Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Md Abulkalam Azad

Multi-label Video Classification for Underwater Ship Inspection

Master's thesis in Marine and Maritime Intelligent Robotics (MSMIR)
Supervisor: Prog. Martin Ludvigsen
Co-supervisor: Ahmed Mohammed, Maryna Waszak
June 2023

NTNU
Norwegian University of
Science and Technology

SINTEF

Md Abulkalam Azad

Multi-label Video Classification for Underwater Ship Inspection

Master's thesis in Marine and Maritime Intelligent Robotics (MSMIR)
Supervisor: Prog. Martin Ludvigsen
Co-supervisor: Ahmed Mohammed, Maryna Waszak
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

Multi-label Video Classification for Underwater Ship Inspection

Erasmus Mundus Master's in Marine and Maritime Intelligent Robotics (MSMIR)

at

The Department of Marine Technology, NTNU

In collaboration with

SINTEF Digital

The Master's Thesis by:
Md Abulkalam Azad

Under the supervision of:
Ahmed Mohammed (SINTEF)
Maryna Waszak (SINTEF)
Prof. Martin Ludvigsen (NTNU)

11th June 2023

MARINE & MARITIME INTELLIGENT ROBOTICS

Co-funded by the Erasmus+ Programme of the European Union

Abstract

Today ship hull inspection including the examination of the external coating, detection of defects, and other types of external degradation such as corrosion and marine growth is conducted underwater by means of Remotely Operated Vehicles (ROVs). The inspection process consists of a manual video analysis which is time-consuming and labour-intensive. To address this, we propose an automatic video analysis system using deep learning and computer vision to improve upon existing methods that only consider spatial information on individual frames in underwater ship hull video inspection. By exploring the benefits of adding temporal information and analyzing frame-based classifiers, we propose a multi-label video classification model that exploits the self-attention mechanism of transformers to capture spatiotemporal attention in consecutive video frames. Apart from utilizing off-the-shelf vision transformers for extracting spatial information, we have incorporated the transformer from the original language model to extract temporal information from the video. We have specifically highlighted the underlying distinct characteristics between these transformers and made empirical modifications to optimize their performance and achieve the best results. Furthermore, our investigation delved into the self-attention mechanism, which serves as a critical component of the transformer architecture. We introduced a different but light approach known as single query attention computation, which has proven instrumental in enhancing the robustness of the model. By utilizing attention scores as weights, we were able to improve the overall performance and strengthen the reliability of our approach. In a nutshell, we have showcased three distinct approaches to multi-label video classification, and the outcomes have demonstrated great potential, positioning this work as a benchmark for future research and development in underwater video inspection applications.

Preface

I would like to begin by recalling the memory of my mother who passed away in a terrible road accident. I would not have been here today without her utmost diligence.

This master's thesis builds upon the specialization project I undertook in the previous semester, and I am grateful for the opportunity to continue this research within the Computer Vision group at SINTEF Digital in Oslo. I would like to express my gratitude to SINTEF Digital for providing me with the platform to work on this project as part of my master's thesis. I am specifically grateful for the initial opportunity I had as a summer student at SINTEF Digital, which laid the foundation for my involvement in this thesis. During this time, I had the privilege of working under the careful supervision of Dr. Ahmed Mohammed. His expertise and intellectual guidance have been instrumental in my learning of state-of-the-art techniques in the field of 3D computer vision and video understanding. I am thankful to Dr. Mohammed for his mentorship, and I believe this collaboration is just the beginning. I look forward to future projects under his supervision, as I am confident that they will further contribute to my growth and development in the field.

Secondly, I would like to show my utmost gratitude to Dr. Maryna Waszak for engaging me to the LIACi (Lifecycle Inspection, Analysis, and Condition Information system) project and invaluable contribution to this thesis work. From the very beginning, she provided me with the necessary guidance and introduced me to the initial project requirements, setting the foundation for my research. Her continuous support and close guidance have been instrumental in overcoming challenges and doubts, allowing me to progress smoothly and effectively towards achieving the main objectives of this thesis. I am particularly grateful for Dr. Waszak's unwavering support, which has not only facilitated the successful completion of this thesis but has also played a significant role in the publication of our research at renowned conferences such as OCEANS and NORA. Her dedication and assistance have been invaluable throughout this journey, and I am truly appreciative of her

involvement in this project.

I would like to express my gratitude to the co-authors of this thesis for their invaluable support, feedback, and guidance throughout the project. In particular, I would like to extend my appreciation to Prof. Martin Ludvigsen for his academic supervision and guidance from the Department of Marine Technology, NTNU. His expertise and involvement have been instrumental in shaping the academic aspects of this research. I would also like to acknowledge and thank Helene Schulerud, the research manager of the Computer Vision group at SINTEF, for her unwavering trust in my abilities and providing me with various opportunities, starting from my initial summer job at SINTEF. Her support and belief in my potential have been crucial in my academic and professional development.

Moreover, I would like to express my gratitude to the European Commission for awarding me the Erasmus Mundus scholarship, which has enabled me to pursue a master's degree in Marine and Maritime Intelligent Robotics (MSMIR). This thesis project has been an integral part of my studies within the MSMIR program, and I am thankful for the opportunity to delve into this research area through the support of the scholarship.

Finally, I would like to highlight some important aspects of my thesis work. I had the privilege of presenting a portion of my research at the esteemed OCEANS 2023 conference held in Limerick, Ireland. I am delighted to inform you that a 10-page article with the title: **Multi-label Video Classification for Underwater Ship Inspection** based on this work will be published by IEEE. Furthermore, an extended version of our approach with the title: **MViST: A Multi-label Vision Spatiotemporal Transformer**, which includes significant improvements, was accepted and presented at the NORA 2023 annual conference by my supervisor, Dr. Ahmed Mohummed, on June 6th. This recognition has also resulted in an invitation to submit the extended version of our work to the prestigious Nordic Machine Intelligence (NMI) level-1 journal. In preparation for this submission, I am currently working on an additional 10-page article, even after completing the thesis. Both the papers and presentation slides are included as attachments to provide you with comprehensive references. I hope you will find the thesis engaging and informative with the wealth of information provided.

.....
Azad, Md Abulkalam
Student No: 581732
MSMIR, Marine Technology
NTNU
11th June 2023

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

1.1 Underwater ship hull inspection

Inspection of marine vessels is a regular phenomenon in the maritime industry which plays a significant role in monitoring the life cycle and analyzing the condition of the hull. It examines the external coating and detects potential defects. For instance, corrosion, marine growth, or other external degradation can damage the hull and reduce its lifespan. Ship hull inspections are nowadays shifting to underwater operation from dry-dock to reduce the high cost and downtime of the ship. This underwater inspection is conducted by a Remotely Operated Vehicle (ROV) to further cut down the cost and prevent the risk of a human diver. The overall workflow as illustrated in Fig. 1.1 consists of three phases;

1. Collection of videos of the ship hull by operating an ROV
2. Extensive analysis of the videos by skilled personnel
3. Preparation of the inspection report based on the analysis for certification

The manual video analysis in phase 2 is time-consuming, tedious, and prone to error from human cognitive fatigue. Therefore, with the advancement of deep learning in computer vision and autonomy in underwater vehicles, an automatic video analysis system can significantly improve underwater inspection and expedite the entire process.

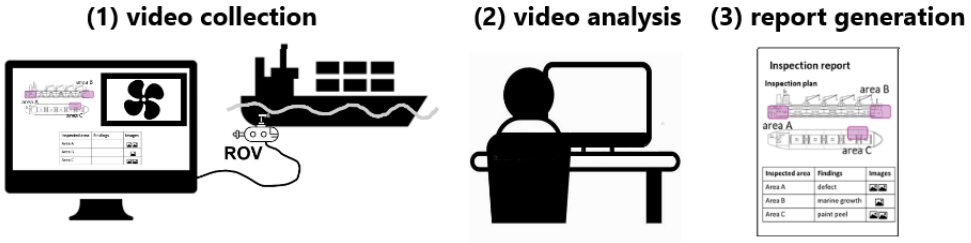


Figure 1.1: The workflow of current underwater ship hull inspection using ROVs.

1.2 Frame-wise video analysis

A trivial approach to underwater video analysis is to inspect each frame of the entire video individually and identify potential threats such as defects, corrosion, or marine growth. This approach can assign higher values to frames that are more likely to contain plausible dangers, allowing inspectors to focus only on those frames for analysis. It simply needs an efficient and robust multi-label image classifier, and there are many such off-the-shelf models available online that can be used for this purpose. We can use a pre-trained image classification model and apply an effective deep transfer learning technique as suggested in [1] to fine-tune the model for our domain. One preceding work [2] under the LIACi[1] project utilized transfer learning to train a multi-label image classifier using Microsoft Custom Vision [3] framework to classify individual frames in the video. The trained model can predict nine different class labels on the surface of the ship hull which are stated in chapter 3 of materials & methods.

However, this approach has a significant limitation as it only considers spatial information from static image frames and lacks the temporal insight that is essential for Video Understanding [4]. Besides, the underwater environment presents various difficulties for vision-based tasks. The quality of the underwater images and videos is affected by the attenuation of light and the presence of particles in the water. It becomes difficult to distinguish the objects and the background as their colours do not remain the same over time. The visibility in underwater environments is also limited, leading to low-resolution and blurry images. Moreover, the ROV motion creates distortions and blurs in the images and videos. All these factors collectively make it extremely cumbersome in an underwater environment to detect and classify potential dangers. This leads to the problem of prediction inconsistency and temporal instability in frame-based classification models during underwater video analysis.

Prediction inconsistency occurs when there is a significant difference in the model's prediction confidences on consecutive frames, even though there are negligible spatial changes between the frames. Temporal instability in frame-based models refers to the model's inability to consider dynamic information during predictions. For instance, in underwater environments, an ROV creates motion and results in incomplete target objects in most of the frames. Consequently, the model may exhibit low confidence in predicting target objects or potential dangers during video analysis.

In the following section, we present our objectives to mitigate the issues and improve the performance of the model for underwater video analysis.

[1] Lifecycle Inspection, Analysis, and Condition information system (https://www.sintef.no/en/projects/2021/liaci/)

1.3 Main objective

In order to alleviate the issues, it is necessary to train a model by learning spatiotemporal information from videos which can improve the automatic video analysis of underwater ship hull inspections. Our approach starts with enhancing the performance of the frame-based multi-label classifier by incorporating better spatial information. Next, we integrate temporal features into the trained frame-based model to develop our video model. Unlike temporal action recognition and localization [5] that consider dynamic foreground and background objects, our videos only have static scenes including ROV motion with a dynamic camera. Hence, the benefit of utilizing the temporal aspects can facilitate stabilization during the video analysis by paying attention to adjacent frames in the temporal dimension. Our core focus is to enhance the consistency and stability of the model's predictions during underwater video analysis. Therefore, in this paper, we investigate the consistency and stability of image-based classifiers which can help us in understanding the advantages and limitations of using an image-based multi-label classifier for this purpose. Furthermore, we propose a video classification model which can extract both spatial and temporal features from the video. In summary, the contributions of this work are;

a. Analysis of image-based classifiers (benefits and limitations) and improvement.
b. Exploration of the benefits of adding temporal information.
c. Identification of a deep learning multi-label video classifier for labeling video frames based on spatiotemporal attention.

The rest of the thesis is divided into five chapters. Related works are described in chapter 2, whereas chapter 3 unveils the methods & materials that we have utilized within this work. Chapters 4 and 5 include the results of our works and ablation study. Finally, we conclude in chapter 6 by leaving some discussion and direction for further research and development in the same area.

Chapter 2

Related Works

In this chapter, we present a comprehensive review of relevant works that have contributed to our methods and materials, either implicitly or explicitly.

2.1 Convolutional Neural Network (CNN)

Computer vision has been used in automating various industries worldwide. While artificial intelligence enables machines to think, computer vision provides them with the ability to see. It has been used in many diverse fields such as agriculture, autonomous vehicle, facial recognition, medical imaging, manufacturing, and many more. Convolutional Neural Network (CNN) is widely recognized as a breakthrough innovation in this area which was introduced in 1998 [6] for handwritten digit recognition tasks from images. CNN can extract spatial information from images including low-level features such as edges, corners, and blobs which help with the recognition and classification tasks. The hierarchical architecture within CNN allows lower layers to learn simple features and higher layers combine these low-level simple features to learn more complex features. As a result, it is able to capture both local and global spatial information from an image which assists in any vision task. Consequently, several groundbreaking innovations [7–9] have been achieved to improve this technology further. Therefore, we can utilize a CNN-based architecture to extract spatial information from video frames to accomplish the automatic underwater video analysis system.

2.2 Vision Transformer (ViT)

Following the immense success of the Self-attention based Transformer [10] in the field of Natural Language Processing (NLP), it has also evolved in a wide range of applications within Computer Vision. Researchers thrived to adapt the self-attention mechanism in the Computer Vision area and introduced the Vision

Transformer [11] in 2020 as the counterpart of the original Transformer. ViT addresses image recognition tasks by dividing an input image into patches and applying self-attention to these patches to obtain spatial contextual relations between them. The self-attention mechanism allows ViT to extract global and high-level features from the input image and focus on different parts of the image simultaneously. Thus, it has been adapted together with traditional CNN architectures for various image recognition tasks [12–14]. The revolution of the ViT has also shifted through different variations to other vision tasks including object detection [15, 16], and image segmentation [17].

We are particularly interested to train a multi-label ViT image classifier on our underwater image dataset because of its underlying self-attention mechanism. This facilitates better spatial feature extraction on frames during video analysis. ViT applies a standard NLP-suited transformer on an image which is first split into fixed-size patches in order to make the fewest possible adjustments. The list of patches is similar to tokens or words of NLP applications which are fed to the transformer network as inputs. This approach is called patch embedding. In order to get positional information, standard 1D position encoding is added along with the input sequence of patches. The rest of the architecture consists of transformer encoder layers where a learnable embedding is prepended to the embedded patches sequence. Nonetheless, ViT has a notable drawback that it requires pre-training on large-scale datasets and subsequently fine-tuning on smaller datasets to attain comparable or superior performance to CNNs for various vision applications. While pre-training, a Multi-layer perceptron (MLP) based classification head is integrated with one hidden layer. The MLP layer is later replaced by one single linear layer during fine-tuning. Recently, a study [18] has shown that ViT can outperform CNN models of similar size when trained on ImageNet from scratch without strong data augmentations which overcome the large-scale pretraining limitation. Therefore, it is apparent that ViT holds promises for the underwater video analysis domain as well.

Since self-attention and ViT are two instrumental parts of our approach, we have provided an in-depth explanation of the mathematical theory and architecture within the Materials & Methods in chapter 3. This will facilitate readers and future students to develop a better understanding of these complex theories and architectures.

2.3 Temporal Action Localization (TAL)

To study video understanding, we need to start with extracting temporal information from the frames of a video. Temporal Action Localization (TAL) [5] refers to determining the time intervals in a video that contains a target action. The target action is usually a dynamic activity (e.g., marine plant waving, fish swimming) but can be a stationary fact as well such as *anode* in the ship hull. Besides, there

can be two types of motion in the videos; object or camera motion. As ROV is moving, camera motion is present in our case. TAL mainly performs two tasks; recognition and localization. Recognition denotes the detection of the class labels whereas localization determines the start and end time of the detected actions. The latter does not apply to our work at the moment as we only focus on multi-label class recognition.

Generally, there are two types of TAL methods: single-stage and two-stage; single-stage: generates several temporal action segments (start to end) proposals in an untrimmed long video and classifies these actions simultaneously, two-stage: first proposes segments and classifies actions and then regresses the boundaries. In addition, there are a couple more variations depending on the data annotations;

- **Fully-Supervised Temporal Action Localization (F-TAL):** It refers to the training when the dataset contains both the video-level category classes and the temporal annotations (start and end time) of the action segments.
- **Weakly-Supervised Temporal Action Localization (W-TAL):** Unlike F-TAL, W-TAL tackles the challenge of localizing action boundaries in untrimmed videos where only video-level category labels are available [19]. In this case, obtaining precise temporal annotations is difficult due to the presence of numerous irrelevant frames that are not related to the target actions. This lack of detailed temporal information makes the task of temporal action localization even more harder in the weakly-supervised setting.

W-TAL indeed coincides with our case as we have only untrimmed underwater videos without annotations. However, the implementation of video classification requires video annotation. This needs extensive time to prepare the data for training a deep learning video classifier. Hence, we follow a similar W-TAL approach to train our multi-label video classifier.

In contrast to TAL, which focuses on classifying dynamic actions in video snippets and localizing their boundaries, our approach deals with classifying static objects within individual frames by extracting spatiotemporal features from adjacent frames. Our method provides prediction confidence for each frame index in the temporal dimension of the video to facilitate automatic video analysis. We aim to demonstrate the distinct nature of our work from TAL by discussing TAL and highlighting these differences here.

2.4 Spatiotemporal features in Video Classification

In video understanding, the improved Dense Trajectories (iDT) proposed in [20] was the state-of-the-art hand-crafted feature for classification tasks. The iDT descriptor demonstrates the ability to extract temporal features differently from that spatial information. Consequently, 3D ConvNets was proposed in [21] to learn spatiotemporal features from videos. It also overcomes the limitation of 2D ConvNets which

loses temporal information of the input signal right after every convolution operation. The best architecture proposed in their experiment, called C3D net, is homogeneous and comprises 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. The 3D convolution kernels in this network are 3x3x3 with a stride of 1 in both spatial and temporal dimensions. They also claimed that a trained C3D network can serve as a potential spatiotemporal feature extractor for other video analysis tasks which could be advantageous in our scenario.

TimeSformer [22] is among the first video models to incorporate self-attention mechanisms in video understanding inspired by the success of self-attention mechanisms in ViT. It utilizes self-attention over both spatial and temporal dimensions of an input video sequence rather than using 3D CNN to extract temporal features along the frames. The model takes an input snippet consisting of 8 RGB frames of size 224x224, decomposes each frame into 16x16 patches, and applies self-attention along the temporal patches for these 8 consecutive frames. During inference, it uses 3 spatial crops from the temporal clip and predicts by averaging the scores. In contrast to our approach of using consecutive frames to predict static class labels in the current frame, TimeSformer samples the 8 frames of an input video at a rate of 1/32, and these frames are not necessarily consecutive. Their experiments have demonstrated that the best performance is achieved when temporal and spatial attention are applied separately. Adopting this approach could be crucial in training our model video classifier.

ViViT [23] is another example of a transformer-based video classification model that benefits from the self-attention mechanism. They propose four variations of their model by factorizing the spatial and temporal dimensions in different ways, ranging from simple to complex architectures. They also explain how to utilize pre-trained ViT image models to train a video classifier on small datasets along with effective regularization techniques which could be particularly advantageous for our purposes. They emphasize the operational flexibility of a variable number of input frames which is similar to the original transformer's ability to handle any sequence of input tokens. While there are similarities with TimeSformer [22], the rich ablation study presented in ViViT provides a strong foundation for us to begin with our own video model.

In essence, the spatiotemporal feature extraction strategies in video models based on 3D CNN or transformers provide a promising research direction for developing a suitable multi-label video classifier for underwater ship inspection. Although the underlying architecture of our model will fall into a similar sort of these models, we will emphasize improving our model to serve a different purpose. Our model will predict static classes instead of dynamic actions by absorbing the disrupted motions in the video and will stabilize the prediction confidence along the temporal dimension.

2.5 Image-to-Video Transfer Learning

The authors of the paper [24] have created a spatiotemporal bottleneck adapter model, which can be integrated into a base architecture. As a result, it is only necessary to train the parameters of the adapter model during fine-tuning. Specifically, the base architecture refers to a Video Understanding model which can adapt a large pre-trained image model into its bottleneck. A pre-trained image model lacks the necessary capacity to infer temporal structured information, which is a crucial aspect of video understanding. Despite this limitation, state-of-the-art video models are usually constructed to learn the temporal dimension utilizing existing image models. Consequently, the proposed ST-Adapter can extract and utilize the existing knowledge of a large image model, resulting in improved video understanding while minimizing the number of parameters to be updated during training. This method is known as a parameter-efficient image-to-video transfer learning approach.

According to the authors, existing efficient tuning methods fall broadly into three categories and those are stated below in their exact language;

- **Task-specific adapter:** An adapter consists of lightweight modules inserted between layers of a pre-trained model. To be parameter-efficient, only those newly added adapter modules need to be updated during task fine-tuning, whilst all the parameters of the large pre-trained model, which takes the majority proportion of the whole solution, are frozen.
- **Prompt tuning:** Instead of manipulating the network architecture, these methods prepend a set of learnable tokens at the input point of the model or intermediate layers. Similarly, only these added tokens need to be optimized for each downstream task.
- **Learning weight approximation:** In particular, only the low-rank matrices for approximating the weights need to be updated during training.

However, this paper moves a step further to consider the more challenging adaptation problem from a pre-trained image model without temporal knowledge to video understanding tasks. The authors chose two representative video ViT models, TimeSformer and XViT [25] in their performance benchmark. The adapter module is specifically composed of a down-projection linear layer followed by a spatiotemporal operator using a standard depth-wise 3D convolution layer, a non-linear activation function, and an up-projection linear layer.

Comprehending the insight of image-to-video transfer learning can assist us in integrating the parameter-efficient training strategy into our video model. Even though our approach is targeted to be a post-processing video analysis technique, effective training can serve as a guide in adapting the model in real-time for future work.

Chapter 3

Materials & Methods

This chapter provides a meticulous explanation of all the methods and materials employed to achieve our objective. Since our contributions are divided into three phases, we have included methodologies that sequentially address all the phases, allowing us to gradually construct our final video model.

3.1 Datasets

Starting with a description of the datasets we are using to develop our methods provides a ground understanding of the platform we are building upon. Since our objective is to improve the frame-based video classification and then propose a multi-label video model, the following two subsections provide descriptions of the respective image and video datasets.

3.1.1 Image dataset

The image dataset for underwater ship Lifecycle Inspection, Analysis, and Condition Information (LIACI) is publicly available, published in [26], and hence called the LIACI dataset. The dataset comprises 1893 RGB images extracted from 17 inspection videos of various ships. There are 10 class labels as depicted in Fig. 3.1 divided into two different categories;

- **Ship components:** *Anode*, *Bilge keel*, *Overboard valve*, *Propeller*, *Sea chest grating*, and *Ship hull*.
- **Common marine coating issues:** *Marine growth*, *Paint peel*, *Corrosion*, and *Defect*.

The latter category classes are more challenging to classify and differentiate from each other as they mostly evolve together. The images were annotated pixel-wise for semantic segmentation which is also suitable for multi-label classification. Table 3.1 presents the number of images extracted from the number of different

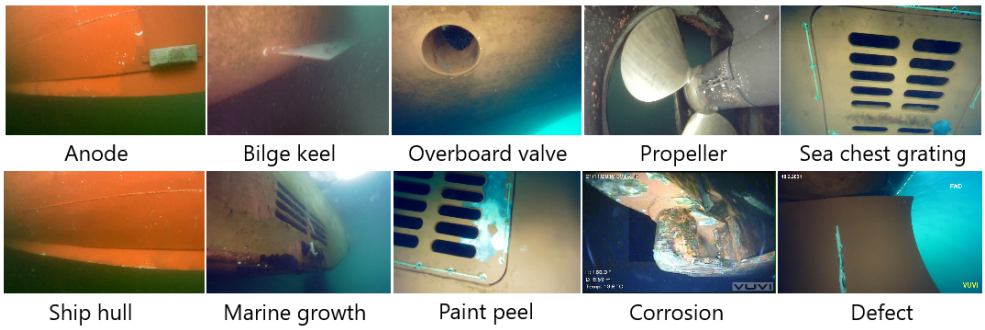


Figure 3.1: Visualization of 10 class labels of two different categories in LIACI dataset.

ships alongside their resolution. On the other hand, Fig. 3.2 represents the number of instances per class label.

Table 3.1: Number of images and their resolutions from different ship inspection videos

Image resolution	Image count	Number of different ships
640 x 480	284	5
1920 x 1080	725	4
1280 x 720	837	7
1920 x 1080	47	1

However, we exclude the *Ship hull* class during the training of our deep learning model as it is present in all images. We only used 1561 images from the LIACI dataset to train and test our model as recommended by the authors [26]. The remaining 332 images were considered too spatially similar to other images in the dataset (Cosine similarity cut-off of 0.90). The class instance distribution of the remaining images in our dataset in Fig. 3.3 indicates that while the dataset is not perfectly balanced, it is not severely imbalanced either.

3.1.2 Video dataset

We hardly find any relevant off-the-shelf video dataset that meets the requirement to train our video model for our domain. Therefore, we need to create and annotate our own dataset to train and evaluate our proposed model. Since generating and annotating a video dataset requires a significant amount of hard work and time, we follow a weakly supervised approach as mentioned in the related works.

We have acquired the corresponding videos of LIACI training images which are untrimmed and unstructured video data. We were able to extract 755 corresponding video snippets out of 1893 images contained in the dataset. Each snippet consisted of seven consecutive frames, with the middle frame representing the

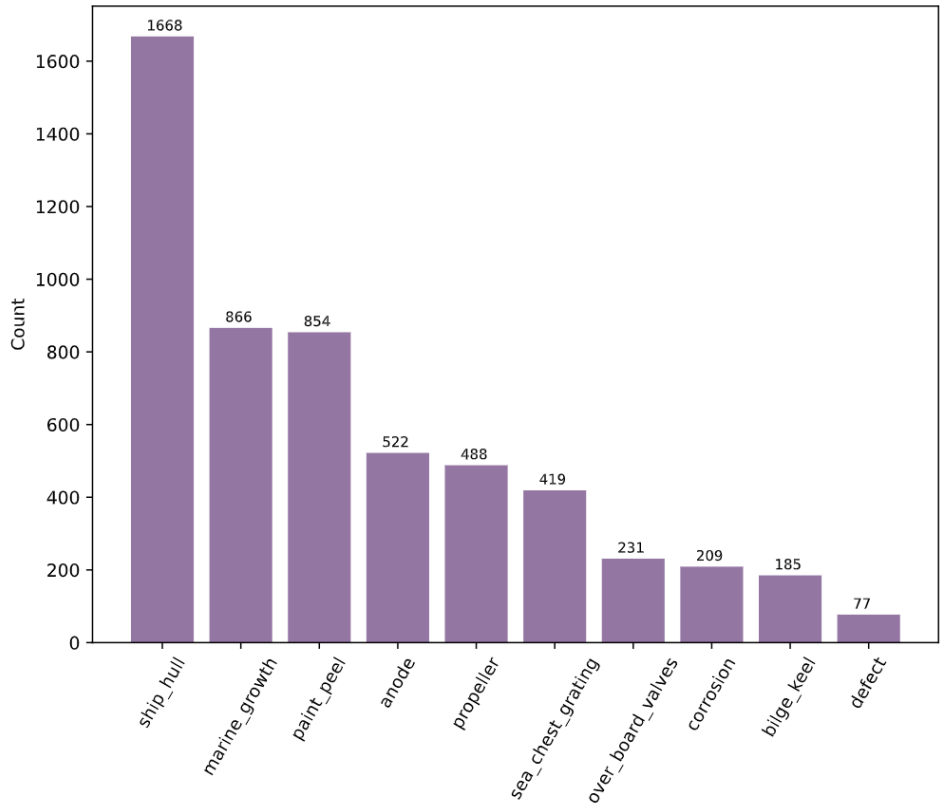


Figure 3.2: Number of class instances per class labels in LIACI image dataset.

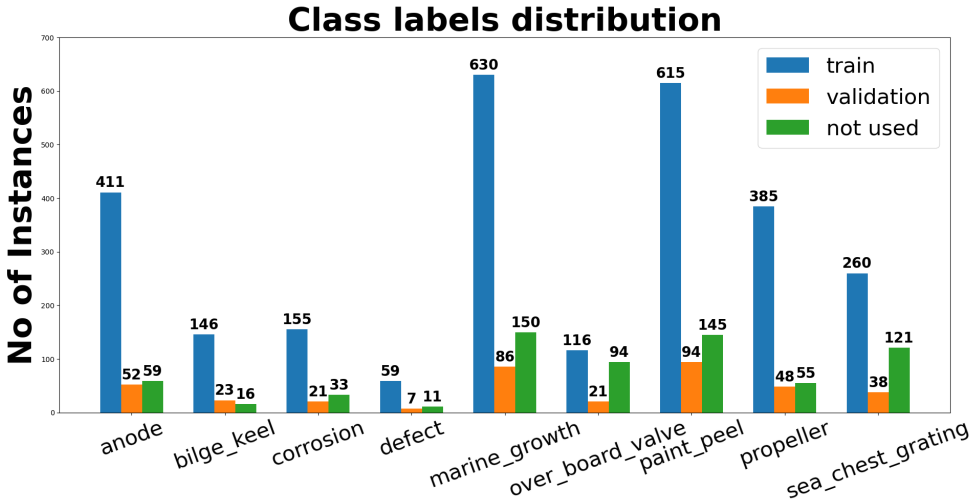


Figure 3.3: Distribution of class instances in LIACI image dataset.

original image from the LIACI dataset. The class labels of the middle frame are also considered the labels for the entire snippet during training. This approach may be considered a weakly supervised data annotation. The snippets were split into 584 for training, 87 for validation, and 84 that were not used by following the same splitting convention of the image dataset. The class instance distribution of this generated video dataset is reported in Fig. 3.4. It is worth noting that the generated video dataset contains fewer snippets than half of the number of images in the LIACI dataset. As a result, it may not be sufficient to train a robust video video model compared to the image model.

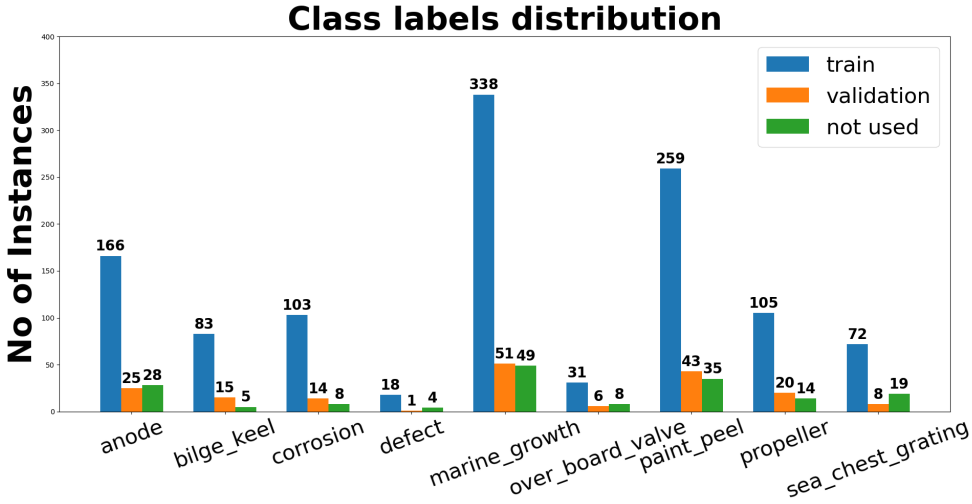


Figure 3.4: Distribution of class instances in generated video dataset.

Furthermore, to comprehensively analyze and evaluate the inspection performance of trained models, we have selected 8 key clips of 1920x1080 resolution from an untrimmed underwater inspection video. These clips were chosen randomly and each clip is approximately 14 seconds long. Table 3.2 provides descriptions of the physical content of the clips that are easily recognizable to human eyes. However, distinguishing between *marine_growth*, *corrosion*, and *paint_peel* with human visual perception can be quite challenging most of the time. The results of the analysis and evaluation are documented in chapter 4 and 5.

3.2 Self-Attention Mechanism

Self-attention [10] is an attention mechanism that finds contextual relationships between different positions of an input sequence to create a coherent understanding of the entire sequence. An input sequence, for instance, can be a sentence consisting of a sequence of words. Specifically, self-attention computes scores to determine the level of attention each word should have towards the other words in the sentence. These words are technically referred to as input embeddings or

Table 3.2: Ground truth physical contents of the randomly selected 8 key video clips.

Serial	Major physical real contents
1	anode, paint_peel
2	bilge_keel, paint_peel, over_board_valve, anode
3	propeller, paint_peel, corrosion, marine_growth
4	paint_peel, marine_growth, propeller
5	marine_growth, propeller, corrosion
6	paint_peel
7	propeller, marine_growth
8	sea_chest_grating, paint_peel, corrosion

vectors within the input sequence. To complete the self-attention mechanism, the Scaled Dot-Product Attention function is utilized to calculate the attention outputs.

3.2.1 Scaled Dot-Product Attention

The Scaled Dot-Product Attention function takes a query and a set of key-value pairs and maps them to an attention output. Both the inputs and the output are in vector forms. Basically, the output is the weighted sum of the value vectors, where the weights correspond to the attention scores, which are calculated as the scaled dot product of the query and keys. In practice, the computation is performed simultaneously on a set of queries that are stacked together in a matrix $Q \in \mathbb{R}^{N\times D}$. N is the number of input vectors and D represents the dimension of each vector. This allows for efficient parallel computing, which can significantly speed up the calculation process. Assuming that the keys and values are also stacked in matrices $K \in \mathbb{R}^{N\times D}$ and $V \in \mathbb{R}^{N\times D}$ respectively, the resulting attention output matrix $\in \mathbb{R}^{N\times D}$ can be represented as follows:

$$\textbf{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \textbf{softmax}(\frac{\mathbf{QK^T}}{\sqrt{\mathbf{D}}})\mathbf{V} \tag{3.1}$$

The softmax function is applied to obtain the weight distribution across the values, ensuring the sum of the weights is up to 1. However, in our methodology, we will also require to compute attention for a single query, which is similar to the equation mentioned above and is represented as:

$$\textbf{Attention}(\mathbf{q},\mathbf{K},\mathbf{V}) = \textbf{softmax}(\frac{\mathbf{qK^T}}{\sqrt{\mathbf{D}}})\mathbf{V} \tag{3.2}$$

where $q \in \mathbb{R}^{1\times D}$ refers to a single query vector. To ensure that the attention output remains in the shape of $\mathbb{R}^{N\times D}$, we need to replace the last matrix multiplication operation with element-wise multiplication. Fig. 3.5 illustrates a visual representation comparing the normal Dot-Product Attention with the computation for a

single query. Please also refer to the appendix B.1 for the corresponding code to get the intuition of the attention computation.

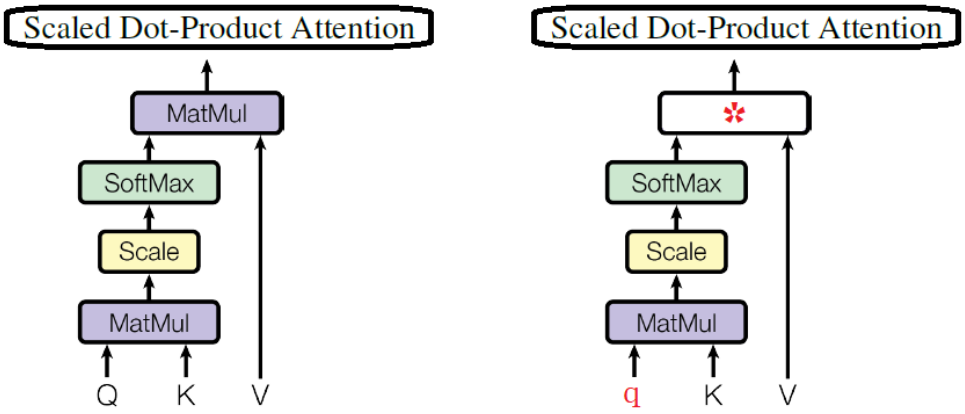


Figure 3.5: (left) General Scaled Dot-Product Attention where the attention matrix after the softmax is multiplied with the value matrix. (right) Scaled Dot-Product Attention for a single query where the MatMul is replaced by element-wise multiplication (*) as the attention is a vector of the single query.

3.2.2 Multi-Head Attention

Another pivotal aspect of the self-attention mechanism is the utilization of Multi-headed Self-Attention (MSA). In MSA, the queries, keys, and values are linearly projected h times using separate learnable parameters. This process results in h different sets of queries, keys, and values to facilitate parallel computations of the attention function. The dimension d of the projected queries, keys, and values can vary depending on individual preference but is typically set to D/h to maintain a consistent computational cost as that of a single-head ($h = 1$) attention with full model dimension D. This choice ensures that the total computational complexity remains stable across different numbers of attention heads. Consequently, D in the attention equations above is replaced by d during multi-headed self-attention computation. The yielded h sets of attention values from the multiple heads are concatenated and further projected to obtain the final set of values. This additional projection step helps to consolidate and refine the information extracted from the attention heads, resulting in the final output values. Fig. 3.6 illustrates the computations of the MSA as represented by the following equations:

$$\mathbf{MultiHead(Q,K,V) = Concat(head_1, head_2, ..., head_h)W^O} \tag{3.3}$$

$$\mathbf{head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)}$$

where the projected weight matrices $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{D\times d}$ and $W^O \in \mathbb{R}^{hd\times D}$. Also note that, for a single query attention computation, Q and W_i^Q will be replaced

by $q \in \mathbb{R}^{1\times D}$ and W_i^q respectively. The code for computing multi-head attention is attached in appendix B.2.

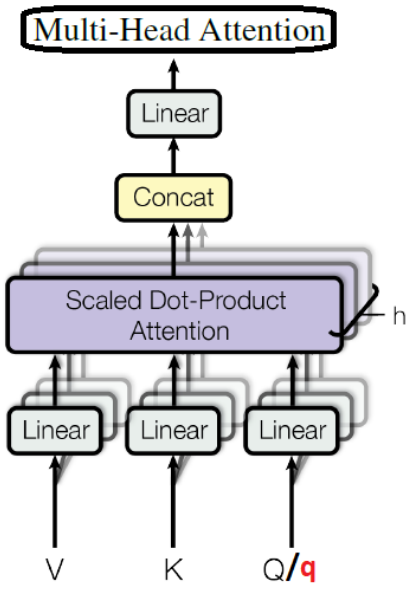


Figure 3.6: Multi-headed Self-Attention Computation process. Q matrix will be replaced by a vector q for the single query attention.

N.B. The diagrams of Fig. 3.5 and 3.6 are borrowed from the original paper [10] and modified to align with our approach and explanation.

3.3 Transformers

Transformers play pivotal roles as key components in our approaches. We have specifically demonstrated the overview of the Vision Transformer in the related works chapter. This section delves into the fundamental architecture of the transformer and highlights our approach to adapting it within our work. To achieve our objectives, we need to implement both image and video classification models. Consequently, the contents of the following two subsections will be integrated into our models.

3.3.1 Spatial Transformer

The Vision Transformer (ViT) serves as our spatial transformer as it possesses the ability to extract spatial information from images. We leverage the ViT as our multi-label image model as well as the spatial encoder part in our video model. Following the brief description in the related works, Fig. 3.7, building upon the

original diagram presented in [11], provides an overview of the ViT architecture, specifically focusing on the modifications made in our approach while retaining its simplicity. This diagram aims to highlight the key adaptations and modifications we have incorporated to enhance the understanding of our methodology.

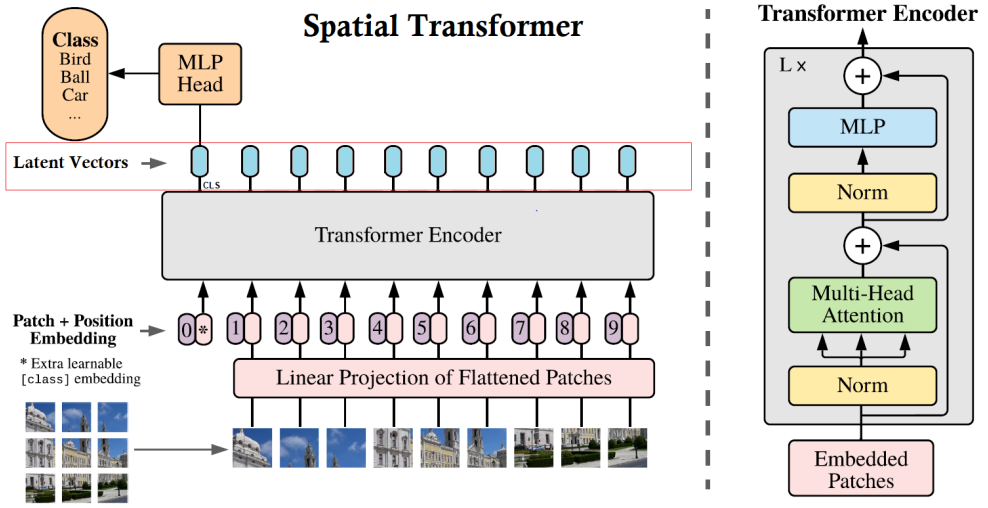


Figure 3.7: The standard ViT architecture as our Spatial Transformer. The highlighted latent vectors will be utilized in our video model architecture.

In the initial step, the input image undergoes non-overlapping convolutional operations to be split into multiple patches. Consequently, the patch embedding which represents the flattened patches is trainable using various convolutional kernels. The dimension of the patch embedding is determined by the number of kernels employed. This process effectively accomplishes the "Linear Projection of Flattened Patches" as depicted in the diagram. Practically, the input image size for our Spatial Transformer is set to $(H \times W) = (224 \times 224)$. We use the kernel size $k = (16 \times 16)$ and the stride size is also set to the same size as the kernel to ensure non-overlapping patches. We employ a total of 768 kernels, which corresponds to the model dimension D as mentioned in the self-attention section. The number of flattened patches we get after the convolution is $N = (224 \times 224) \circledast (16 \times 16) = (14 \times 14) = 196$. Afterward, an additional learnable classification token of the same shape is prepended making it a total of 197 patches. Now, to provide positional information into the model, **a learnable 1D positional vector** of the same length as the patch embedding is added to each patch. This operation does not alter the shape of the embeddings but rather modifies their values, effectively encoding the positional information into the patches. By adding the positional vector, the model becomes aware of the spatial arrangement of the patches within the image. Finally, these patch embeddings are fed into the transformer encoder which encompasses the self-attention mechanism discussed in the previous section. Apart from the multi-head atten-

tion, the architecture also employs layer normalization, residual connection, and multi-layer perceptron (MPL). This combination enables the model to analyze relationships between the patches and capture contextual information for feature extraction. In our spatial transformer, we use a total of $L = 12$ encoder layers and $h = 12$ in MSA to obtain the latent vectors for each of the input patch embeddings. Among the 197 latent vectors generated, only the additional classification token (CLS) is connected to an MLP head for the final classification prediction. This CLS token serves as an aggregate representation of the entire image and is specifically used for making the classification decision. The standard implementation of Vision Transformer (Spatial Transformer) is available in PyTorch and attached in appendix B.3 for the sake of completeness.

However, we have specifically highlighted the latent vectors in Fig. 3.7 because we intend to remove the MLP head and utilize these vectors in our video models. Apart from using the CLS token for final classification, we can leverage the information contained in the latent vectors for further analysis and processing within our video models.

3.3.2 Temporal Transformer

In implementing our temporal transformer, we follow the design principles outlined in the original transformer [10], despite the fact that the underlying architecture remains similar to the spatial transformer. This will ensure leveraging the proven effectiveness of the transformer architecture for temporal processing in our video models.

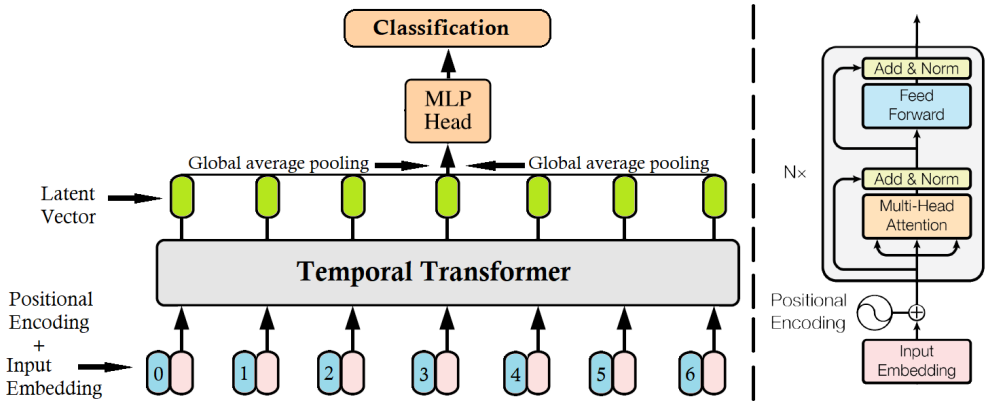


Figure 3.8: Temporal Transformer architecture with no classification token embedding. A fixed sine-cosine function is used for positional encoding.

