced and the number of instances or labels per class.

Figure 9 shows that both class label 0 and class label 1 are approximately the same numbers of instances or samples. So, we can conclude that our dataset is balanced. Furthermore, since our dataset is binary, we can use the step function to visualize the dataset in a sequence to understand which consecutive samples belong to which class.
Figure 9. Graph visualizing the dataset balance between the two classes (0) for non-seepage images and (1) for images containing seepage with both having 1050 images.

The below graph shows primarily two values from samples 0 to 2100, i.e., either 0 label or one label, like a step function. We can see from the chart that where we have consecutive lines overlapped with blue color shade, it means we have consecutive one label and white spaces but the horizontal line around 0 of y x-axis, then it means 0 labels are consecutive. This visualization can be more helpful if we want to visualize further the data based on time or time series. This spread has been obtained by shuffling the data, reducing variance, and ensuring that models remain general and overfit less. We also shuffle the data to ensure that our training, test, and validation sets represent the overall distribution of the data.

Figure 10. Graph visualizing the class spread. Both class are distributed evenly after the data was shuffled.
Plotting of Kernel (Learned Weights) in Sample Domain

In neural network terminology, the learned filters are simply weights. Thanks to the specialized two-dimensional structure of the filters, the weight values have a spatial relationship and plotting each filter as a two-dimensional image is meaningful. We have 1-D CNN, so we cannot draw spatial relationships, but we can draw the kernel values or weights in the sample domain.

We can access all model layers via the model.layers property. Each layer has a layer. Name property, where the convolutional layers have a naming convolution like block#_conv#, where the ‘#’ is an integer. Therefore, we can check the name of each layer and skip any that do not contain the string ‘Conv.’

Each convolutional layer has two sets of weights. One is the block of filters, and the other is the block of bias values. These are accessible via the layer.get_weights() function. We can retrieve these weights and then summarize their shape. We have an input shape of (4098 × 3), where 4098 is the length of each 1-D vector corresponding to the three values of R, G, and B. To draw or plot the 1-D CNN kernel weights, we can get filter values from the following line of code.

```
filters, biases = model.layers[0].get_weights()
```

We will not plot biases here; we are only plotting the weight coefficients of the kernel from the first layer as we are using 32 filters and a kernel size of 5 at the first layer. Figure 11 depicts the output filters with (5 × 3 × 32) shape. On the X-axis, we have kernel values from 0 to 4, and on the Y-axis, we have kernel weights coefficient values, i.e., 3 for each kernel value. We can have 32 similar graphs for the first layer.

![Figure 11. One of the filters with Kernel weight coefficients of the RGB vector. on the X-axis and kernel values on the Y-axes.](image)

Chapter 5 Experimental setup

In this section, we put light on the experimental composition of our research, including initial selection, training, and validation of the dataset, feature extraction, and engineering of the predictive model.

A baseline model is a point of reference to compare against our model. Our baseline models were created by taking existing convolutional neural networks trained on similar tasks, fine-tuning them, and using transfer learning. We used weight decay and regularization that are provided in Kera’s library. For example, in VGG19, we froze the top 15 layers and tuned the last four layers, followed by dense neural networks.

Our baseline models are VGG16, VGG19, Xception, InceptionNetv3, ResNet50, EfficientNetB0, and EfficientNetB1

Overview of the baseline models

VGG16 and VGG19
Those two architectures have several convolution layer module structures connected to three full connection layers and classified through a SoftMax layer.

Xception
This neural network is 36 layers deep. In addition, Xception contains depth-wise separable layers like MobileNet, and it also includes shortcuts where the output of specific layers is summed with the output from previous layers.

Inception v3
This model comprises symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concatenations, dropouts, and fully connected layers.

ResNet-50
This neural network has 50 layers deep. It is a subclass of convolutional neural networks; the main innovation of ResNet is the skip connection. As deep networks often suffer from vanishing gradients, i.e., the gradient gets smaller as the model backpropagates. The gradients can make learning intractable. The skip connection builds a deeper network by offsetting the vanishing gradient and allowing the network to skip through layers.

EfficientNet
This architecture is a convolutional neural network and scaling method that uniformly scales all dimensions using a compound coefficient. EfficientNet-B0 is the baseline network, and Efficient-B1 is just a scaled-up baseline network.
Table 3. Performance evaluation of baseline models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG19</td>
<td>94.32%</td>
</tr>
<tr>
<td>VGG16</td>
<td>92.77%</td>
</tr>
<tr>
<td>Xception</td>
<td>80.43%</td>
</tr>
<tr>
<td>InceptionNetv3</td>
<td>78.00%</td>
</tr>
<tr>
<td>ResNet50</td>
<td>65.14%</td>
</tr>
<tr>
<td>EfficientNet-B0</td>
<td>53.00%</td>
</tr>
<tr>
<td>EfficientNet-B1</td>
<td>51.57%</td>
</tr>
</tbody>
</table>

The experiments were performed using a 3.80 GHz AMD Ryzen Threadripper 3960X 24-Core Processor and 2 24 GB GeForce RTX 3090 GPU.
Chapter 6 Proposed method

This chapter discusses the technical approaches we applied to solve the problem definition associated with our dataset.