We describe the key differences of the temporal transformer through Fig. 3.8. In contrast to the spatial transformer, the temporal transformer takes ready-made embedding vectors as inputs instead of image patches. There are two ways to ob-

tain these vectors from the spatial transformer. Firstly, we can directly extract the CLS latent vector, which serves as a latent representation for the entire image. Alternatively, we can compute the global average of all the latent vectors, resulting in a single vector that captures an aggregated representation of the image. This implies that a single image input to the spatial transformer generates one input embedding for the temporal transformer. However, in order to execute the temporal transformer, we need a total of seven such input embeddings. This indicates that we process a sequence of seven consecutive frames to capture the temporal aspects and analyze the relationships between them through temporal attention.

A significant distinction in our approach is the absence of the extra classification token. Instead, we depend solely on the sequence of input embeddings to capture temporal attention and make predictions in our video models. Additionally, we utilize **a fixed sine and cosine function for positional encoding**, following the original transformer design. The implementation of the positional encoding is attached in appendix B.4. This fixed encoding scheme ensures consistent and reliable positional information throughout the sequence, without the need for learnable parameters as in spatial transformer. We also make adjustments to the number of heads ($h = 8$) in the MSA component of the Temporal transformer. Additionally, through empirical analysis, we determined that a number of encoder layers ($L = 4$) yielded the best results, as documented in the ablation chapter. Consequently, we maintain $L = 4$ for all of our video models, based on these findings. Finally, the global average pooling is used to aggregate the temporal attention into a single latent vector passed through the MLP head for the final prediction. We have implemented the temporal transformer differently from the spatial transformer. In appendix B.5, you will find the code for the temporal transformer, which incorporates the attention mechanism, multi-headed self-attention (MSA), and positional encoding that were discussed earlier. This code provides a detailed implementation of the temporal transformer architecture used in our methodology.

3.4 Multi-label Image Classifier

To initiate our work, we reproduce the multi-label image model that was previously trained as part of the LIACI project, as mentioned in the introduction chapter. This serves as our starting point, providing us with a baseline model to build upon and gradually improve the performance. By reproducing the existing model, we establish a solid foundation for our subsequent improvements and experiments.

3.4.1 ResNet Model

To perform frame-wise video analysis, a ResNet [27] model based on CNN architecture was trained using Microsoft custom vision [3] on the LIACI image dataset as a multi-label image classifier. We were given access to the private repository of

the project to reproduce and analyze the model's performance. We have imported the project along with the trained model, added a new script which is included in appendix B.6, and modified some of the scripts. This new script allows us to reproduce the corresponding multi-label classification on our selected key video snippets, as presented in table 3.2. It also generates one CSV file containing the prediction probabilities for all the class labels at each individual frame and draws a diagram for visual interpretation. Fig. 3.9 represents the visual interpretation of the first snippet in the table. It shows that the *paint_peel* class severely appears in the last half of the clip. Besides, *anode* and *propeller* class labels are more or less visible throughout the entire clip. The rest of the 7 visual performance interpretations are added in appendix A.1.

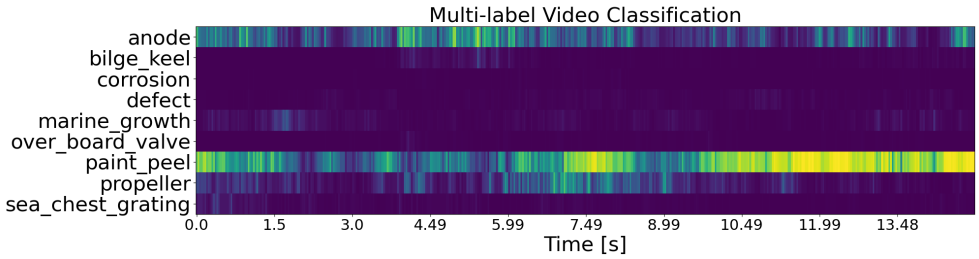


Figure 3.9: Visual interpretation for multi-label classification of snippet no. 1 in table 3.2 using the ResNet classifier.

3.4.2 ViT Model

To enhance the multi-label classification for LIACI video analysis at the frame level, we propose to integrate a pretrained ViT model. The advantages of the self-attention mechanism within ViT offer a promising approach to improve the accuracy and performance of multi-label classification in video analysis within the context of the LIACI project. Importantly, we aim to leverage the ViT architecture as the basis for our attention-based video model, which will allow us to efficiently transfer the knowledge learned from the image model to the video model.

In the original ViT paper [11], a few variants of the architecture were proposed that differ in model size and input patch size. For instance, the ViT-L/16 refers to the "Large" variant and is composed of 24 training layers with a 16x16 input patch size. The PyTorch [28] vision package includes several ViT models as shown in table 3.3 that can be easily implemented. Besides, PyTorch enables access to the models' underlying architecture and allows us to modify them through retraining or fine-tuning conveniently. Based on the model's capacity, our requirements, and computing resources we have selected the ViT-B/16 architecture. The size of the model is 330.3MB with 86M trainable parameters and it has 95.318%@5 accuracy on ImageNet 1K dataset [29]. Besides, it resembles the architecture we have stated as our spatial encoder in the previous section.

Table 3.3: Available pretrained ViT architectures in PyTorch.

	Model	**img size**	**patch size**	**Layers**	**Dim**	**MLP size**	**Heads**	**Params**	**acc@5** (ImageNet1k)
1	ViT_B/16	224	16	12	768	3072	12	86M	95.318
2	ViT_B/32	224	32	12	768	3072	12	88M	92.466
3	ViT_L/16	224	16	24	1024	4096	16	304M	94.368
4	ViT_L/32	224	32	24	1024	4096	16	306M	93.07
5	ViT_H/14	224	14	32	1280	5120	16	633M	98.694

We have decided to train two versions of the selected ViT architecture on the LIACI dataset using PyTorch. One version will be pretrained on the ImageNet 1k dataset, while the other will be pretrained on the COCO 2014 dataset [30]. The goal is to compare the performances of these two models and assess the influence of different pretraining datasets on the ViT's performance with the LIACI data. Although the weights of the ImageNet pre-trained ViT are available in PyTorch, we need to train the COCO version by ourselves in advance. Consequently, we downloaded the COCO dataset using FiftyOne [31] and conducted a complete fine-tuning of a pre-trained ViT model that was originally trained on ImageNet. The COCO dataset consists of 82,783 training images and 40,504 validation images. We trained the model on this extensive dataset for a total of 94 epochs. Detailed information regarding the hyperparameters and data transformations used for training the ViT on the COCO dataset can be found in table 3.4.

Table 3.4: Hyperparameters and data transformations to train ViT on COCO dataset.

Hyperparameters	**Transformation**
BCEWithLogitsLoss	Image Resize(224x224)
Optimizer: SGD	
Learning rate: 0.001	Normalization: Mean[0.485, 0.456, 0.406]
Momentum: 0.9	Normalization: Std [0.229, 0.224, 0.225]
Batch size: 16	
Epochs: 94	RandomHorizontalFlip(only on training set)
Scheduler:StepLR	
(step=20,gamma=0.1)	

Finally, we have trained our two desired ViT models pre-trained from ImageNet and COCO datasets and abbreviated them as IMAGENET_ViT and COCO_ViT respectively. Extensive analysis of various training hyperparameters has been conducted during our training procedures. The findings and outcomes of this ana-

lysis are documented and discussed in detail in the ablation study chapter. The final training hyperparameters and data transformations for both models are kept identical and can be found in table 3.5. The data transformations employed for the COCO dataset differ significantly due to the distinct characteristics of this dataset compared to LIACI. LIACI consists of underwater images captured with ROVs, which introduces motion blur. We applied separate image normalization by computing the corresponding mean and standard deviation on LIACI and COCO datasets. It is noted that only the **Image Resize** and **Normalization** are applied during validation or evaluation. Data augmentation such as AugMix [32] and GuassianBlur are applied randomly with a probability of 0.5. Nonetheless, we investigated various hyperparameters and data augmentations that are exhibited in the ablation study chapter 5. The outcome and the model performance are reported in the results chapter.

Table 3.5: Training hyperparameters and data transformations for IMAGENET_ViT and COCO_ViT.

Hyperparameters	**Transformation**
BCEWithLogitsLoss	Image Resize(224x224)
Optimizer: SGD	RandomHorizontalFlip(p=0.5)
Learning rate: 0.001	Normalization: Mean[0.3485, 0.3699, 0.3520]
Momentum: 0.9	Normalization: Std [0.2495, 0.2446, 0.2062]
Batch size: 16	
Epochs: 100	GaussianBlur(p=0.5)
Scheduler:ReduceLROnPlateau (mode="min", factor=0.1)	AugMix() (p=0.5)[32]

3.5 Multi-label Video Classifier

In the development of our multi-label video models, our primary focus is to leverage the pretrained image models that have been trained on the LIACI dataset. This approach aligns with the image-to-video transfer learning concept that we discussed in the related works chapter. By utilizing the pretrained image models, we aim to transfer their learned knowledge to the video domain. This will enable us to achieve faster training of the video models and dedicate more time to experiments. We intend to implement multiple video models for our ablation study, following three main approaches outlined in the following three subsections.

3.5.1 Naive Video Transformer

Initially, we adopted a straightforward method by utilizing two different spatiotemporal token embedding techniques proposed in [23].

Uniform frame sampling: It uniformly embeds each 2D frame of the video snippet using the same method as ViT and then concatenates together. For instance, if a frame is tokenized into 196 patch embeddings, we will have a total of 1372 embeddings for a single snippet from our video dataset, considering 7 frames in the snippet. In this method, the temporal information is not fused within the embeddings themselves. Instead, it will be incorporated and processed by the transformer architecture at a later stage.

Tubelet embedding: This approach expands the embedding of ViT to 3D, aligning with the concept of 3D Convolution. It extracts non-overlapping spatiotemporal tubes from the video snippet along the temporal, height, and width dimensions. Consequently, these spatiotemporal tubes are linearly projected to generate the final embeddings. This intuitive approach effectively integrates the spatiotemporal information into the embedding representation before being processed by the transformer model.

Now, the naive approach is to extract tokens from the video snippets using either uniform frame sampling or tubelet embedding methods, and then feed these tokens directly into a trained image ViT model. The process is illustrated in Fig. 3.10, and the diagrams used are prepared by combining resources from [23] and [11]. In practice, to implement the model with uniform frame sampling, we extracted 28 patches with dimensions of 32x56 from each frame of a seven-frame input snippet, generating a total of 196 patch embeddings. These embeddings are readily compatible with a base ViT architecture. On the other hand, to achieve the tubelet embedding as depicted in Fig.3.10, we utilized a pretrained 3D ResNet18 model to extract C3D features from the input snippet as explained in the related works chapter. The results and analysis are reported in the corresponding chapters.

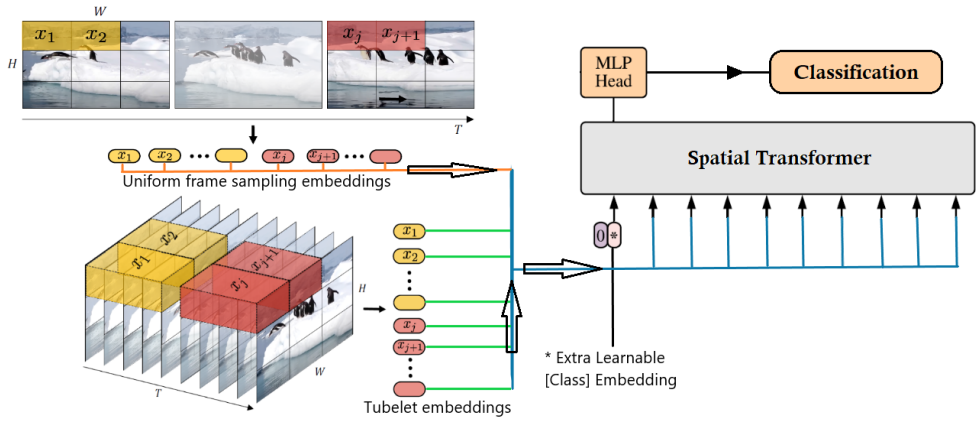


Figure 3.10: A simple approach to video model using the same architecture as the image classifier. Spatiotemporal feature extraction is applied by either uniform frame sampling or tubelet embedding.

3.5.2 Late-Fusion Spatiotemporal Transformer

Our second approach for the video classification model involves a two-stage process. In the first stage, a spatial transformer is exploited to extract spatial features from the seven consecutive frames of the input video snippet. These spatial features are then passed to a temporal transformer in the second stage, which predicts the classes of the snippet. The spatial and temporal transformers, as well as their internal architectures, have been visually demonstrated in Figs 3.7 and 3.8 respectively. Furthermore, Fig 3.11 provides an overview of our second video model approach, which combines these two transformers to create an integrated architecture.

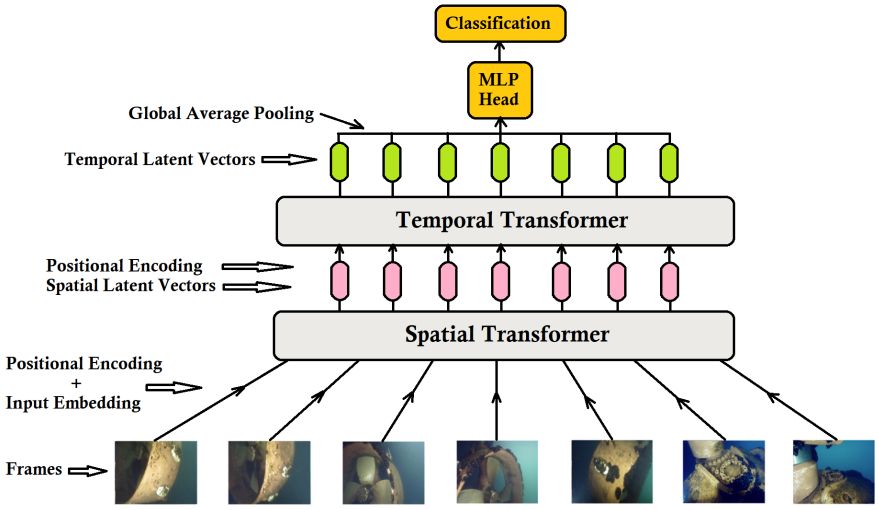


Figure 3.11: Spatiotemporal transformer architecture of the video model. The spatiotemporal feature is extracted separately by spatial and temporal transformers respectively.

We adopted the strategy from Model 2 proposed in [23] which is also similar to the TimeSformer method presented in [22]. Moreover, the architecture provides the flexibility to seamlessly integrate a pretrained image model trained on the LIACI dataset as the spatial transformer component. While we are already aware of the distinct characteristics of the spatial and temporal transformers, there are a couple of modifications that need to be made in order to effectively integrate them into a unified video model. First, we eliminate the MLP head from the spatial transformer and aggregate the spatial features of the image into a single latent vector by following any of the two ways mentioned in the respective section. The second modification involves removing the input embedding from the temporal transformer when attaching it to the spatial transformer. The adjustment is made because the latent vectors generated by the spatial transformer are inherently compatible and align with the shape of the embeddings used by the temporal

transformer. Hence, there is no need for additional input embedding when connecting the spatial and temporal transformers. Since the spatial features are fused with the temporal features at the later stage, it is called a late-fusion spatiotemporal transformer.

This method is designed to address the issue of overfitting on smaller datasets such as ours and provides a more sophisticated model for video classification. A previously trained ViT image classifier is adopted as the spatial transformer encoder, while a new standard transformer is employed as the temporal transformer. During training, we usually freeze the weights of the spatial transformer and solely update the temporal transformer. This approach resulted in a notable acceleration of the training process and facilitated the adaptation of the models to finetuning tasks. We have trained six variants of this architecture that are reported and documented in the results and ablation study chapters.

3.5.3 Attention weighted Spatiotemporal Transformer

Our final approach is also based on the spatiotemporal transformer framework. Nonetheless, our goal is to modify the computation of the multi-head attention in the temporal transformer component. The MSA within the temporal transformer is based on Eq. 3.1 which calculates the global attention by taking the weighted sum of the value vectors across all frames. However, in our LIACI video data, the snippets are labeled based on the classes of the middle frame. Our objective is to predict this middle frame by incorporating temporal attention from the surrounding frames. The MSA attends to all seven frames collectively, including non-relevant information for the middle frame. Hence, this can result in reduced prediction confidence or potentially misclassifying objects within the middle frame. For instance, the middle frame may feature the ship's propeller, which could be obscured in the last frame due to an abrupt movement of the ROV.

Therefore, within this approach, we propose calculating single query attention using Eq. 3.2. In this equation, the embedding of the middle frame serves as the query to attend to all the neighboring frames, including itself. Based on the attention theory, our hypothesis suggests that the attention score for the middle frame is expected to be higher compared to the other frames. The adjacent frames that are more relevant will yield higher attention scores, while less related frames may have lower scores or even zero attention if they do not contribute at all. As the scores are the result of a softmax function, their sum will always equal 1.

Two categories of the approach can be obtained. The first category utilizes the same spatiotemporal transformer architecture but incorporates the single query MSA. The second one employs these attention scores as weights and calculates the weighted average prediction from all seven frames of the spatial transformer, which serves as the final prediction. The latter strategy is adopted from [33] and depicted in Fig. 3.12. In addition, it deviates from the late-fusion architecture in

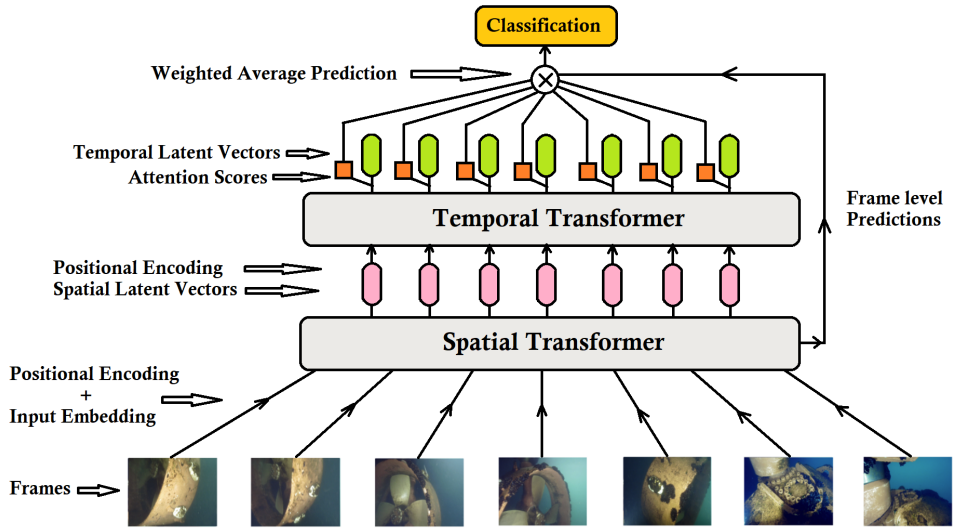


Figure 3.12: Temporal attention weighted spatiotemporal transformer architecture. The spatial features are extracted by the spatial transformer and then attention scores are generated by the temporal transformer. Finally, the predictions of each frame from the spatial transformer are multiplied by their respective attention scores.

terms of the positioning of the MLP head. Specifically, we detach the MLP head from the temporal transformer and reattach it to the spatial transformer. This arrangement enables frame-level predictions, as depicted in the figure. Fig. 3.13 illustrates the input data flow for a batch of two snippets, which provides a clearer understanding of the internal computations within the model. The data flow also indicates that altering the position of the MLP head leads to the learning of attention scores based on our hypothesis, as it directly contributes to the final prediction. Consequently, the optimization process will directly impact the updating of scores during the full finetuning of the model. The outputs and analysis of both variants are documented in the results and ablation study chapters.

3.6 Analyzing Materials

One of our key contributions involves an extensive analysis of multi-label image and video models. To facilitate this analysis, we have implemented a few tools and materials. The following subsections outline the materials we have utilized in order to accomplish our objectives.

3.6.1 Prediction Confidence and Temporal Characteristics

To analyze a trained model's confidence behaviour, we leverage OpenCV [34] to process a video snippet and observe the model's prediction confidence on each

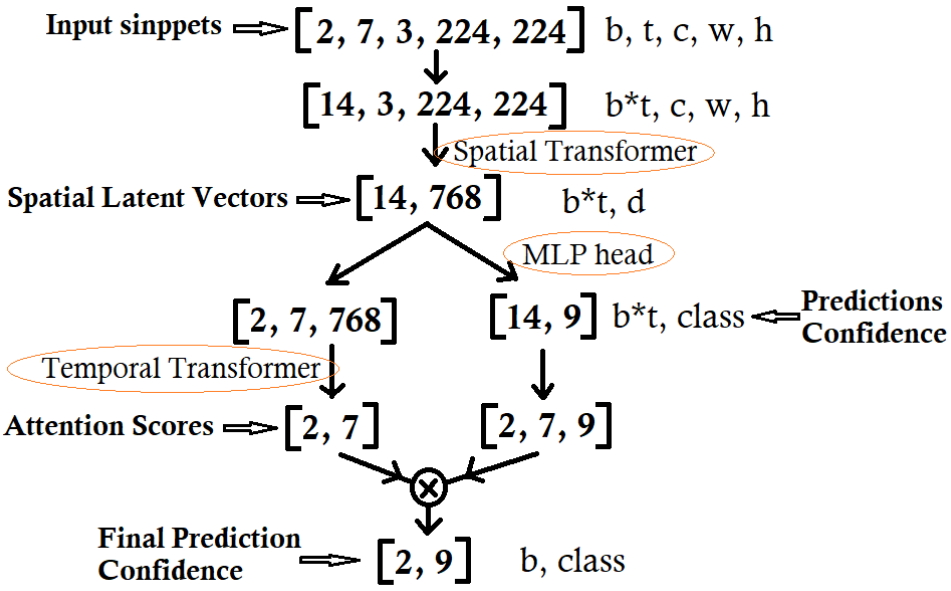


Figure 3.13: Input data flow through the attention-weighted spatiotemporal transformer. Where b, t, c, w, and h refer to batch size, number of frames, number of channels, width, and height respectively.

frame, as illustrated in Fig. 3.14. This approach also enabled us to evaluate a model's ability to predict multiple class labels simultaneously on a per-frame basis.

To integrate temporal reasoning into our model, it is necessary to examine and analyze the model's temporal consistency throughout the development process. To achieve this, we utilize OpenCV to observe the temporal aspect of the model's confidence for different labels during an inspection. An example has been depicted in Fig. 3.15. This is useful to qualitatively assess the temporal stability of a trained model and is reported in the result section. The tool can load a video and conduct the spatial and temporal inspection alongside playing the video.

3.6.2 Underwater Image Quality Metrics

In underwater image or video tasks, measuring image quality is a grave concern as it directly impacts any vision-based operation. Poor-quality images can significantly degrade the performance. To measure frame quality, we employed two separate image quality metrics - UCIQE [35] and UIQM [36] - to establish a correlation between the model's prediction confidence and frame quality. Both metrics are no-reference and meticulously designed for underwater images. One example of inspecting the frame quality using these two metrics is illustrated in Fig. 3.16. This has been useful during our analysis to measure the quality of each frame in a video.

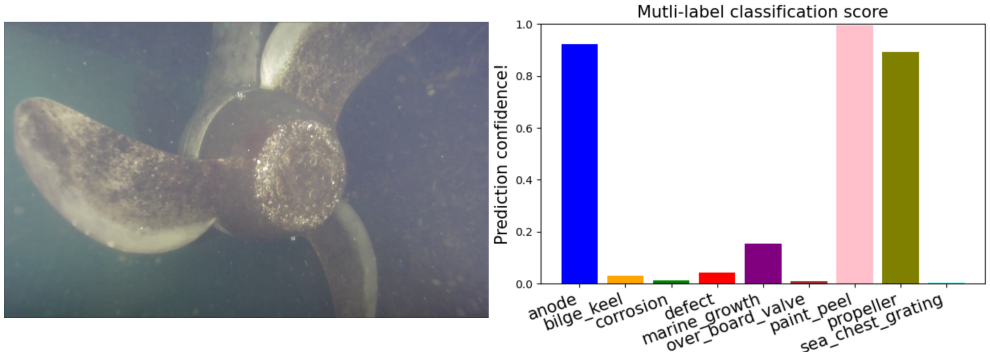


Figure 3.14: Model's multi-label prediction confidence on each frame during a video inspection. It facilitates the frame-wise spatial analysis of the model's confidence.

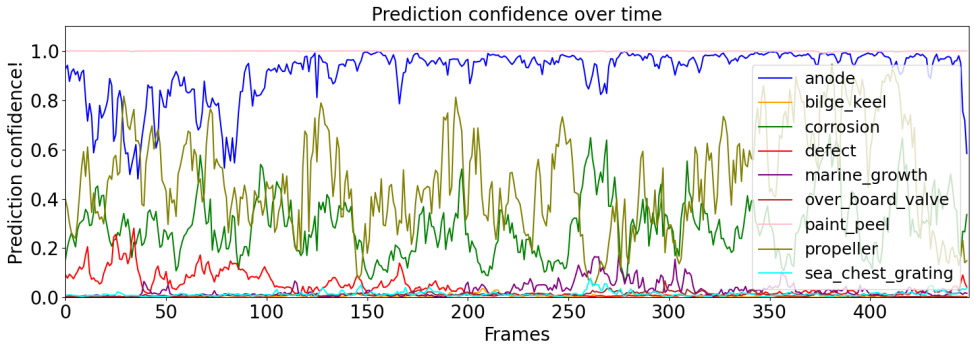


Figure 3.15: An example of the temporal observation of a model's prediction confidence during a video inspection.

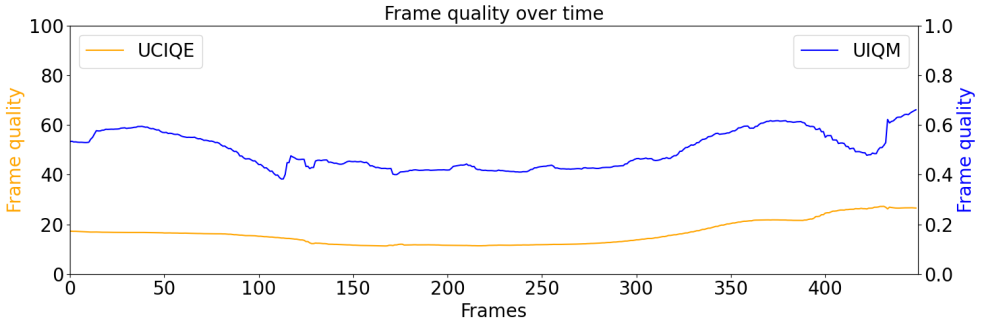


Figure 3.16: An example of an inspection of frame quality based on UCIQE and UIQM metrics in a video snippet.

3.6.3 Multi-label Evaluation Metrics

The computation of multi-label classification evaluation metrics is different from multi-class classification. The Scikit-learn Python package [37] provides essential tools to easily compute different metrics. We report accuracy, precision, recall, and f1-score on the validation set of LIACI data for our image and video models in chapter 4. These metrics are calculated along the instances and averaged over them. The mathematical equations are as follows in Eq. (3.4), (3.5), (3.6), and (3.7) where n is the number of images, y is the ground truth, and $\hat{y}$ is the predicted label. Besides, we computed class-wise evaluation metrics during some analysis in chapter 5.

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|} \tag{3.4}$$

$$Precision = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|\hat{y}_i|} \tag{3.5}$$

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|y_i|} \tag{3.6}$$

$$F1-score = \frac{1}{n}\sum_{i=1}^{n}\frac{2|y_i \cap \hat{y}_i|}{|y_i| + |\hat{y}_i|} \tag{3.7}$$

3.7 Hardware Resources

We used NVIDIA RTX 2080 Ti (11GB) and RTX A6000 (48GB) GPUs to train both of our image and video models. For inference and testing, we used a local system that constitutes of NVIDIA GTX 980 (4GB) with Intel(R) Xeon(R) CPU E5-1650v3 @3.50GHz and 32GB RAM.

Chapter 4

Results

As our contribution focuses on the development of multi-label image and video classifiers for underwater ship inspection video analysis, the results chapter will be divided into two sections. This division will showcase the performances of these classifiers on the LIACI image and video datasets individually. Before delving into the results of our models, it is important to highlight the reproduced limitation of the ResNet classifier that was trained on the LIACI image dataset in a previous study [2]. By doing so, we can gain insights into the improvements made by our image models and reflect upon their significance.

4.1 Multi-label ResNet Classifier

The temporal observation of video snippet no.3 from table 3.2 is illustrated in Fig. 4.1 using the ResNet model. Although the model successfully detects a couple of classes, the confidence values for consecutive frames fluctuate significantly. We noticed similar behaviour for other snippets even though the spatial changes between frames are negligible. The bottom row of Fig. 4.1 displays the output of the two image quality metrics on the same video snippet, while comparing them against a model's temporal prediction confidence.

Due to the different value ranges of UCIQE and UIQM metrics, we have plotted them on two different scales within the same plot. It is clear that UCIQE does not show any correlation with the observed fluctuation. However, the UIQM values indicate a consistent prediction trend, with higher UIQM values observed between frames 250 to 450. In contrast, the highlighted confidence values at frames 70 and 78 exhibit a substantial difference, with values of 0.12 and 0.81, respectively. Surprisingly, this discrepancy occurs despite the negligible spatial difference between these frames, as depicted in Fig. 4.2. Therefore, in the subsequent sections, we present the results and performances of our models to illustrate the extent to which they can gradually address and overcome the aforementioned issue.

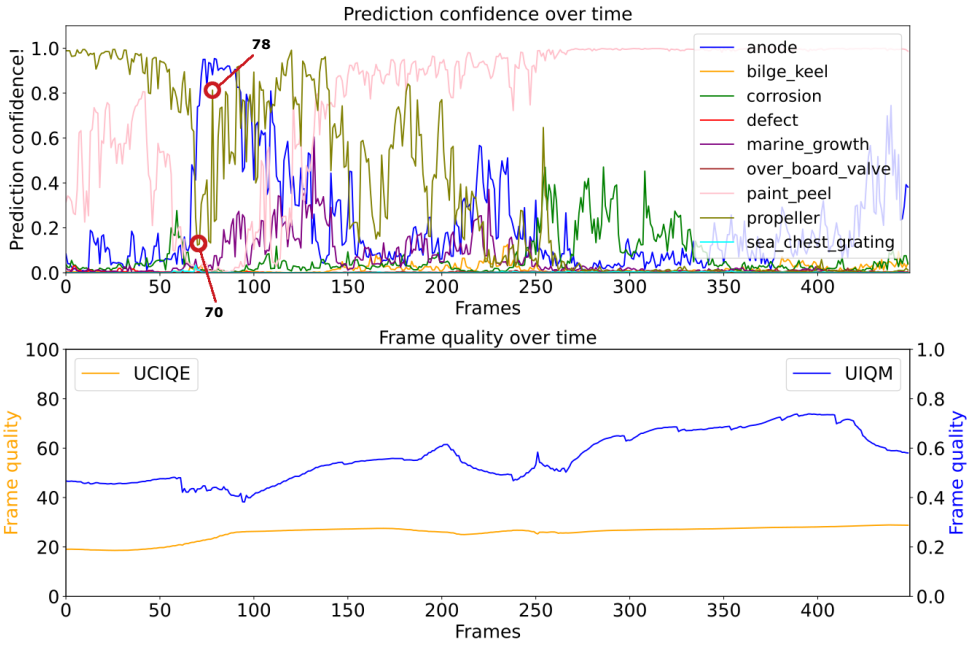


Figure 4.1: Temporal observation with UCIQE and UIQM frame quality metrics on the video snippet no.3 from table 3.2.

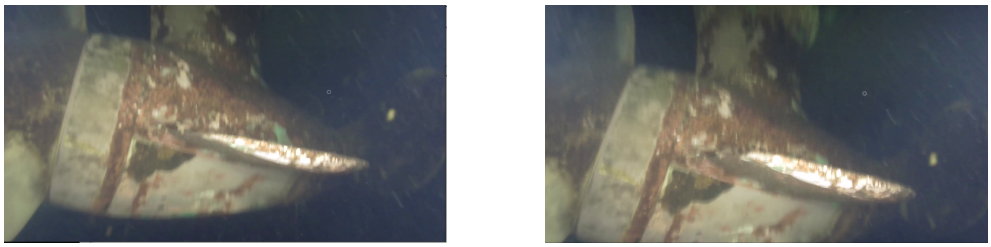


Figure 4.2: Frame 70 and 78 (left to right) of the video snippet no.3 from table 3.2.

4.2 Multi-label Image Classifers

After initiating the training process for our image models, we undertook a thorough analysis involving four different variations from each of COCO_ViT and IMAGENET_ViT to identify the most optimal model. Consequently, we discovered the best performances by employing the hyperparameters and transformations outlined in our methodology, as shown in table 3.5. The documentation of the analysis conducted to obtain these optimal hyperparameters can be found in the ablation study chapter. We have included a comparative quantitative evaluation in Fig. 4.3, which highlights the performance of both our models in comparison to the ResNet model. Although both of our models demonstrate nearly comparable performances across all the evaluation metrics, COCO_ViT exhibited slightly better results than IMAGENET_ViT in all metrics except precision. However, it is important to note that both models significantly outperformed the ResNet model, as illustrated in table 4.1.

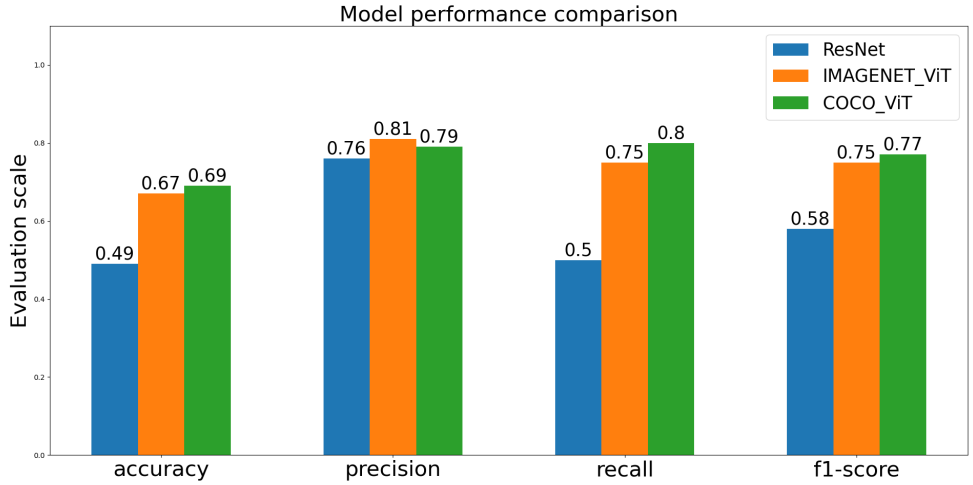


Figure 4.3: Evaluation metrics comparison between our ViT-based image models and the ResNet on the validation dataset.

Table 4.1: The gain of COCO_ViT and IMAGENET_ViT over ResNet model on the validation dataset.

Models	Accuracy	Precision	Recall	F1-score
COCO_ViT	39.6%	5.3%	46.0%	29.0%
IMAGENET_ViT	31.2%	1.1%	44.0%	24.0%

On the other hand, our models enhanced the stability of the prediction confidence in temporal observation by facilitating the learning of abrupt ROV motion

during inspections. Fig. 4.4 demonstrates that both COCO_ViT and IMAGENET_ViT models improved the stability of temporal confidence, particularly in detecting the *paint_peel* class on the video snippet no.3 from table 3.2, in contrast to the ResNet model. Furthermore, the models exhibited a more exploratory nature in detecting other class labels during the inspection which indicates improvement in multi-label competency. Similar improvements in temporal consistency were observed for the remaining testing snippets which are attached in appendix A.2.1.

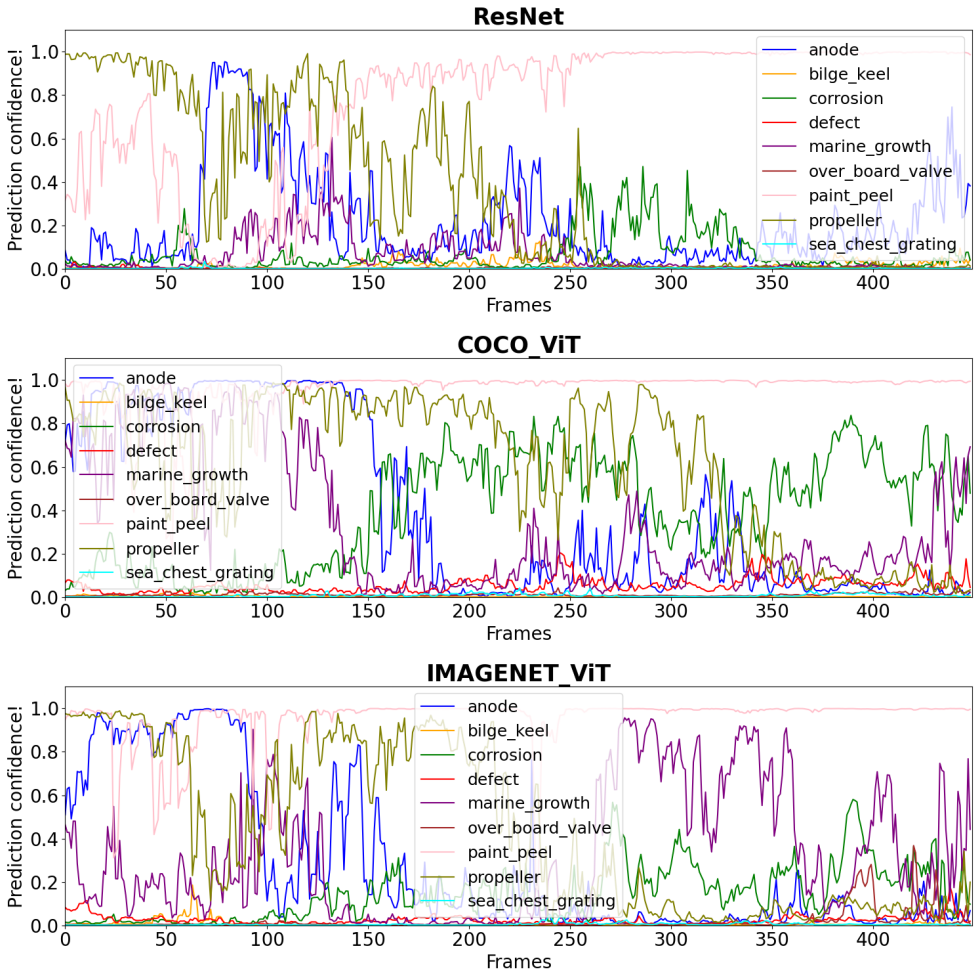


Figure 4.4: Temporal observation of IMAGENET_ViT and COCO_ViT on the video snippet no.3 in table 3.2 compared with the ResNet model.

While the detailed analysis is documented in the ablation study chapter, it is important to highlight the gradual improvements we have achieved in performance. Figs 4.5 and 4.6 visually depict the incremental enhancements of the COCO_ViT and IMAGENET_ViT models, respectively, from the initial ResNet model to our final variants. These figures serve to showcase the notable progress made in terms of performance throughout our iterative development process.

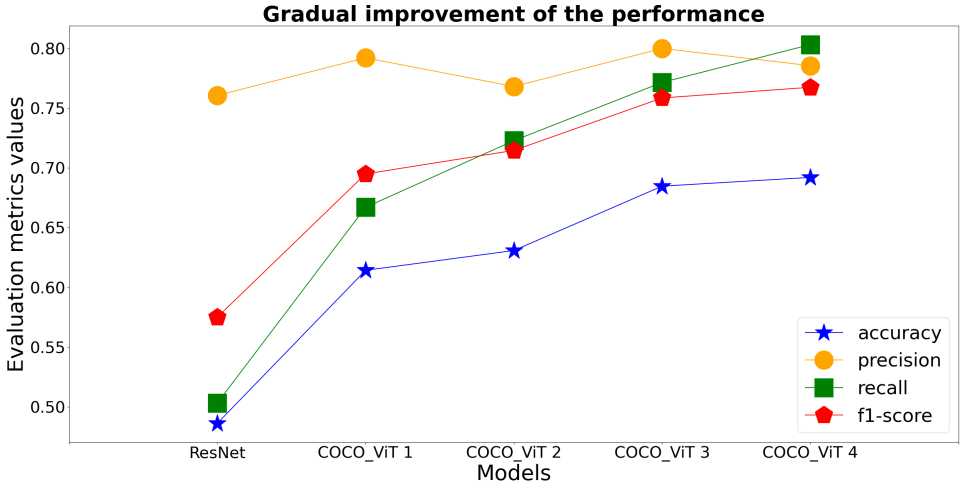


Figure 4.5: Gradual improvement of the COCO_ViT models.

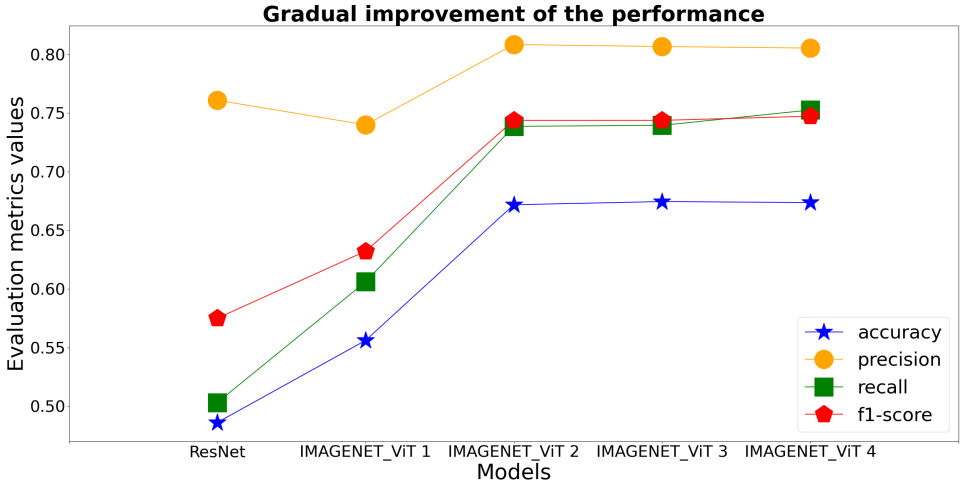


Figure 4.6: Gradual improvement of the IMAGENET_ViT models.

4.3 Multi-label Video Classifiers

Since we posed three different approaches for multi-label video classifiers, our intention is to showcase each outcome separately in the next three subsections. We will ultimately show a comparison of their performances at the end of this section.

4.3.1 Naive Video Transformer

We have trained three different variations as part of our ablation study for each of the embedding techniques employed in the naive approach. It comes as no surprise that our initial variant model, which utilized uniform frame sampling, exhibited significantly poor performance. Conversely, the model variants that utilized the tubelet embedding method demonstrated promising results. Table 4.2 presents the evaluation metric values for each technique on the LIACI validation video dataset. We have reported the best metric values achieved across the variants as well as the best variant for both embedding techniques. The comprehensive details pertaining to the variants, along with additional information, are extensively discussed in the ablation study chapter.

Table 4.2: Evaluation metrics of the naive video models on the LIACI video validation dataset.

Embedding	Best	Loss	Accuracy	Precision	Recall	F1-score
Uniform	values	0.43	0.44	0.64	0.56	0.56
	model 1	0.45	0.45	0.64	0.56	0.56
Tubelet	values	0.33	0.58	0.76	0.69	0.68
	model 2	0.38	0.58	0.73	0.69	0.67

4.3.2 Late-Fusion Spatiotemporal Transformer

In the methodology chapter, we have mentioned about six variants of this approach, which are primarily categorized based on two different strategies for aggregating the latent vector from the spatial transformer for each frame. We can either choose the classification token embedding as our latent vector or compute the mean pooling from all the patch embeddings. Similar to the previous subsection, we have presented the evaluation metrics for the best values across variants and the best single model from these two strategies in table 4.3. The results indicate that the overall performance of this approach surpasses the naive approach by a moderate margin. The rest of the details are discussed and documented in the ablation study chapter.

Table 4.3: Evaluation metrics of the late-fusion video models on the LIACI video validation dataset.

Spatial Latent Vector	Best	Loss	Accuracy	Precision	Recall	F1-score
CLS	values	0.29	0.67	0.80	0.78	0.76
	model 3	0.31	0.67	0.80	0.78	0.76
Mean	values	0.31	0.64	0.78	0.78	0.73
	model 4	0.33	0.64	0.76	0.78	0.73

4.3.3 Attention weighted Spatiotemporal Transformer

In our final approach, we have also divided it into two categories, as explained in the methodology, and trained three variants for each category. Likewise, we have presented the best values across variants and the best model in table 4.4. The results align to some extent with our hypothesis, as the approach succeeds in enhancing the model's quantitative performance compared to the previous two approaches, as reflected by the evaluation metric values. The additional information about the training and models is documented in the ablation study chapter.

Table 4.4: Evaluation metrics of the attention-weighted video models on the LIACI video validation dataset.