Images are two-dimensional with width and height. Colored images have an additional dimension for RGB color. We usually use 2D convolutions for black and white images and 3D convolutions for colored images. 2D or 3D convolutions would have been a promising approach for natural detection. However, areal images present some unique challenges.

1. The major one is figuring out the area of interest in the large image. The scale of the Seepage has small scales relative to the high-resolution images. Therefore, the Seepage occupies a tiny portion. In addition, seepage areas can be sparse, nonuniform, and concentrated in certain regions. Therefore, it is essential to focus on those regions instead of the whole image. To classify efficiently, we had to reduce dimensionality while preserving important image information so our model could focus higher entropy region.

2. Handling the high resolution of the images. A regular CNN will take a long time to train. Training a 2D or 3D neural network using the raw areal images is a massive volume of data to work with due to their high resolution. Working with such data is computationally very demanding. Reducing the overall image resolution leads to losing features critical to a good prediction. We aim to reduce the images into a more manageable form to process without losing important information.

3. Achieve better accuracy with fewer mathematical operations by reducing the dimensions of feature sets

To overcome the above, we used the Discrete Cosine Transform (DCT) in conjunction with Thepade’s Sorted Block Truncation Coding (TSBTC) to achieve higher performance by making the model focus on the critical areas of an image. Those compression techniques extract more relevant feature sets after dimensionality reduction and focus on the high entropy region. First, we took our 3D input is converted it to DCT coefficients using a multi-channel discrete cosine transform block, then feedforward to the Thepade’s Sorted n-ary Block Truncation coding (TSBTC). Thepade’s Sorted n-ary Block Truncation coding flattens our 3D data into a sorted array. Therefore, using a One-dimensional (1D) CNN to extract features in the TBTC domain is more suitable. Lastly, Our Sorted Block Truncation Coding (TSBTC) output is fed to our models. In addition, the 1DCNN is less complex and less demanding and could make our model accessible to other devices without the need to use cutting-edge GPUs to run the predictions. Opening doors to mobile computing and IoT devices.

The presented method uses four different classifiers with an ensemble combination. We used The tenfold cross-validation approach for training these classifiers. Tenfold cross-validation is one of the best approaches for training classifiers. It provides all samples from the dataset a chance to be used as training or test data, resulting in a trained classifier that is less biased.
Figure 12. shows a block diagram of our approach. From left to right, our dataset goes through a preprocessing stage, (DTC) stage, then (TBSTC) stage before being fed to the 1DCNN model for classification. Finally, we can see a path going from the model back to the dataset representing the mislabeled images which are candidates for augmentation.

![Image of block diagram]

**Figure 12.** Block diagram of the proposed method, showcasing the different steps from preprocessing to training and cross validation

**Discrete Cosine Transform (DCT)**

The discrete cosine transform (DCT) was first proposed by Ahmed [35] in 1972. DCT transforms a signal or image from the spatial domain to the frequency domain [36].

DCT exploits interpixel redundancies to render excellent decorrelation for most natural images. The Discrete Cosine Transform (DCT) attempts to decorrelate the image data like other transforms. After decorrelation, each transform coefficient can be encoded independently without losing compression efficiency [36]. Furthermore, the decorrelation characteristics of The Discrete Cosine Transform (DCT) should render a decrease in the entropy of an image. As a result, the number of bits required to represent the image reduces. This operation allows the quantizer to discard coefficients with relatively small amplitudes without introducing visual distortion in the reconstructed image. In addition, The Discrete Cosine Transform (DCT) exhibits excellent energy compaction for highly correlated images [36].

The Discrete Cosine Transform (DCT) is a compression algorithm [37], [38] that offers modest dimension reduction in feature sets. First, our image is divided into $(8 \times 8)$ squares, and each square’s Discrete Cosine Transform (DCT) is computed. See figure 11. The DCT coefficients with sufficiently large absolute values are put into the compressed file. Usually, high-frequency components that correspond to noise and small details in the image, insignificant to us, are skipped [39].

Since The Discrete Cosine Transform (DCT) can do energy compaction, we do not need the entire output of The Discrete Cosine Transform (DCT) image. We can take a cut from the top left side of the image as it contains the highest amount of information. The information reduces as we move diagonally in the picture. The Discrete Cosine Transform (DCT) can operate mathematically in any dimension; however, images are two-dimensional surfaces; so the 2D
DCT transform is used, which is given by equation 1 below. We applied the reverse (DCT) to reconstruct our image using the $(8 \times 8)$ blocks containing the most informative coefficients.

\[
T[i,j] = c(i,j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} V[x,y] \times \cos \left( \frac{2y+1)i\pi}{2N} \right) \times \cos \left( \frac{2x+1)j\pi}{2N} \right)
\]

\[c(i,j) = \begin{cases} 
\frac{2}{N}, & \text{i and j \neq 0} \\
\frac{1}{N}, & \text{i and j = 0}
\end{cases}
\]

**Figure 13.** Our $(512 \times 512)$ pixels image is broken into $(8 \times 8)$ squares containing 4096 pixels; for each pixel with values between 0 and 255, the DCT works on pixel values ranging from -128 to 127. The original block is leveled by subtracting 128 from each pixel value before performing the Discrete Cosine Transform, which is done by matrix multiplication. The DCT coefficients are then rearranged, with the highest values in the top right decreasing as we move diagonally to the bottom right. A similar process happens in each one of the blocks of our image.