Attention Weighting	Best	Loss	Accuracy	Precision	Recall	F1-score
Values (V)	values	0.29	0.67	0.80	0.79	0.76
	model 5	0.30	0.67	0.80	0.79	0.76
Spatial Prediction	values	0.29	0.67	0.80	0.79	0.76
	model 6	0.30	0.67	0.80	0.79	0.76

If you notice that the evaluation metric values for the best values and model are identical, it indicates that the best model achieved the highest metric values across all evaluation metrics. In other words, the best model outperformed the other variants consistently in all the evaluation metrics.

4.3.4 Performance Comparison

To determine the best model from these three approaches, we compare the performances of all the video models across variants. Fig. 4.7 illustrates the performance comparison of the best models from all the approaches together. Model 1-6 represents the models that have been reported in tables 4.2, 4.3, and 4.4. These visualizations aid in selecting the appropriate model for video analysis tasks.

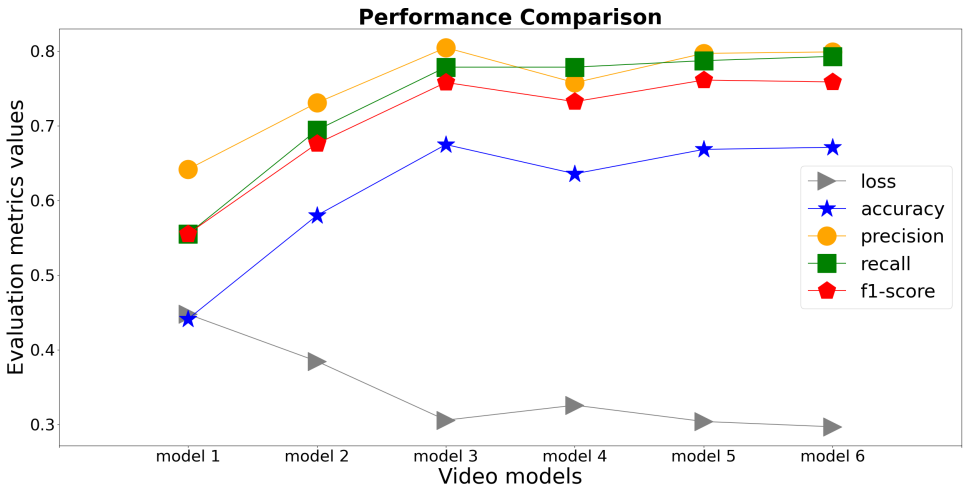


Figure 4.7: Performance comparisons of the best video models from all the variants of different approaches.

4.3.5 Temporal Performance

For temporal performance analysis, we will begin by showcasing the temporal observations of each of the best models from each approach on random video snippets. Subsequently, we will present a comparative temporal observation between the best image and video models side by side to visualize the prediction stabilization of the models during inspections. Figs. 4.8, 4.9, and 4.10 illustrate the temporal observation of the best video models from each approach, presented in the order of their appearances in this section. These observations were respectively conducted on video snippets number 1, 5, and 8, as listed in Table 3.2. The remaining temporal observations for the rest of the video snippets can be found in appendix A.2.2.

Now, we will see the temporal performance comparisons between our best image and video models. Fig. 4.11 represents the temporal performance comparison between the final COCO_ViT model and the best attention-weighted value video model on all eight test video snippets. On the contrary, Fig. 4.12 shows the temporal performance comparison between the final COCO_ViT model and the best attention-weighted spatial prediction video model on all eight test video snippets. The figures demonstrate that the temporal observation using video models resulted in more stable prediction confidences in the temporal dimension of all the video snippets. A detailed discussion of their performances is included in the ablation study.

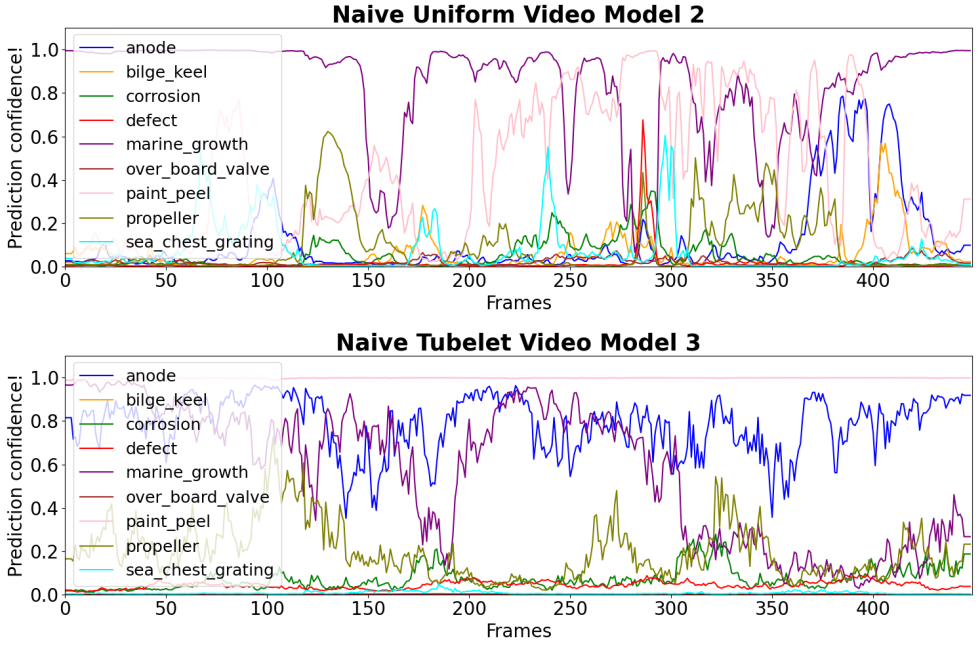


Figure 4.8: Temporal observation of the best two naive video models on the video snippet no.1 from table 3.2.

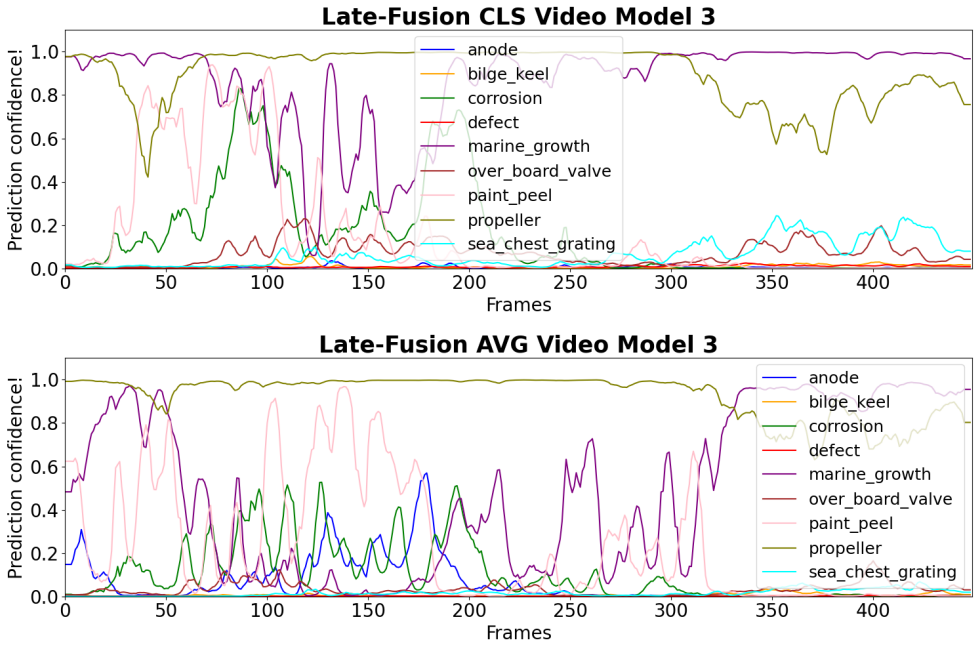


Figure 4.9: Temporal observation of the best two late-fusion video models on the video snippet no.5 from table 3.2.

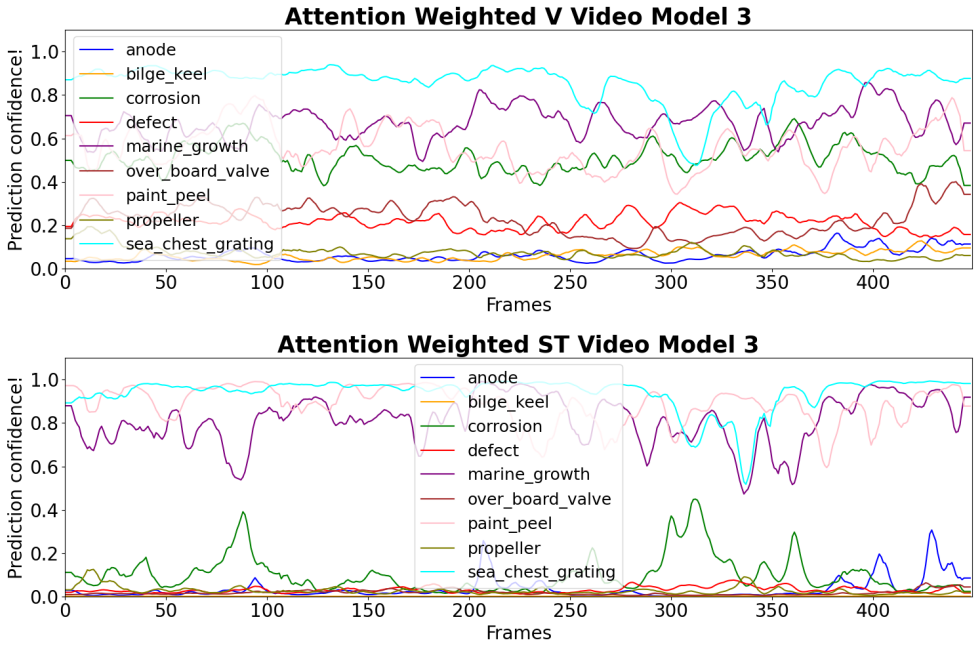


Figure 4.10: Temporal observation of the best two attention-weighted video models on the video snippet no.8 from table 3.2.

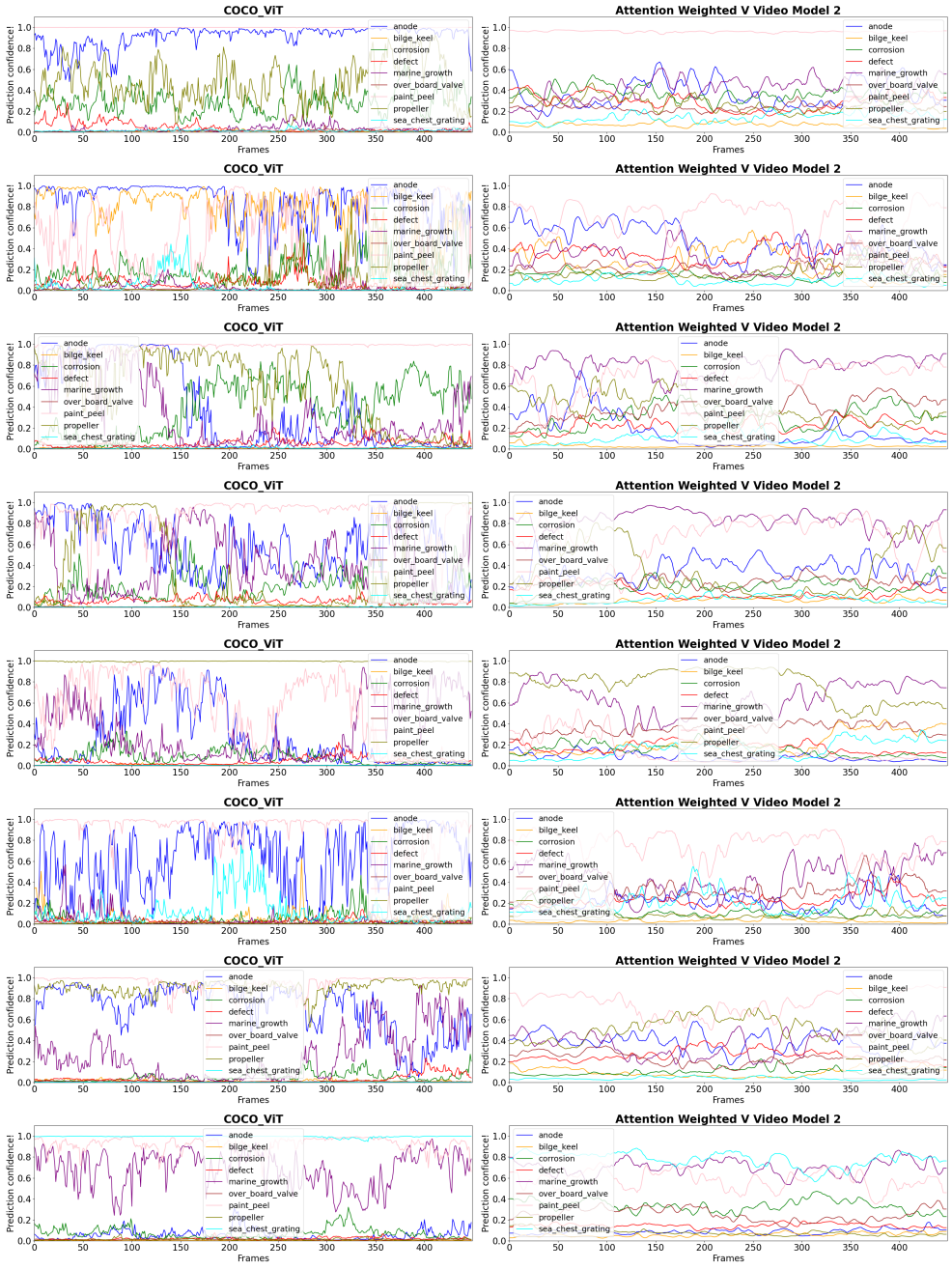


Figure 4.11: Temporal observation of the final COCO_ViT and the attention-weighted value video models on the video snippets of table 3.2.

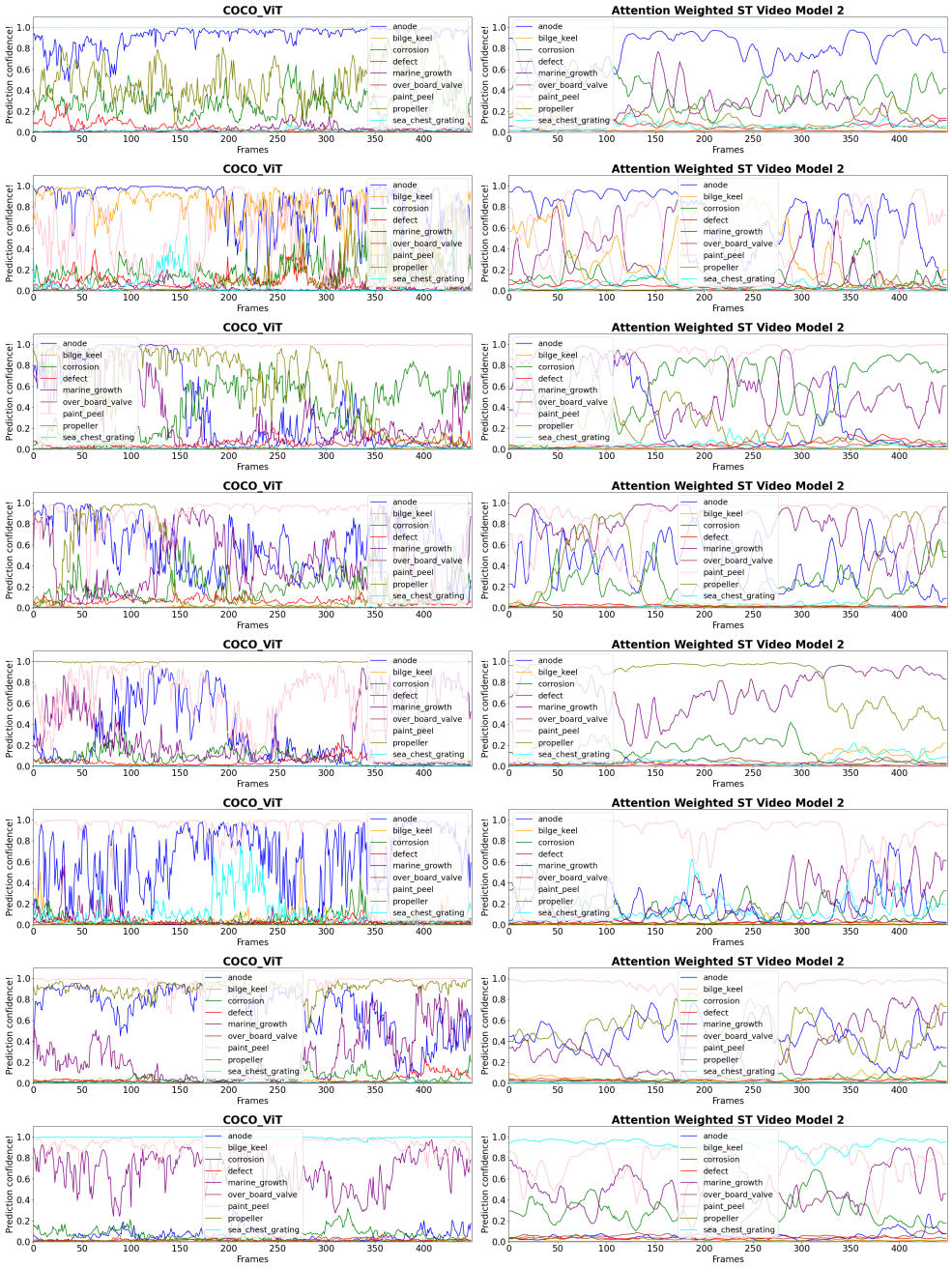


Figure 4.12: Temporal observation of the final COCO_ViT and the attention-weighted ST video models on the video snippets of table 3.2.

Chapter 5

Ablation study

In this ablation study, we performed various analyses on both image and video-based approaches for different components. Through these analyses, we demonstrated gradual improvements in performance. The following sections are dedicated to reporting and explaining some key points we would like to highlight.

5.1 Frame-based Video Classification

To enhance this approach, the initial focus should be on improving our image models. The subsequent subsections will examine various components of the image models and demonstrate a step-by-step enhancement process.

5.1.1 Hyperparameters and Transformations

We trained our image models initially using the hyperparameters and transformations listed in table 5.1. The IMAGENET_ViT and COCO_ViT models were fully finetuned for over 300 epochs to gain insights into their behaviour. Fig. 5.1 visually represents the loss decline across the training epochs for both models on the training and validation datasets. Based on the figure, it is evident that both models, IMAGENET_ViT and COCO_ViT exhibit signs of overfitting. This is expected since the LIACI dataset is relatively small in scale. Furthermore, the evaluation metric values in table 5.2 for both models indicate that COCO_ViT shows similar or even better performance compared to IMAGENET_ViT.

To mitigate the overfitting issue, we adopted a regularization technique by performing another round of fine-tuning on the models. However, this time we employed partial fine-tuning. Specifically, we kept all the weights of the transformer frozen and solely updated the MLP head. The outcome is visualized in Fig. 5.2, which shows that IMAGENET_ViT outperforms COCO_ViT as a feature extractor. Also, it is worth noting that the difference between the train and validation losses for both models is not significantly high, indicating that the models

Table 5.1: Initial training hyperparameters and data transformations.

Hyperparameters	Transformation
BCEWithLogitsLoss	Image Resize(224x224)
Optimizer: SGD	RandomHorizontalFlip(p=0.5)
Learning rate: 0.001	Normalization: Mean[0.3485, 0.3699, 0.3520]
Momentum: 0.9	Normalization: Std [0.2495, 0.2446, 0.2062]
Batch size: 16	
Epochs: 100	
Scheduler:StepLR	
(step=20,gamma=0.1)	

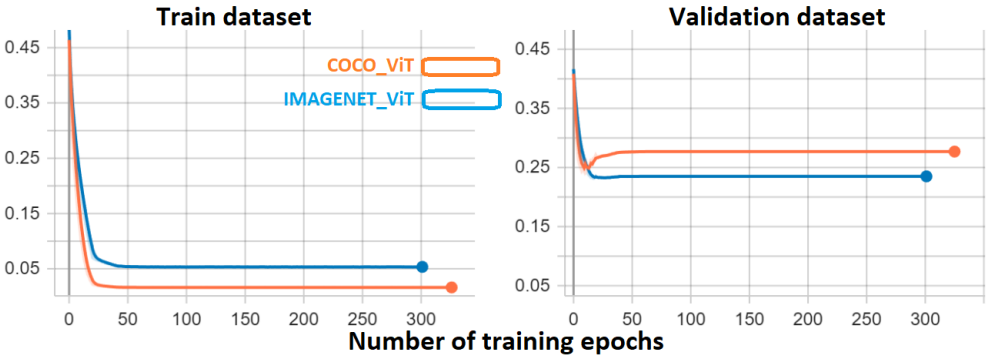


Figure 5.1: The loss behaviour during the training of the IMAGENET_ViT and COCO_ViT.

Table 5.2: Initial evaluation metric values of the image-based models on the validation dataset.

Model	Accuracy	Precision	Recall	F1-score
IMAGENET_ViT	0.66	0.80	0.72	0.73
COCO_ViT	0.67	0.80	0.76	0.75

are well-regularized. However, provided the superior performance observed on the validation dataset, our decision is to proceed with further development and refinement of the fully fine-tuned models.

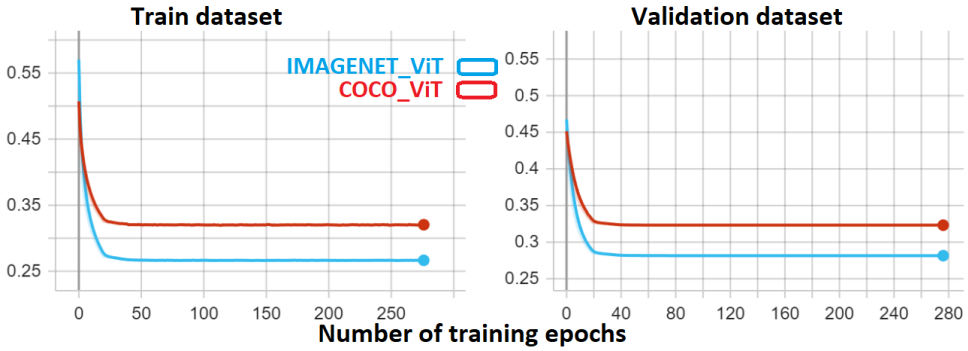


Figure 5.2: The loss behaviour during partial fine-tuning of the IMAGENET_ViT and COCO_ViT.

We also replaced the optimizer SGD with Adam and noticed a decline in performance. Hence, we reverted back to using SGD to continue our analysis. On the other hand, the ReduceLROnPlateau learning rate scheduler aids in finding better local minima on the validation loss. Fig. 5.3 shows that the final model was able to find the minimal loss on validation compared to the initial one in both cases. The optimal validation loss for IMAGENET_ViT is within 20 to 30 epochs. Increasing the loss of the final model during training compared to the initial model and subsequently reducing the loss more on the validation set leads to better regularization of COCO_ViT. Besides, the utilization of Gaussian blur and AugMix [32] enhanced the stability of the model's confidence in temporal analysis by facilitating the learning of abrupt ROV motion during inspections. Consequently, Fig. 5.4 demonstrates that both models improved the stability of temporal confidence, particularly in detecting the *Paint peel* class, in contrast to Fig. 4.1 at the beginning of the result chapter. Furthermore, the models exhibited a more exploratory nature in detecting other class labels during the inspection which indicates improvement in multi-label competency. Similar improvements in temporal consistency were observed for the remaining testing snippets which are shown in Figs. 4.11 and 4.12 alongside the outputs from video models in the results chapter.

5.1.2 Prediction Confidence Evolution

We investigated how the confidence of a model's predictions on different class labels evolves throughout the training epochs. We anticipated that the confidence for the true class labels would progressively increase while diminishing the remaining class labels. Fig. 5.5 illustrates an example of this confidence acceleration phenomenon across the training epochs. In the first row of the figure, a gradual increase in prediction confidence can be observed for the classes *propeller*

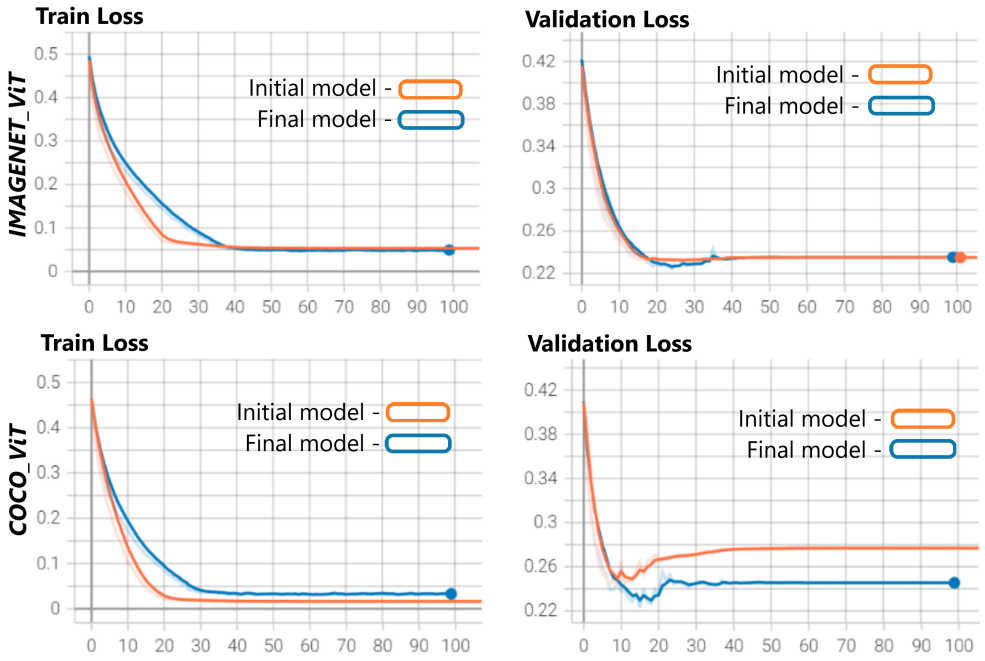


Figure 5.3: Comparison between the initial and final models in finding minimal loss for the IMAGENET_ViT and COCO_ViT.

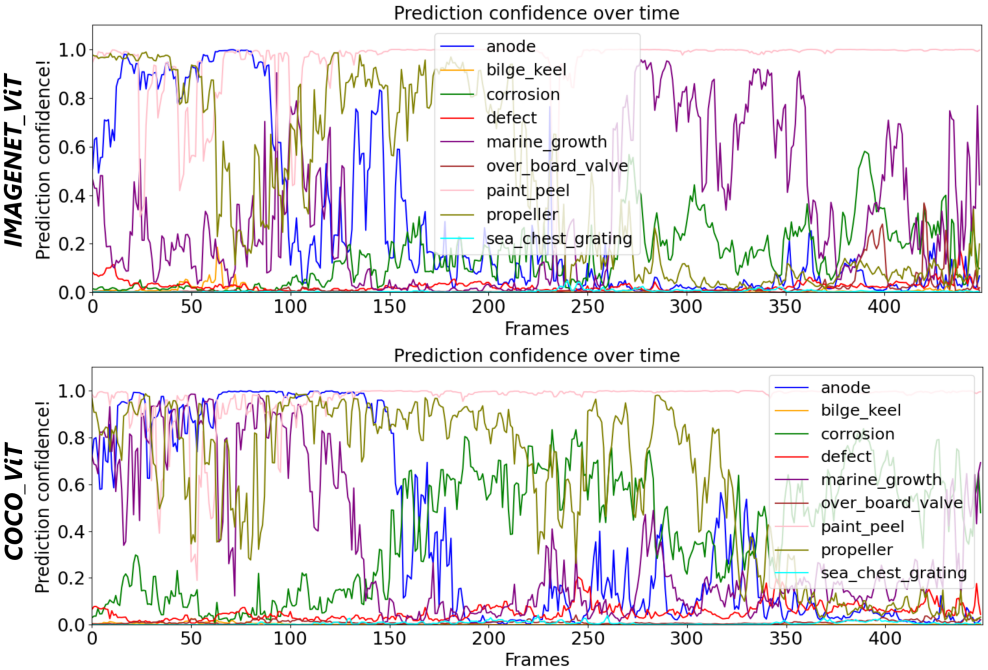


Figure 5.4: Temporal observation of the final IMAGENET_ViT and COCO_ViT on the same video snippet as in Fig. 4.1.

and *paint_peel*. Similarly, the second row demonstrates a similar trend of confidence growth for the classes *marine_growth*, *paint_peel*, and *sea_chest_grating*. This observation is important as it provides a basis for introducing reinforcement techniques, such as active learning, during the model training process. These techniques can be utilized to address the poor performance of certain class labels identified through the analysis of the confidence evolution. In our experiments, we attempted to utilize guided cropping as a data augmentation technique to improve the performance of weakly performed class labels. However, we observed that this approach had a negative impact on the overall performance. The reason behind this is that when we cropped a single class from a training frame, the other classes present in the frame were neglected, resulting in a loss of important information. As a result, we have identified the need to further refine and develop the strategy of guided cropping as a potential area for future work and improvement.

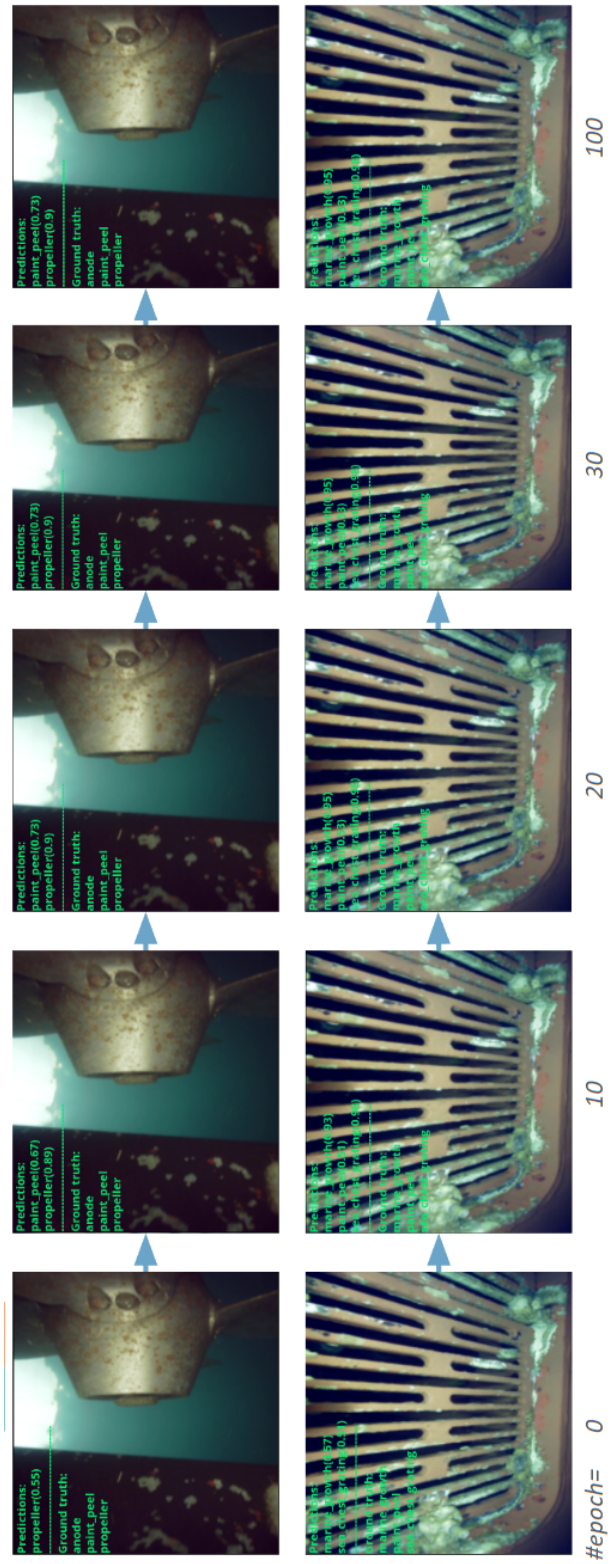


Figure 5.5: Prediction confidence evolution during a training of a model.

5.1.3 Multi-label ViT Image Models

To extract the best performance from image-based models for underwater ship hull inspection, several models were trained with gradual improvements by addressing the limitations of the LIACI dataset. The COCO 2014 dataset is a large-scale dataset that contains images with multiple object classes labelled in each image. In contrast, the IMAGENET dataset is primarily used for conventional image classification tasks where each image belongs to a single class. Hence, enabling our model to have multi-label classification capability, we initially train a ViT model on the COCO 2014 dataset using the hyperparameters and transformations mentioned in Table 3.5. The COCO dataset consists of 82783 train and 40504 validation images and the model was trained for 94 epochs with a batch size of 16. We observed the model stops learning approximately after 30 epochs as both the training and validation losses become extremely low despite the accuracy still being confined under 0.7. Subsequently, we perform full finetuning of our two initial ViT models on the LIACI dataset. Table 5.3 includes the analysis of these initial models in rows 2 and 3, whereas row 1 corresponds to the COCO model. It is apparent from the F1-score or other metric values of these two initial models that the ViT pre-trained on COCO performs better than the one pre-trained on IMAGENET.

We investigated which model performs best in extracting features from the LIACI data. To devise this, we trained variants of the COCO and IMAGENET models using partial finetuning, where all the pre-trained weights except the classification part are frozen. The results are included in rows 4 and 5 of table 5.3 which imply that the IMAGENET version outperforms the COCO model in feature extraction. However, the overall performance of the partial finetuning approach is still below the full finetuning approach. Therefore, we decided to keep the partial finetuning approach apart from our experiments. Additionally, we experiment with changing the optimizer from SGD to Adam with a weight decay of 0.3 to train both models but this led to a significant degradation in performance. We conducted experiments to explore the effects of different step sizes on the performance of the COCO and IMAGENET models. Along with the StepLR learning rate scheduler with a gamma value of 0.1 and test two more different step sizes: 5 and 50. To summarize, using a step size of 5 led to further regularization of the COCO model, but it also induced a decline in the overall performance for both models, as shown in rows 6 and 7 of table 5.3. On the other hand, the step size of 50 had a tendency to overfit the training for both models as assigned in rows 8 and 9. Finally, we deduced the best models with the configuration mentioned in the result section by considering both the quantitative evaluation measures and qualitative temporal performance which are also added in rows 10 and 11. COCO_ViT is the best frame-based model which dominates all the validation evaluation metrics except the precision which is dominated by its counterpart IMAGENET_ViT.

Table 5.3: Analysis of different models & results. FF = fully finetune & PF = partial finetune.

	Model	Pretrain weight	#Epochs	Loss		Accuracy		Precision		Recall		F1-score	
				Train	Val	Train	Val	Train	Val	Train	Val	Train	Val
1	COCO ViT	IMAGENET 1K (FF)	94	0.042	0.051	0.708	0.601	0.895	0.838	0.745	0.692	0.790	0.731
2	LIACI ViT (initial)	IMAGENET 1K (FF)	301	0.054	**0.235**	0.951	0.659	0.968	**0.798**	0.952	0.723	0.958	0.729
3	LIACI ViT (initial)	COCO 2014 (FF)	326	0.016	0.277	0.970	0.673	0.971	0.797	0.969	0.760	0.970	0.749
4	LIACI ViT (extractor)	IMAGENET 1K (PF)	277	0.267	0.281	0.593	0.565	0.764	0.741	0.648	0.621	0.672	0.642
5	LIACI ViT (extractor)	COCO 2014 (PF)	276	0.320	0.323	0.484	0.479	0.648	0.637	0.536	0.534	0.559	0.552
6	LIACI ViT (step=5)	IMAGENET 1K (FF)	99	0.263	0.288	0.597	0.556	0.778	0.740	0.651	0.606	0.678	0.632
7	LIACI ViT (step=5)	COCO 2014 (FF)	99	0.170	0.260	0.779	0.614	0.902	0.792	0.810	0.667	0.834	0.695
8	LIACI ViT (step=50)	IMAGENET 1K (FF)	99	0.018	0.277	0.971	0.672	**0.972**	0.808	0.971	0.739	0.971	0.744
9	LIACI ViT (step=50)	COCO 2014 (FF)	99	**0.010**	0.315	**0.972**	0.631	**0.972**	0.768	**0.972**	0.723	**0.972**	0.715
10	LIACI ViT (final)	IMAGENET 1K (FF)	99	0.034	**0.235**	0.961	0.674	0.969	0.805	0.962	0.753	0.964	0.747
11	LIACI ViT (final)	COCO 2014 (FF)	99	0.071	0.240	0.915	**0.692**	0.936	0.786	0.947	**0.803**	0.935	**0.768**

5.2 Video-based Classification

5.2.1 Spatiotemporal-based Video Classification

With the uniform frame sampling tokenization, we attempted to train our video models utilizing both image models and experimented with different learning rate schedulers. However, none of these approaches resulted in convergence during training. It is important to note that we were limited to using a dependent patch size to generate a total of 196 image patch embeddings from 7 frames, which were then fed into a ViT model. In addition to the video models discussed earlier, we also explored an approach that involved combining 3D CNN and ViT which we referred to as the tubelet embedding approach. Specifically, we extracted C3D features utilizing a pretrained 3D ResNet architecture and subsequently passed these features through our trained ViT-based image models. Although this approach resulted in convergence during training, the performance was not competitive enough to be included in the paper.

The spatial-temporal video model we reported in the paper has a total of 161.399M trainable parameters, with 75.600M of them belonging to the temporal transformer. Since we are utilizing a pre-trained ViT classifier as the spatial transformer, we freeze its weights during training and only update the weights of the temporal transformer, resulting in a substantial reduction in training time. One significant challenge that can contribute to poor performance is the limitation of transformers, which require pretraining on a large-scale dataset to optimize their performance. This is particularly relevant for the temporal transformer in our models, as its weights are initialized randomly, which can limit its ability to learn from the available data and lead to poor performance.

5.2.2 Multi-label Video Classifiers

For analysis purposes, we trained six variants from each of the three video model approaches mentioned in the methodology chapter. In the case of the naive approach, the first three variants utilized the uniform frame sampling embedding technique. Specifically, the first variant used partial fine-tuning, the second variant went through full fine-tuning, and the third variant was fully fine-tuned starting from the first variant. The partial finetuning here refers to only optimizing the new embedding layer in the network. Similarly, the other three variants are trained using tubelet embedding technique. The only distinction in the case of partial fine-tuning is that we focused on optimizing the C3D network, which serves as our tubelet embedding method. This approach involved fine-tuning the C3D network while keeping the remaining parts of the model unchanged. Table 5.4 listed all the variants including the other information.

Table 5.4: Analysis of the naive approach. FF = fully finetune & PF = partial finetune.

	Train Type	Embedding	Model size	Trainable parameters	Best epoch	Loss		Accuracy		Precision		Recall		F1-score	
						Train	Val	Train	Val	Train	Val	Train	Val	Train	Val
1	PF	Uniform	359MB	4.130M (89.935M)	20	0.33	0.43	0.46	0.44	0.67	0.64	0.50	0.55	0.54	0.55
2	FF	Uniform	684MB	89.935M (89.935M)	28	0.13	0.45	0.84	0.44	0.91	0.64	0.86	0.56	0.87	0.56
3	FF	Uniform	684MB	89.935M (89.935M)	32	0.11	0.48	0.87	0.44	0.93	0.64	0.89	0.55	0.90	0.55
4	PF	Tubelet	1185MB	110.387M (200.322M)	76	0.07	0.33	0.94	0.55	0.97	0.75	0.95	0.62	0.95	0.64
5	FF	Tubelet	1513MB	200.322M (200.322M)	57	0.11	0.34	0.86	0.57	0.92	0.76	0.89	0.65	0.89	0.67
6	FF	Tubelet	1513MB	200.322M (200.322M)	5	0.13	0.38	0.80	0.58	0.88	0.76	0.85	0.69	0.85	0.68

Table 5.5 presents all six variants from the late-fusion approach, which were trained in a similar manner. The variants are categorized based on the aggregation method used for the latent vector obtained from the spatial transformer for each video frame. The "cls" category denotes taking the output vector of the classification token, while the "avg" category indicates calculating the mean of all the patches. The partial finetuning here refers to optimizing only the temporal transformer while keeping the spatial transformer frozen.

Within the final approach, which is the attention-weighted spatiotemporal transformer, we also trained six variants. The primary distinction among these variants lies in how we incorporate attention-weighted prediction. Table 5.6 presents the details of these variants where the term "value" indicates that the attention weights are applied to the value (V) matrix of the self-attention mechanism, while "ST" signifies taking the attention-weighted prediction from the spatial transformer. The rest are similar as the late-fusion approach.

Table 5.5: Analysis of the late-fusion approach. FF = fully finetune & PF = partial finetune.

	Train Type	Latent Vector(ST)	Model size	Trainable parameters	Best epoch	Loss		Accuracy		Precision		Recall		F1-score	
						Train	Val	Train	Val	Train	Val	Train	Val	Train	Val
1	PF	cls	510MB	22.071M (107.870M)	33	0.16	0.29	0.78	0.64	0.87	0.79	0.82	0.73	0.83	0.73
2	FF	cls	838MB	107.870M (107.870M)	25	0.02	0.35	0.98	0.64	0.98	0.78	0.98	0.74	0.98	0.73
3	FF	cls	838MB	107.870M (107.870M)	7	0.04	0.31	0.97	0.67	0.98	0.80	0.97	0.78	0.97	0.76
4	PF	avg	510MB	22.071M (107.870M)	31	0.19	0.31	0.72	0.62	0.83	0.78	0.78	0.76	0.78	0.72
5	FF	avg	838MB	107.870M (107.870M)	22	0.03	0.31	0.98	0.63	0.98	0.76	0.98	0.76	0.98	0.72
6	FF	avg	838MB	107.870M (107.870M)	11	0.03	0.33	0.97	0.64	0.98	0.76	0.98	0.78	0.98	0.73

Table 5.6: Analysis of the attention-weighted approach. FF = fully finetune & PF = partial finetune.

	Train Type	Attention Weighting	Model size	Trainable parameters	Best epoch	Loss		Accuracy		Precision		Recall		F1-score	
						Train	Val	Train	Val	Train	Val	Train	Val	Train	Val
1	PF	value	510MB	22.071M (107.870M)	29	0.18	0.30	0.74	0.60	0.86	0.77	0.79	0.70	0.80	0.70
2	FF	value	838MB	107.870M (107.870M)	8	0.10	0.29	0.90	0.63	0.94	0.78	0.91	0.73	0.92	0.72
3	FF	value	838MB	107.870M (107.870M)	10	0.03	0.30	0.97	0.67	0.98	0.80	0.97	0.79	0.97	0.76
4	PF	ST	494MB	22.071M (107.870M)	91	0.20	0.31	0.72	0.59	0.83	0.77	0.77	0.67	0.78	0.69
5	FF	ST	821MB	107.870M (107.870M)	9	0.10	0.29	0.89	0.63	0.94	0.79	0.90	0.73	0.91	0.73
6	FF	ST	821MB	107.870M (107.870M)	7	0.05	0.30	0.95	0.67	0.97	0.80	0.96	0.79	0.96	0.76

5.2.3 Number of layers in Temporal Transformer

During our experimentation, we investigated the impact of varying the number of layers in the temporal transformer. The results of our analysis are presented in table 5.7, indicating that the video models with four layers in the temporal transformer achieved the highest performance in the validation dataset. Consequently, we decided to maintain four encoder layers in the temporal transformer architecture for all of our video models.

Table 5.7: Experiments on the number of encoder layers in Temporal Transformer.

	Number of layers	Model size	Trainable parameters	Best epoch	Loss		Accuracy		Precision		Recall		F1-score	
					Train	Val	Train	Val	Train	Val	Train	Val	Train	Val
1	2	426MB	11.043M (96.842M)	20	0.22	0.30	0.70	0.61	0.83	0.79	0.75	0.71	0.77	0.71
2	4	838MB	22.071M (107.870M	33	0.16	0.29	0.78	0.64	0.87	0.79	0.82	0.73	0.83	0.73
3	6	838MB	33.099M (118.898M)	33	0.15	0.30	0.80	0.60	0.88	0.78	0.85	0.73	0.85	0.71
4	8	494MB	44.127M (129.926M)	18	0.19	0.30	0.73	0.62	0.84	0.79	0.78	0.74	0.79	0.72
5	10	821MB	55.155M (140.954M)	30	0.13	0.31	0.83	0.62	0.89	0.79	0.87	0.73	0.87	0.72
6	12	821MB	66.183M (151.982M)	30	0.13	0.31	0.81	0.63	0.89	0.77	0.85	0.76	0.86	0.73

5.2.4 Single Query Attention Inspection

Once the training of our final video models utilizing single query attention was completed, we conducted an analysis of the attention-scoring behaviour. Figure 5.6 illustrates the results of this analysis, where the highlighted middle frames represent the single queries for each snippet. In the first row of the figure, where the frames of the input video snippet are the same frame, we can observe that the attention scores for all frames are evenly distributed, aligning with our expectations. Moving to the second row, which consists of random frames unrelated to the query frame, we observe that there is no attention given to the neighbouring frames, indicating that the model is correctly ignoring irrelevant frames. Finally, the last row presents a real snippet extracted from our validation dataset. Here, we can observe that the query frame receives the highest attention score, gradually decreasing based on the distance from the adjacent frames.

Figure 5.6: Distribution of the attention scores to the query as well as neighbouring frames in different situations.

This observation is highly promising as it suggests that our model's attention mechanism is functioning effectively, which is likely to contribute to improved performance. It provides us with valuable insights and encourages us to pursue further development and refinement of our model.

Chapter 6

Conclusion & Future Work

We have trained several multi-label ViT image classifiers and gradually improved them on the LIACI dataset to conduct framewise video inspections. Our image-based models successfully surpassed the performance of the ResNet model that was previously used in the LIACI project. In fact, the same improved image models are also utilized during training multi-label video classifiers through different state-of-the-art approaches and modifying them based on our requirements. However, while frame-based ViT classifiers are limited by their inability to capture temporal information, video classifiers overcome this limitation by extracting both spatial and temporal features from the video. Spatial features are dominant in some videos, making image classifiers suitable for evaluation. Our single query attention-based video models showed adequate capability in distributing attention scores to neighbouring frames in the temporal dimension. Considering temporal attention from the adjacent frames during classification improves the robustness of the task, making it more effective for difficult video inspections like ours, and also stabilizes the model's prediction in the temporal dimension.

Although we conducted an exhaustive analysis, we believe still there are rooms in improving the performance of both image and video-based classifiers for an underwater environment. For example, exploring other pretraining strategies or designing custom architectures may yield better results. Additionally, gathering more diverse and high-quality data can also improve the performance of these models. Incorporating other techniques such as data augmentation, transfer learning, or ensembling can also be explored to improve the overall performance. Besides, introducing a quantitative metric to evaluate the temporal performance of the video-based classifiers would indeed be a useful research direction. By quantifying the temporal performance, we can have a measurement of how well the model is able to capture temporal information in the videos. This could potentially lead to further improvements in the model architecture or training process and ultimately result in better performance for video-based classification tasks in underwater environments.

Designing a new spatial transformer architecture that is compatible with the uniform frame sampling tokenization of seven frames could potentially overcome the convergence issue observed previously. Pretraining this new architecture on large-scale datasets before fine-tuning it for the LIACI dataset could also improve its performance. One significant challenge we faced is the limited size and weakly supervised nature of our video dataset. To address this, it is better to explore options such as acquiring a larger fully supervised dataset, using techniques like data augmentation and regularization to enhance generalization, or incorporating pretrained weights for the temporal transformer. By doing so, we could improve the robustness and effectiveness of our video inspection models.

In conclusion, we hope this work provides a benchmark for the development of image and video-based classifiers in underwater environments for ship hull inspection. The analysis will help researchers and developers to improve the accuracy and effectiveness of these classifiers and our findings will facilitate the application of these methods in real-world scenarios. Furthermore, we will also continue to focus on improving the video model and developing quantitative metrics to evaluate the temporal performance of video-based classifiers to improve their reliability and robustness.

Bibliography

[1] J. Plested and T. Gedeon, 'Deep transfer learning for image classification: A survey,' *arXiv preprint arXiv:2205.09904*, 2022.

[2] J. Hirsch, B. Elvesæter, A. Cardaillac, B. Bauer and M. Waszak, 'Fusion of multi-modal underwater ship inspection data with knowledge graphs,' in *OCEANS 2022, Hampton Roads*, IEEE, 2022, pp. 1–9.

[3] M. Salvaris, D. Dean, W. H. Tok, M. Salvaris, D. Dean and W. H. Tok, 'Cognitive services and custom vision,' *Deep Learning with Azure: Building and Deploying Artificial Intelligence Solutions on the Microsoft AI Platform*, pp. 99–128, 2018.

[4] D.-A. Huang, V. Ramanathan, D. Mahajan, L. Torresani, M. Paluri, L. Fei-Fei and J. C. Niebles, 'What makes a video a video: Analyzing temporal information in video understanding models and datasets,' in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7366–7375.

[5] H. Xia and Y. Zhan, 'A survey on temporal action localization,' *IEEE Access*, vol. 8, pp. 70 477–70 487, 2020.

[6] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, 'Gradient-based learning applied to document recognition,' *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] K. Simonyan and A. Zisserman, 'Very deep convolutional networks for large-scale image recognition,' *arXiv preprint arXiv:1409.1556*, 2014.

[8] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'Imagenet classification with deep convolutional neural networks,' *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, 'Going deeper with convolutions,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, 'Attention is all you need,' *Advances in neural information processing systems*, vol. 30, 2017.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, 'An image is worth 16x16 words: Transformers for image recognition at scale,' 2021.

[12] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou and M. Douze, 'Levit: A vision transformer in convnet's clothing for faster inference,' in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12 259–12 269.

[13] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, 'Swin transformer: Hierarchical vision transformer using shifted windows,' in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[14] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo and L. Shao, 'Pyramid vision transformer: A versatile backbone for dense prediction without convolutions,' in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 568–578.

[15] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, 'End-to-end object detection with transformers,' in *European conference on computer vision*, Springer, 2020, pp. 213–229.

[16] Z. Dai, B. Cai, Y. Lin and J. Chen, 'Up-detr: Unsupervised pre-training for object detection with transformers,' in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 1601–1610.

[17] Z. Li, W. Wang, E. Xie, Z. Yu, A. Anandkumar, J. M. Alvarez, P. Luo and T. Lu, 'Panoptic segformer: Delving deeper into panoptic segmentation with transformers,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 1280–1289.

[18] X. Chen, C.-J. Hsieh and B. Gong, 'When vision transformers outperform resnets without pre-training or strong data augmentations,' 2022.

[19] L. Huang, L. Wang and H. Li, 'Weakly supervised temporal action localization via representative snippet knowledge propagation,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3272–3281.

[20] H. Wang and C. Schmid, 'Action recognition with improved trajectories,' in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 3551–3558.

[21] D. Tran, L. Bourdev, R. Fergus, L. Torresani and M. Paluri, 'Learning spatiotemporal features with 3d convolutional networks,' in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[22] G. Bertasius, H. Wang and L. Torresani, 'Is space-time attention all you need for video understanding?' In *ICML*, vol. 2, 2021, p. 4.

[23] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić and C. Schmid, 'Vivit: A video vision transformer,' in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6836–6846.

[24] J. Pan, Z. Lin, X. Zhu, J. Shao and H. Li, 'St-adapter: Parameter-efficient image-to-video transfer learning,' *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 462–26 477, 2022.

[25] A. Bulat, J. M. Perez Rua, S. Sudhakaran, B. Martinez and G. Tzimiropoulos, 'Space-time mixing attention for video transformer,' *Advances in Neural Information Processing Systems*, vol. 34, pp. 19 594–19 607, 2021.

[26] M. Waszak, A. Cardaillac, B. Elvesæter, F. Rødølen and M. Ludvigsen, 'Semantic segmentation in underwater ship inspections: Benchmark and data set,' *IEEE Journal of Oceanic Engineering*, 2022.

[27] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, 'Pytorch: An imperative style, high-performance deep learning library,' *Advances in neural information processing systems*, vol. 32, 2019.

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, 'Imagenet: A large-scale hierarchical image database,' in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, 'Microsoft coco: Common objects in context,' in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755.

[31] B. E. Moore and J. J. Corso, 'Fiftyone,' *GitHub. Note: https://github.com/voxel51/fiftyone*, 2020.

[32] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer and B. Lakshminarayanan, 'Augmix: A simple data processing method to improve robustness and uncertainty,' *arXiv preprint arXiv:1912.02781*, 2019.

[33] A. Mohammed, I. Farup, M. Pedersen, S. Yildirim and Ø. Hovde, 'Ps-devcem: Pathology-sensitive deep learning model for video capsule endoscopy based on weakly labeled data,' *Computer Vision and Image Understanding*, vol. 201, p. 103 062, 2020.

[34] G. Bradski, 'The OpenCV Library,' *Dr. Dobb's Journal of Software Tools*, 2000.

[35] M. Yang and A. Sowmya, 'An underwater color image quality evaluation metric,' *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 6062–6071, 2015.

[36] K. Panetta, C. Gao and S. Agaian, 'Human-visual-system-inspired underwater image quality measures,' *IEEE Journal of Oceanic Engineering*, vol. 41, no. 3, pp. 541–551, 2015.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, 'Scikit-learn: Machine learning in Python,' *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Paper I

This is the paper that has been accepted to be presented on 8th June at OCEANS 2023 conference in Limerick, Ireland and will be published by IEEE. The presentation slides are also attached to the paper. This paper includes the preliminary results we produced from our video models. However, we have improved the models in a later stage that is included in the thesis and out of the scope of this paper.

Multi-label Video Classification for Underwater Ship Inspection

Md Abulkalam Azad*†‡, Ahmed Mohammed*, Maryna Waszak*, Brian Elvesæter*, and Martin Ludvigsen‡
*SINTEF AS, Forskningsveien 1, 0373 Oslo, Norway
†Faculty of Sciences and Technology, University of Toulon (UTLN), Toulon, France
‡Department of Marine Technology, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

***Abstract*—Today ship hull inspection including the examination of the external coating, detection of defects, and other types of external degradation such as corrosion and marine growth is conducted underwater by means of Remotely Operated Vehicles (ROVs). The inspection process consists of a manual video analysis which is a time-consuming and labor-intensive process. To address this, we propose an automatic video analysis system using deep learning and computer vision to improve upon existing methods that only consider spatial information on individual frames in underwater ship hull video inspection. By exploring the benefits of adding temporal information and analyzing frame-based classifiers, we propose a multi-label video classification model that exploits the self-attention mechanism of transformers to capture spatiotemporal attention in consecutive video frames. Our proposed method has demonstrated promising results and can serve as a benchmark for future research and development in underwater video inspection applications.**

***Index Terms*—Video Classification, Vision Transformer, Underwater Inspection, Deep Learning, Computer Vision**

I. Introduction

A. Underwater ship hull inspection

Inspection of marine vessels in the maritime industry plays a significant role in monitoring the life cycle and analyzing the condition of the hull. It examines the external coating and detects potential defects. Corrosion, marine growth, or other external degradation can damage the hull and reduce its lifespan. Ship hull inspections are nowadays shifting to underwater operation from dry-dock to reduce the cost and downtime of the ship. These are conducted by a Remotely Operated Vehicle (ROV) to further cut down the cost and prevent the risk of a human diver. The general procedure as illustrated in Fig. 1 consists of a) collection of videos of the ship hull using an ROV, b) intensive analysis of the videos, and c) preparation of the inspection report. The manual video analysis within the process is time-consuming, tedious, and prone to human error. Therefore, with the advancement of deep learning in computer vision and autonomy in underwater vehicles, automatic video analysis can greatly improve underwater inspection.

B. Frame-wise classification

A trivial approach to video analysis is to classify each frame of the entire video separately and identify potential threats such as defects or corrosion. This approach only needs an efficient and robust multi-label image classifier and many such off-the-shelf models are available online. We can use a pre-trained image classification model and apply an effective deep transfer learning technique as suggested in [1] to fine-tune the model for our domain. A preceding work under the LIACi[1] project [2] also utilized transfer learning to train a multi-label image classifier using the Microsoft Custom Vision [3] framework on the LIACi dataset to classify individual frames in the video. The trained model can predict nine different class labels as illustrated in the methods & materials section on the surface of the ship hull. However, this approach has a significant limitation as it only considers spatial information from static image frames and lacks the temporal insight that is essential for Video Understanding [4]. As a result, the model becomes temporally unstable.

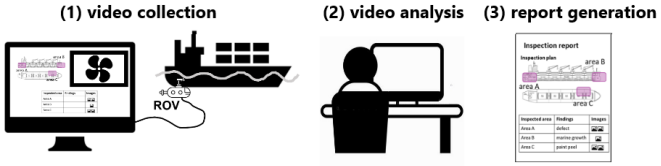


Fig. 1. The workflow of current underwater ship inspection using ROVs.

C. Main objective

In order to alleviate the issue, it is necessary to train a model by learning spatiotemporal information from videos which can improve the automatic video analysis of underwater ship hull inspections. Unlike temporal action recognition and localization [5] that consider dynamic foreground and background objects, our videos only have static scenes including ROV motion with a dynamic camera. Hence, the benefit of utilizing the temporal aspects can facilitate stabilization during the video analysis. Our core focus is to enhance the consistency and stability of the model's predictions during underwater video analysis. Therefore, in this paper, we investigate the consistency and stability of image-based classifiers which can help us in understanding the advantages and limitations of using an image-based multi-label classifier for this purpose. Furthermore, we propose a video classification model that takes into account both temporal and spatial information. In summary, the contributions of this work are;

a. Analysis of image-based classifiers (benefits and limitations).

[1]Lifecycle Inspection, Analysis, and Condition information system (https://www.sintef.no/en/projects/2021/liaci/)

b. Exploration of the benefits of adding temporal information.
c. Identification of a deep learning multi-label video classifier for labeling video frames based on spatiotemporal attention.

The rest of the paper is divided into five sections. Related works are described in section II, whereas section III unveils the methods & materials we utilize within this work. Sections IV and V include the results of our works and ablation study. Finally, we conclude in section VI by leaving some discussion and direction for further research and development in the same area.

II. Related works

A. Computer vision technology

Computer vision has been used in automating various industries worldwide. While artificial intelligence enables machines to think, computer vision provides them with the ability to see. It has been used in many diverse fields such as agriculture, autonomous vehicle, facial recognition, medical imaging, manufacturing, and many more. Convolutional Neural Network (CNN) is widely recognized as a breakthrough innovation in this area which was introduced in 1998 [6] for hand-written digit recognition tasks from images. CNN extracts spatial information from images which helps with the recognition and classification tasks. Since then, several groundbreaking innovations [7]–[9] have been achieved to improve this technology further. Therefore, utilizing a CNN-based architecture to extract spatial features from video frames is a valuable addition to automatic underwater video analysis.

B. Vision Transformer (ViT)

Following the immense success of the Self-attention based Transformer [10] in the field of Natural Language Processing (NLP), it has also evolved in a wide range of applications within Computer Vision. Researchers thrived to adapt the self-attention mechanism in the Computer Vision area and introduced the Vision Transformer [11] in 2020 as the counterpart of the original Transformer. ViT addresses image recognition tasks by dividing an input image into patches and applying self-attention to these patches to obtain spatial contextual relations between them. Thus, it has been adapted together with traditional CNN architectures for image recognition tasks [12]–[14]. The revolution of the ViT has also shifted through different variations to other vision tasks including object detection [15], [16], and image segmentation [17].

We are particularly interested to train a multi-label ViT image classifier on LIACI dataset because of its outstanding self-attention mechanism. This facilitates better spatial feature extraction on frames during video analysis. ViT applies a standard NLP-suited transformer on an image which is first split into fixed-size patches in order to make the fewest possible adjustments. The list of patches is similar to tokens or words of NLP applications which are fed to the transformer network as inputs. This approach is called patch embedding. In order to get positional information, standard 1D position encoding is added along with the input sequence of patches. The rest of the architecture is designed by the transformer encoder layers where a learnable embedding is prepended to the embedded patches sequence. One major limitation of ViT is that it needs to be pre-trained on large-scale datasets and then fine-tuned on smaller datasets to surpass CNN for downstream tasks. While pre-training, a Multi-layer perceptron (MLP) based classification head is integrated with one hidden layer. The MLP layer is later replaced by one single linear layer during fine-tuning. Recently, a study [18] has shown that ViT can outperform CNN models of similar size when trained on ImageNet from scratch without strong data augmentations which overcome the large-scale pretraining limitation. Therefore, it is apparent that ViT holds promises for the underwater video analysis domain as well.

C. Temporal Action Localization (TAL)

To study video understanding, we need to start with extracting temporal information from the frames of a video. Temporal Action Localization (TAL) [5] refers to determining the time intervals in a video that contains a target action. The target action is usually a dynamic activity (e.g., marine plant waving, fish swimming) but can be a stationary fact as in our case which lasts for an indefinite duration such as corrosion in a ship hull. TAL mainly performs two tasks; recognition and localization. Recognition denotes the detection of the class labels whereas localization determines the start and end time of the detected actions. The latter does not apply to our work at the moment as we only focus on multi-label class recognition.

Generally, there are two types of TAL methods: single-stage and two-stage; single-stage: generates several temporal action segments (start to end) proposals in an untrimmed long video and classifies these actions simultaneously, two-stage: first proposes segments and classifies actions and then regresses the boundaries. In addition, there are a couple more variations depending on the data annotations;

- **Fully-Supervised Temporal Action Localization (F-TAL):** It refers to the training when the dataset contains both the video-level category classes and the temporal annotations (start and end time) of the action segments.
- **Weakly-Supervised Temporal Action Localization (W-TAL):** In the realistic scenario, most of the videos are untrimmed with no temporal information and contain many frames that are not relevant to target actions. So it is very difficult to acquire temporal annotations.

W-TAL indeed coincides with our case as we have only untrimmed underwater videos without annotations. However, the implementation of video classification requires video annotation. This needs extensive time to prepare the data for training a deep learning video classifier. Hence, we follow a similar W-TAL approach to train our multi-label video classifier.

D. Spatiotemporal features in video classification

In video understanding, the improved Dense Trajectories (iDT) proposed in [19] was the state-of-the-art hand-crafted feature for classification tasks. The iDT descriptor demonstrates the ability to extract temporal features differently from that spatial information. Consequently, 3D ConvNets was proposed in [20] to learn spatiotemporal features from videos. It also overcomes the limitation of 2D ConvNets which loses temporal information of the input signal right after every convolution operation. The best architecture proposed in their experiment, called C3D net, is homogeneous and comprises 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. The 3D convolution kernels in this network are 3x3x3 with a stride of 1 in both spatial and temporal dimensions. They also claimed that a trained C3D network can serve as a potential spatiotemporal feature extractor for other video analysis tasks which could be advantageous in our scenario.

TimeSformer [21] is among the first video models to incorporate self-attention mechanisms in video understanding inspired by the success of self-attention mechanisms in ViT. It utilizes self-attention over both spatial and temporal dimensions of an input video sequence rather than using 3D CNN to extract temporal features along the frames. The model takes an input snippet consisting of 8 RGB frames of size 224x224, decomposes each frame into 16x16 patches, and applies self-attention along the temporal patches for these 8 consecutive frames. During inference, it uses 3 spatial crops from the temporal clip and predicts by averaging the scores. In contrast to our approach of using consecutive frames to predict static class labels in the current frame, TimeSformer samples the 8 frames of an input video at a rate of 1/32, and these frames are not necessarily consecutive. Their experiments have demonstrated that the best performance is achieved when temporal and spatial attention are applied separately. Adopting this approach will be crucial in training our model video classifier.

ViViT [22] is another example of a transformer-based video classification model that benefits from the self-attention mechanism. They propose four variations of their model by factorizing the spatial and temporal dimensions in different ways, ranging from simple to complex architectures. They also explain how to utilize pre-trained ViT image models to train a video classifier on small datasets along with effective regularization techniques which could be particularly advantageous for our purposes. They emphasize the operational flexibility of a variable number of input frames which is similar to the original transformer's ability to handle any sequence of input tokens. While there are similarities with TimeSformer [21], the rich ablation study presented in ViViT provides a strong foundation for us to begin with our own video model.

In essence, the video models based on 3D CNN or transformers provide a promising research direction for developing a suitable multi-label video classifier for underwater ship inspection. Although the underlying architecture of our model will be similar to these models, it will serve a different purpose. Our model will predict static classes instead of dynamic actions by absorbing the disrupted motions in the video and will stabilize the prediction confidence along the temporal dimension.

III. Materials & Methods

A. Datasets

The LIACI dataset for underwater ship Lifecycle Inspection, Analysis, and Condition Information is publicly available and has been published in [23]. The dataset comprises 1893 RGB images extracted from 17 inspection videos of various ships. There are 10 class labels as depicted in Fig. 2 divided into two different categories;

- **Ship components:** *Anode*, *Bilge keel*, *Overboard valve*, *Propeller*, *Sea chest grating*, and *Ship hull*.
- **Common marine coating issues:** *Marine growth*, *Paint peel*, *Corrosion*, and *Defect*.

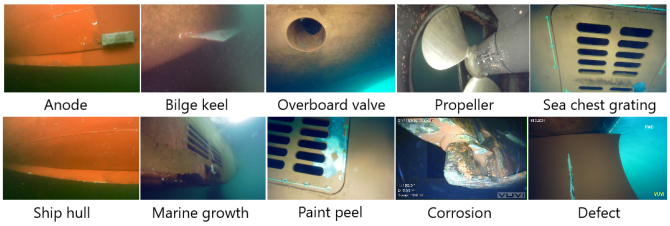


Fig. 2. Visualization of 10 class labels of two different categories.

However, we exclude the *Ship hull* class during the training of our deep learning model as it is present in all images. We only used 1561 images from the LIACI dataset to train and test our model as recommended by the authors [23]. The remaining 332 images were considered too spatially similar to other images in the dataset (Cosine similarity cut-off of 0.90). The class instance distribution in Fig. 3 indicates that while the dataset is not perfectly balanced, it is not severely imbalanced either.

Furthermore, to comprehensively analyze and evaluate the performance of trained models, we selected 8 key clips of 1920x1080 resolution from an underwater inspection video. These clips were chosen randomly from untrimmed inspection

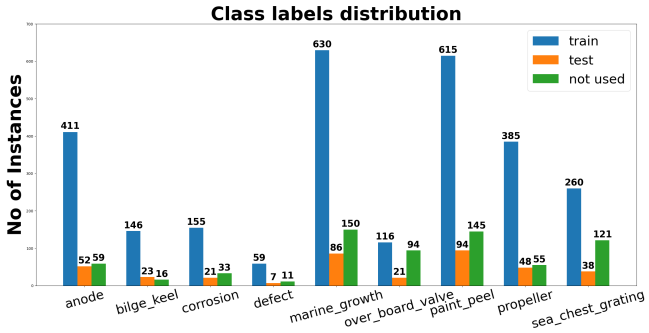


Fig. 3. Distribution of class instances.

videos and each clip is approximately 14 seconds long. Table I provides descriptions of the physical content of the clips that are easily recognizable to human eyes. However, distinguishing between *marine_growth*, *corrosion*, and *paint_peel* with human visual perception can be quite challenging most of the time. The results of the analysis and evaluation are documented in sections IV and V.

TABLE I
CONTENTS OF THE 8 KEY VIDEO CLIPS.

Serial	Major physical real contents
1	anode, paint_peel
2	bilge_keel, paint_peel, over_board_valve, anode
3	propeller, paint_peel, corrosion, marine_growth
4	paint_peel, marine_growth, propeller
5	marine_growth, propeller, corrosion
6	paint_peel
7	propeller, marine_growth
8	sea_chest_grating, paint_peel, corrosion

B. Multi-label ViT Image Classifiers

In [11], a few variants of ViT models are proposed that differ in model size and input patch size. For instance, the ViT-L/16 refers to the "Large" variant and is composed of 24 training layers with a 16x16 input patch size. The PyTorch [24] vision package includes several ViT models that can be easily implemented. Besides, PyTorch enables access to the models' underlying architecture and allows us to modify them through retraining or fine-tuning conveniently. Based on the model's capacity, our requirements, and computing resources we selected the ViT-B/16 architecture. The size of the model is 330.3MB with 86M trainable parameters and it has 95.318%@5 accuracy on ImageNet 1K dataset [25].

We decided to train two versions of the ViTs on LIACI data with pre-trained on ImageNet 1k and COCO 2014 [26] datasets respectively and compare their performances. Although the ImageNet pre-trained ViT is readily available in PyTorch, we need to train the COCO version by ourselves in advance. We downloaded the COCO dataset using FiftyOne [27] and fully finetuned an ImageNet pre-trained ViT model on COCO. Finally, we trained our two desired ViT models pre-trained from ImageNet and COCO datasets and abbreviated them as IMAGENET_ViT and COCO_ViT respectively. The training hyperparameters are the same for both as shown in Table II along with the data transformations. It is noted that we applied separate image normalization by computing respective mean (M) and standard deviation (S) on LIACI and COCO datasets. Also, only the Image Resize and Normalization are applied during validation or evaluation. Nonetheless, we investigated various hyperparameters and data augmentations that are exhibited in section V.

C. Prediction Confidence and Temporal Characteristics

To analyze a trained model's confidence behavior, we leverage OpenCV [28] to process a video snippet and observe the model's prediction confidence on each frame, as illustrated in Fig. 4. This approach also enabled us to evaluate a model's ability to predict multiple class labels simultaneously on a per-frame basis.

To integrate temporal reasoning into our model, it is necessary to examine and analyze the model's temporal consistency throughout the development process. To achieve this, we utilize OpenCV to observe the temporal aspect of the model's confidence for different labels during an inspection. This is useful to qualitatively assess the temporal stability of a trained model and is depicted in the result section.

TABLE II
TRAINING HYPERPARAMETERS AND DATA TRANSFORMATIONS FOR IMAGENET_ViT AND COCO_ViT

Type	***IMAGENET_ViT***	***COCO_ViT***
Loss function	BCEWithLogitsLoss	BCEWithLogitsLoss
Optimizer	SGD	SGD
Learning rate	0.001	0.001
Momentum	0.9	0.9
Batch size	16	16
Scheduler	StepLR (step=20, gamma=0.1)	StepLR (step=20, gamma=0.1)
Data Transformations		
Image Resize	224x224	224x224
Normalization (LIACI Data)	M[0.348, 0.369, 0.352] S[0.249, 0.244, 0.206]	M[0.348, 0.369, 0.352] S[0.249, 0.244, 0.206]
Normalization (COCO Data)	N/A N/A	M[0.485, 0.456, 0.406] S[0.229, 0.224, 0.225]
Random Horizontal Flip	p=0.5	p=0.5

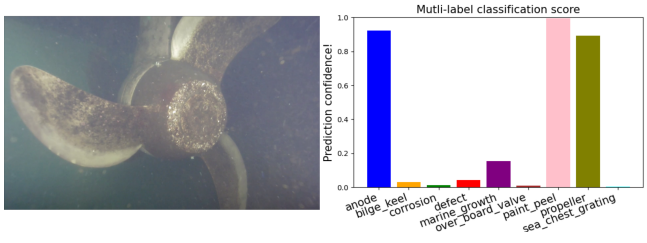


Fig. 4. Model's prediction confidence on a frame during a video inspection.

D. Underwater Image Quality Metrics

In underwater image or video tasks, measuring image quality is a grave concern as it directly impacts any vision-based operation. Poor-quality images can significantly degrade the performance. To measure frame quality, we employed two separate image quality metrics - UCIQE [29] and UIQM [30] - to establish a correlation between the model's prediction confidence and frame quality. Both metrics are no-reference and meticulously designed for underwater images. The qualitative output of these two metrics is reported in the result section.

E. Video data Generation and Annotation

We have acquired the corresponding videos of LIACI training images which are untrimmed and unstructured video data. We were able to extract 755 corresponding video snippets out of 1893 images contained in the dataset. Each snippet consisted of seven consecutive frames, with the middle frame representing the original image from the LIACI dataset and

its class labels considered as the labels for the entire snippet during training. This approach may be considered a weakly supervised data annotation. The snippets were split into 584 for training, 87 for validation, and 84 that were not used by following the same splitting convention of the image dataset. It is worth noting that the generated video dataset contains fewer snippets than half of the number of images in the LIACI dataset. As a result, it may not be sufficient to train a robust video model compared to the image model.

F. Multi-label Video Classifiers

We have implemented and trained 6 different variants of ViT-based multi-label video classifiers. Initially, we adopted a straightforward method by utilizing the spatiotemporal token embedding techniques proposed in [22]. We trained our first 2 variants by extracting tokens from the video snippets using either uniform frame sampling or tubelet embedding methods, and then input these tokens directly into a base ViT encoder. The process is illustrated in Fig. 5, and the diagrams used are borrowed from [22] and [11]. To implement uniform frame sampling, we extracted 28 patches with dimensions of 32x56 from each frame of a seven-frame input snippet, generating a total of 196 patch embeddings. These embeddings are readily compatible with a base ViT architecture. On the other hand, to achieve the tubelet embedding as depicted in Fig.5, we utilized a pretrained 3D ResNet18 model to extract C3D features from the input snippet.

The rest of the 4 video classifiers are implemented by applying different underlying strategies based on Model 2 proposed in [22] which is similar to the TimeSformer method presented in [21]. This approach uses a ViT base architecture called a spatial transformer encoder to extract spatial features from each frame. These consecutive spatial features are then passed through a temporal transformer to combine with temporal features, followed by an MLP head to predict class labels. This method is designed to address the issue of overfitting on smaller datasets such as ours and provides a more sophisticated model for video classification. A previously trained ViT image classifier is adopted as the spatial transformer encoder, while a new standard transformer is employed as the temporal one. During training, we froze the weights of the spatial transformer and solely updated the temporal transformer. This approach resulted in a notable acceleration of the training process and facilitated the adaptation of the models to finetuning tasks.

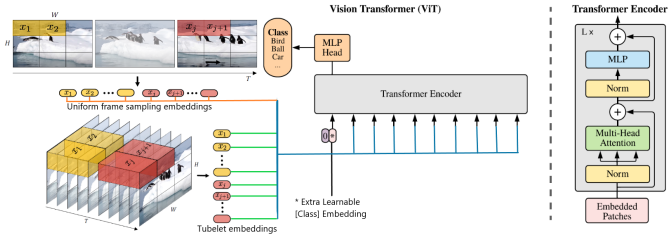


Fig. 5. A simple approach to video model using the same architecture as the image classifier.

G. Multi-label Evaluation Metrics

The computation of multi-label classification evaluation metrics is different from multi-class classification. The Scikit-learn Python package [31] provides essential tools to easily compute different metrics. We report accuracy, precision, recall, and f1-score on the validation set of LIACI data for our image and video models in section IV. These metrics are calculated along the instances and averaged over them. The mathematical equations are as follows in Eq. (1), (2), (3), and (4) where n is the number of images, y is the ground truth, and $\hat{y}$ is the predicted label. Besides, we computed class-wise evaluation metrics during some analysis in section V.

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|} \tag{1}$$

$$Precision = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|\hat{y}_i|} \tag{2}$$

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i \cap \hat{y}_i|}{|y_i|} \tag{3}$$

$$F1 - score = \frac{1}{n}\sum_{i=1}^{n}\frac{2|y_i \cap \hat{y}_i|}{|y_i| + |\hat{y}_i|} \tag{4}$$

H. Hardware Resources

We used NVIDIA RTX 2080 Ti (11GB) and RTX A6000 (48GB) GPUs to train both of our image and video models. For inference and testing, we used a local system that constitutes of NVIDIA GTX 980 (4GB) with Intel(R) Xeon(R) CPU E5-1650v3 @3.50GHz and 32GB RAM.

IV. Results

The temporal observation of video clip no.3 from Table I is illustrated in Fig. 6 using a model trained on the LIACI dataset through Microsoft Custom Vision in [2]. Although the model successfully detects a couple of classes, the confidence values for consecutive frames fluctuate significantly. We noticed similar behavior for other snippets even though the spatial changes between frames are negligible. The bottom row of Fig. 6 displays the output of the two image quality metrics mentioned in section III on a single video clip, while comparing them against a model's temporal prediction confidence. Since UCIQE and UIQM have different value ranges, we plot these metrics on two different scales within the same plot. Consequently, it is evident that UCIQE does not exhibit any correlation with the observed fluctuation, whereas UIQM indicates that the prediction tends to be consistent with higher UIQM values between frames 250 to 450. On the other hand, the highlighted confidence values for frames 70 and 78, differ significantly at 0.12 and 0.81, respectively, despite a negligible spatial difference between them, as shown in Fig. 7. Therefore, the rest of this section demonstrates to what extent our image and video models gradually overcome the issue.

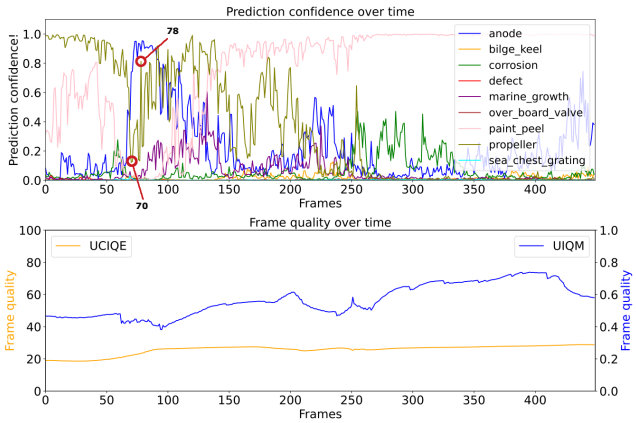


Fig. 6. Temporal observation with UCIQE and UIQM metrics on a video snippet.

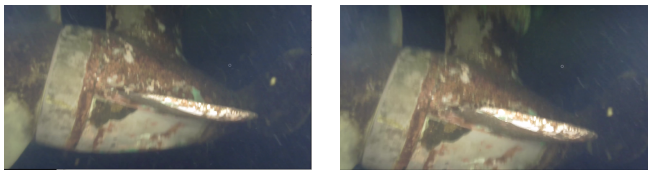


Fig. 7. Frame 70 and 78 (left to right) of a video snippet.

A. *IMAGENET_ViT and COCO_ViT Image Classifers*

Once we began the training process using the hyperparameters and transformations outlined in section III, we conducted an extensive analysis to determine the optimal models. Consequently, we found the best performances by utilizing the hyperparameters and transformations presented in Table III. A comparative quantitative evaluation for both of our models is shown in Fig. 8. While both models exhibit almost similar performances in each evaluation metric, COCO_ViT outperformed IMAGENET_ViT by a small margin in all metrics except precision.

TABLE III
OPTIMAL HYPERPARAMETERS AND TRANSFORMATIONS FOR IMAGENET_VIT AND COCO_VIT

Hyperparameters	
Loss function	BCEWithLogitsLoss
Optimizer	SGD
Learning rate	0.001
Momentum	0.9
Batch size	16
Scheduler	ReduceLROnPlateau mode='min', factor=0.1
Data Transformations	
Image Resize	224x224
Normalization	M[0.348, 0.369, 0.352] S[0.249, 0.244, 0.206]
GaussianBlur	kernel_size=(5, 9), sigma=(0.1, 5), p=0.5
AugMix() [32]	p=0.5
Random Horizontal Flip	p=0.5

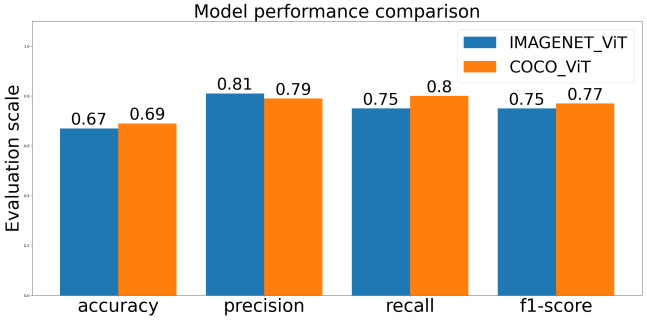


Fig. 8. Evaluation metrics comparison between our IMAGENET_ViT and COCO_ViT models on the validation dataset.

The ReduceLROnPlateau learning rate scheduler aids in finding better local minima on the validation loss. Fig. 9 shows that the final model was able to find the minimal loss on validation compared to the initial one in both cases. Increasing the loss of the final model during training compared to the initial model and subsequently reducing the loss more on the validation set leads to better regularization of COCO_ViT. On the other hand, the utilization of Gaussian blur and AugMix [32] enhanced the stability of the model's confidence in temporal analysis by facilitating the learning of abrupt ROV motion during inspections. Hence, Fig. 10 demonstrates that both models improved the stability of temporal confidence, particularly in detecting the *Paint peel* class, in contrast to Fig. 6. Furthermore, the models exhibited a more exploratory nature in detecting other class labels during the inspection which indicates improvement in multi-label competency. Similar improvements in temporal consistency were observed for the remaining testing snippets which are shown in Figure 11 alongside the outputs from video models.

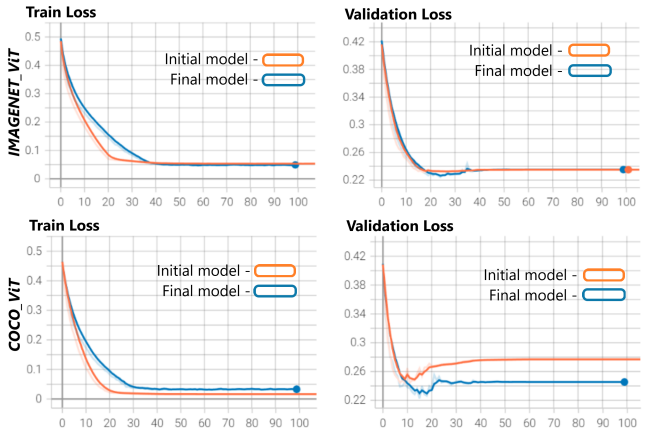


Fig. 9. Comparison between the initial and final models in finding local minima for the loss during training. The optimal validation loss for IMAGENET_ViT is within 20 to 30 epochs. COCO_ViT exhibits more regularization than the initial model.

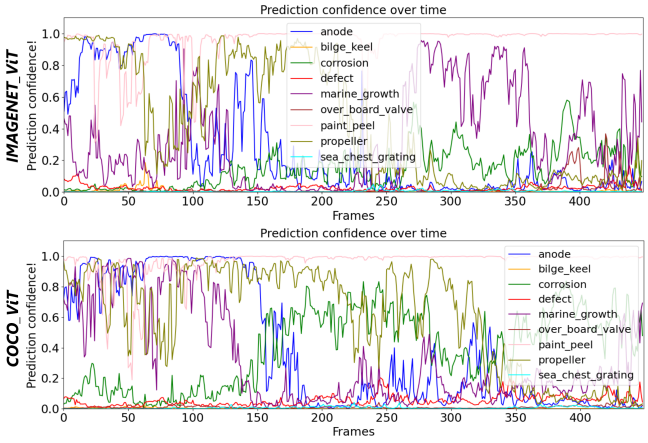


Fig. 10. Temporal observation of the final IMAGENET_ViT and COCO_ViT on the same video snippet as in Fig. 6.

B. Multi-label Video Classifiers

Our initial attempt at implementing the video model utilizing uniform frame sampling did not result in convergence. Even after training for 1000 epochs, it exhibited a train and validation loss plateauing around 0.44. Also, the second variant using C3D features as tubelet embeddings did not yield a comparative performance. Nonetheless, our final approach produced promising results in terms of video classification performance. We trained 4 variants of video models within this approach by altering the weights of the spatial transformer encoder and the underlying feature pooling strategy for both the spatial and temporal transformers. Table IV outlines the performance evaluations of these video models on the validation video dataset. A detail of all the different training experiments is provided in the ablation study. Table IV indicates that model number 3 performs slightly better than the others. Accordingly, we have included the temporal observations of this model in Fig. 11, alongside the best image model. It is evident that the video model generates smoother temporal prediction confidence scores than the image model by stabilizing the predictions along the temporal dimension. While it has introduced some variance within the same class label, we discussed further improvement in the future work section which may overcome this limitation.

TABLE IV
EVALUATION METRICS OF VIDEO MODELS ON THE VALIDATION DATASET. ST = SPATIAL TRANSFORMER, TT = TEMPORAL TRANSFORMER, AND POOL = FEATURE EXTRACTION.

	Weights (ST)	Pool (ST)	Pool (TT)	Loss	Acc	Prec	Rec	F1-score
1	COCO_ViT	cls	cls	0.30	0.59	**0.78**	0.72	0.69
2	IMAGENET_ViT	cls	cls	0.30	0.60	0.74	0.70	0.69
3	COCO_ViT	cls	avg	0.30	**0.62**	**0.78**	**0.73**	**0.72**
4	COCO_ViT	avg	avg	**0.29**	0.59	**0.78**	0.72	0.69

V. ABLATION STUDY

A. Frame-based Video Classification

To extract the best performance from image-based models for underwater ship hull inspection, several models were trained with gradual improvements by addressing the limitations of the LIACI dataset. The COCO 2014 dataset is a large-scale dataset that contains images with multiple object classes labeled in each image. In contrast, the IMAGENET dataset is primarily used for conventional image classification tasks where each image belongs to a single class. Hence, enabling our model to have multi-label classification capability, we initially train a ViT model on the COCO 2014 dataset using the hyperparameters and transformations mentioned in Table III. The COCO dataset consists of 82783 train and 40504 validation images and the model was trained for 94 epochs with a batch size of 16. We observed the model stops learning approximately after 30 epochs as both the training and validation losses become extremely low despite the accuracy still being confined under 0.7. Subsequently, we perform full finetuning of our two initial ViT models on the LIACI dataset. Table V includes the analysis of these initial models in rows 2 and 3, whereas row 1 corresponds to the COCO model. It is apparent from the F1-score or other metrics values of these two initial models that the ViT pre-trained on COCO performs better than the one pre-trained on IMAGENET.

We investigated which model performs best in extracting features from the LIACI data. To devise this, we trained variants of the COCO and IMAGENET models using partial finetuning, where all the pre-trained weights except the classification part are frozen. The results are included in rows 4 and 5 of Table V which imply that the IMAGENET version outperforms the COCO model in feature extraction. However, the overall performance of the partial finetuning approach is still below the full finetuning approach. Therefore, we decided to keep the partial finetuning approach apart from our experiments. Additionally, we experiment with changing the optimizer from SGD to Adam with a weight decay of 0.3 to train both models but this led to a significant degradation in performance. We conducted experiments to explore the effects of different step sizes on the performance of the COCO and IMAGENET models. Along with the StepLR learning rate scheduler with a gamma value of 0.1 and test two more different step sizes: 5 and 50. To summarize, using a step size of 5 led to further regularization of the COCO model, but it also induced a decline in the overall performance for both models, as shown in rows 6 and 7 of Table V. On the other hand, the step size of 50 had a tendency to overfit the training for both models as assigned in rows 8 and 9. Finally, we deduced the best models with the configuration mentioned in the result section by considering both the quantitative evaluation measures and qualitative temporal performance which are also added in rows 10 and 11. COCO_ViT is the best frame-based model which dominates all the validation evaluation metrics except the precision which is dominated by its counterpart IMAGENET_ViT.

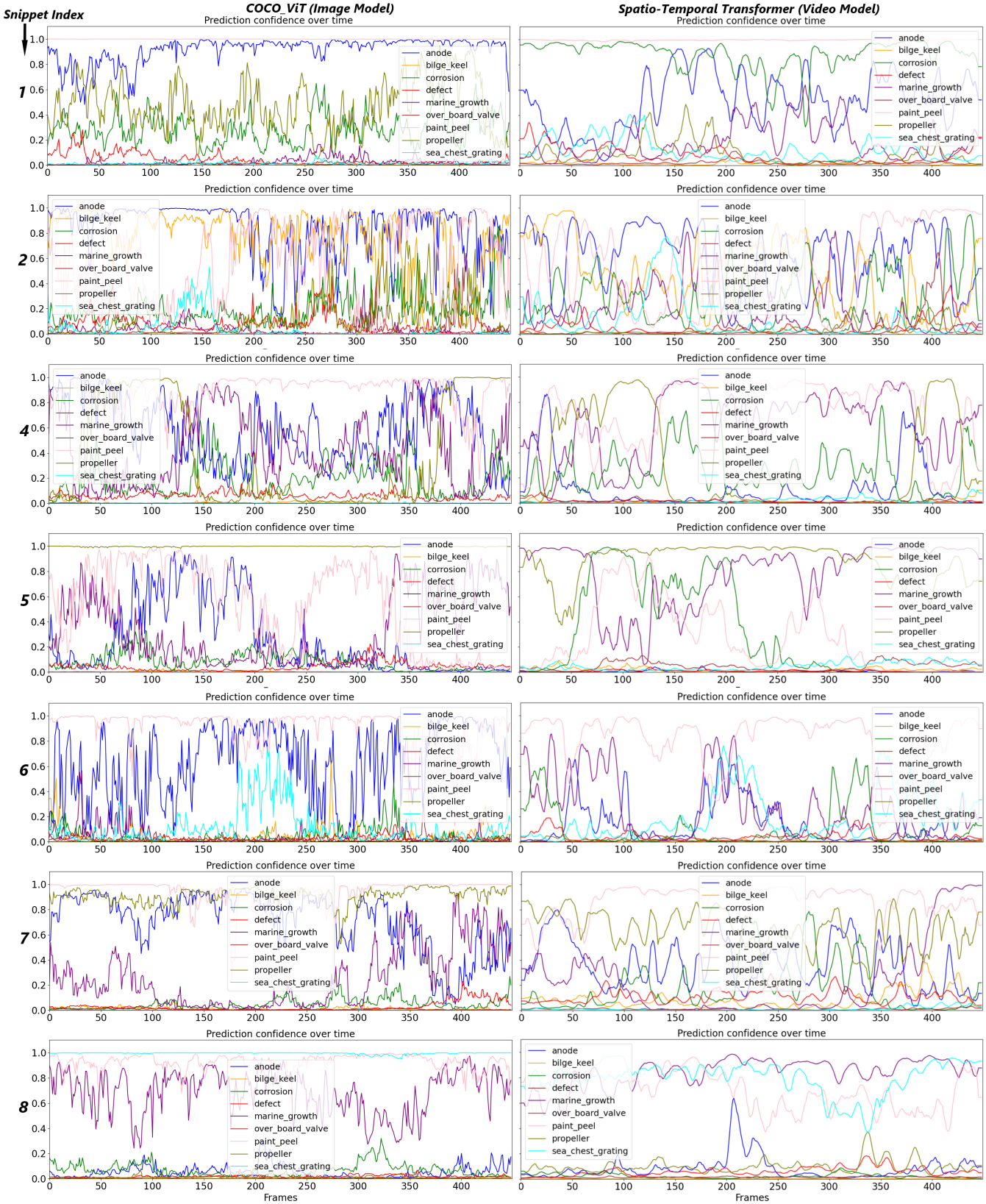


Fig. 11. Temporal consistency comparison between the IMAGENET_ViT and COCO_ViT models on the snippets from Table I.