![Figure 13. A (64 × 64) Pixels image slice. Each pixel with a value between 0 and 255 to DCT coefficients on a](image)

**Figure 14.** A $(64 \times 64)$ cut from our image. by applying DCT to the image, we transform the image to the frequency domain generating the DCT coefficient matrix we can see on the right side of the figure. Our now compressed image gives complete information with the highest value at the top left (White pixels are the maximum DCT coefficient) and decreases as we move to the bottom right (black pixels represent the minimum DCT coefficient). Thus, a small cut of the DCT matrix at the top left of the matrix will contain the highest information from the image giving Attention to essential points in the picture. Note that the DCT coefficient values are absolute.
Figure 14. Pixels to the frequency domain on a (64 × 64) image slice. The DCT coefficient matrix with the lower frequencies on the upper left corner. The bottom right ones represents the less important frequencies.

**Thepadé’s Sorted n-ary Block Truncation Coding (TSBTC n-ary)**

Block truncation [40-42] coding was developed in 1979 for grayscale images [43]. However, Block truncation coding proves to be better in color feature extraction [38].

Here Ti indicates the i\textsuperscript{th} cluster centroids of the image. Here the image is converted into a one-dimensional sorted array; using the sorted array, the TSBTC-array feature vector is computed as [T1, T2], as shown in the equation below. TR\textsubscript{1} here means a red first/upper centroid cluster, as shown in Figure 13. The centroid cluster can explain most of the characteristics of the whole cluster, which can be given by the formula below.

\[ TR_1 = \frac{2}{d \times j} \sum_{r=1}^{\frac{d \times j}{2}} \text{sort}(r) \]  \hspace{1cm} (2)

\[ TR_2 = \frac{2}{d \times j} \sum_{r=\frac{d \times j}{2}+1}^{d \times j} \text{sort}(r) \]  \hspace{1cm} (3)

\[ TG_1 = \frac{2}{d \times j} \sum_{r=1}^{\frac{d \times j}{2}} \text{sort}(r) \]  \hspace{1cm} (4)

\[ TG_2 = \frac{2}{d \times j} \sum_{r=\frac{d \times j}{2}+1}^{d \times j} \text{sort}(r) \]  \hspace{1cm} (5)

\[ TB_1 = \frac{2}{d \times j} \sum_{r=1}^{\frac{d \times j}{2}} \text{sort}(r) \]  \hspace{1cm} (6)

\[ TB_2 = \frac{2}{d \times j} \sum_{r=\frac{d \times j}{2}+1}^{d \times j} \text{sort}(r) \]  \hspace{1cm} (7)
Figure 15. Show the reconstructed output image of DCT \((512 \times 512 \times 3)\) is rolled up and converted into one dimensional over all the RGB channels \((4096 \times 3)\); the color planes R, G, and B are converted into a one-dimensional array sorted as sortR, sortG, and sortB. Using these sorted one-dimensional color planes, the TSBTC-2ary feature vector is computed as \([TR1, TR2, TG1, TG2, TB1, TB2]\) \([44, 45]\). The calculated two centroids along each channel of RGB are appended to the rolled-up \((4096 \times 3)\) output of The Discrete Cosine Transform (DCT), making the final feature vector of size \((4098 \times 3)\) over n images in the dataset. Centroid contains the complete cluster information.

![Diagram of TSBTC transformation](image)

Figure 15. TSBTC transforming the input to a flattened and sorted array vector with two centroids appended.

**One-dimensional Convolutional Neural Network**

Convolutional Neural Network (CNN) is a way to implement an artificial neural network. CNNs are the go-to model for every image-related problem. CNNs are Artificial Neural Networks with alternating convolutional and subsampling layers for effective feature extraction, especially from images. CNNs combine feature extraction and classification tasks into a single body without preprocessing steps or utilizing fixed and handmade features that could require high computational complexity as traditional Artificial Neural Networks do. Instead, CNNs learn complex patterns by extracting learned features from the raw data to maximize classification accuracy.

One-dimensional (1D) CNNs operate in a multi-scale manner from local to global. They are usually used with sequential data sets. Such as speed, pressure, temperature, and humidity. Generally, 2DCNN and 3DCNN are more common in image recognition applications since images are two-dimensional matrices. However, some scholars successfully used 1DCNN to realize fault diagnosis. For example, in [47], the authors used 1DCNN for Motor Fault Detection by using raw data (signal) without using a feature extraction algorithm. Experimental results showed more efficient and effective results. [48] used it for Fault Diagnosis of Wheelset Bearings in High-Speed Trains. Compared to 2DCNN, 1DCNN has a better ability for feature extraction. Furthermore,
since 1DCNN has only one-dimensional convolution, which makes it more straightforward with a small number of parameters hence lower computational complexity, i.e., an image with the dimensions of \((N \times N)\) convolve with kernel \(K \times K\) will have a computational complexity \(\sim O(N^2K^2)\) while in the 1D convolution is \(\sim O(NK)\).