TABLE V
ANALYSIS OF DIFFERENT MODELS & RESULTS. FF = FULLY FINETUNE & PF = PARTIAL FINETUNE.

	Model	Pretrain weight	#Epochs	Loss		Accuracy		Precision		Recall		F1-score	
				Train	Val	Train	Val	Train	Val	Train	Val	Train	Val
1	COCO ViT	IMAGENET 1K (FF)	94	0.042	0.051	0.708	0.601	0.895	0.838	0.745	0.692	0.790	0.731
2	LIACI ViT(initial)	IMAGENET 1K (FF)	301	0.054	**0.235**	0.951	0.659	0.968	**0.798**	0.952	0.723	0.958	0.729
3	LIACI ViT(initial)	COCO 2014 (FF)	326	0.016	0.277	0.970	0.673	0.971	0.797	0.969	0.760	0.970	0.749
4	LIACI ViT(extractor)	IMAGENET 1K (PF)	277	0.267	0.281	0.593	0.565	0.764	0.741	0.648	0.621	0.672	0.642
5	LIACI ViT(extractor)	COCO 2014 (PF)	276	0.320	0.323	0.484	0.479	0.648	0.637	0.536	0.534	0.559	0.552
6	LIACI ViT(step=5)	IMAGENET 1K (FF)	99	0.263	0.288	0.597	0.556	0.778	0.740	0.651	0.606	0.678	0.632
7	LIACI ViT(step=5)	COCO 2014 (FF)	99	0.170	0.260	0.779	0.614	0.902	0.792	0.810	0.667	0.834	0.695
8	LIACI ViT(step=50)	IMAGENET 1K (FF)	99	0.018	0.277	0.971	0.672	**0.972**	0.808	0.971	0.739	0.971	0.744
9	LIACI ViT(step=50)	COCO 2014 (FF)	99	**0.010**	0.315	**0.972**	0.631	**0.972**	0.768	**0.972**	0.723	**0.972**	0.715
10	LIACI ViT(final)	IMAGENET 1K (FF)	99	0.034	**0.235**	0.961	0.674	0.969	0.805	0.962	0.753	0.964	0.747
11	LIACI ViT(final)	COCO 2014 (FF)	99	0.071	0.240	0.915	**0.692**	0.936	0.786	0.947	**0.803**	0.935	**0.768**

B. Spatiotemporal-based Video Classification

With the uniform frame sampling tokenization, we attempted to train our video models utilizing both image models and experimented with different learning rate schedulers. However, none of these approaches resulted in convergence during training. It is important to note that we were limited to using a dependent patch size to generate a total of 196 image patch embeddings from 7 frames, which were then fed into a ViT model. In addition to the video models discussed earlier, we also explored an approach that involved combining 3D CNN and ViT which we referred to as the tubelet embedding approach. Specifically, we extracted C3D features utilizing a pretrained 3D ResNet architecture and subsequently passed these features through our trained ViT-based image models. Although this approach resulted in convergence during training, the performance was not competitive enough to be included in the paper.

The spatial-temporal video model we reported in the paper has a total of 161.399M trainable parameters, with 75.600M of them belonging to the temporal transformer. Since we are utilizing a pre-trained ViT classifier as the spatial transformer, we freeze its weights during training and only update the weights of the temporal transformer, resulting in a substantial reduction in training time. One significant challenge that can contribute to poor performance is the limitation of transformers, which require pretraining on a large-scale dataset to optimize their performance. This is particularly relevant for the temporal transformer in our models, as its weights are initialized randomly, which can limit its ability to learn from the available data and lead to poor performance.

VI. CONCLUSION & FUTURE WORK

We have trained several multi-label ViT image classifiers and gradually improved them on the LIACI dataset to conduct framewise video inspections. In fact, the same trained model is also utilized during training multi-label video classifiers through different state-of-the-art approaches. However, while frame-based ViT classifiers are limited by their inability to capture temporal information, video classifiers can overcome this limitation by extracting both spatial and temporal features from the video. Spatial features are dominant in some videos, making image classifiers suitable for evaluation. Considering temporal features during video classification improves the robustness of the task, making it more effective for difficult video inspections like ours, and also stabilizes the model's prediction in the temporal dimension.

Although we conducted an exhaustive analysis, we believe there is still room for improving the performance of both image and video-based classifiers in an underwater environment. For example, exploring other pretraining strategies or designing custom architectures may yield better results. Additionally, gathering more diverse and high-quality data can also improve the performance of these models. Incorporating other techniques such as data augmentation, transfer learning, or ensembling can also be explored to improve the overall performance. Besides, introducing a quantitative metric to evaluate the temporal performance of the video-based classifiers would indeed be a useful research direction. By quantifying the temporal performance, we can have a more objective measure of how well the model is able to capture temporal information in the videos. This could potentially lead to further improvements in the model architecture or training process and ultimately result in better performance for video-based classification tasks in underwater environments.

Designing a new Vision Transformer architecture that is compatible with the uniform frame sampling tokenization of 7 frames could potentially overcome the convergence issue observed previously. Pretraining this new architecture on large-scale datasets before fine-tuning it for the LIACI dataset could also improve its performance. One significant challenge we faced is the limited size and weakly supervised nature of our video dataset. To address this, it is better to explore options such as acquiring a larger fully supervised dataset, using techniques like data augmentation and regularization to enhance generalization, or incorporating pretrained weights for the temporal transformer. By doing so, we could improve the robustness and effectiveness of our video inspection models.

In conclusion, we hope this work provides a benchmark for the development of image and video-based classifiers in underwater environments. The analysis will help researchers and developers to improve the accuracy and effectiveness of these classifiers and our findings will facilitate the application

of these methods in real-world scenarios. Furthermore, we will also continue to focus on improving the video model and developing quantitative metrics to evaluate the temporal performance of video-based classifiers to improve their reliability and robustness.

ACKNOWLEDGMENT

The authors express their gratitude to the collaborators within the LIACi project, funded by the Research Council of Norway under project No 317854. The first author would like to thank the Erasmus Mundus MIR program funded by the European Union for providing his master's scholarship to study at the University of Toulon and the Norwegian University of Science and Technology. He also acknowledges Helene Schulerud, research manager of the Computer Vision group at SINTEF, for hosting him to conduct the master's thesis within the group.

REFERENCES

[1] J. Plested and T. Gedeon, "Deep transfer learning for image classification: a survey," *arXiv preprint arXiv:2205.09904*, 2022.

[2] J. Hirsch, B. Elvesæter, A. Cardaillac, B. Bauer, and M. Waszak, "Fusion of multi-modal underwater ship inspection data with knowledge graphs," in *OCEANS 2022, Hampton Roads*. IEEE, 2022, pp. 1–9.

[3] M. Salvaris, D. Dean, W. H. Tok, M. Salvaris, D. Dean, and W. H. Tok, "Cognitive services and custom vision," *Deep Learning with Azure: Building and Deploying Artificial Intelligence Solutions on the Microsoft AI Platform*, pp. 99–128, 2018.

[4] D.-A. Huang, V. Ramanathan, D. Mahajan, L. Torresani, M. Paluri, L. Fei-Fei, and J. C. Niebles, "What makes a video a video: Analyzing temporal information in video understanding models and datasets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7366–7375.

[5] H. Xia and Y. Zhan, "A survey on temporal action localization," *IEEE Access*, vol. 8, pp. 70 477–70 487, 2020.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[12] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, "Levit: a vision transformer in convnet's clothing for faster inference," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12 259–12 269.

[13] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[14] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 568–578.

[15] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[16] Z. Dai, B. Cai, Y. Lin, and J. Chen, "Up-detr: Unsupervised pre-training for object detection with transformers," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 1601–1610.

[17] Z. Li, W. Wang, E. Xie, Z. Yu, A. Anandkumar, J. M. Alvarez, P. Luo, and T. Lu, "Panoptic segformer: Delving deeper into panoptic segmentation with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 1280–1289.

[18] X. Chen, C.-J. Hsieh, and B. Gong, "When vision transformers outperform resnets without pre-training or strong data augmentations," *arXiv preprint arXiv:2106.01548*, 2021.

[19] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 3551–3558.

[20] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[21] G. Bertasius, H. Wang, and L. Torresani, "Is space-time attention all you need for video understanding?" in *ICML*, vol. 2, no. 3, 2021, p. 4.

[22] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "Vivit: A video vision transformer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6836–6846.

[23] M. Waszak, A. Cardaillac, B. Elvesæter, F. Rødølen, and M. Ludvigsen, "Semantic segmentation in underwater ship inspections: Benchmark and data set," *IEEE Journal of Oceanic Engineering*, 2022.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[27] B. E. Moore and J. J. Corso, "Fiftyone," *GitHub. Note: https://github.com/voxel51/fiftyone*, 2020.

[28] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[29] M. Yang and A. Sowmya, "An underwater color image quality evaluation metric," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 6062–6071, 2015.

[30] K. Panetta, C. Gao, and S. Agaian, "Human-visual-system-inspired underwater image quality measures," *IEEE Journal of Oceanic Engineering*, vol. 41, no. 3, pp. 541–551, 2015.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty," *arXiv preprint arXiv:1912.02781*, 2019.

OCEANS 2023 Limerick, 5-8 June

Multi-label Video Classification for Underwater Ship Inspection

M. A. Azad, A. Mohammed, M. Waszak, B. Elvesæter, M. Ludvigsen

Azad, Md Abulkalam
Erasmus Mundus Master's student
Marine and Maritime Intelligent Robotics
Department of Marine Technology, NTNU

SINTEF NTNU

Content

Background

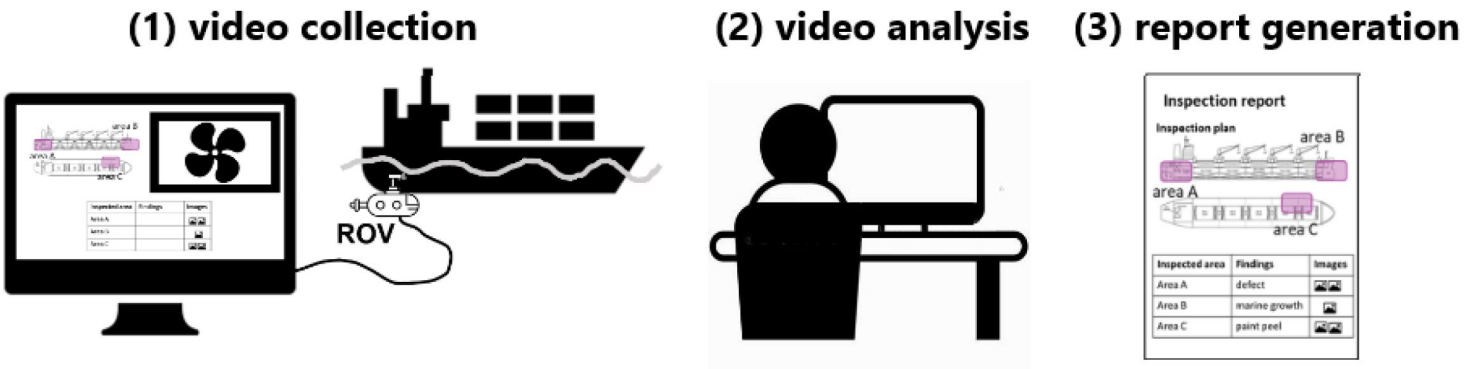


Fig. 1. The workflow of current underwater ship inspection using ROVs.

Introduction

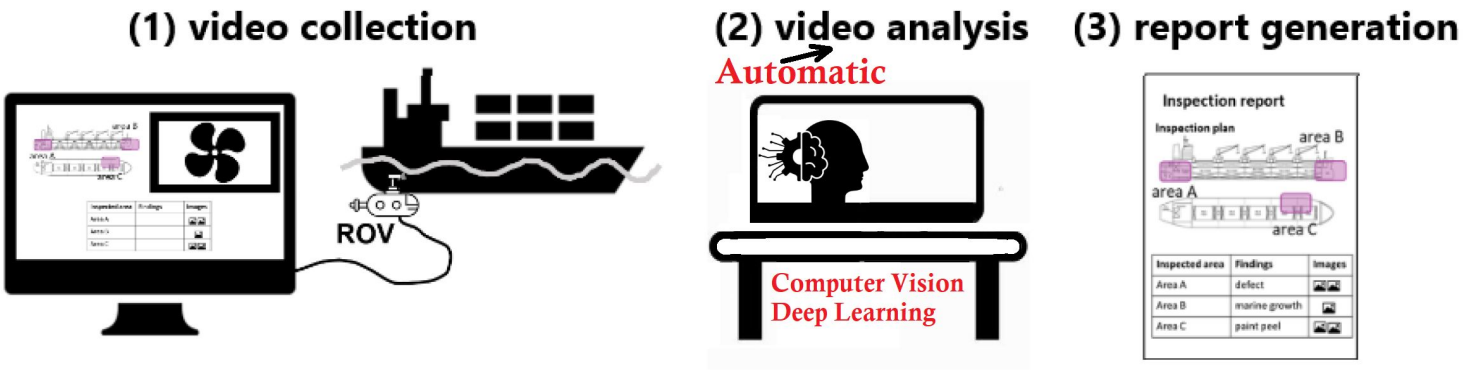


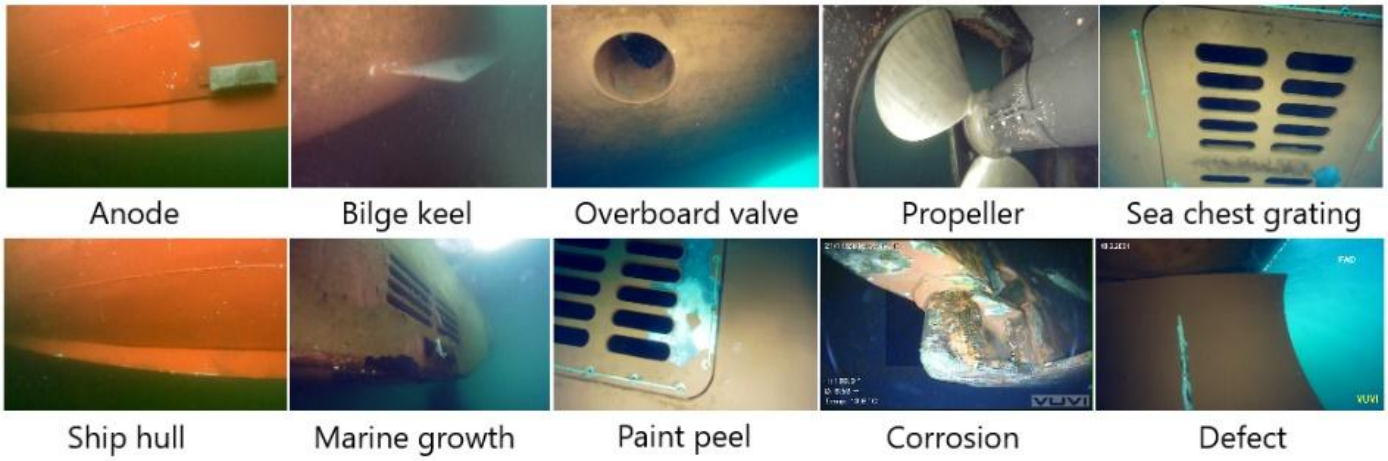
Fig. The workflow of ~~current~~ Automatic underwater ship inspection using ROVs.

Objective

1. Analysis of Image-based classifiers (benefits and limitations)
2. Exploration of temporal information in videos;
   - **Model stabilization and consistency** (**primary**)
   - **Confidence robustness in underwater environment**
3. Identification of a deep learning Multi-label Video Classifier

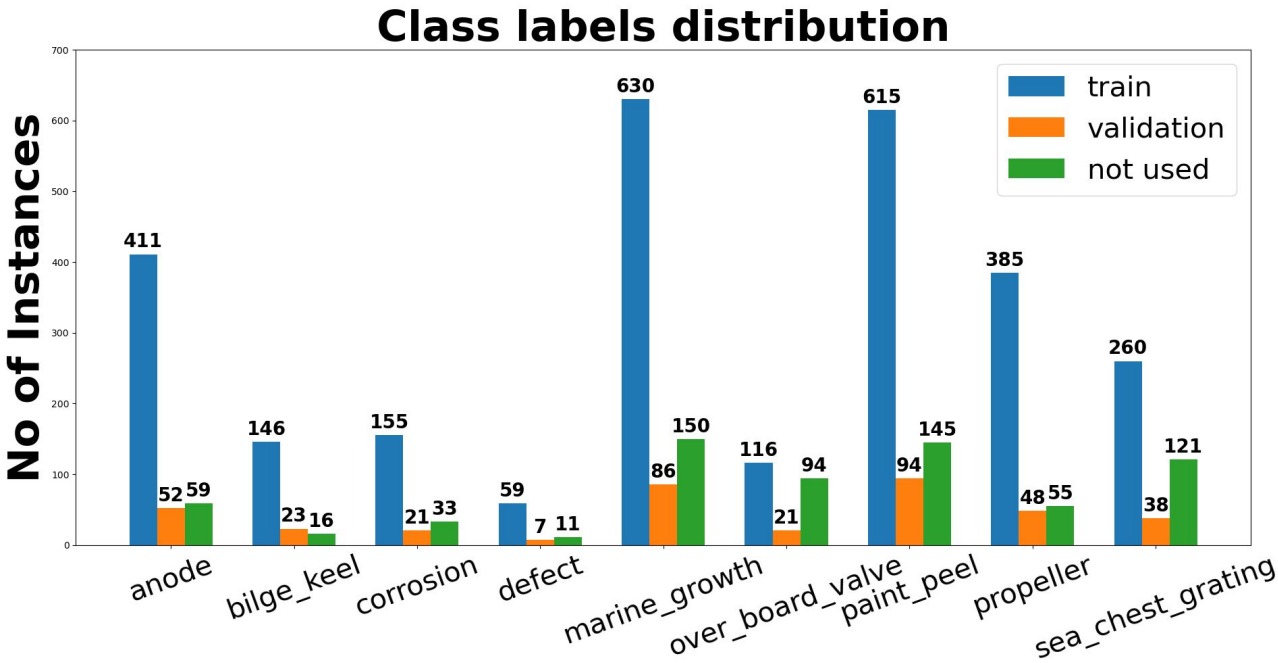
Methods & Materials

LIACI Image Dataset

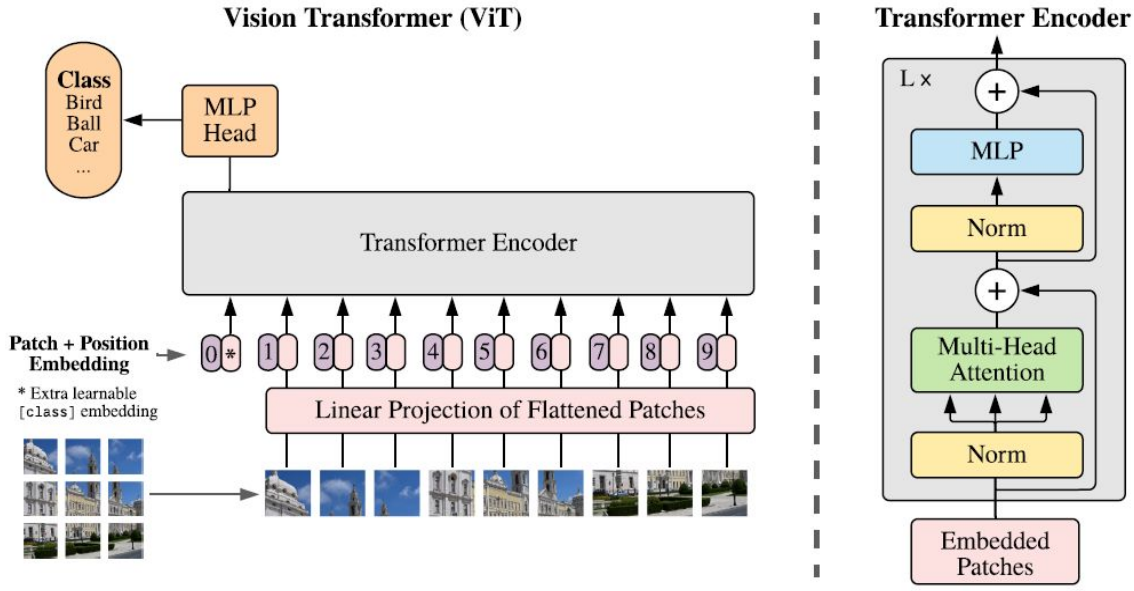


Total images = 1893, Train = 1370, Validation = 191, Not used = 332

LIACI Image Dataset

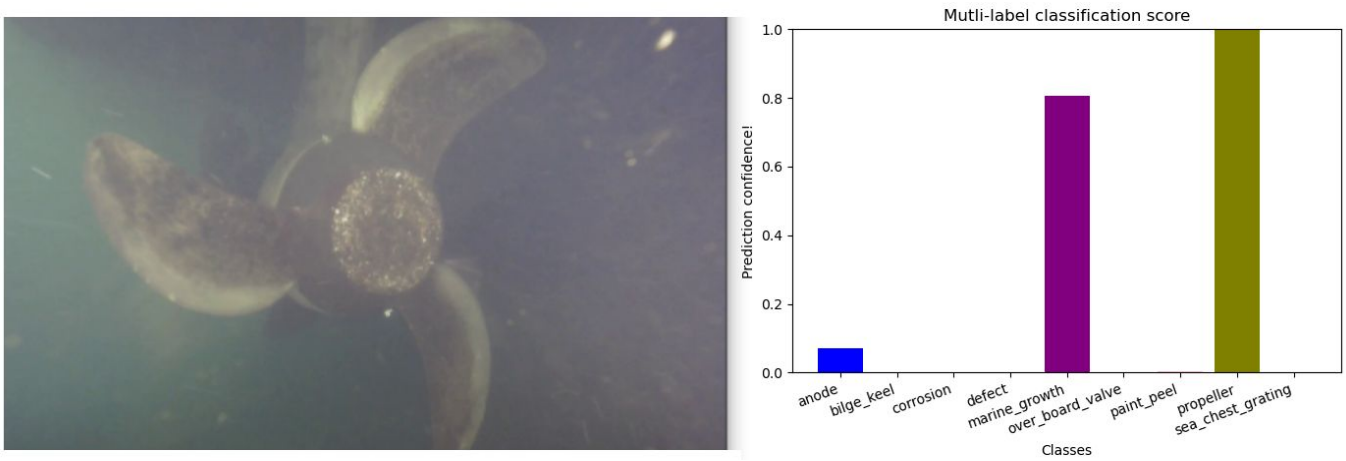


Multi-label Image Classifier



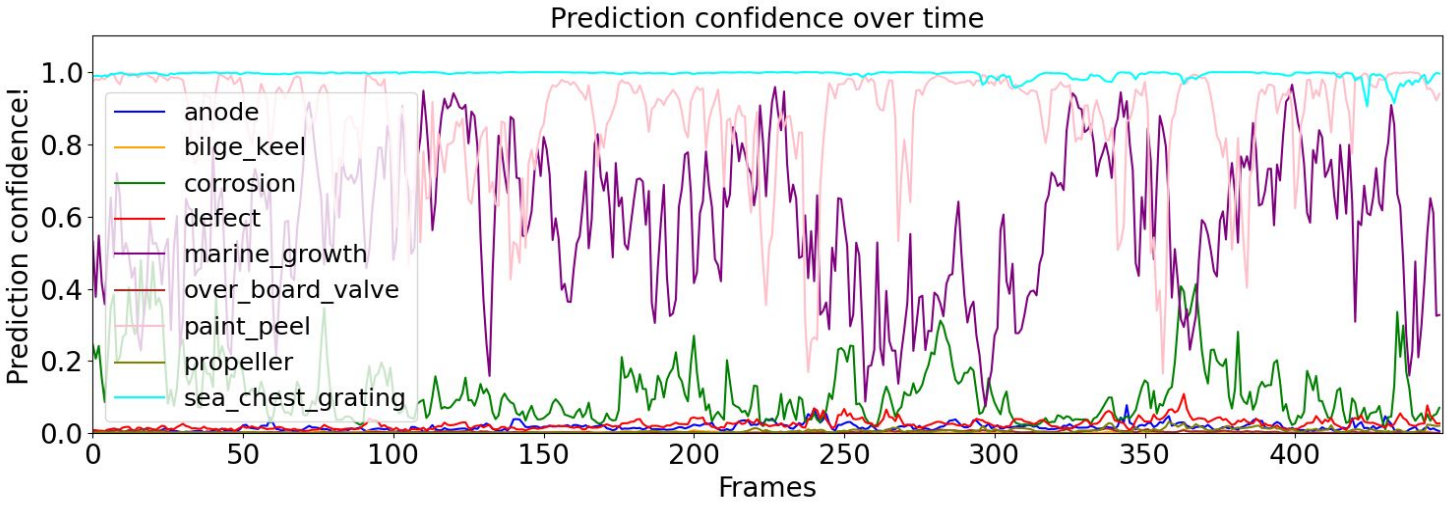
Model Architecture of Vision Transformer (ViT) [2]

Spatial Prediction Observation



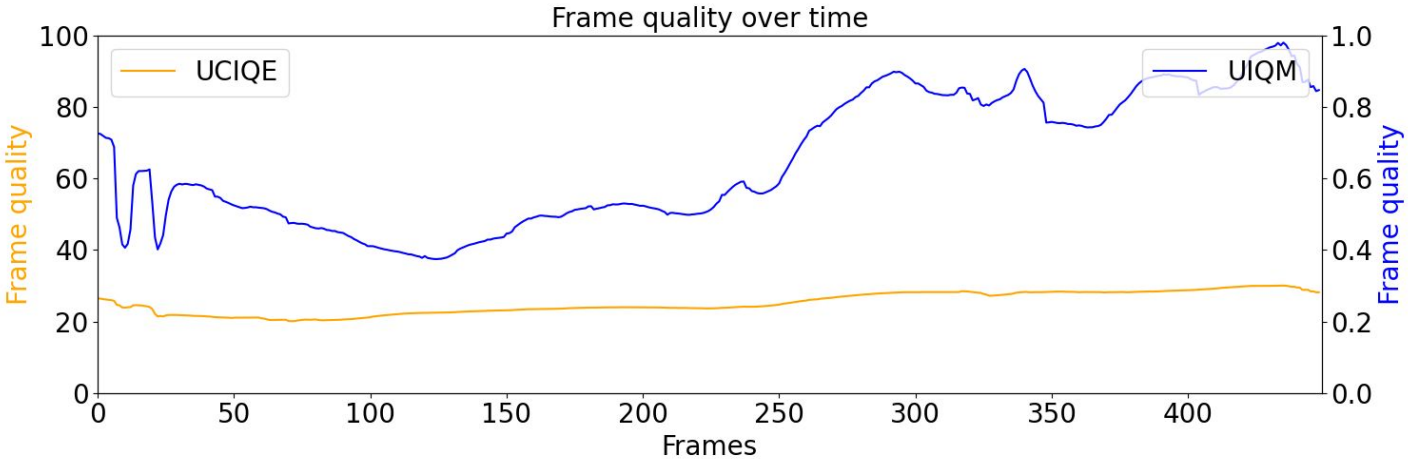
Frame-wise multi-label prediction confidence observation.

Temporal Prediction Observation



Temporal prediction confidence observation on a video snippet.

Underwater Image Quality Metrics

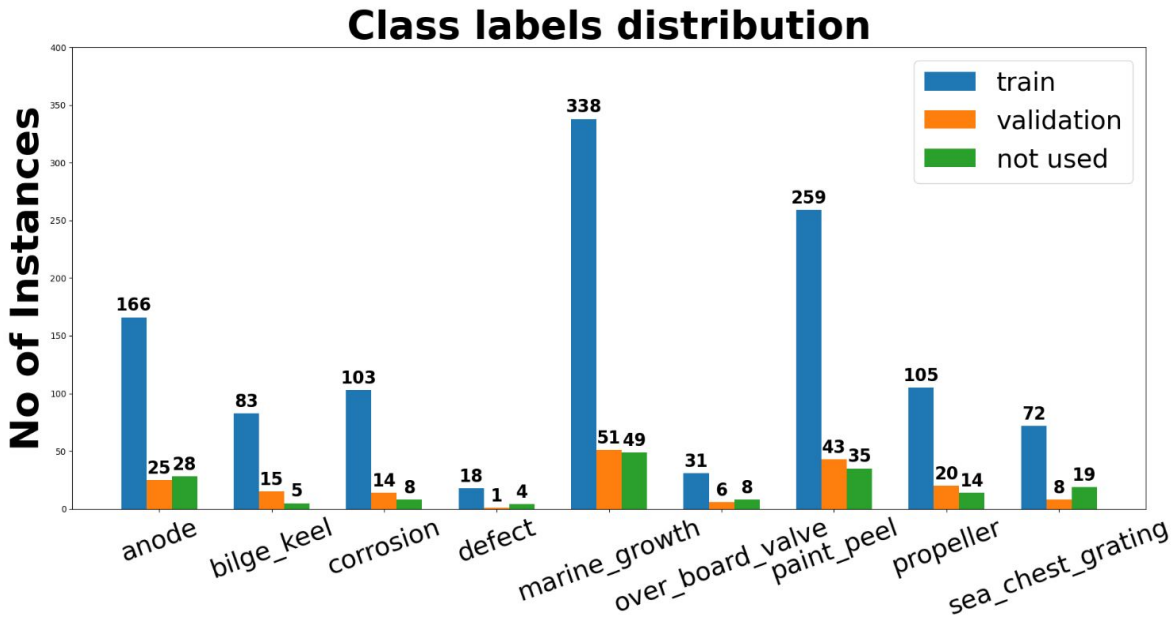


Frame quality observation using UCIQE [3] and UIQM [4].

Video Dataset

Each snippet:

- 7 Frames
- Label = middle frame



Total snippets = 755, Train = 584, Validation = 87, Not used = 84

Multi-label Video Classifier

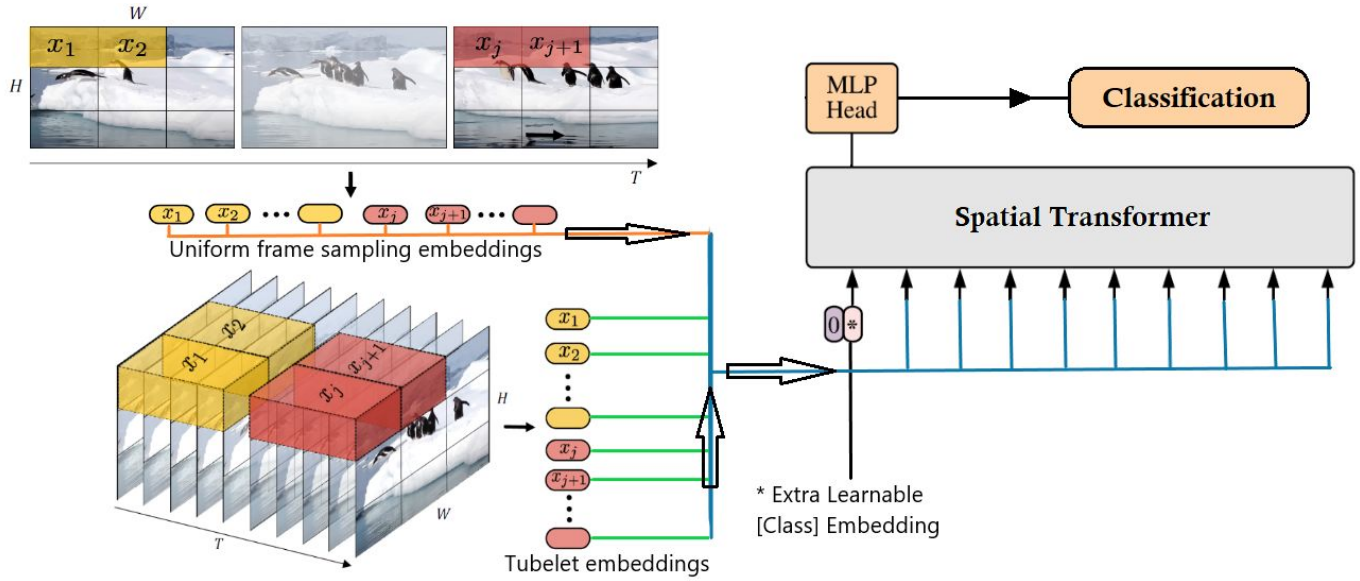
Naïve Approach (model 1):

- Spatiotemporal tokenization;
  - Uniform frame sampling [5]
  - Tubelet embedding [5]
- Image model architecture

Two-stage Approach (model 2):

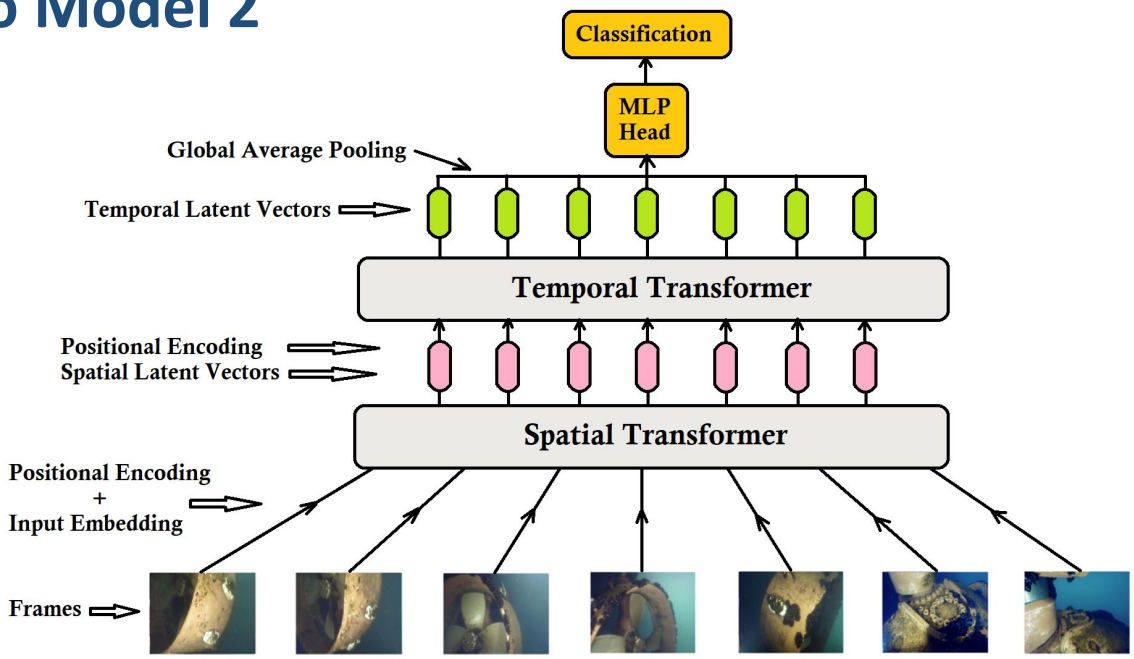
- Spatial Transformer
- Temporal Transformer
- Latent vector;
  - CLS token
  - Average

Video Model 1



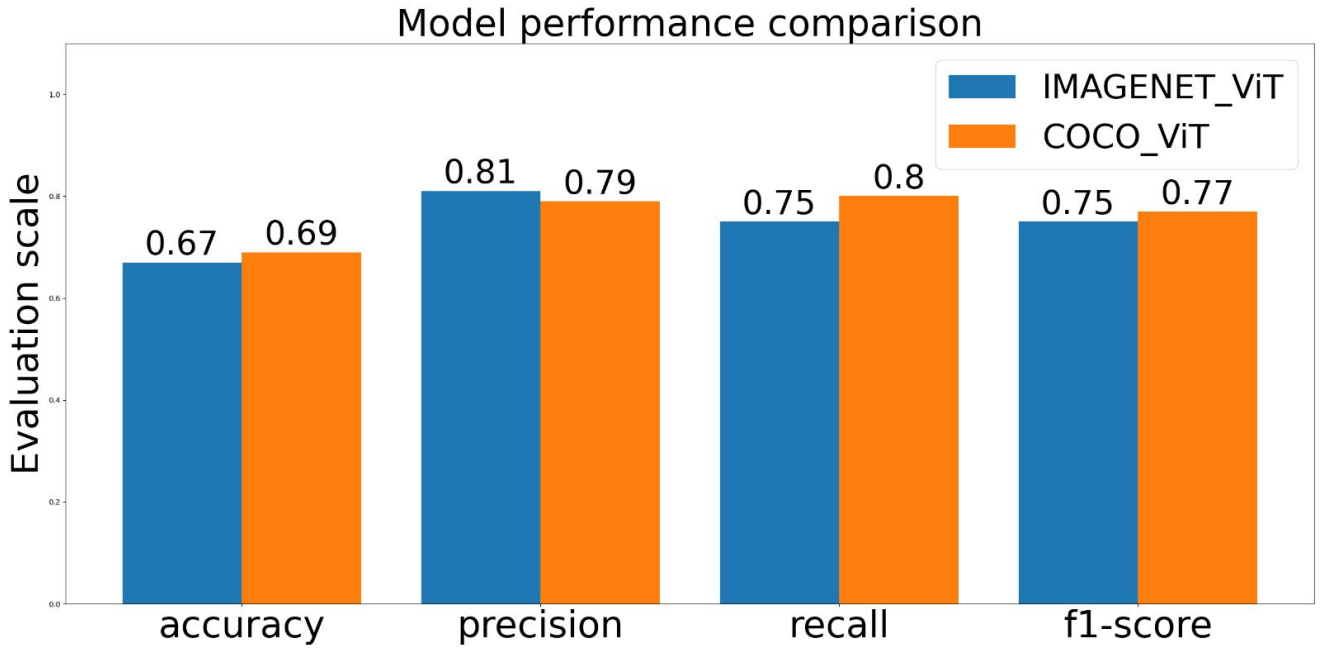
A naïve approach to video model using the image model.

Video Model 2



Spatiotemporal transformer-based video model.

Result: Image Classifiers



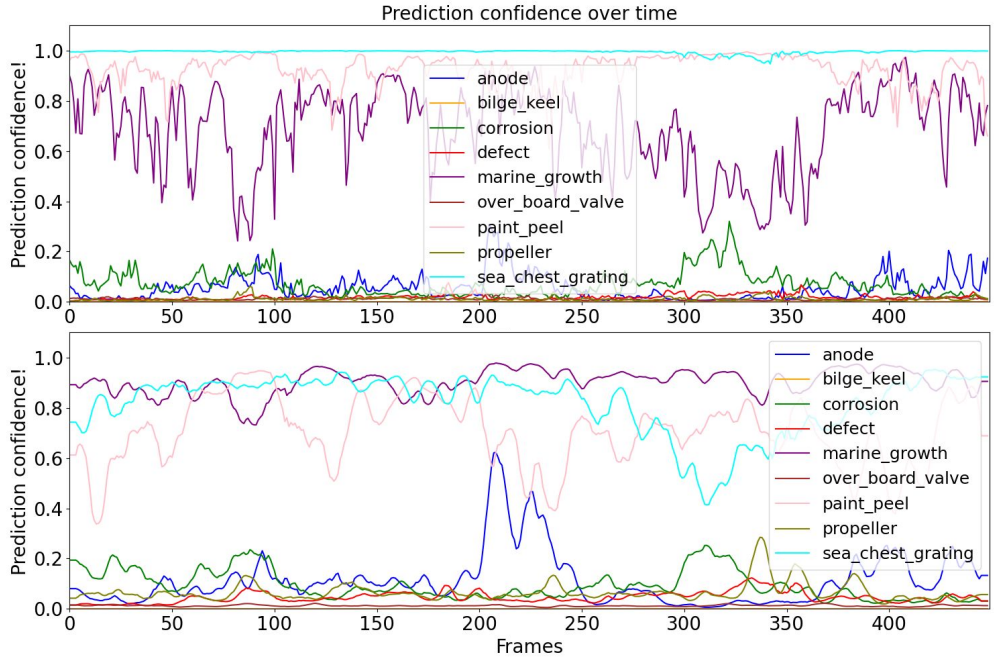
Result: Video Classifiers

EVALUATION METRICS OF VIDEO MODELS ON THE VALIDATION DATASET. ST = SPATIAL TRANSFORMER, TT = TEMPORAL TRANSFORMER, AND POOL = FEATURE EXTRACTION.

	Weights (ST)	Pool (ST)	Pool (TT)	Loss	Acc	Prec	Rec	F1-score
1	COCO_ViT	cls	cls	0.30	0.59	**0.78**	0.72	0.69
2	IMAGENET_ViT	cls	cls	0.30	0.60	0.74	0.70	0.69
3	COCO_ViT	cls	avg	0.30	**0.62**	**0.78**	**0.73**	**0.72**
4	COCO_ViT	avg	avg	**0.29**	0.59	**0.78**	0.72	0.69

The results are from the approach 2 and approach 1 did not result in convergence.

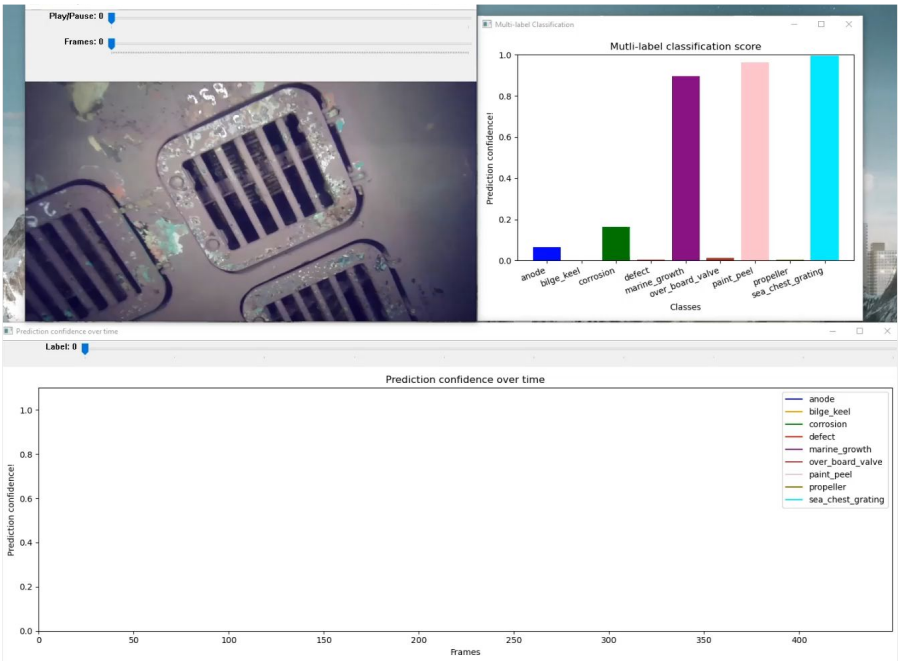
Result: Temporal Observation



Best Image Model

Best Video Model

Result: Temporal Observation



Future Work

1. **Video classifier improvement**
   - Large-scale video dataset
   - Pretrained temporal transformer
   - Customized temporal transformer architecture
   - Temporal attention-weighted prediction
   - Single query attention

2. **Quantification of temporal performance**

Conclusion

- Multi-label ViT image classifiers are trained and improved
- Different spatiotemporal tokenization is exploited
- Two different approaches towards multi-label video models are explored
- Spatial and temporal analysis are conducted
- Limitation and further research direction are provided

Acknowledgement

The Research Council of Norway

MIR MARINE & MARITIME INTELLIGENT ROBOTICS

Co-funded by the Erasmus+ Programme of the European Union

Helene Schulerud
Research Manager
Computer Vision, SINTEF Digital

References

1. M. Waszak, A. Cardaillac, B. Elvesæter, F. Rødølen, and M. Ludvigsen, “Semantic segmentation in underwater ship inspections: Benchmark and data set,” IEEE Journal of Oceanic Engineering, 2022.
2. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” arXiv preprint arXiv:2010.11929, 2020.
3. M. Yang and A. Sowmya, “An underwater color image quality evaluation metric,” IEEE Transactions on Image Processing, vol. 24, no. 12, pp. 6062–6071, 2015.
4. K. Panetta, C. Gao, and S. Agaian, “Human-visual-system-inspired underwater image quality measures,” IEEE Journal of Oceanic Engineering, vol. 41, no. 3, pp. 541–551, 2015.
5. A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lu˘ci´c, and C. Schmid, “Vivit: A video vision transformer,” in Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 6836–6846.