One-dimensional (1D) Convolutional Neural Networks have shown advantageous and more efficient for specific applications than their 2D counterparts [12–22]. The convolutional layer uses a one-dimensional convolution kernel to perform the convolution calculation for the local region of the input signal to produce the corresponding one-dimensional feature map, and different convolution kernels extract various features in the input signals [49].

As in the conventional 2D CNNs, the input layer receives the raw 1D signal, and the output layer is a Multi-Layer Perceptron with the number of neurons equal to the number of classes. The convolution layer is typically situated just after the input layer; first performs a sequence of convolutions, the sum passed through the activation followed by the sub-sampling operation. Then process the raw 1D data and learn to extract features; each convolution detects specific features on input feature maps to achieve weight sharing on the same input feature map. Those features are then used in the classification.

The MLP-layers are combined into one process. These 1D convolutions are linear weighted sums of two 1D arrays and can execute as a parallel Forward and Back-Propagation.

<table>
<thead>
<tr>
<th>#</th>
<th>Layer(s)</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Conv 1D (32 k: 5)</td>
<td>((N \times 4094 \times 32))</td>
</tr>
<tr>
<td>3</td>
<td>Conv (32 k: 5)</td>
<td>((N \times 4094 \times 32))</td>
</tr>
<tr>
<td>4</td>
<td>MaxPool 1D (P: 2)</td>
<td>((N \times 2045 \times 32))</td>
</tr>
<tr>
<td>5</td>
<td>Conv 1D (32 k: 5)</td>
<td>((N \times 2041 \times 64))</td>
</tr>
<tr>
<td>6</td>
<td>Conv (32 k: 5)</td>
<td>((N \times 2041 \times 64))</td>
</tr>
<tr>
<td>7</td>
<td>MaxPool 1D (P: 2)</td>
<td>((N \times 1018 \times 64))</td>
</tr>
<tr>
<td>8</td>
<td>Conv 1D (32 k: 5)</td>
<td>((N \times 1016 \times 128))</td>
</tr>
<tr>
<td>9</td>
<td>Conv (32 k: 5)</td>
<td>((N \times 1016 \times 128))</td>
</tr>
<tr>
<td>10</td>
<td>MaxPool 1D</td>
<td>((N \times 507 \times 128))</td>
</tr>
<tr>
<td>11</td>
<td>Dropout (0.25)</td>
<td>((N \times 507 \times 128))</td>
</tr>
<tr>
<td>12</td>
<td>Flatten ()</td>
<td>64896</td>
</tr>
<tr>
<td>13</td>
<td>Dense (128, relu, K.L.12) (L=0.1)</td>
<td>128</td>
</tr>
<tr>
<td>14</td>
<td>Dropout (0.5)</td>
<td>128</td>
</tr>
<tr>
<td>15</td>
<td>Dense (1, sigmoid)</td>
<td>1</td>
</tr>
</tbody>
</table>

The structure consists of one or more convolutional layers; each convolution kernel detects specific features. Because of the convolution layers, the number of feature maps increases, which is not conducive to calculation; a pooling layer is necessary to reduce dimensionality, followed by a fully connected layer to classify the results. Higher layers use data from lower layers to recognize more complex patterns during this process. First, the input layer
takes our images; then, the convolutional and pooling layers do the feature extraction and processing.

Our 1D-CNN model is trained on the generated dataset (N × 4098 × 3) with batch normalization, regularization, and dropout. We used Ten folds Cross-validation to validate the model’s performance.

First, the input layer takes the feature vector tensors. Next, the model randomly initializes Weights and bias. Then, the input is propagated forward through the convolution, pooling, and fully connected layers to obtain the output value. The model repeated this procedure using different kernels to form as many output feature maps as desired. At the end of each forward propagation, the model calculates the error between the output and expected values. Finally, the model updates the weights and bias until the acceptable error margin is fulfilled [50]. The weights that minimize the error function are considered a solution to the learning problem.

**Ensemble meta-estimator**

The ensemble is strategies for improving the predictive ability of a machine learning model, also known as Bootstrap Aggregation. A costume is a set of individually trained models combined with predictions when classifying new cases through a voting process. The voting process can be based on their average value, also known as soft voting, or based on the most significant sum of votes from models referred to as hard voting. Generally, ensemble learning is used to lower variance in accuracy for classification tasks. We used a Bagging as our ensemble mechanism, which fits each model on a subset of data.

Our ensemble consists of four different versions of the base model. For example, in Table 5, Our Base model is one, followed by the three different. Model 5 is a hard voting ensemble, and models 6 and 7 are soft voting ensembles, with the 7th having a higher threshold. On the right side of the table, there is a short description of how each model is different.