SINTEF

Thank You

Technology for a better society

Paper II

This is the abstract that was accepted for an oral presentation on 6th June at NORA 2023 annual conference in Tromsø, Norway. The presentation slides are also attached. We have been also invited to submit an extended version of the work that will be published in the Nordic Machine Intelligence (NMI) level-1 journal after peer review. As my thesis is covering broad works and analyses, we planned to submit a 4/10 pages article for this journal by September 2023.

MViST: A Multi-label Vision Spatiotemporal Transformer for Underwater Ship Inspection

Md Abulkalam Azad[a,b], Ahmed Mohammed[a], Maryna Waszak[a], Brian Elvesæter[a], and Martin Ludvigsen[b]
[a]*SINTEF AS, Forskningsveien 1, Oslo 0373, Norway*
[b]*Department of Marine Technology, Norwegian University of Science and Technology, Trondheim 7491, Norway*

Keywords: *Video Classification, Vision Transformer, Computer Vision, Underwater Inspection.*

The inspection of ship hulls today involves human visual analysis of underwater videos to assess the external coating, identify defects, and detect external degradation such as corrosion and marine growth. This process is time-consuming, labor-intensive, lacks reproducibility and is prone to error. To alleviate these issues, an automatic video analysis utilizing deep learning and computer vision could significantly enhance this process. At present, the video analysis is conducted on a framewise basis using multi-label image classifiers that only consider spatial information from individual frames as presented in [1]. We propose incorporating spatiotemporal information to improve the automatic indexing and summarization of underwater ship hull inspection videos. Therefore, our study aims to investigate the frame-based Vision Transformer (ViT) [2] classifier, explore the advantages of including temporal information utilizing parameter-efficient image-to-video transfer learning, and propose a novel video classification model that considers both spatial and temporal aspects of the video.

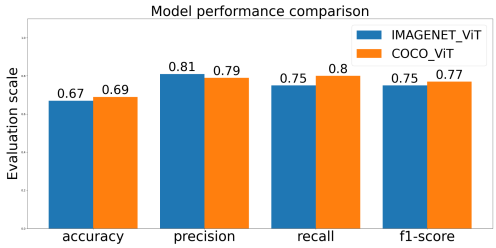


Figure 1: Evaluation metrics of different ViTs.

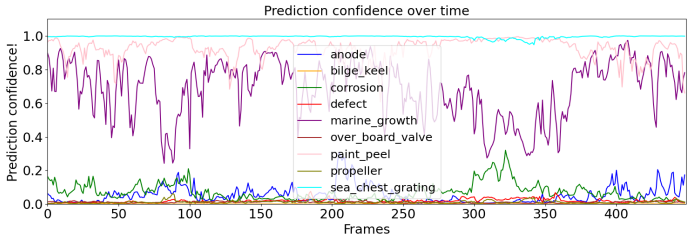


Figure 2: Temporal analysis of the model's confidence on a 14s video snippet containing sea_chest_grating, paint_peel, and marine_growth.

The LIACI dataset as introduced in [3] was used to extract the labels for different inspection categories of underwater ship hulls such as *corrosion*, *defect*, or *anode*. We have trained multiple variations of ViTs as multi-label image classifiers by analyzing our data and adequately applying relevant hyperparameters and data augmentations. Hence, our ViT models pretrained on ImageNet and COCO datasets provided good classification results as shown in Fig. 1. The temporal consistency of the model's confidence during a video inspection is also adequate as depicted in Fig. 2. The model shows high consistency in detecting sea*_chest_grating* and *paint_peel* but struggles with *marine_growth*. To further enhance stability and consistency, we plan to develop and train a novel spatiotemporal ViT model using image-to-video transfer learning from already trained ViT models. This approach will be highly efficient, as it will not require updating all weights during training, making it easy to integrate into an application. Therefore, we hope this work will be useful and serve as a benchmark for future research and development in underwater ship hull inspection.

References

[1] Hirsch, J., Elvesæter, B., Cardaillac, A., Bauer, B., and Waszak, M. (2022) "Fusion of Multi Modal Underwater Ship Inspection Data with Knowledge Graphs." *OCEANS 2022: Hampton Roads*: pp. 1–9.

[2] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., and Uszkoreit, J. (2020) "An image is worth 16x16 words: Transformers for image recognition at scale."*arXiv preprint arXiv:2010.11929*.

[3] Waszak, M., Cardaillac, A., Elvesæter, B., Rødølrn, F., and Ludvigsen, M. (2022) "Semantic Segmentation in Underwater Ship Inspections: Benchmark and Dataset." *IEEE Journal of Oceanic Engineering (Early access)*: pp. 1–12.

NORA Annual Conference 2023
Tromsø, 5-6 June

MViST: A Multi-label Vision Spatiotemporal Transformer for Underwater Ship Inspection

M. A. Azad, **A. Mohammed,** M. Waszak, B. Elvesæter, M. Ludvigsen

SINTEF Digital, Oslo

Content

Background

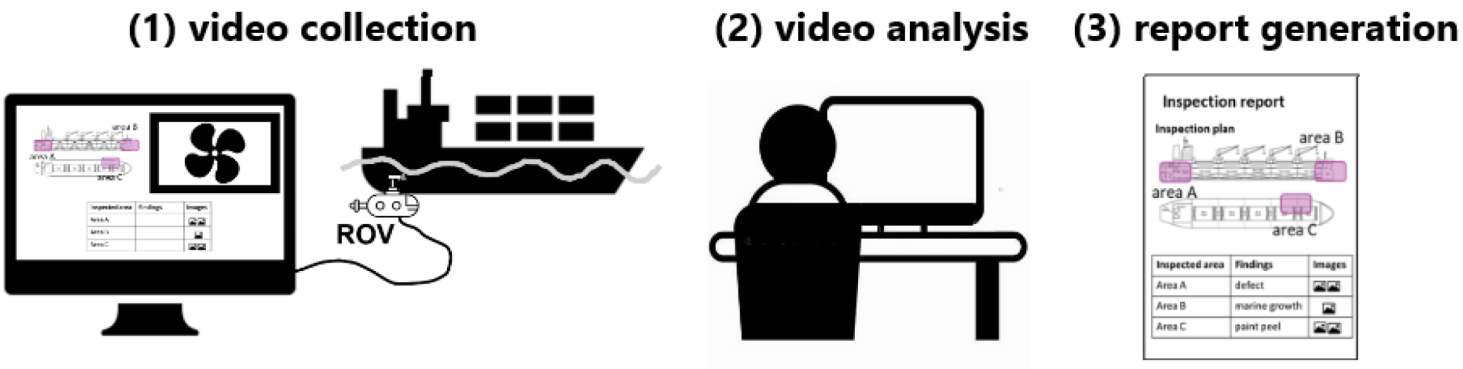


Fig. 1. The workflow of current underwater ship inspection using ROVs.

Introduction

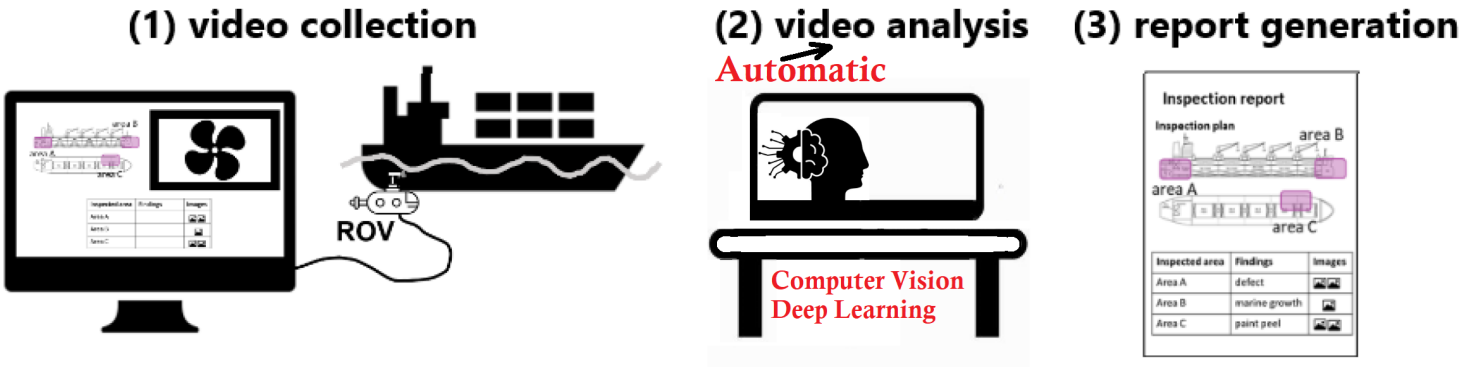
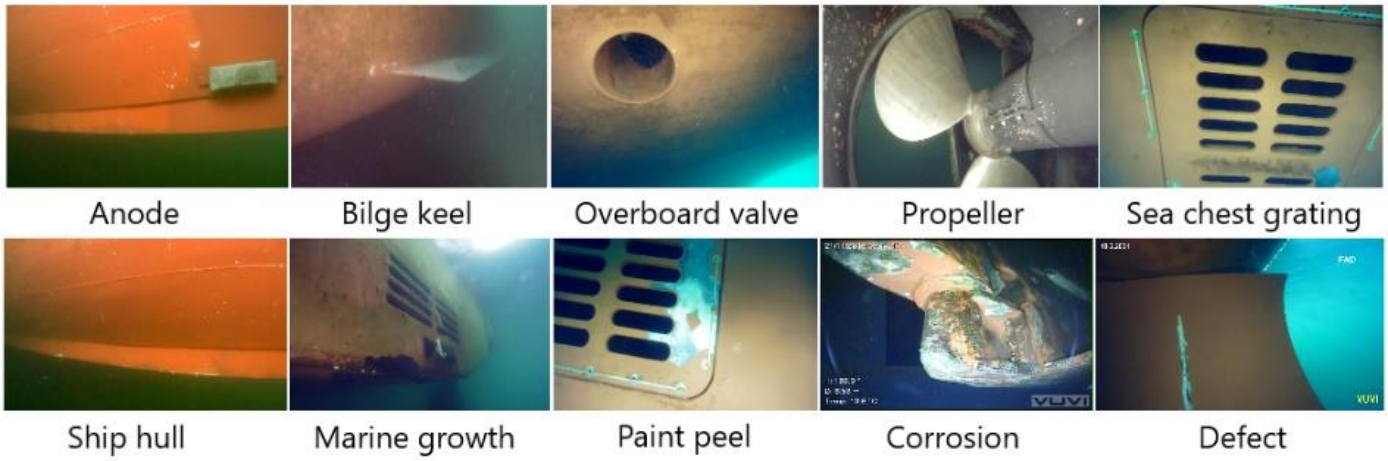


Fig. The workflow of ~~current~~ Automatic underwater ship inspection using ROVs.

Objective

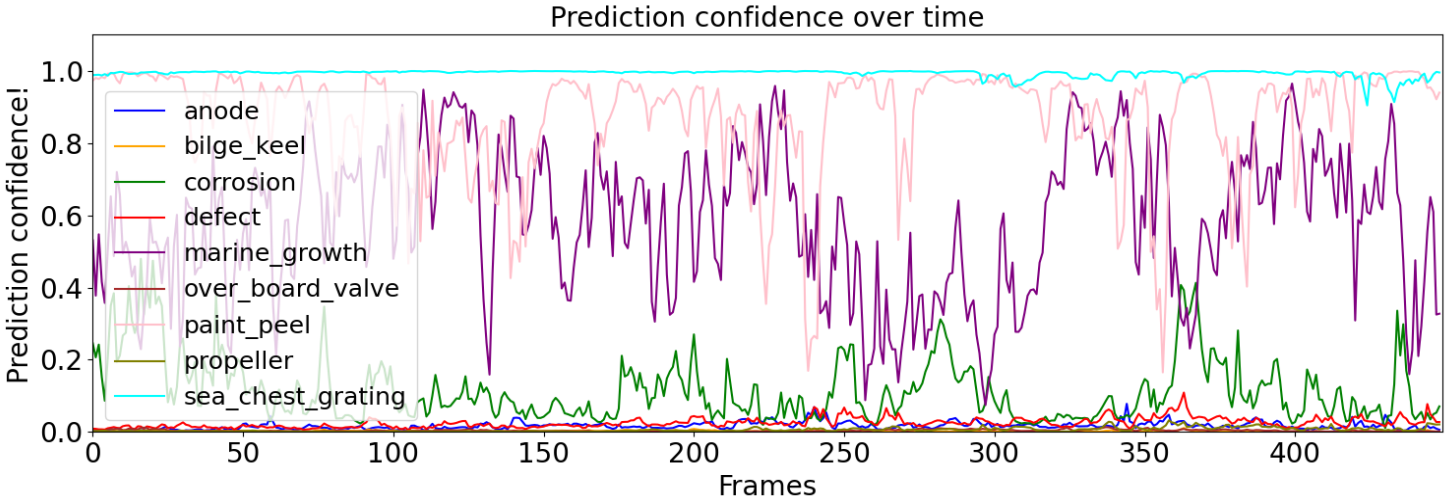
1) Analysis of the image-based (spatial-information) classifier (benefits and limitations)
2) Exploration of the advantages of adding temporal information such as;
   a. **Model stabilization and consistency** (**primary**)
   b. **Confidence robustness**
3) Weakly supervised video classification

LIACI Image Dataset



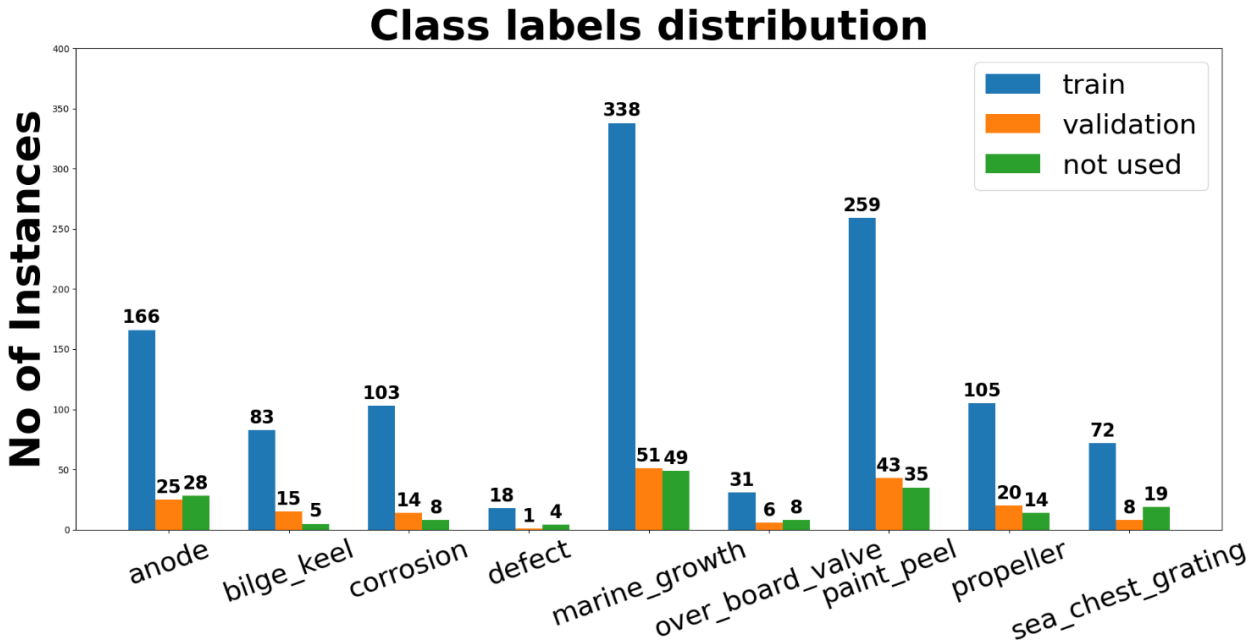
Total images = 1893, Train = 1370, Validation = 191, Not used = 332

Image based classifier



Temporal prediction confidence observation on a video snippet.

LIACI Video Dataset



Approach 1: Naive Video Transformer

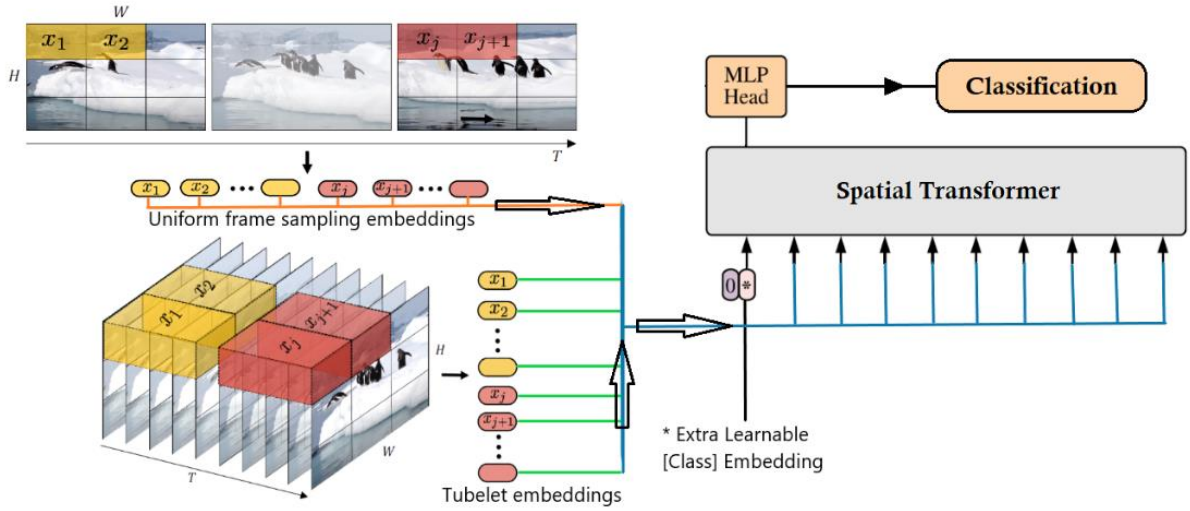


Figure 3.10: A simple approach to video model using the same architecture as the image classifier.

Approach 2: Late-Fusion Spatiotemporal Transformer

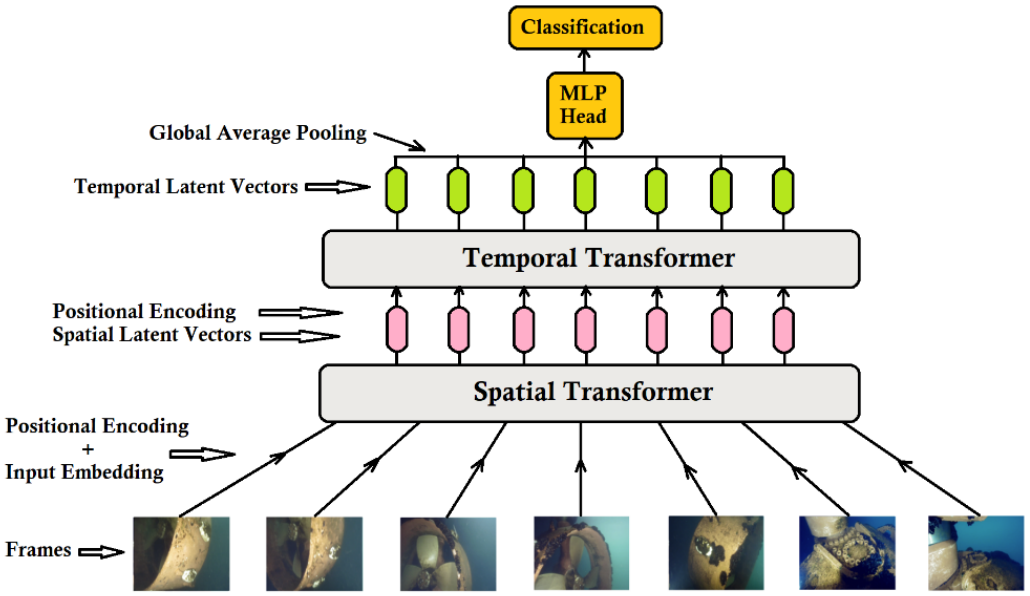


Figure 3.11: Spatiotemporal transformer architecture.

Approach 3: Attention Weighted Spatiotemporal Transformer

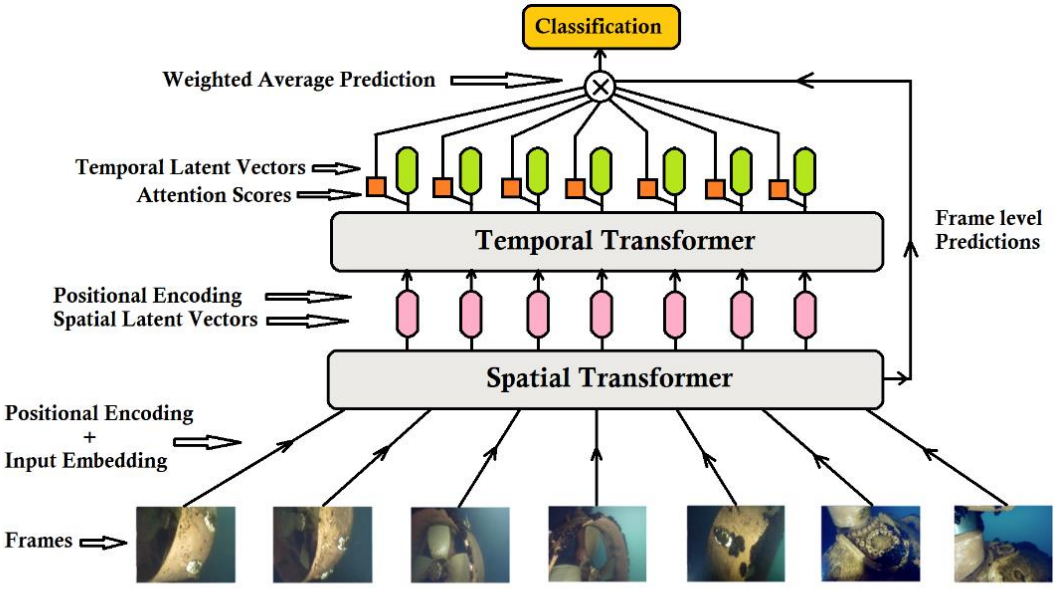


Figure 3.12: Temporal attention weighted spatiotemporal transformer architecture.

Results: Multi-label Video Classifiers

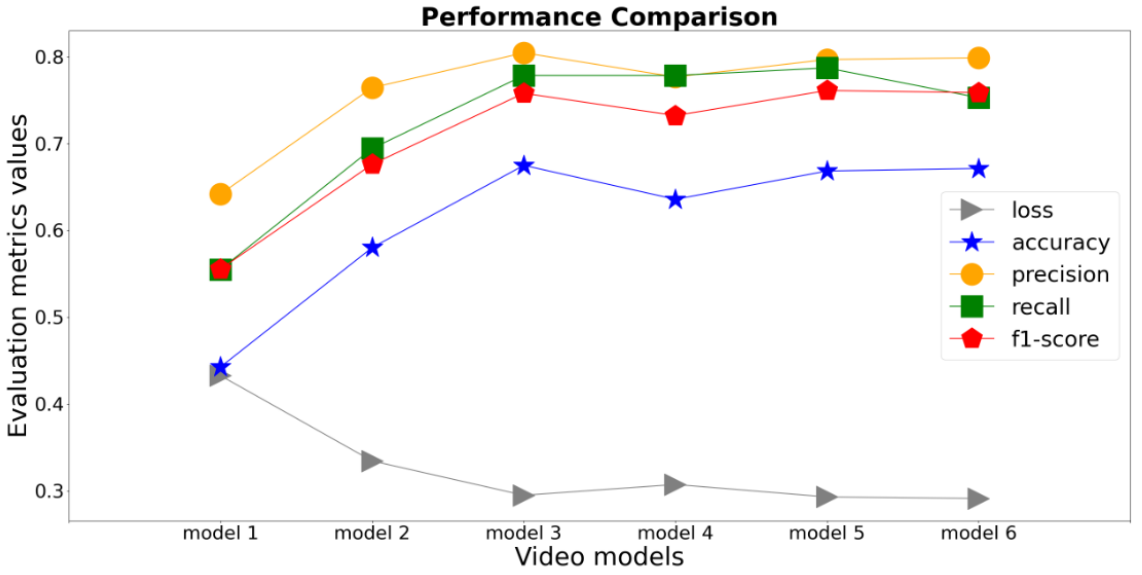
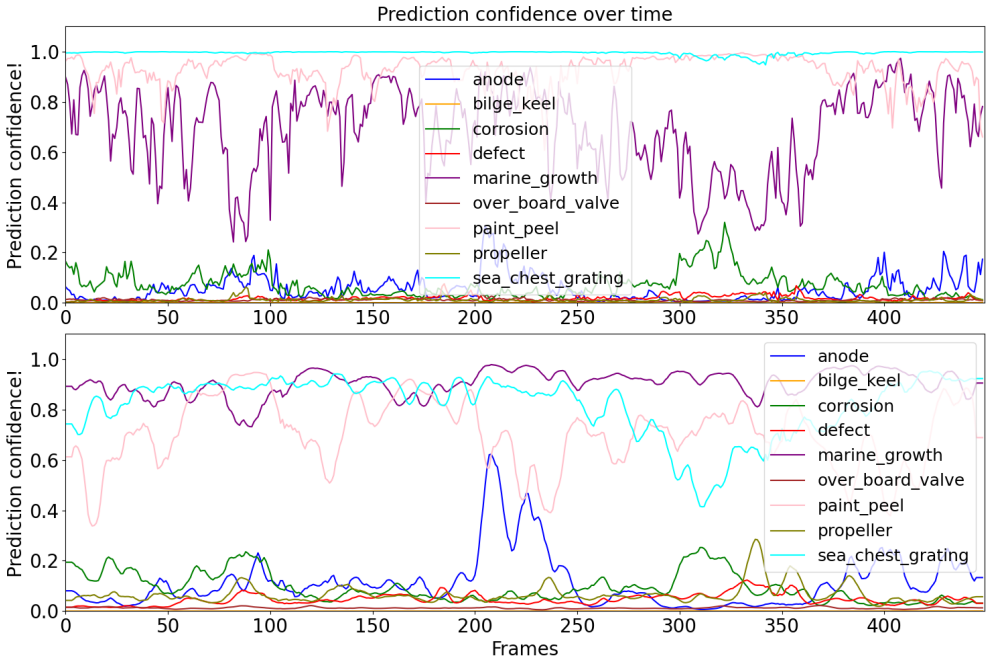


Figure 4.7: Best evaluation metric values across all the variants.

Model	Embedding
1	Uniform
2	Tubelet
	Spatial Latent Vector
3	CLS
4	Mean
	Attention Weighting
5	Values (V)
6	Spatial Prediction

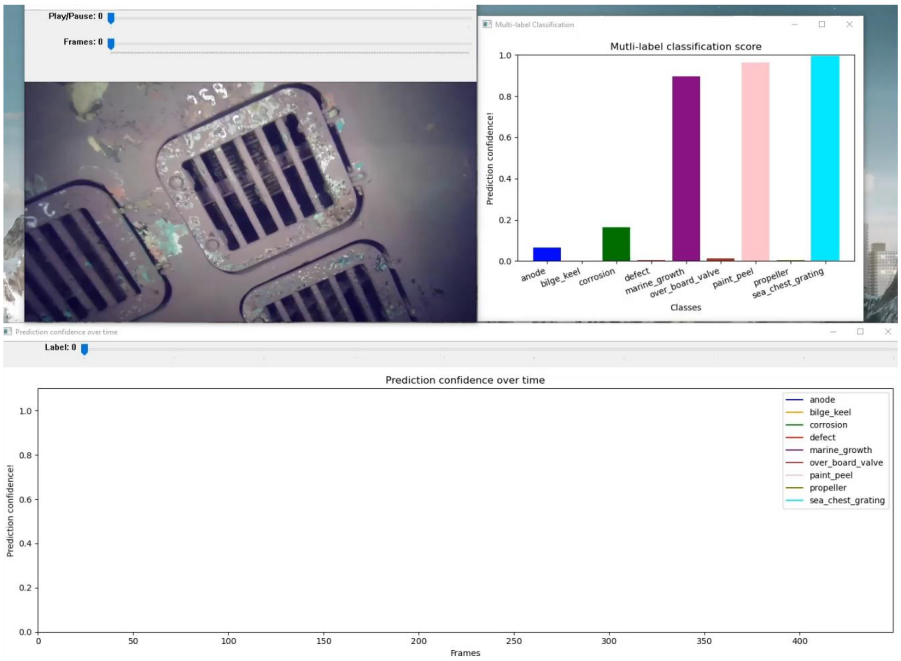
Result: Temporal Observation



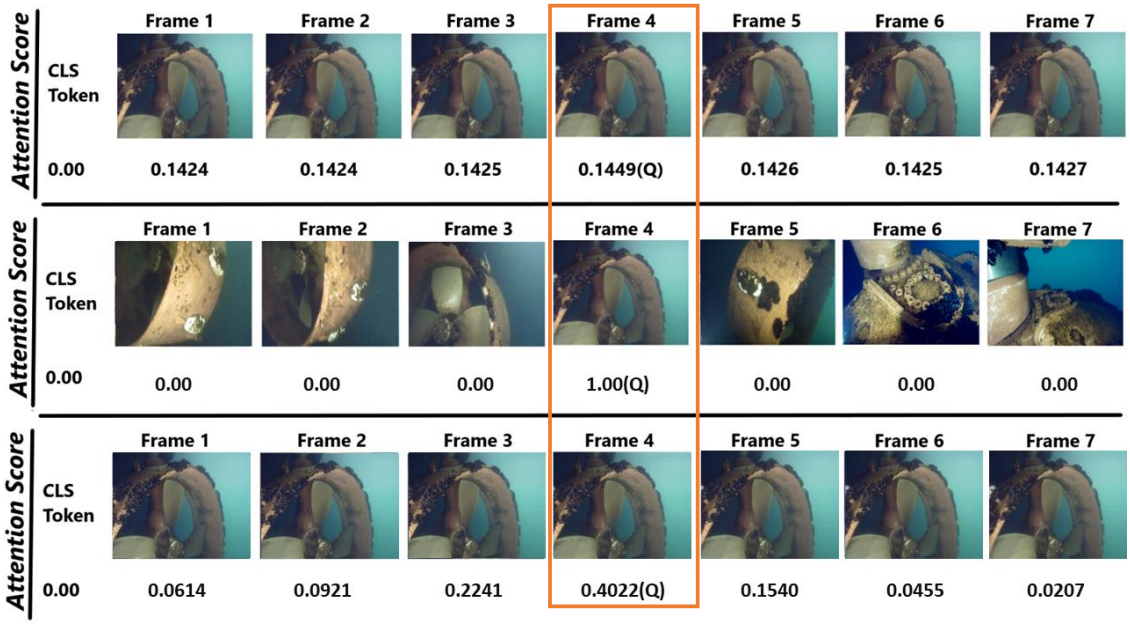
Best Image Model

Best Video Model

Result: Temporal Observation



Ablation Study: Temporal Attention Scores



The same image in all frames -> Attention is evenly distributed

Random neighboring frames with Q frame -> Attended only on the query frame

The validation snippet with the same Q frame -> Q frame got the highest attention

Conclusion & Future Work

- Work on data
- Pretraining strategies
- Temporal information improves the robustness and stabilizes the prediction confidence.

SINTEF

Thank You

Technology for a better society

Poster I

This is the poster (A2 size) that I designed for the Department of Marine Technology, NTNU.

Multi-label Video Classification for Underwater Ship Inspection

Azad, Md Abulkalam
Erasmus Mundus MSMIR
Email: mdaaz@stud.ntnu.no

Supervisor(s): Ahmed Mohammed (SINTEF), Maryna Waszak (SINTEF), and Prof. Martin Ludvigsen

1. Introduction

Problem: The manual video analysis in phase (2) of current underwater ship hull inspection in Fig. 1 is time-consuming, tedious, and prone to human error.

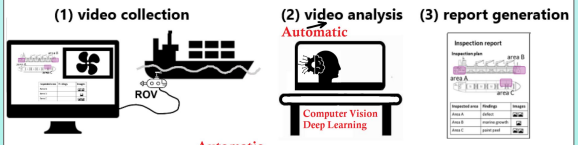


Fig. 1.The workflow of ~~current~~ Automatic underwater ship inspection using ROVs.

Objective: Deep learning and computer vision based automatic video analysis system can significantly improve the inspection and expedite the entire process. The following objectives are set to accomplish the goal.

1. **Analysis of image based classifiers (pros & cons)**
2. **Exploration of temporal information in videos;**
   a. **Model stabilization and consistency**
   b. **Robustness in underwater environment**
3. **Identification of a Multi-label Video Classifier**

2. Image & Video Datasets

LIACI Image Dataset: Total images = 1893, train = 1370, validation = 191, and not used = 332.

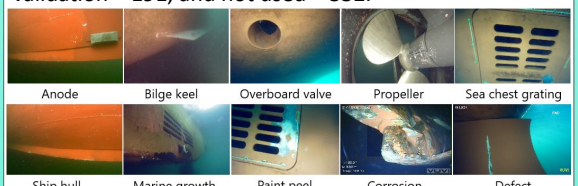


Fig. 3. Visualization of the class labels of LIACI

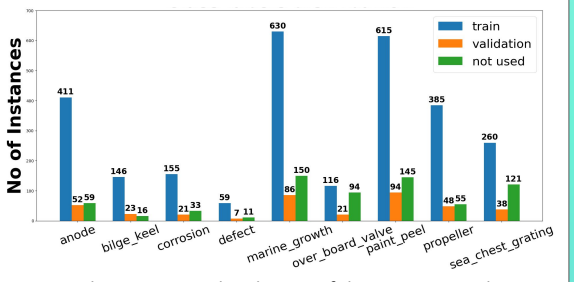


Fig. 3. Class instances distribution of the LIACI image dataset.

Generated Video Dataset: Total snippets = 755, train = 584, validation = 87, and not used = 84.

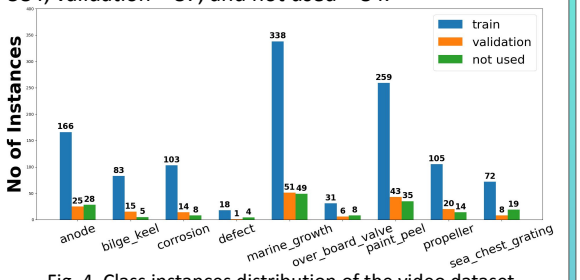


Fig. 4. Class instances distribution of the video dataset.

5. Analysis

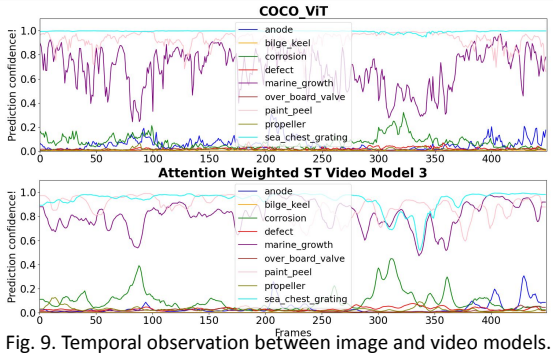


Fig. 9. Temporal observation between image and video models.

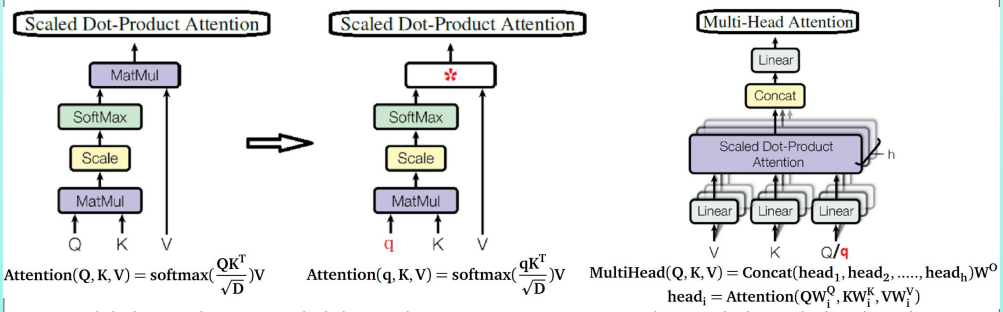
Various analysis can be found in the thesis report.

6. References

References are available in the thesis report or in preprint ResearchGate: http://dx.doi.org/10.13140/RG.2.2.27960.32007/1

3. Methodology

Self-Attention: $Q, K, V \in \mathbb{R}^{N \times D}$ where N is the number of queries, D is the dimension and $q \in \mathbb{R}^{1 \times D}$ represents a single query. $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{D \times d}$ and $W^O \in \mathbb{R}^{hd \times D}$ where $d = D/h$.



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{D}})V \qquad \text{Attention}(q, K, V) = \text{softmax}(\frac{qK^T}{\sqrt{D}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O$$
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Fig. 5. Global vs single query scaled-dot product attention computation along with the multi-head mechanism.

Multi-label Video Classifier: Three approaches; Naive, Late-Fusion, and Attention-weighted models.

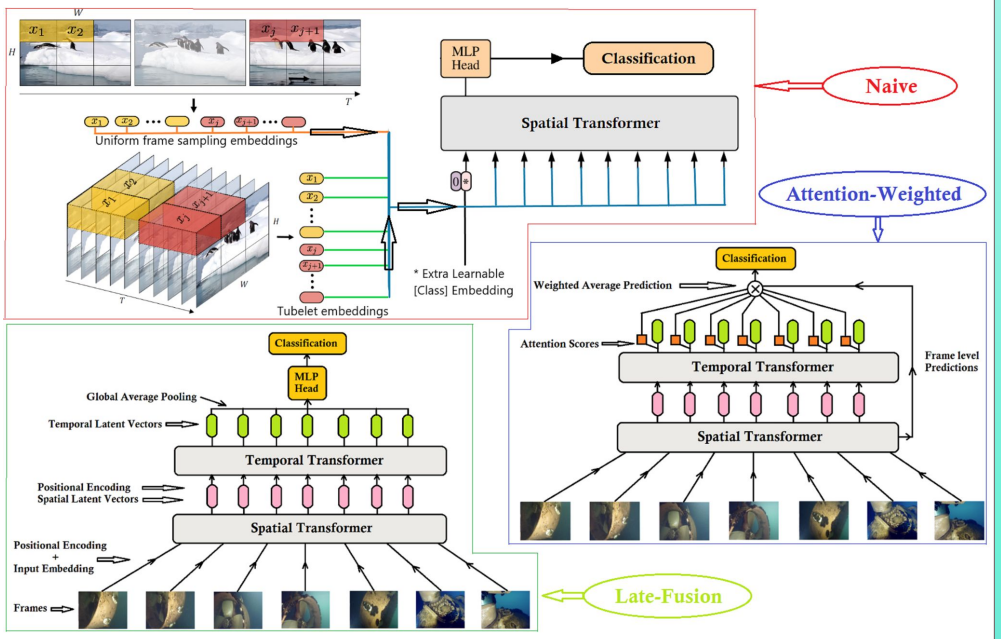


Fig. 6. Three different approaches to multi-label video classification.

4. Results

Multi-label Image Classifier Improvement: Two models; IMAGENET_ViT and COCO_ViT.

Metric	IMAGENET_ViT	COCO_ViT
Accuracy	31.2%	39.6%
Precision	1.1%	5.3%
Recall	44.0%	46.0%
F1-Score	24.0%	29.0%

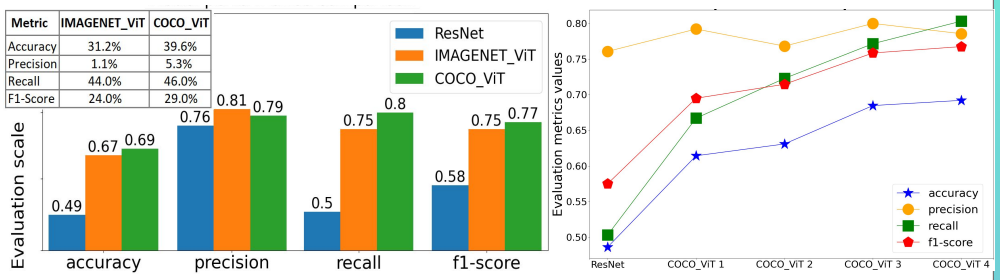


Fig. 7. (left)Performance comparison between ResNet and our models. (right)Gradual improvement of COCO_ViT.

Multi-label Video Classifier: Three approaches; Naive, Late-Fusion, and Attention-weighted models.

Embedding	Best	Loss	Accuracy	Precision	Recall	F1-score
Uniform	values	0.4326	0.4425	0.6418	0.5550	0.5554
	model	0.4479	0.4414	0.6418	0.5550	0.5554
Tubelet	values	0.3343	0.5801	0.7644	0.6943	0.6761
	model	0.3849	0.5801	0.7308	0.6943	0.6761
Spatial Latent Vector	**Best**	**Loss**	**Accuracy**	**Precision**	**Recall**	**F1-score**
CLS	values	0.2947	0.6749	0.8046	0.7785	0.7580
	model	0.3061	0.6749	0.8046	0.7785	0.7580
Mean	values	0.3071	0.6358	0.7768	0.7785	0.7323
	model	0.3261	0.6358	0.7577	0.7785	0.7323
Attention Weighting	**Best**	**Loss**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Values (V)	values	0.2928	0.6684	0.7969	0.7872	0.7612
	model	0.3043	0.6684	0.7969	0.7872	0.7612
Spatial Prediction	values	0.2910	0.6713	0.7989	0.7929	0.7587
	model	0.2972	0.6713	0.7989	0.7929	0.7587

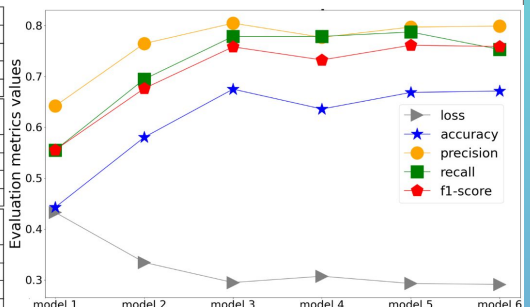


Fig. 8. (left)Best metric values and models across approaches. (right)Performance comparison among the models.

7. Publications

1. (Accepted) **OCEANS 2023** Limerick conference (IEEE) Preprint: https://arxiv.org/abs/2305.17338
2. Accepted to be presented at **NORA 2023** conference and is invited for the **NMI journal**.

8. Acknowledgment

The Research Council of Norway
MIR MARINE & MARITIME INTELLIGENT ROBOTICS
Co-funded by the Erasmus+ Programme of the European Union

Poster II

This is the poster (A1 size) that I designed for our Erasmus Mundus Master MIR Symposium 2023 which will take place in Spain from 20 to 22 June.

Multi-label Video Classification for Underwater Ship Inspection

Azad, Md Abulkalam, Erasmus Mundus MSMIR, mdaaz@stud.ntnu.no
Supervisor(s): Ahmed Mohammed (SINTEF), Maryna Waszak (SINTEF), and Prof. Martin Ludvigsen

MARINE & MARITIME INTELLIGENT ROBOTICS
Co-funded by the Erasmus+ Programme of the European Union

1. Introduction

Problem: The manual video analysis in phase (2) of current underwater ship hull inspection in Fig. 1 is time-consuming, tedious, and prone to human error.

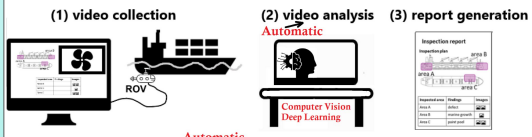


Fig. 1. The workflow of ~~current~~ Automatic underwater ship inspection using ROVs.