<table>
<thead>
<tr>
<th>#</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Base model.</td>
</tr>
<tr>
<td>2</td>
<td>Base model with Batch Normalization (BN).</td>
</tr>
<tr>
<td>3</td>
<td>Base model with Batch Normalization (BN) and Regularization.</td>
</tr>
<tr>
<td>4</td>
<td>Base model with Batch Normalization (BN) on CNN Layer with Matrix.</td>
</tr>
<tr>
<td>5</td>
<td>Hard voting ensemble.</td>
</tr>
<tr>
<td>6</td>
<td>Soft voting ensemble with a threshold 50.</td>
</tr>
<tr>
<td>7</td>
<td>Soft voting ensemble with a threshold 60.</td>
</tr>
</tbody>
</table>

Table 5. Different models composition.
Figure 16. show the training and validation of the four models converging at 50 epochs. The figure gives us an idea of how those models stacked against each other.

![Figure 16. Models’ training and validation accuracy results after 50 epochs. We can see each one of the individual models' performance.](image)

Table 6. shows the models’ correlation. For example, model2 and model4 highlighted in red are highly correlated, which means the predictions of model2 and model4 are 96% the same, or from 100 predictions, only four predictions are different. Since our models are highly correlated, hard voting is more appropriate; however, we also used soft voting with two models, each with a different threshold of 50 and 60.

<table>
<thead>
<tr>
<th></th>
<th>model1</th>
<th>model2</th>
<th>model3</th>
<th>model4</th>
<th>model5</th>
<th>y_true</th>
<th>vot_hard</th>
<th>model6v1</th>
<th>model6v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>model1</td>
<td>1.000000</td>
<td>0.878086</td>
<td>0.861134</td>
<td>0.87901</td>
<td>0.88146</td>
<td>0.877434</td>
<td>0.936512</td>
<td>0.889474</td>
<td>0.910187</td>
</tr>
<tr>
<td>model2</td>
<td>0.878086</td>
<td>1.000000</td>
<td>0.923727</td>
<td>0.965806</td>
<td>0.973134</td>
<td>0.974616</td>
<td>0.977565</td>
<td>0.980595</td>
<td>0.965924</td>
</tr>
<tr>
<td>model3</td>
<td>0.861134</td>
<td>0.923727</td>
<td>1.000000</td>
<td>0.920525</td>
<td>0.927276</td>
<td>0.925681</td>
<td>0.961374</td>
<td>0.934668</td>
<td>0.954036</td>
</tr>
<tr>
<td>model4</td>
<td>0.87901</td>
<td>0.965806</td>
<td>0.920525</td>
<td>1.000000</td>
<td>0.968662</td>
<td>0.967184</td>
<td>0.976944</td>
<td>0.985123</td>
<td>0.956136</td>
</tr>
<tr>
<td>model5</td>
<td>0.88146</td>
<td>0.973134</td>
<td>0.927276</td>
<td>0.968662</td>
<td>1.000000</td>
<td>0.980569</td>
<td>0.974840</td>
<td>0.983559</td>
<td>0.963202</td>
</tr>
<tr>
<td>y_true</td>
<td>0.877434</td>
<td>0.974616</td>
<td>0.925681</td>
<td>0.967184</td>
<td>0.980569</td>
<td>1.000000</td>
<td>0.971661</td>
<td>0.982063</td>
<td>0.958665</td>
</tr>
<tr>
<td>vot_hard</td>
<td>0.936512</td>
<td>0.977565</td>
<td>0.961374</td>
<td>0.976944</td>
<td>0.974840</td>
<td>0.971661</td>
<td>1.000000</td>
<td>0.983321</td>
<td>0.982344</td>
</tr>
<tr>
<td>model6v1</td>
<td>0.889474</td>
<td>0.980595</td>
<td>0.934668</td>
<td>0.985123</td>
<td>0.983559</td>
<td>0.982063</td>
<td>0.983321</td>
<td>1.000000</td>
<td>0.967638</td>
</tr>
<tr>
<td>model6v2</td>
<td>0.910187</td>
<td>0.965924</td>
<td>0.954036</td>
<td>0.96136</td>
<td>0.963202</td>
<td>0.958665</td>
<td>0.982344</td>
<td>0.967638</td>
<td>1.000000</td>
</tr>
</tbody>
</table>
Chapter 7 Results

In this section, we will first outline the training process and then discuss the performances of different models. The default callback saves all the logs of training in History Callback. We retrieved the records for training metrics (i.e., loss and accuracy) for the training and validation set during our training process. Specifically, the fit() function returns the history object once the training is completed on defined epochs. The returned object contains the metrics, including training accuracy, training loss, validation accuracy, and validation loss. The below code snippet can collect the keys of the metrics involved.

We then use those training histories to create the plots shown below. This section will have an in-depth discussion on those plots as they played crucial roles during our model training. We used the plots to examine three pieces of information during our training process outlined below. First, we used to investigate the speed of convergence for learning. The model will take longer to converge if the learning rate is too low.

On the other hand, if the learning rate is too large, the model might not converge or converge slowly. By looking at the loss value over training epochs, we made sure that we used a reasonable learning rate. Second, we used the plot to determine whether our model had converged. It was done by looking at loss and accuracy change over time. If the losses (both training loss and validation loss) stopped decreasing and the accuracy (both training accuracy and validation accuracy), we concluded that our model had converged. Finally, we also used the plots to ensure that our model was not over-fitting the training data. We looked for an inflection point for both validation loss and accuracy.

Base model

Figure 17 shows that the model converged in 50 epochs. Therefore, we terminated our training process at epoch 50. Because the model’s performance on the validation set shows comparable accuracy to the training set, we conclude that in epoch 50, the model has not yet overfitted the training dataset.

![Figure 17. Base model training and test accuracy for 50 epochs.](image)
In figure 16, we can see that the model has comparable performance on both train and validation datasets (labeled test). If the training loss starts to decrease further, but the validation loss stops falling, it might be a sign to stop training at an earlier epoch to avoid overfitting. Like the observation from Figure 18, we conclude that we did not overfit the base model. If we visualize the whole training log, then with a more significant number of epochs, the loss and accuracy of the model on training and testing data converged, thus making the model a stable one.

![Figure 18](image1.png)

**Figure 18.** Base model training and test loss over 50 epochs.

From Figure 19, we can get a detailed insight into the base model’s performance. Reading from the above confusion matrix, we can see that the base model predicted five wrong results out of 88 samples of the validation dataset. Out of the five results, we have three false-negative predictions and two false-positive predictions. The overall accuracy of the base model is 93.5%. For the rest of this section, we will report the training/validation accuracy, training/validation loss, and the confusion matrix for the rest of the models. First, we overview the model, presenting three plots for each model.

![Figure 19](image2.png)

**Figure 19.** Base model's confusion matrix results. Top right being a true positive. Top right: false positive, Bottom left: false-negative, and bottom-right true negative.
Base model with batch normalization

We implemented the first variation using batch normalization between layers. The model was motivated by the empirical observation that batch normalization usually improves the training and model convergence for neural networks. Figure 20 shows the training and validation accuracy, and Figure 21 shows the training and validation loss for the model. Here we can see that our model behaved differently after adding batch normalization, possibly because that batch normalization calculates the mean and variance differently between train and test. The mean and variance are calculated based on a mini-batch during training, while they are calculated using the population statistics in the test [51].

**Figure 20.** Base model with batch normalization training and test accuracy.

**Figure 21.** Base model with batch normalization training and test loss.
The base model with batch normalization has achieved an accuracy score of 90.8%, causing slightly worse performance than the base model. Figure 22 shows the confusion matrix for the Base model with batch normalization, and from the figure, we can see that the model incorrectly predicts eight samples out of 88 pieces in the validation set. Furthermore, the eight incorrect predictions have seven false-negative predictions and one false positive prediction.

![Confusion Matrix for Base Model with Batch Normalization](image.png)

**Figure 22.** Base model with batch normalization confusion matrix results. Top right being a true positive. Top right: false positive, Bottom left: false-negative, and bottom-right true negative.

**Base model with batch normalization and regularization**

The second variation uses batch normalization between layers and a regularization matrix. We used an L2 regularization to estimate the mean of the data to avoid overfitting. Figure 23 shows the training and validation accuracy, and Figure 22 shows the training and validation loss for the model.

![Accuracy vs. Epoch for Base Model with Batch Normalization and Regularization](image.png)

**Figure 23.** Base model with Batch Normalization and Regularization accuracy after training for 50 epochs
The base model with Batch Normalization and Regularization has achieved an accuracy score of 91.3%, yielding 2% performance degradation compared to the base model. Figure 25 shows the confusion matrix for the Base model with Batch Normalization and Regularization. The figure shows that the model incorrectly predicts three out of 88 samples in the validation set. We have 0 false-negative predictions and all three false-positive predictions among the three incorrect predictions.

Base model with batch normalization on CNN layers

We implemented the third variation using batch normalization on CNN layers. We used Batch normalization after the convolutional layers, input, and output. MaxPooling1D and 0.25 Dropout followed the convolutional layers, while we used the ReLU activation function after the input layer.
and sigmoid on the output layer. Figure 26 shows the training and validation accuracy, and Figure 27 shows the training and validation loss for the model.

**Figure 26.** Base model with batch normalization on CNN layers accuracy.

**Figure 27.** Base model with batch normalization on CNN layers loss.
Figure 28. Base model with batch normalization on CNN layers confusion matrix results. Top right being a true positive. Top right: false positive, Bottom left: false-negative, and bottom-right true negative.

The base model with batch normalization on CNN layers Confusion Matrix has achieved an accuracy score of 92.4%, yielding around 1% performance degradation compared to the base model. Figure 23 shows the confusion matrix for the Base model with batch normalization on CNN layers Confusion Matrix. The figure shows that the model incorrectly predicts 15 out of 88 samples in the validation set. Among the 15 incorrect predictions, we have 14 false-negative predictions and one false positive prediction.