Objective: Deep learning and computer vision based automatic video analysis system can significantly improve the inspection and expedite the entire process.
The following objectives are set to accomplish the goal.

1. **Analysis of image based classifiers (pros & cons)**
2. **Exploration of temporal information in videos;**
   a. **Model stabilization and consistency**
   b. **Robustness in underwater environment**
3. **Identification of a Multi-label Video Classifier**

2. Image & Video Datasets

LIACI Image Dataset: Total images = 1893, train = 1370, validation = 191, and not used = 332.

Fig. 3. Visualization of the class labels of LIACI

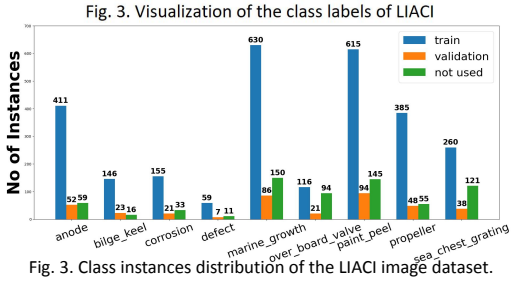


Fig. 3. Class instances distribution of the LIACI image dataset.

Generated Video Dataset: Total snippets = 755, train = 584, validation = 87, and not used = 84.

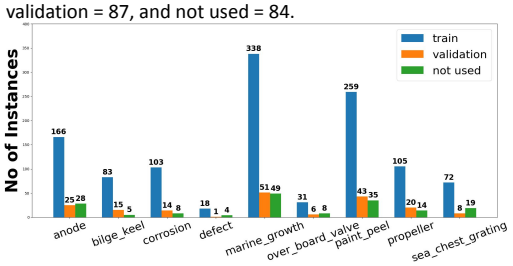


Fig. 4. Class instances distribution of the video dataset.

5. Discussion

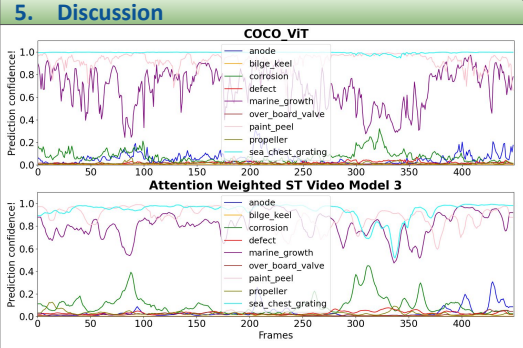


Fig. 9. Temporal observation between image and video models.

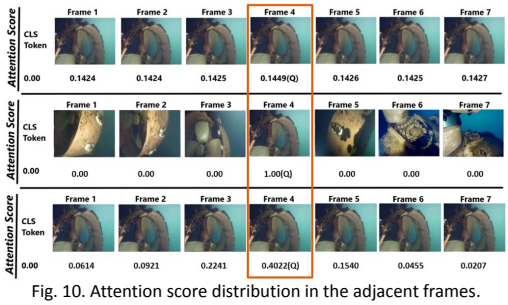
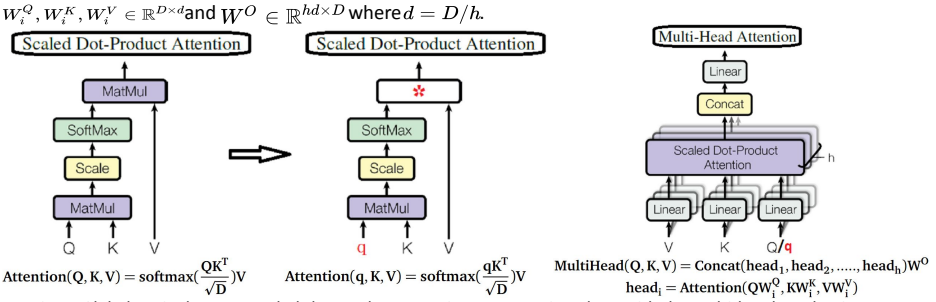


Fig. 10. Attention score distribution in the adjacent frames.

3. Methodology

Self-Attention: $Q, K, V \in \mathbb{R}^{N\times D}$ where N is the number of queries, D is the dimension and $q \in \mathbb{R}^{1\times D}$ represents a single query. $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{D\times d}$ and $W^O \in \mathbb{R}^{hd\times D}$ where $d = D/h$.



$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{D}})V \qquad \text{Attention}(q,K,V) = \text{softmax}(\frac{qK^T}{\sqrt{D}})V$$

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O$$
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Fig. 5. Global vs single query scaled-dot product attention computation along with the multi-head mechanism.

Multi-label Video Classifier: Three approaches; Naive, Late-Fusion, and Attention-weighted models.

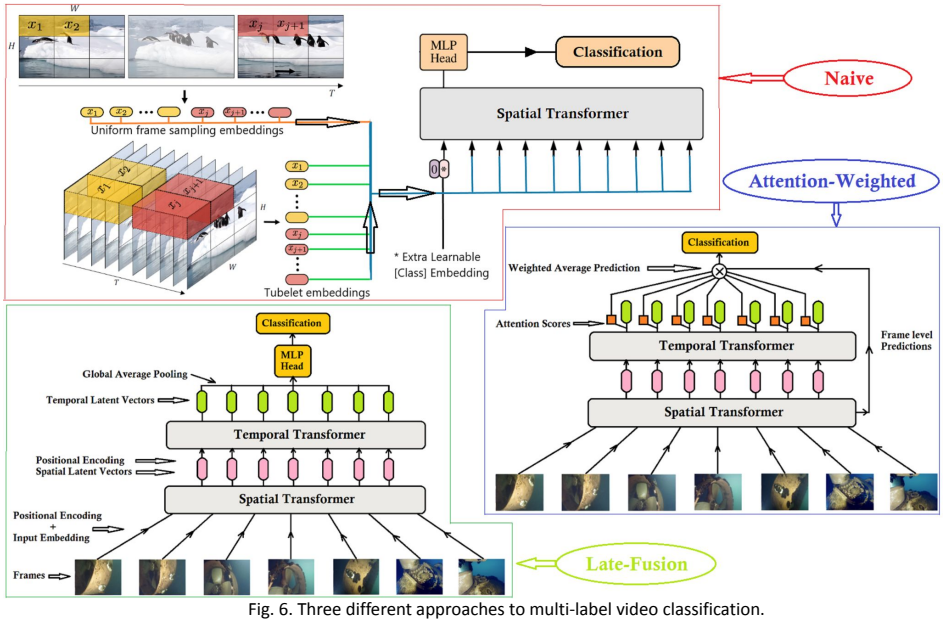


Fig. 6. Three different approaches to multi-label video classification.

4. Results

Multi-label Image Classifier Improvement: Two models; IMAGENET_ViT and COCO_ViT.

Metric	IMAGENET_ViT	COCO_ViT
Accuracy	31.2%	39.6%
Precision	1.1%	5.3%
Recall	44.0%	46.0%
F1-Score	24.0%	29.0%

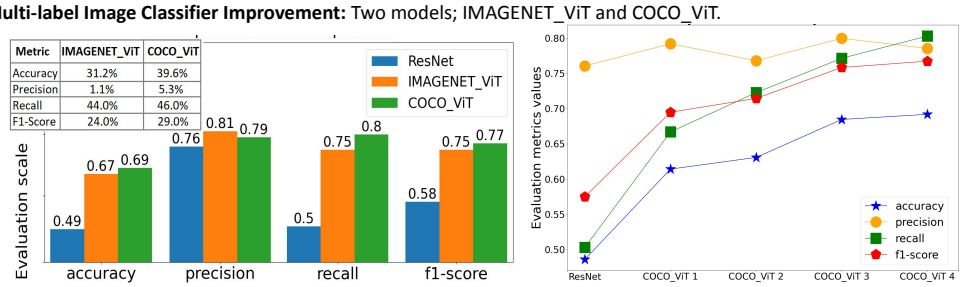


Fig. 7. (left)Performance comparison between ResNet and our models. (right)Gradual improvement of COCO_ViT.

Multi-label Video Classifier: Three approaches; Naive, Late-Fusion, and Attention-weighted models.

Embedding	Best	Loss	Accuracy	Precision	Recall	F1-score
Uniform	values	0.4326	0.4425	0.6418	0.5550	0.5554
	model	0.4479	0.4414	0.6418	0.5550	0.5554
Tubelet	values	0.3343	0.5801	0.7644	0.6943	0.6761
	model	0.3849	0.5801	0.7308	0.6943	0.6761
Spatial Latent Vector	**Best**	**Loss**	**Accuracy**	**Precision**	**Recall**	**F1-score**
CLS	values	0.2947	0.6749	0.8046	0.7785	0.7580
	model	0.3061	0.6749	0.8046	0.7785	0.7580
Mean	values	0.3071	0.6358	0.7768	0.7785	0.7323
	model	0.3261	0.6358	0.7577	0.7785	0.7323
Attention Weighting	**Best**	**Loss**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Values (V)	values	0.2928	0.6684	0.7969	0.7872	0.7612
	model	0.3043	0.6684	0.7969	0.7872	0.7612
Spatial Prediction	values	0.2910	0.6713	0.7989	0.7929	0.7587
	model	0.2972	0.6713	0.7989	0.7929	0.7587

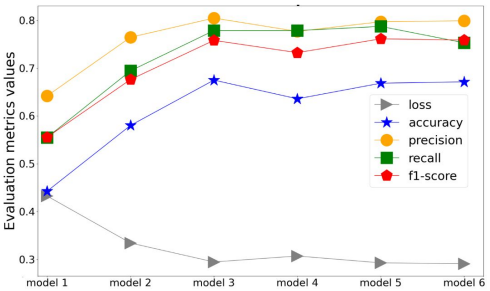


Fig. 8. (left)Best metric values and models across approaches. (right)Performance comparison among the models.

6. Conclusion & Future Work

- Multi-label ViT image classifiers are trained and improved
- Video dataset is generated and annotated (weakly supervised)
- State-of-the-art spatiotemporal feature extraction is exploited
- Three different approaches to video models are introduced
- Single query in self-attention mechanism is introduced and analyzed to compute attention scores
- Further work to prepare large-scale video dataset is required
- Other pretraining strategies should be explored
- Limitation and further research direction are provided

7. Publications

1. Presented at **OCEANS 2023** Limerick conference (IEEE) Preprint: https://arxiv.org/abs/2305.17338
2. Presented at **NORA Annual 2023** conference.
3. Invited for the **Nordic Machine Intelligence (NMI) level-1 journal**.

8. References

References are available in the thesis report or in the preprint on ResearchGate:
http://dx.doi.org/10.13140/RG.2.2.27960.32007/1

SINTEF
The Research Council of Norway
LIACi
NTNU

Appendix A

Additional Information

A.1 Multi-label classification using ResNet

Below the 7 diagrams present the visual multi-label classification reports of 7 key video snippets shown in table 3.2.

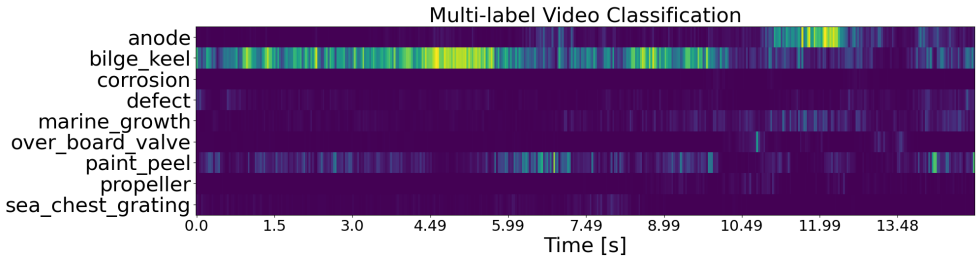


Figure A.1: Visual report of multi-label classification on snippet no. 2 in table 3.2 using ResNet.

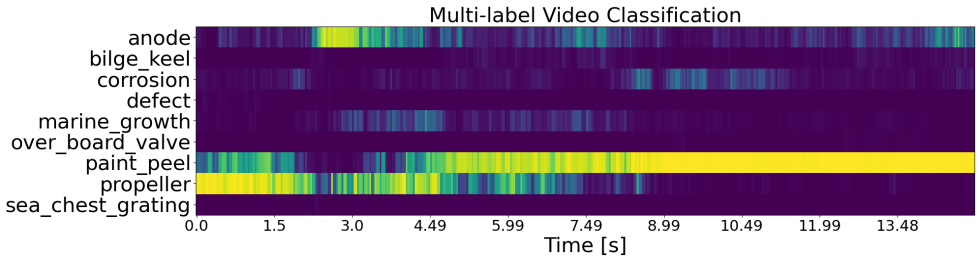


Figure A.2: Visual report of multi-label classification on snippet no. 3 in table 3.2 using ResNet.

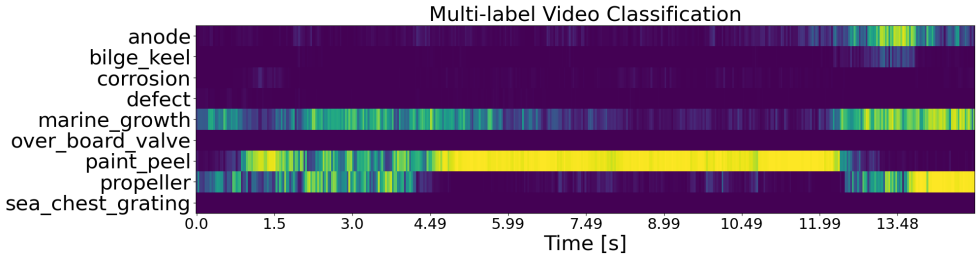


Figure A.3: Visual report of multi-label classification on snippet no. 4 in table 3.2 using ResNet.

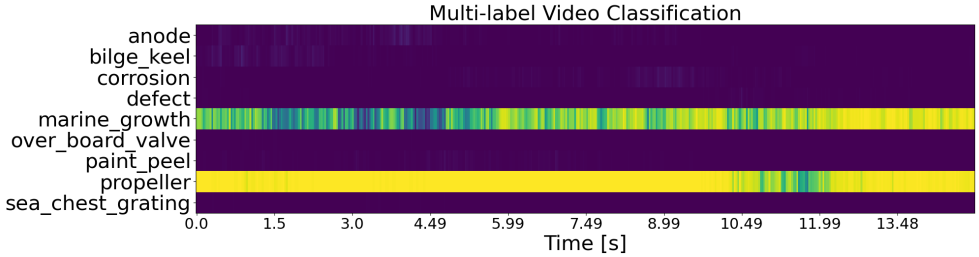


Figure A.4: Visual report of multi-label classification on snippet no. 5 in table 3.2 using ResNet.

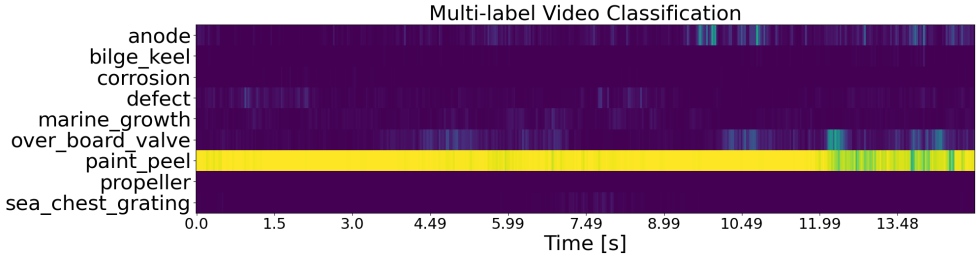


Figure A.5: Visual report of multi-label classification on snippet no. 6 in table 3.2 using ResNet.

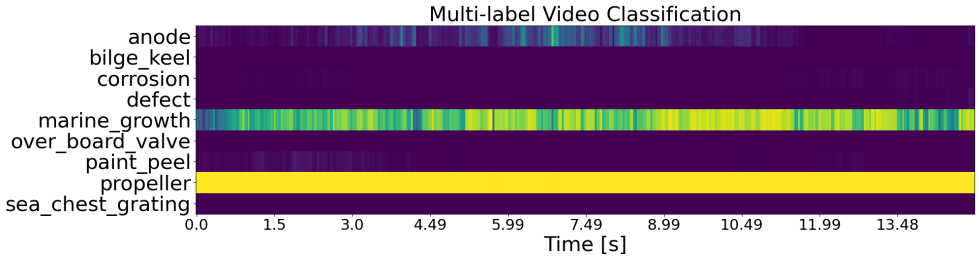


Figure A.6: Visual report of multi-label classification on snippet no. 7 in table 3.2 using ResNet.

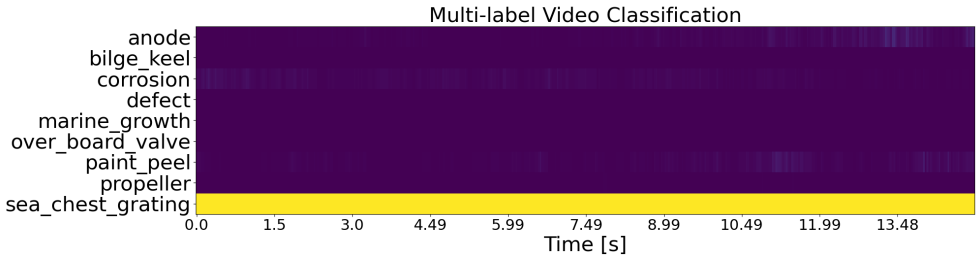


Figure A.7: Visual report of multi-label classification on snippet no. 8 in table 3.2 using ResNet.

A.2 Temporal results comparisons

A.2.1 ResNet VS COCO_ViT and IMAGENET_ViT

Below the seven diagrams represent the temporal observations of COCO_ViT and IMAGENET_ViT comparing with the output of the ResNet model on the rest of the video snippets from table 3.2 in Figures A.8 to A.14.

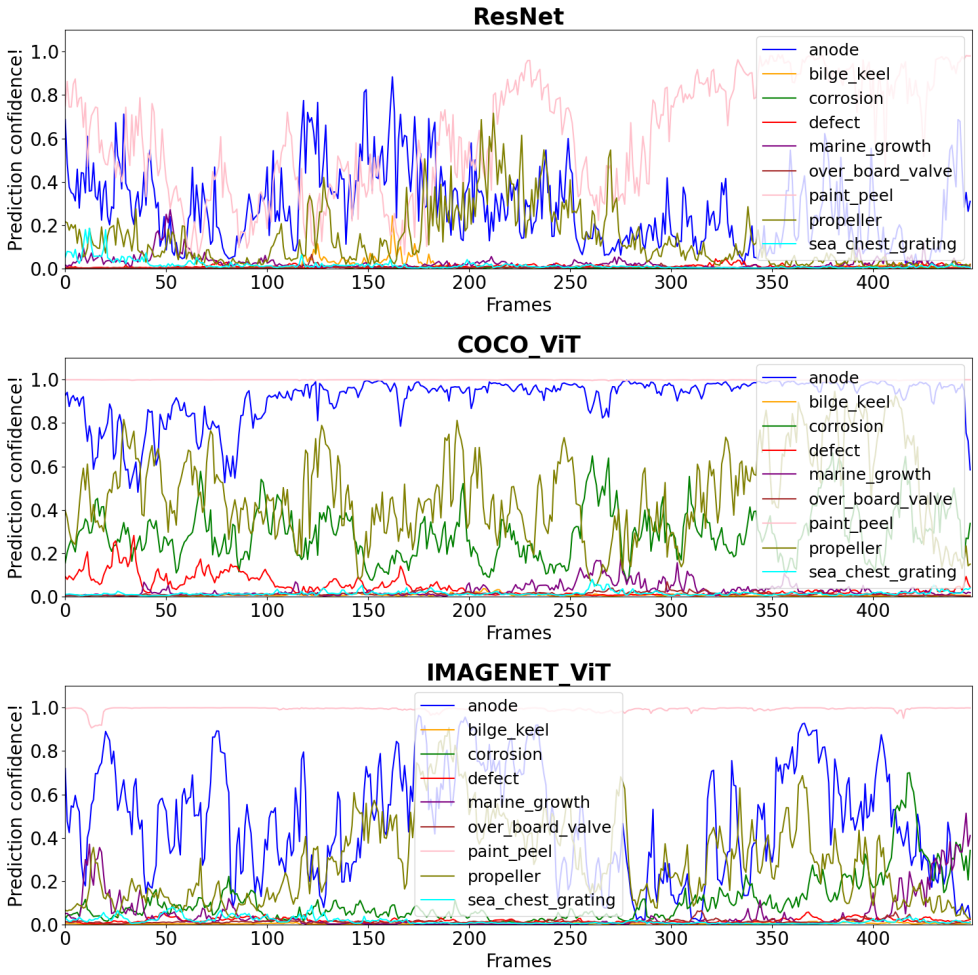


Figure A.8: Temporal observation on the video snippet no. 1 from table 3.2.

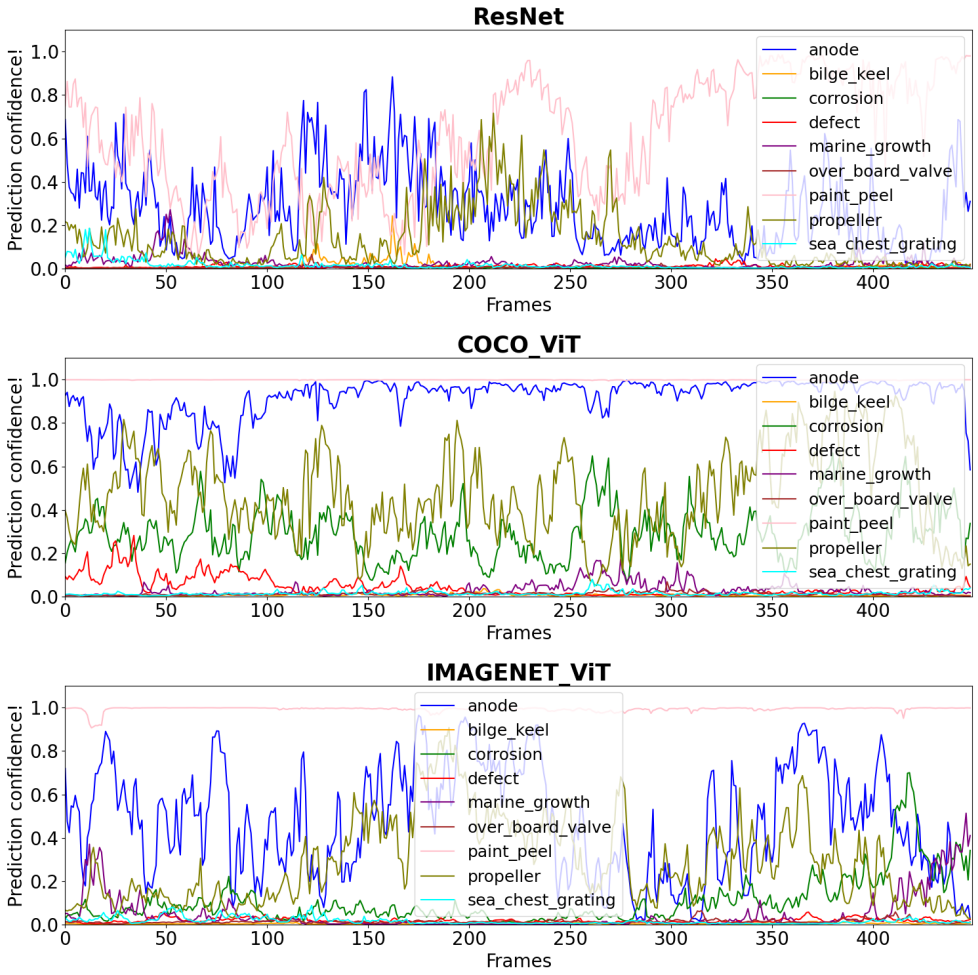


Figure A.9: Temporal observation on the video snippet no. 2 from table 3.2.

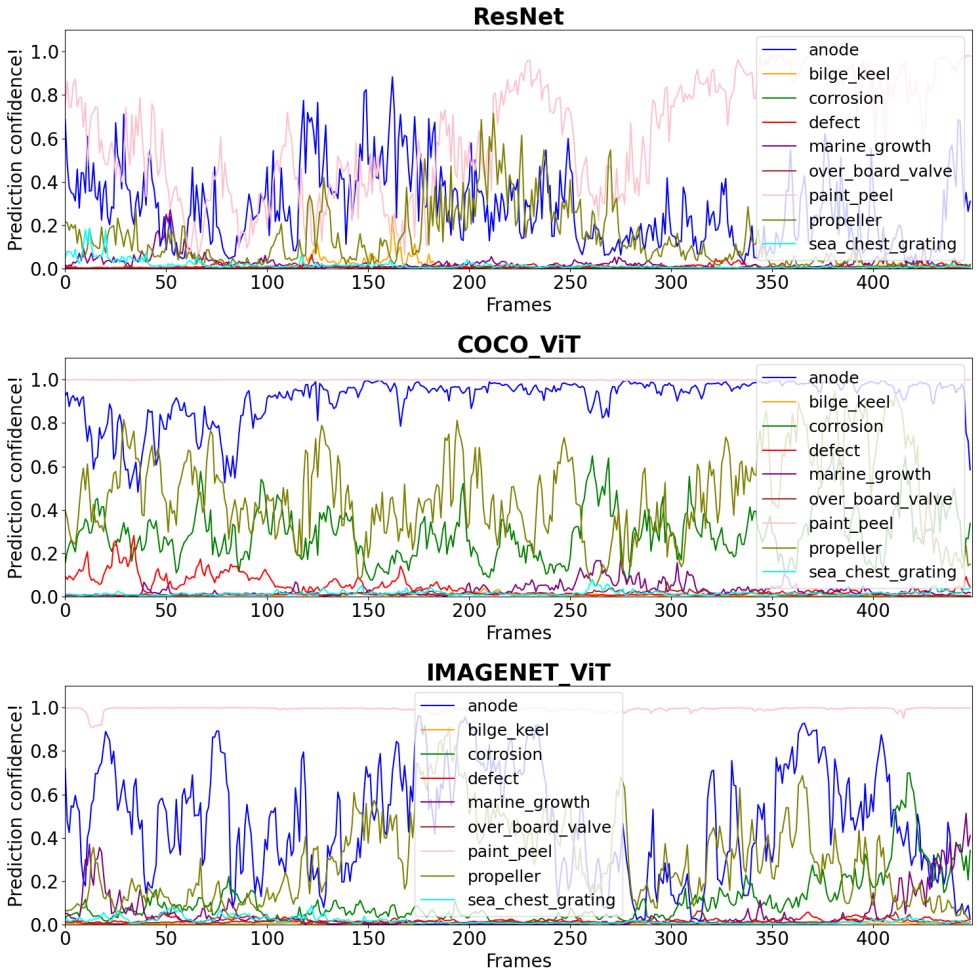


Figure A.10: Temporal observation on the video snippet no. 4 from table 3.2.

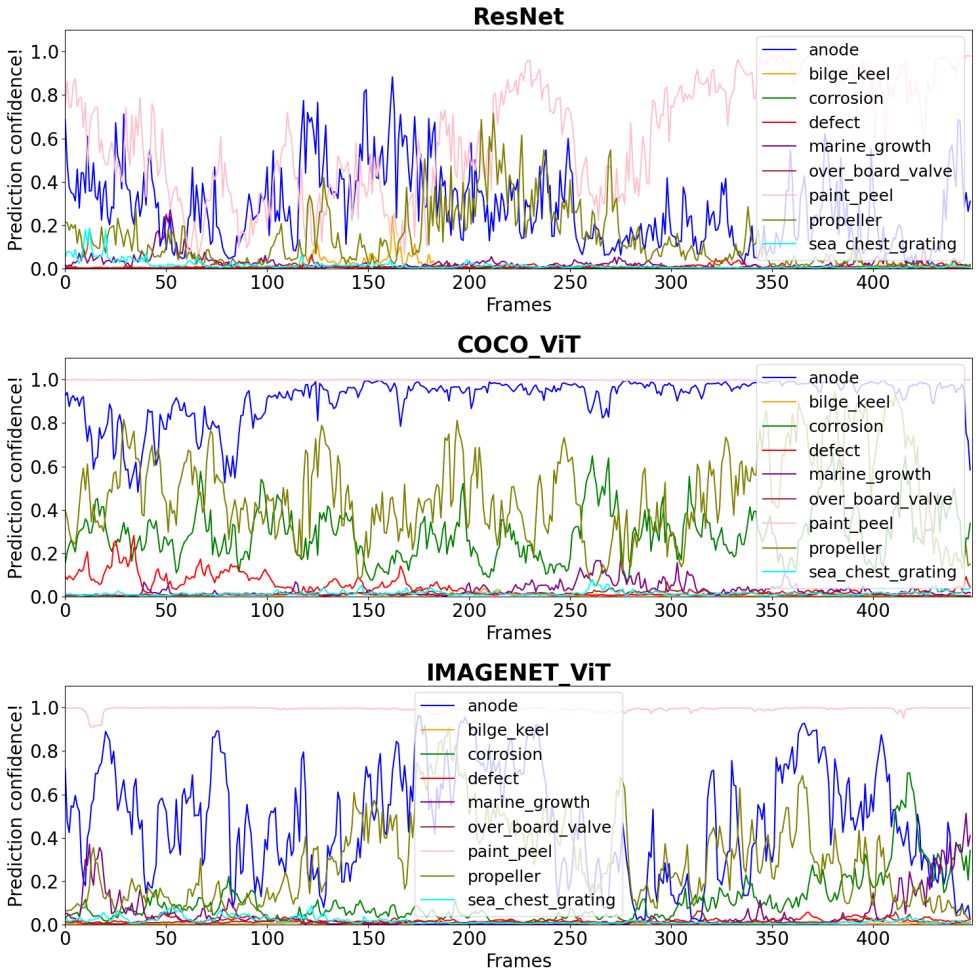


Figure A.11: Temporal observation on the video snippet no. 5 from table 3.2.

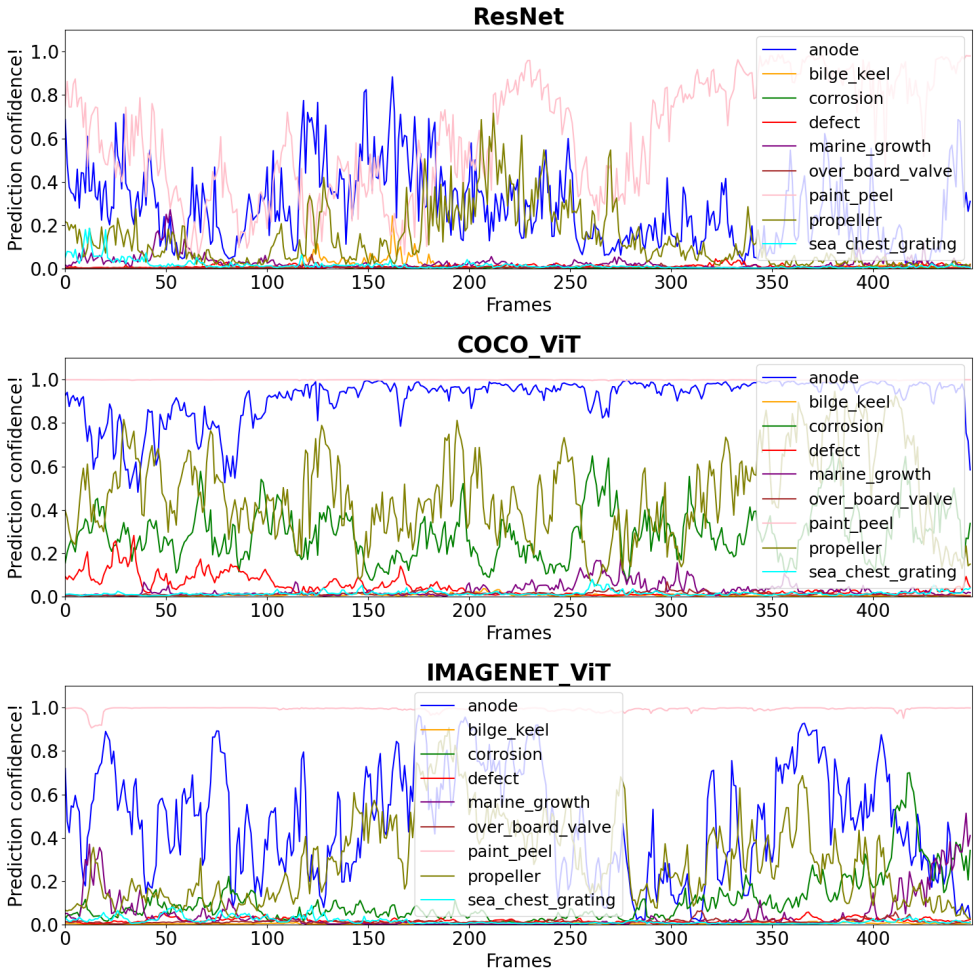


Figure A.12: Temporal observation on the video snippet no. 6 from table 3.2.

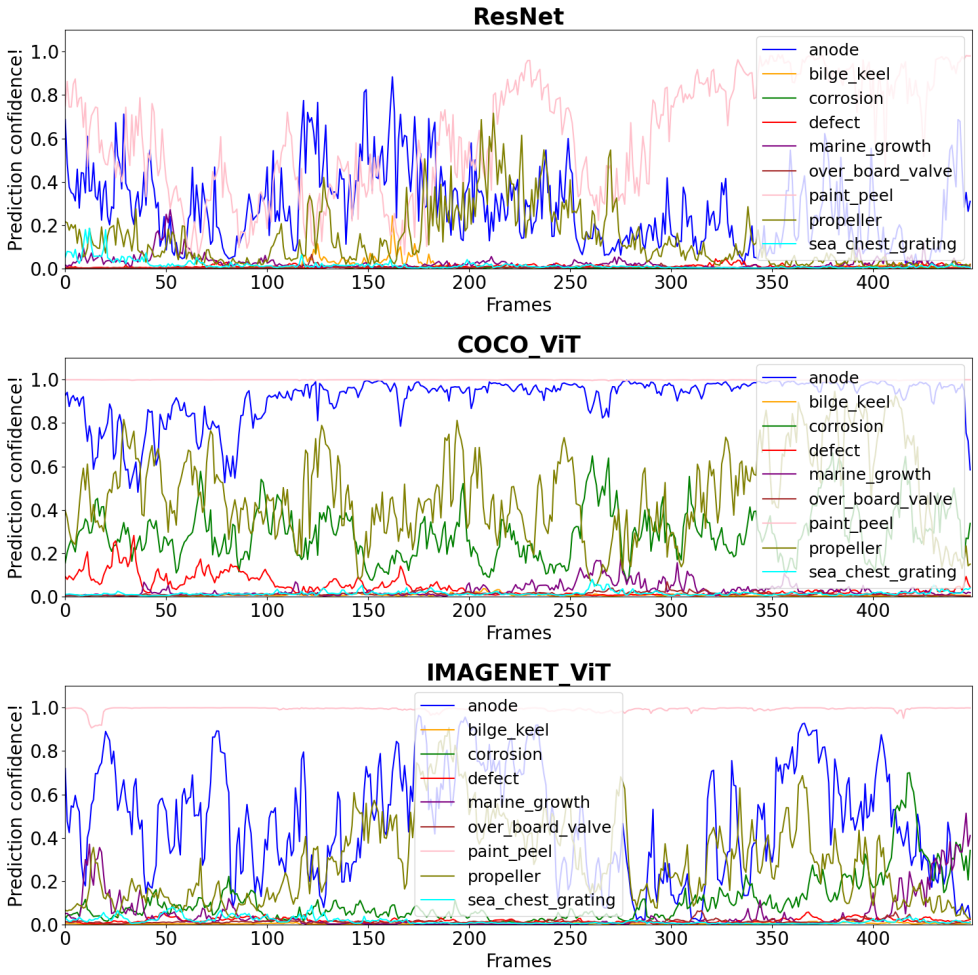


Figure A.13: Temporal observation on the video snippet no. 7 from table 3.2.

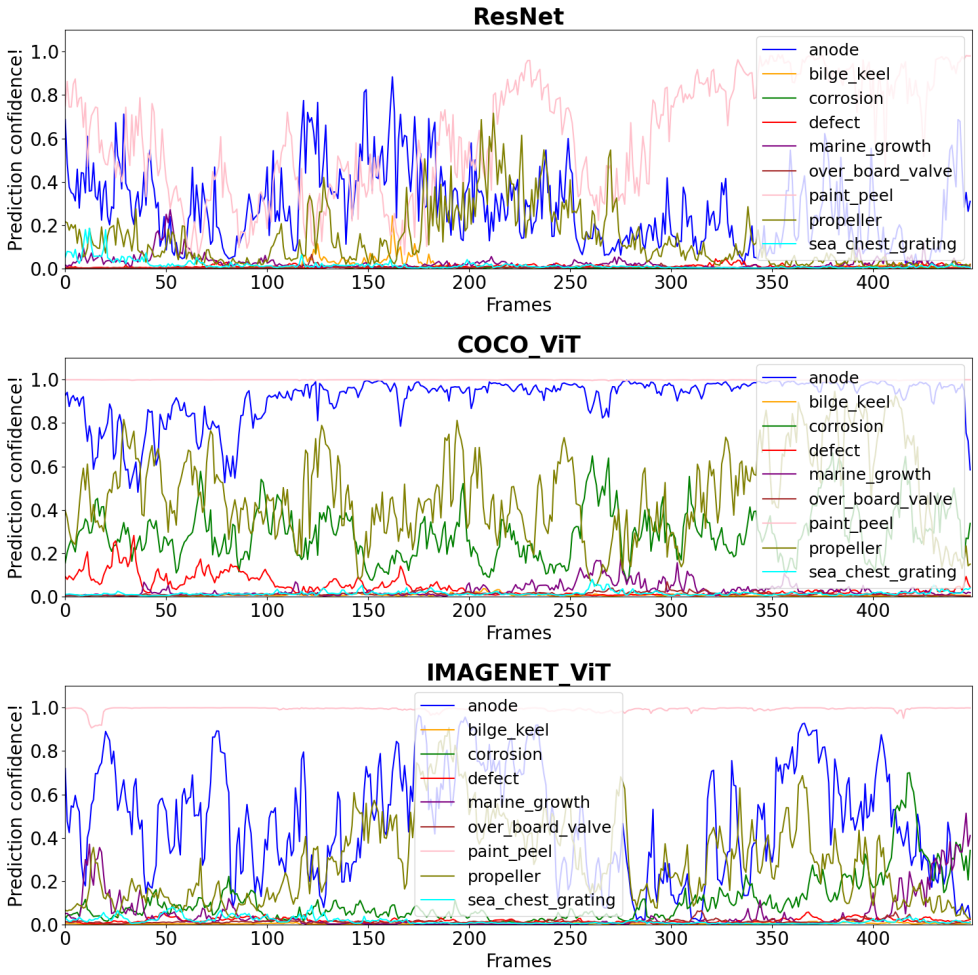


Figure A.14: Temporal observation on the video snippet no. 8 from table 3.2.

A.2.2 Video Models

Temporal observations for video models on the rest of the video snippets are added here.

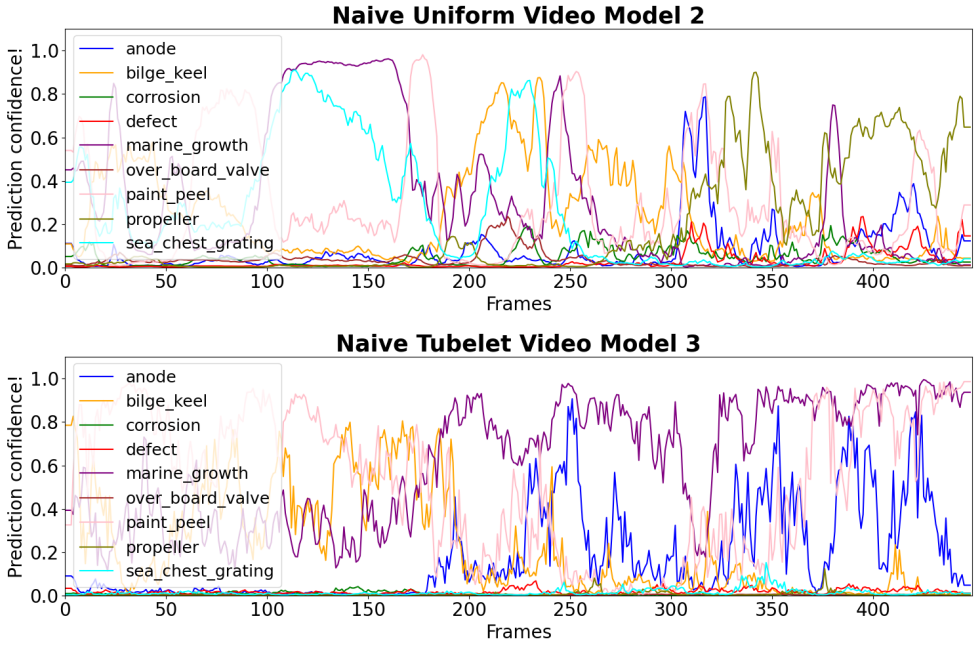


Figure A.15: Temporal observation of the best two naive video models on the snippet no.2 from table 3.2.

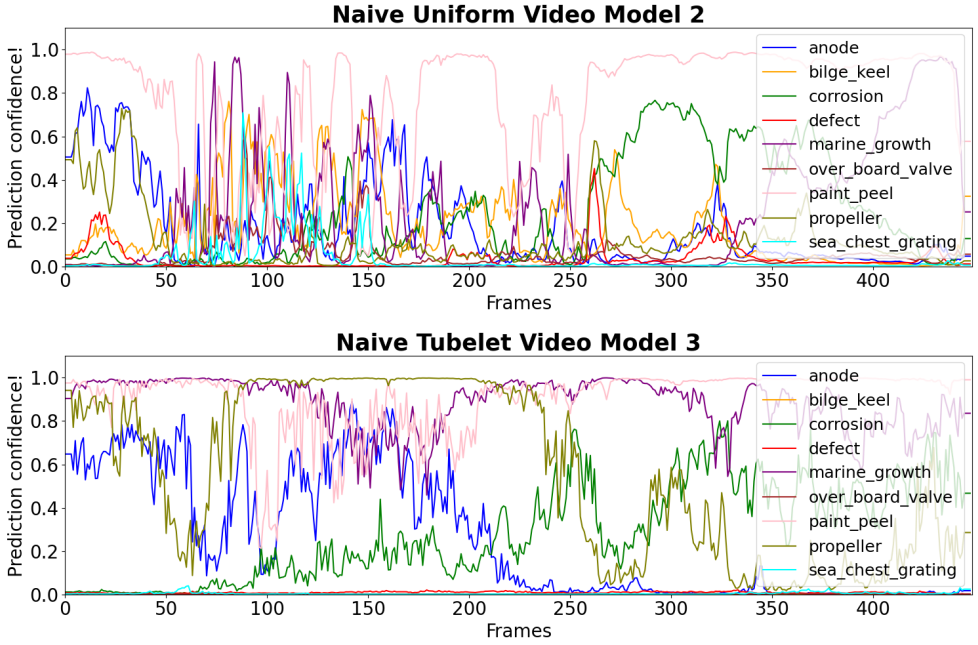


Figure A.16: Temporal observation of the best two naive video models on the snippet no.3 from table 3.2.

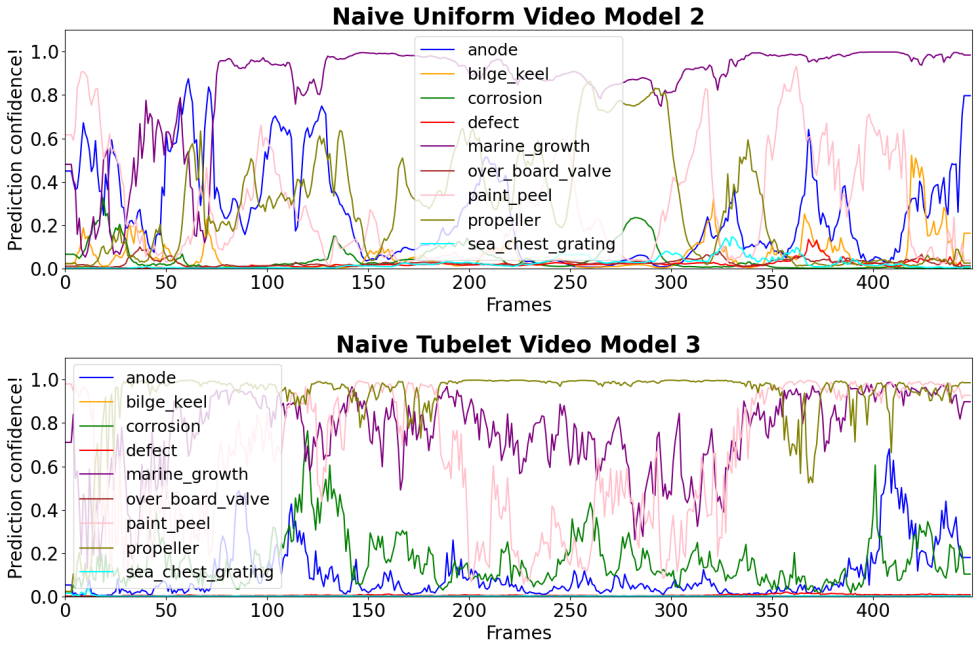


Figure A.17: Temporal observation of the best two naive video models on the snippet no.4 from table 3.2.