**Ensemble Learning**

In this section, we will compare the performance of the models above. Furthermore, we will report the model performance with data augmentation and ensemble learning. In Table 5, the performance of the four models, *i.e.*, the Base model, the Base model with Batch Normalization and Regularization, the Base model with batch normalization on CNN layers) is summarized in the first four rows. We present the model performance with ensemble learning in the next two rows. Namely, we have implemented ensemble learning with both hard and soft voting. The constituent models include the four models presented above. Compared to the performance of individual models, the ensemble-learning models show significant performance improvements in the two metrics we look at - accuracy score and False Negative Rate. This observation is within the expectation of ensemble learning. The statistical learning theory claims that ensemble-learned models tend to obtain better predictive performance than could be obtained from any of the constituent models alone.
Table 7. Models’ performance shows accuracy and false-negative (FN) rates.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Cross-validation</th>
<th>FN Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base model</td>
<td>93.5%</td>
<td>97.6%</td>
<td>0.037</td>
</tr>
<tr>
<td>Base model with batch normalization</td>
<td>90.8%</td>
<td>99.1%</td>
<td>0.007</td>
</tr>
<tr>
<td>Base model with batch normalization and regularization</td>
<td>91.3%</td>
<td>97.6%</td>
<td>0.032</td>
</tr>
<tr>
<td>Base model with batch normalization on CNN layers</td>
<td>92.4%</td>
<td>98.9%</td>
<td>0.052</td>
</tr>
<tr>
<td>Ensemble soft voting</td>
<td>99.0%</td>
<td>-</td>
<td>0.008</td>
</tr>
<tr>
<td>Ensemble hard voting</td>
<td>99.1%</td>
<td>-</td>
<td>0.007</td>
</tr>
</tbody>
</table>

Table 8 presents the model performances with data augmentation (four individual and two ensemble-learning models). Overall, data augmentation slightly improved the performance of all models across the board. Data augmentation is a common practice in deep learning. It reduces overfitting and, as a result, makes the model more robust against noise. Our experiment observed this improvement.

Table 8. Models’ performance using augmentation and Ensemble learning.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>FN Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base model</td>
<td>97.6%</td>
<td>0</td>
</tr>
<tr>
<td>Base model with batch normalization</td>
<td>98.1%</td>
<td>0.35</td>
</tr>
<tr>
<td>Base model with batch normalization and regularization</td>
<td>97.5%</td>
<td>0</td>
</tr>
<tr>
<td>Base model with batch normalization on CNN layers</td>
<td>97.8%</td>
<td>0.047</td>
</tr>
<tr>
<td>soft voting Ensemble</td>
<td>99.5%</td>
<td>0</td>
</tr>
<tr>
<td>hard voting Ensemble</td>
<td>98.8%</td>
<td>0</td>
</tr>
</tbody>
</table>
Chapter 8 Segmentation

The scientific community utilized Aerial Semantic Segmentation in various tasks like building, roads, and vehicles. Seepage Segmentation is very similar to these preexisting problems. The only challenging part of Seepage Segmentation is the smaller area of interest and the needed focus to segment the size and shape of the Seepage appropriately. Here, we leveraged preexisting research done on Aerial Segmentation of roads and tried both U-Net and Attention U-Net.

U-Net

The U-Net was developed by Olaf Ronneberger et al. for Biomedical Image Segmentation. The architecture contains two paths. The first path is the contraction path (also called the encoder) used to capture the context of the image. Consecutive (3 × 3) convolutional and (2 × 2) max-pooling layers handle the input image. The activation function for each layer is a rectified linear unit (ReLU) at each max-pooling. The second path is the symmetric expanding path (also called the decoder) which enables precise localization using transposed convolutions. The symmetric expanding path is an end-to-end, fully convolutional network containing Convolutional and no Dense layers. This path aims to combine the features extracted by up-convolution. This approach consists of a concatenation of feature channels, which cut by half the number of channels. From this perspective, the architecture is symmetric in both paths, using the same convolutional size (3 × 3) and the ReLU as an activation function. The training process uses stochastic gradient descent. This architecture was trained using weighted cross-entropy error as a loss function and measured with traditional classification metrics such as accuracy, precision, and recall.

Attention U-Net

Attention U-Net was written in 2018 and proposed a novel Attention Gate (AG) mechanism that allows the U-Net to focus on target structures of varying size and shape, thus serving our purpose. In picture segmentation, Attention is a technique for highlighting only the relevant activations during training, reducing the computational resources wasted on irrelevant activations, and providing the network with better generalization power. Essentially, the network can pay “attention” to certain image parts.

To understand why Attention is beneficial in the U-Net, we need to look at the skip connections. During upsampling in the expanding path, spatial information recreated is imprecise. To counteract this problem, the U-Net uses skip connections that combine spatial information from the downsampling path with the up-sampling path. However, feature representation is weak in the earliest layers, resulting in redundant low-level feature extractions. Soft Attention implemented at the skip connections will actively suppress activations in irrelevant regions, reducing the number of redundant features brought across, thus improving the accuracy.
Metrics

- Jaccard Index (IoU):

The Intersection-Over-Union (IoU) Jaccard Index is among the most used metrics in semantic segmentation. The IoU is a practical and straightforward metric. IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth. It ranges from 0–1 (0–100%), with 0 signifying no overlap and 1 signifying perfectly overlapping segmentation.

- Dice Score:

The Dice Coefficient is two times the Area of Overlap divided by the total number of pixels in both images. The Dice coefficient is very similar to the IoU. They are positively correlated, meaning if one says model A is better than model B at segmenting an image, the other will say the same. Like the IoU, they range from 0 to 1, with 1 indicating the most significant similarity between predicted and truth [52].

Table 9. Attention U-net and U-net accuracy and dice score comparison.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Dice score</th>
</tr>
</thead>
<tbody>
<tr>
<td>U-Net</td>
<td>96.67</td>
<td>85</td>
</tr>
<tr>
<td>Attention U-net</td>
<td>83.5</td>
<td>74.7</td>
</tr>
</tbody>
</table>

When training, we resized the images into (512 × 512) since feeding the original size of the image does not seem feasible. As a result, we only could train the models with a batch size of 9. We also tried different learning rate values starting from 0.1 and decreasing by 10 to 0.00001, with 0.001 performing the best. We also used data augmentation, which increased the IOU score.

Our U-Net model yelled 96.67% accuracy with an 85 dice score; however, the precise shape segmentation was lacking. On the other hand, Attention U-Net yelled a lower accuracy score and had a dice score of 74.7.

Although U-Net performed better metric-wise, Attention U-Net could understand and replicate the same shape as ground truth masks its prediction suffered from blurry edges. U-Net failed to learn the shape of Seepage properly since they cannot focus on the shape and Seepage properly. Attention U-Net was a clear winner.

Attention U-Net had a total number of parameters of 35,239,445, and we can see its training and loss performance below.
Figure 29. shows training accuracy and validation accuracy. As training progresses, the network slowly overfits the training set as the accuracy on the validation set no longer increases.

![Model Accuracy Graph](image)

**Figure 29.** Attention U-Net model training and validation accuracy.

Figure 30 shows the training and validation loss. Validation loss is greater than the training loss and no longer decreasing. This usually indicates that the model is overfitting.

![Model Loss Graph](image)

**Figure 30.** Attention U-Net training and validation loss.
Figure 31 shows a visual comparison of segmentation performance between Attention U-Net and the original ground of truth mask. On the left, we have the prediction and mask on the right. Attention U-Net segmentation is very close to the original mas, with well-defined boundaries between the Seepage and the background. The shape is also very similar to the ground of the truth.

**Figure 31.** Comparison between Attention U-Net prediction and mask.
Conclusions

In this thesis, we proposed a method using Discrete Cosine Transform (DCT) in conjunction with Thepade’s sorted block truncation coding (TSBTC) and 1D CNN to predict Seepage in aerial images. We experimented with different individual models using Batch Normalization, Regularization and adding batch normalization on the CNN layer. In addition, we experimented with different (TSBTC) array combinations, such as four arrays and six arrays. We also tried different image augmentation techniques to allow the neural net to learn augmentations that best improve the ability to classify images correctly. Finally, we leveraged the four Machine Learning classifiers as an ensemble.

While we focused on the metric, we looked at training and test times. We used those metrics to infer model convergence during training time and ensure that the mode was not overfitted. During testing, we used metrics to compare model performances. Through comparison, we concluded that the base standing-alone individual model was the best among the four proposed models, reaching an accuracy of 97.6% when trained on augmented data. However, meta-ensemble learning outperformed its model constituents and achieved an overall accuracy of 99.0% with a soft voting scheme. We also concluded that while data-augmentation slightly improved the individual model performance, the improvement was not apparent in meta-ensemble learning.

In the second part of our thesis, we used Attention U-Net to perform a semantic segmentation on seepage images which has proven better than regular U-Net, which suffered from blurry edges and lack of boundary details. In conclusion, the Attention U-Net dataset had a better prediction of shape.

This is the first time aerial images have been proposed for seepage detection and segmentation, to the best of our knowledge.

We believe that this model has the potential to bring impact to seepage detection methods. 1D CNNS are easy to train and have lower computational complexity, making them great for mobile devices with limited computation that can be used in the field to run real-time predictions.
Limitations and future work

Our research’s main limitation or weakness is that we used synthetic images to mitigate our unbalanced dataset. While synthetic images are a compelling solution, they cannot replace authentic images, as much research has shown that Synthetic Images Influence the Result of Neural Networks for Object Detection. However, the combination of synthetic images and real photos improves the quality accuracy compared to models trained on only a small number of natural images [46].

We explored the use of image metadata during our research by extracting latitude and longitude points present in the images. This approach was motivated by the natural phenomenon that seepages are more likely to appear close to existing seepage areas. We used the haversine denoted by equation (8) to calculate the distance between two points in the angular distance, which was unnecessary since the formula calculates the distance between two points on the surface of a sphere. So, we used the vicinity, denoted by equation (9). Unfortunately, that technique could not be tested thoroughly because some images were synthetic and did not contain metadata. Nevertheless, we would like to experiment more closely in the future if we get an additional dataset and explore the use of thermal images to leverage the thermal signature present in the water.

\[
2r \arcsin \left( \sin^2 \left( \frac{\phi^2 - \phi^1}{z} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right) \right)
\]  

\[
\sin \sigma = \sqrt{(\cos U_2 \sin \lambda)^2 + (\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos \lambda)^2}
\]
References


Vita

Sofiane Benkara was born in Algiers, Algeria. He obtained his bachelor’s degree in Law from the University of Algiers Law School in 2011. He joined the University of New Orleans Computer Science department as a postbaccalaureate student in 2017, then as a master’s program student in 2020.