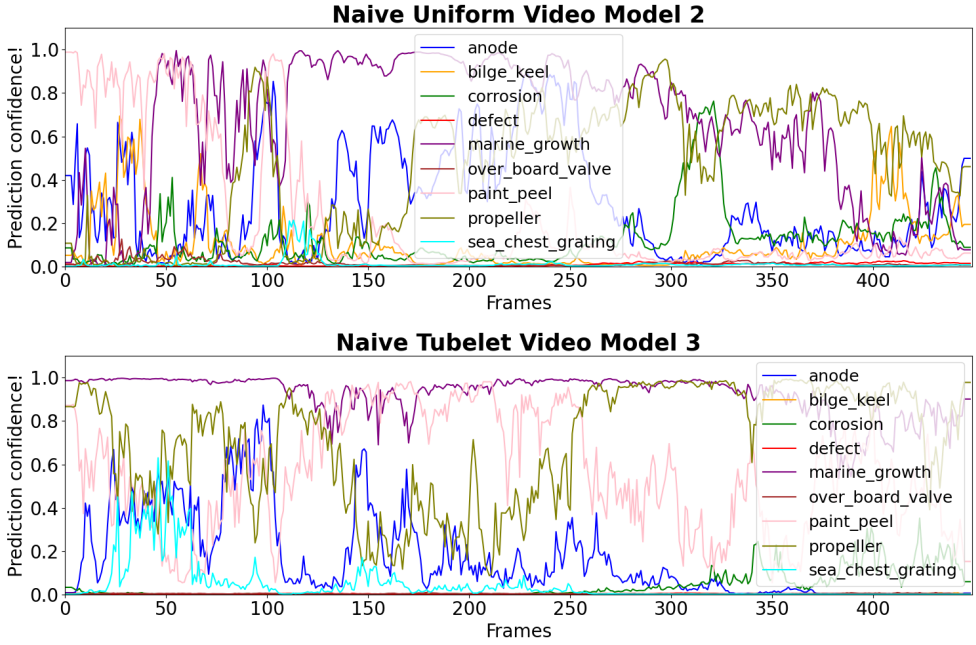


Figure A.18: Temporal observation of the best two naive video models on the snippet no.5 from table 3.2.

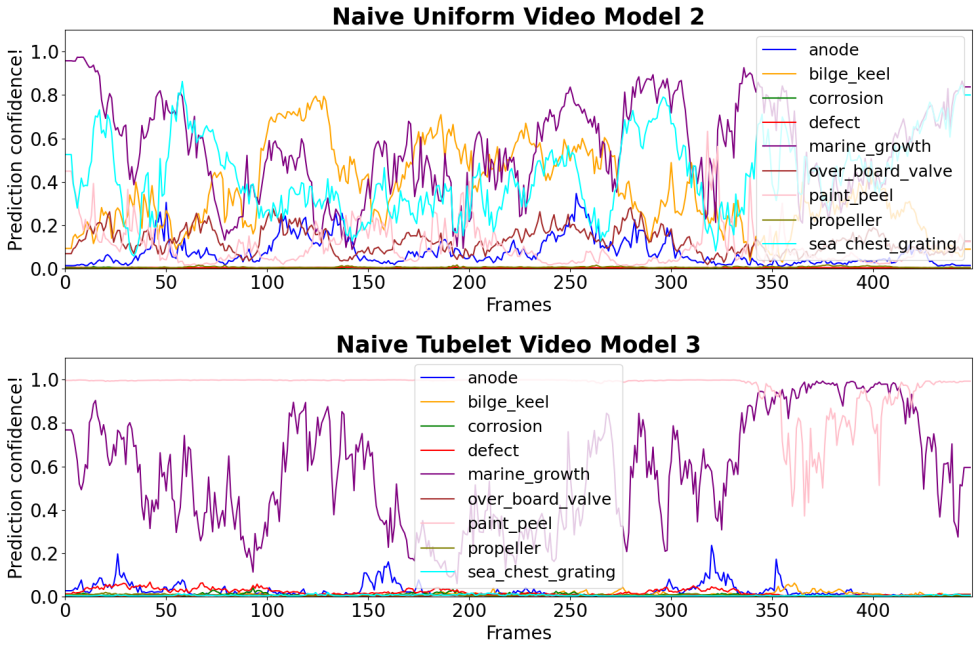


Figure A.19: Temporal observation of the best two naive video models on the snippet no.6 from table 3.2.

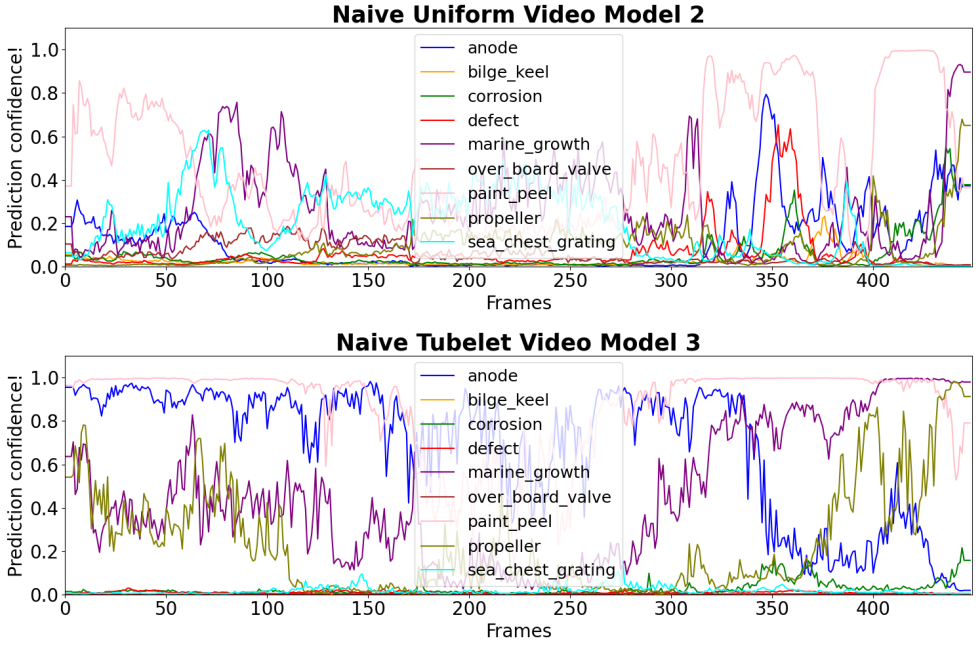


Figure A.20: Temporal observation of the best two naive video models on the snippet no.7 from table 3.2.

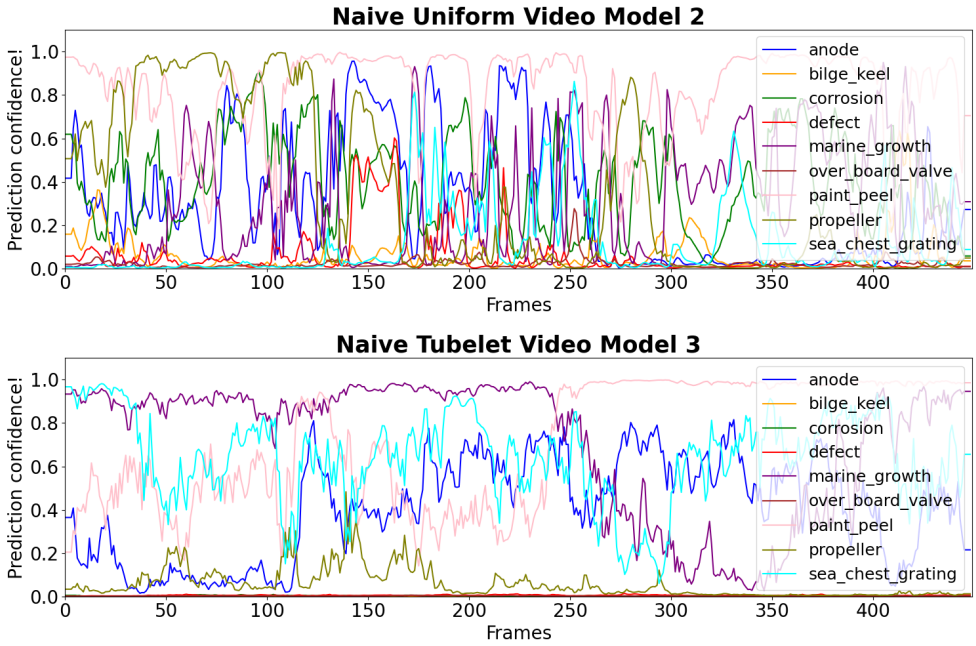


Figure A.21: Temporal observation of the best two naive video models on the snippet no.8 from table 3.2.

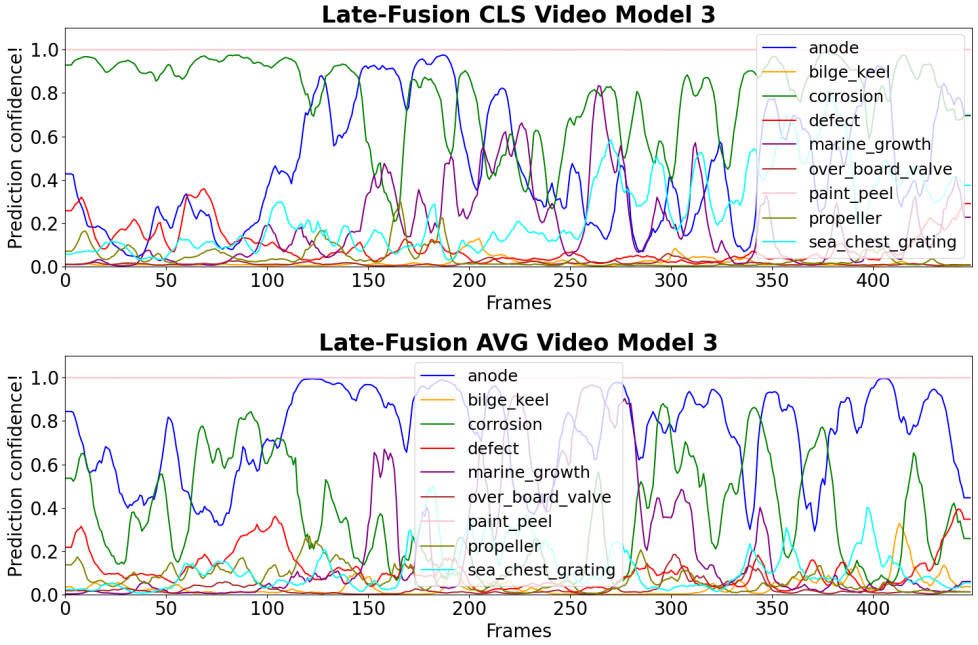


Figure A.22: Temporal observation of the best two late-fusion video models on the snippet no.1 from table 3.2.

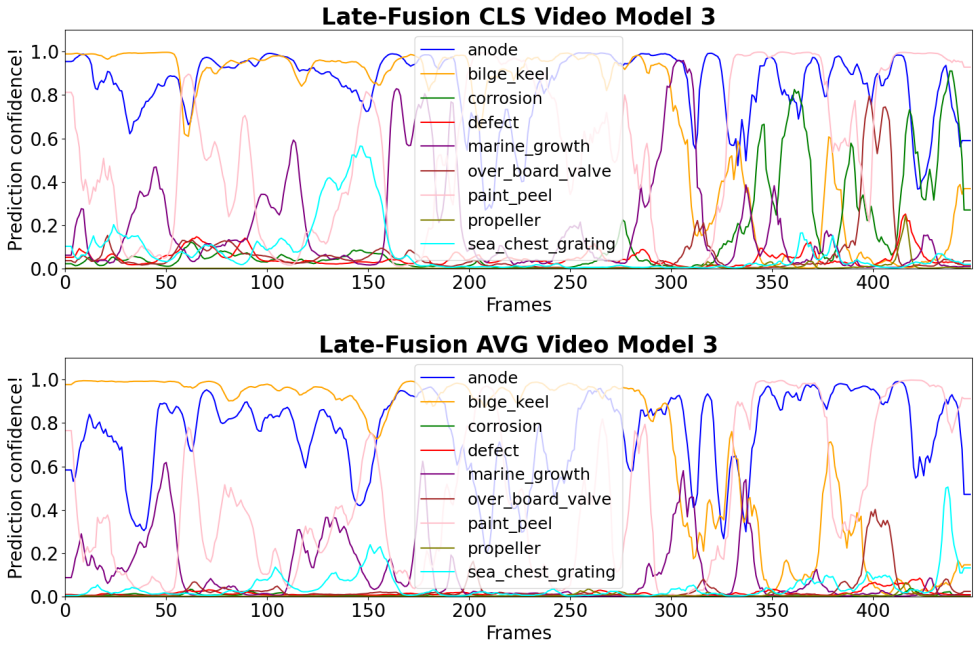


Figure A.23: Temporal observation of the best two late-fusion video models on the snippet no.2 from table 3.2.

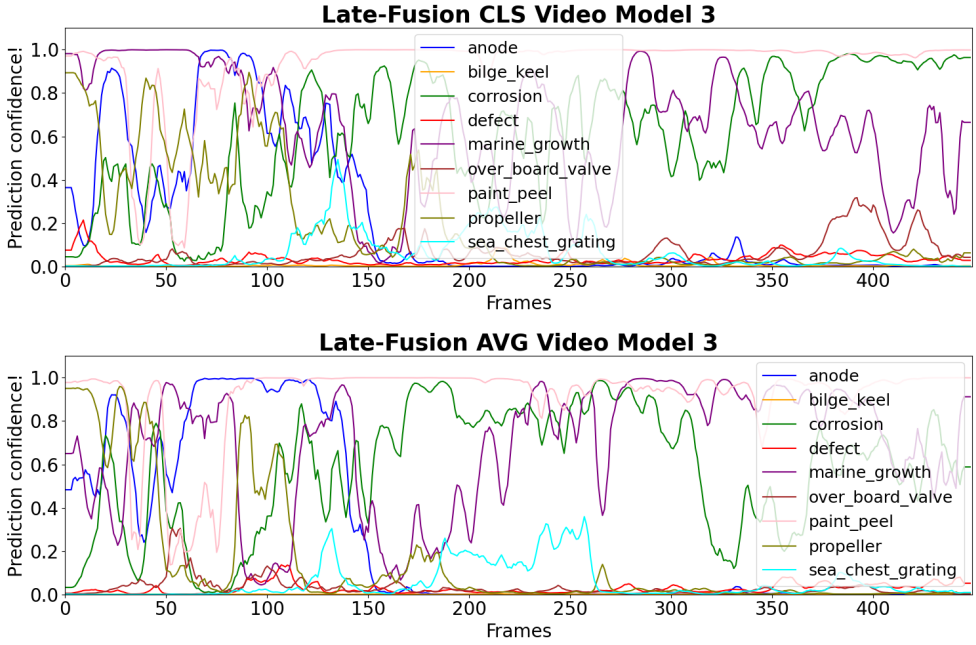


Figure A.24: Temporal observation of the best two late-fusion video models on the snippet no.3 from table 3.2.

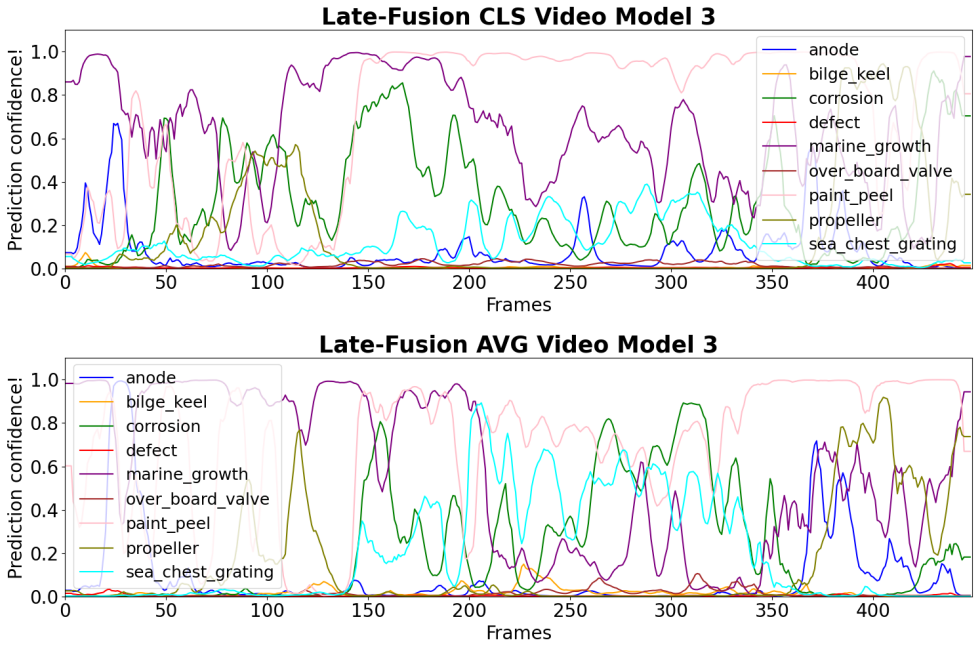


Figure A.25: Temporal observation of the best two late-fusion video models on the snippet no.4 from table 3.2.

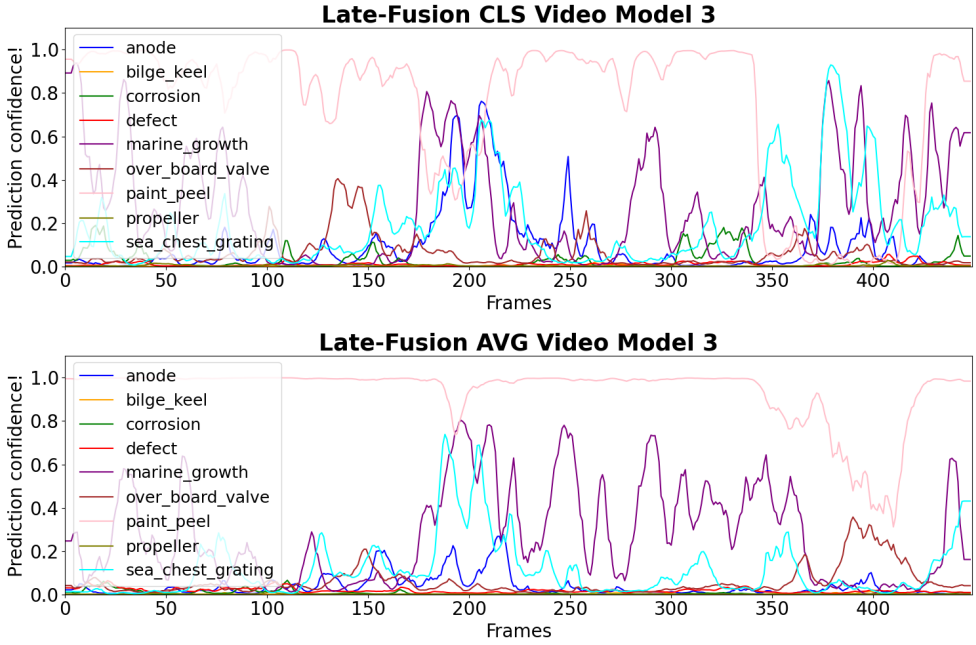


Figure A.26: Temporal observation of the best two late-fusion video models on the snippet no.6 from table 3.2.

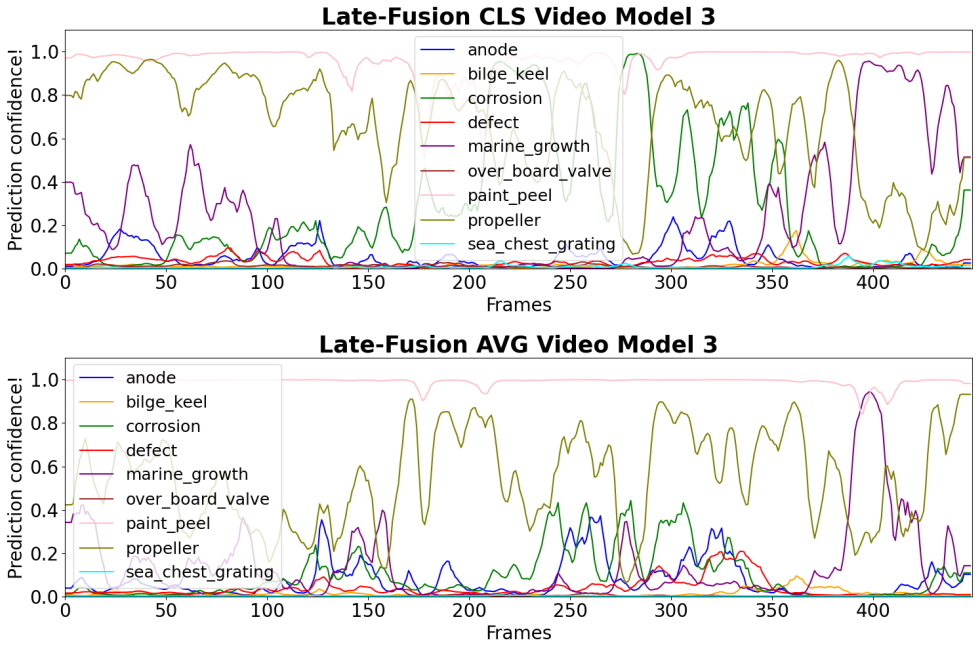


Figure A.27: Temporal observation of the best two late-fusion video models on the snippet no.7 from table 3.2.

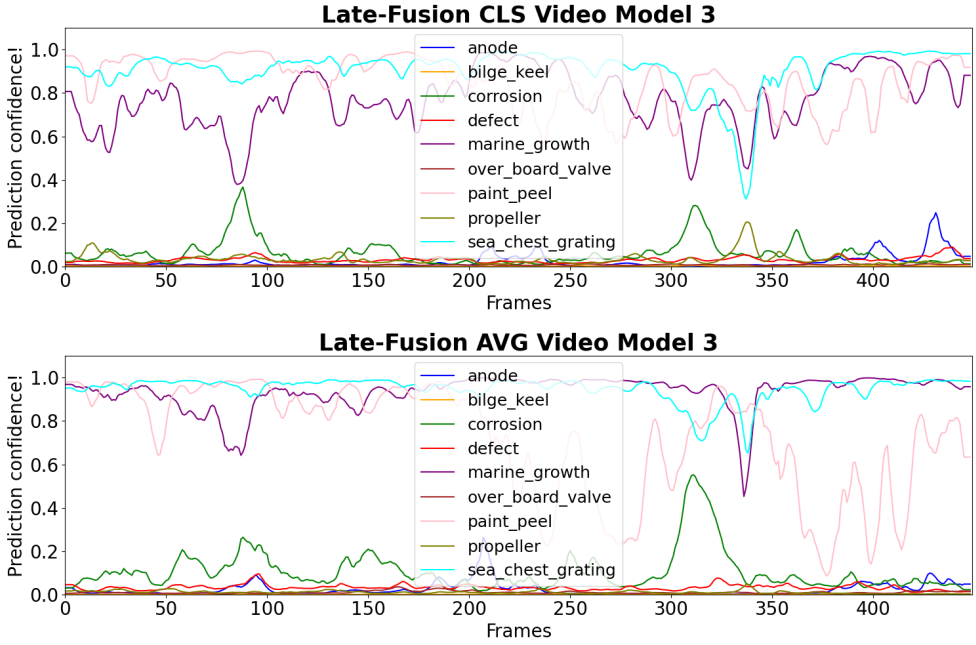


Figure A.28: Temporal observation of the best two late-fusion video models on the snippet no.8 from table 3.2.

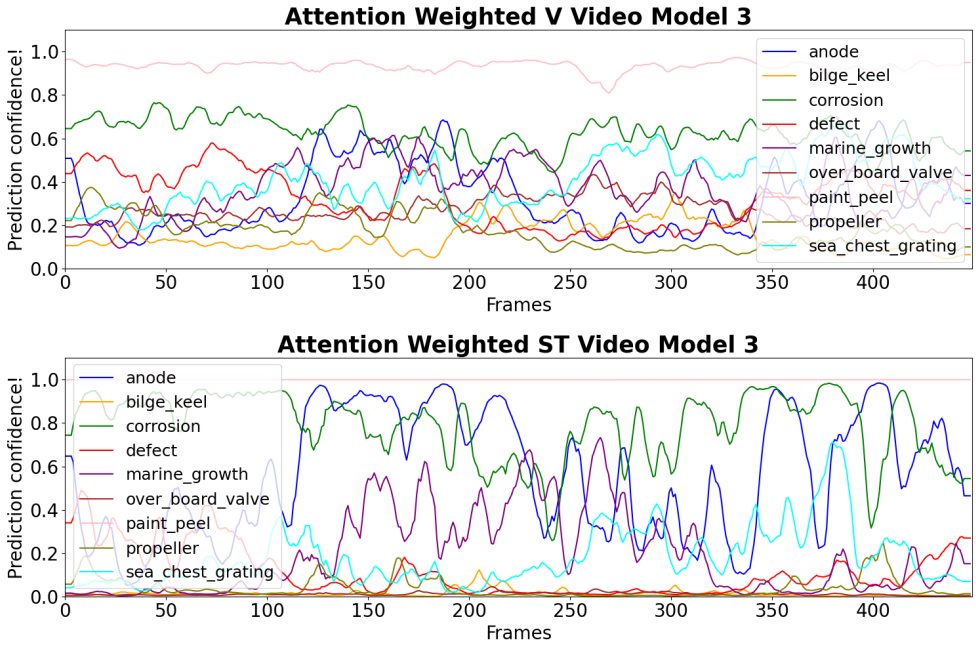


Figure A.29: Temporal observation of the best two attention-weighted video models on the snippet no.1 from table 3.2.

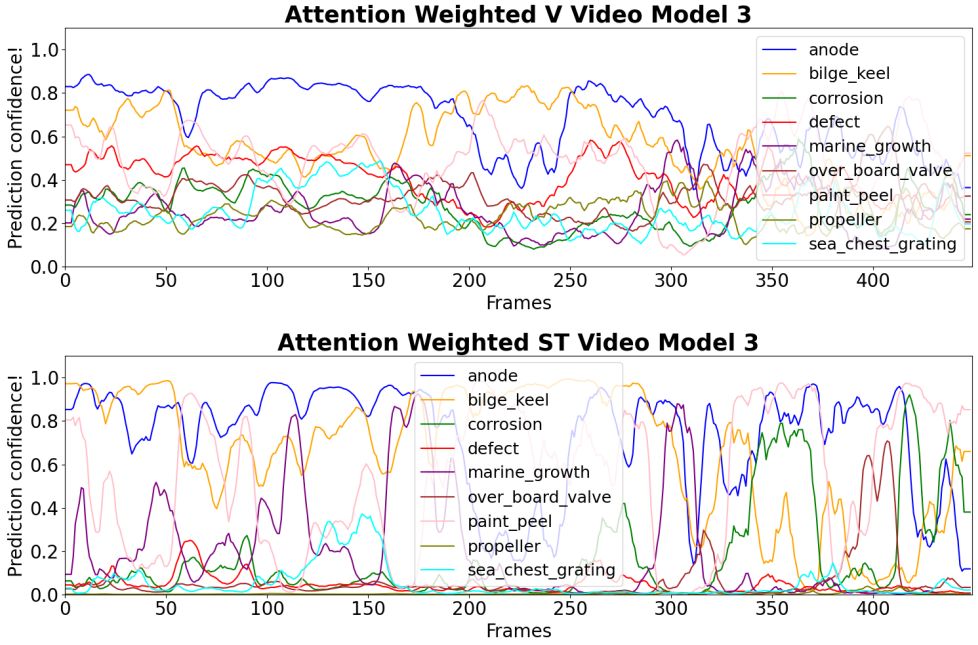


Figure A.30: Temporal observation of the best two attention-weighted video models on the snippet no.2 from table 3.2.

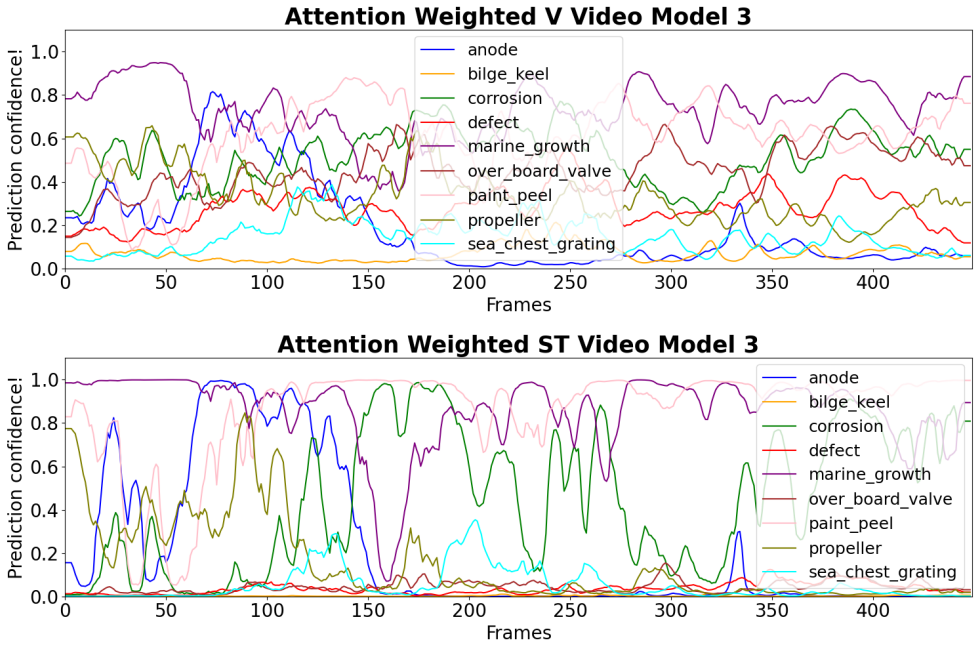


Figure A.31: Temporal observation of the best two attention-weighted video models on the snippet no.3 from table 3.2.

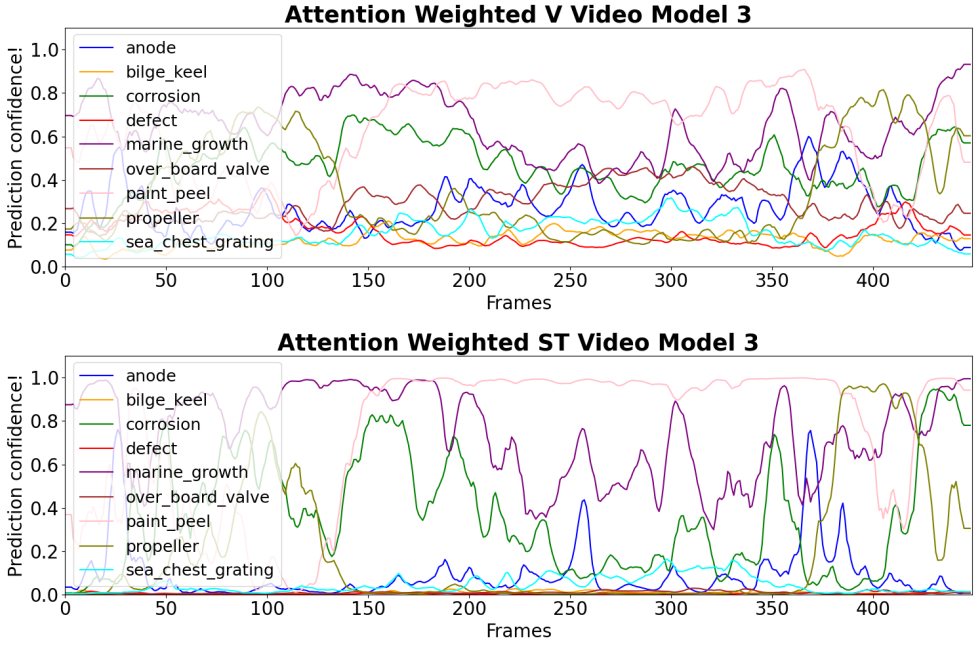


Figure A.32: Temporal observation of the best two attention-weighted video models on the snippet no.4 from table 3.2.

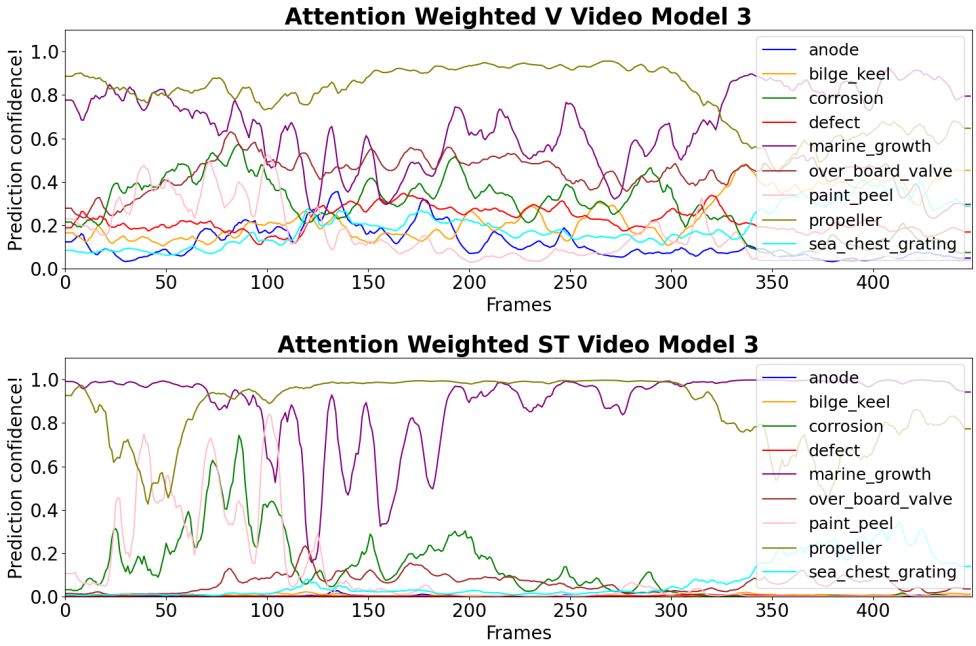


Figure A.33: Temporal observation of the best two attention-weighted video models on the snippet no.5 from table 3.2.

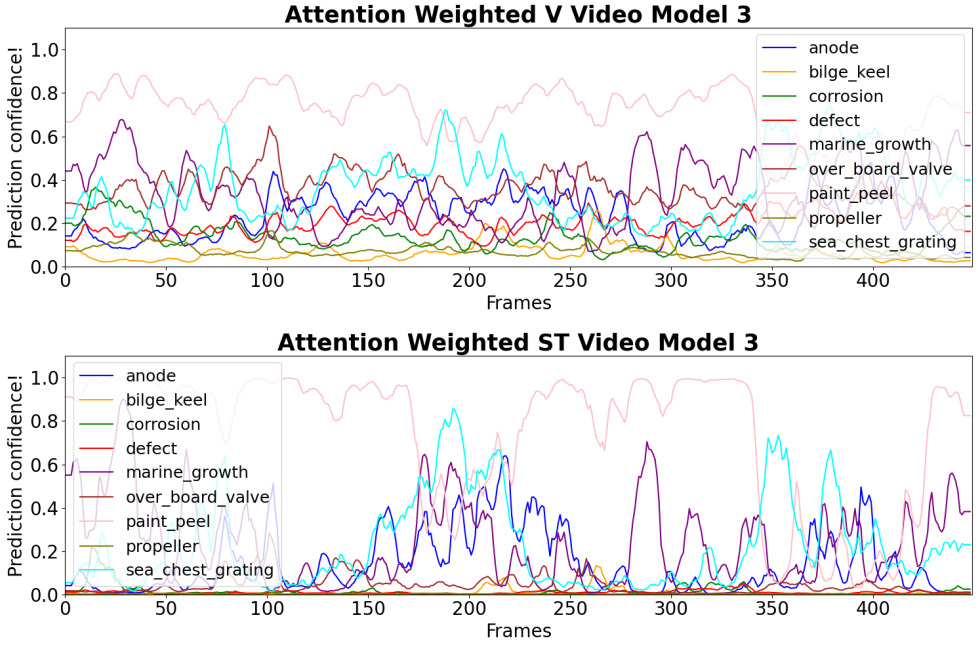


Figure A.34: Temporal observation of the best two attention-weighted video models on the snippet no.6 from table 3.2.

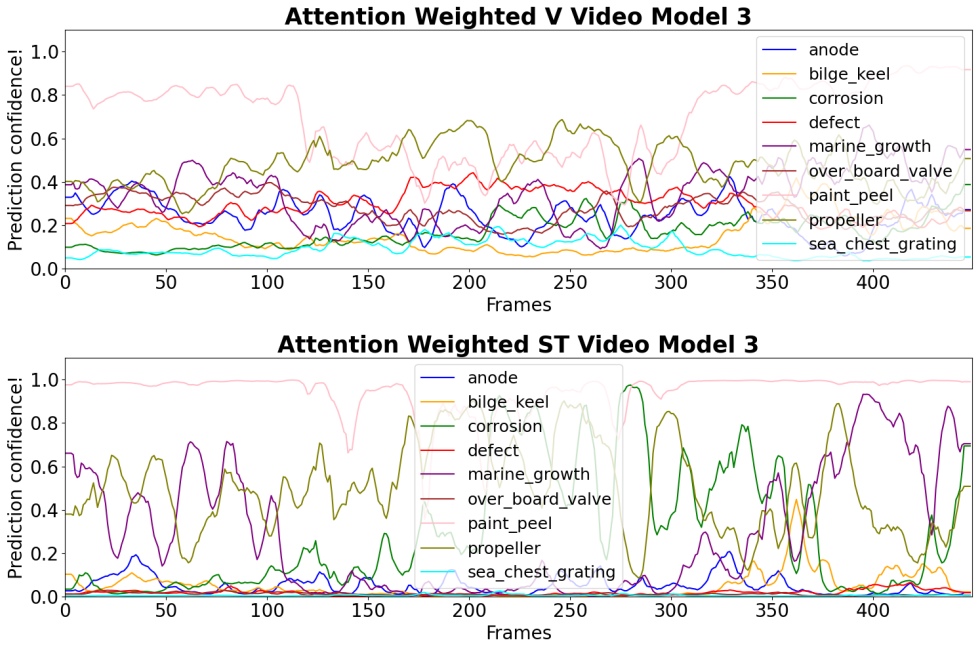


Figure A.35: Temporal observation of the best two attention-weighted video models on the snippet no.7 from table 3.2.

Appendix B

Code Listings

All the reference codes will be attached here in this chapter.

B.1 Attention

The function below computes the attention given to the query, key, and value matrices. It also handles the single query attention computation.

Code listing B.1: The function to compute attention using Q, K, and V

```
def attention(query, key, value, mask=None, dropout=None):
  "Compute␣'Scaled␣Dot␣Product␣Attention'"
  # d_k: head dimension, usually set to D/h
  d_k = query.size(-1)
  # attention score before softmax
  scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)
  # currently, we are not using mask in our project
  if mask is not None:
      scores = scores.masked_fill(mask == 0, -1e9)
  # probability of attention score (after softmax)
  p_attn = scores.softmax(dim=-1)
  if dropout is not None:
      p_attn = dropout(p_attn)
  if query.size(-2) == 1 and not key.size(-2) == 1:
    # if a single query attention, element-wise multiplication
    attn_v = p_attn.transpose(-1,-2) * value
  else:
    # else matrix multiplication
    attn_v = torch.matmul(p_attn, value)
  return attn_v, p_attn
```

B.2 Multi-Head Attention

The below codes execute the multi-head attention using the attention function. If the query is given as a single query, it will compute single query multi-headed attention.

Code listing B.2: Multi-head attention computation.

```
class MultiHeadedAttention(nn.Module):
  def __init__(self, h, d_model, dropout=0.1):
    "Take␣in␣model␣size␣and␣number␣of␣heads."
    super(MultiHeadedAttention, self).__init__()
    assert d_model % h == 0
    # We assume d_v always equals d_k
    self.d_k = d_model // h
    self.h = h
    self.linears = clones(nn.Linear(d_model, d_model), 4)
    self.attn = None
    self.dropout = nn.Dropout(p=dropout)

  def forward(self, query, key, value, mask=None):
    # currently, we are not using mask
    if mask is not None:
      # Same mask applied to all h heads.
      mask = mask.unsqueeze(1)
    # number of batches
    nbatches = query.size(0)

    # 1) Do all the linear projections in batch from d_model => h x d_k
    query, key, value = [
        lin(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
        for lin, x in zip(self.linears, (query, key, value))
    ]

    # 2) Apply attention on all the projected vectors in batch.
    x, self.attn = attention(
        query, key, value, mask=mask, dropout=self.dropout
    )

    # 3) "Concat" using a view and apply a final linear.
    x = (
        x.transpose(1, 2)
        .contiguous()
        .view(nbatches, -1, self.h * self.d_k)
    )
    del query
    del key
    del value
    return self.linears[-1](x)

# composes a stack of N=6 identical layers togather.
def clones(module, N):
    "Produce␣N␣identical␣layers."
    return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
```

B.3 Vision Transformer (Spatial Transformer)

The below codes represent the implementation of the standard Vision Transformer available in torchvision. We use this code also for our spatial transformer.

Code listing B.3: Standard Vision Transformer Implementation.

```
class VisionTransformer(nn.Module):
  """Vision Transformer as per https://arxiv.org/abs/2010.11929."""
```

```
  def __init__(
      self,
      image_size: int,
      patch_size: int,
      num_layers: int,
      num_heads: int,
      hidden_dim: int,
      mlp_dim: int,
      dropout: float = 0.0,
      attention_dropout: float = 0.0,
      num_classes: int = 1000,
      representation_size: Optional[int] = None,
      norm_layer: Callable[..., torch.nn.Module] = partial(nn.LayerNorm, eps=1e-6),
  ):
      super().__init__()
      _log_api_usage_once(self)
      torch._assert(image_size % patch_size == 0,
                    "Input␣shape␣indivisible␣by␣patch␣size!")
      self.image_size = image_size
      self.patch_size = patch_size
      self.hidden_dim = hidden_dim
      self.mlp_dim = mlp_dim
      self.attention_dropout = attention_dropout
      self.dropout = dropout
      self.num_classes = num_classes
      self.representation_size = representation_size
      self.norm_layer = norm_layer
      self.conv_proj = nn.Conv2d(
              in_channels=3, out_channels=hidden_dim,
              kernel_size=patch_size, stride=patch_size
          )

      seq_length = (image_size // patch_size) ** 2

      # Add a class token
      self.class_token = nn.Parameter(torch.zeros(1, 1, hidden_dim))
      seq_length += 1

      self.encoder = Encoder(
          seq_length,
          num_layers,
          num_heads,
          hidden_dim,
          mlp_dim,
          dropout,
          attention_dropout,
          norm_layer,
      )
      self.seq_length = seq_length

      heads_layers: OrderedDict[str, nn.Module] = OrderedDict()
      if representation_size is None:
          heads_layers["head"] = nn.Linear(hidden_dim, num_classes)
      else:
          heads_layers["pre_logits"] = nn.Linear(hidden_dim, representation_size)
          heads_layers["act"] = nn.Tanh()
          heads_layers["head"] = nn.Linear(representation_size, num_classes)

      self.heads = nn.Sequential(heads_layers)
```

```
        # define the pool
        self.pool = 'cls' #'mean'

    """ Process the given input to the model """
    def _process_input(self, x: torch.Tensor) -> torch.Tensor:
        n, c, h, w = x.shape
        p = self.patch_size
        torch._assert(h == self.image_size,
                      f"Wrong image height! Expected {self.image_size} but got {h}!")
        torch._assert(w == self.image_size,
                      f"Wrong image width! Expected {self.image_size} but got {w}!")
        n_h = h // p
        n_w = w // p

        # (n, c, h, w) -> (n, hidden_dim, n_h, n_w)
        x = self.conv_proj(x)
        # (n, hidden_dim, n_h, n_w) -> (n, hidden_dim, (n_h * n_w))
        x = x.reshape(n, self.hidden_dim, n_h * n_w)

        # (n, hidden_dim, (n_h * n_w)) -> (n, (n_h * n_w), hidden_dim)
        # The self attention layer expects inputs in the format (N, S, E)
        # where S is the source sequence length, N is the batch size,
        # E is the embedding dimension
        x = x.permute(0, 2, 1)
        return x

    def forward(self, x: torch.Tensor):
        # Reshape and permute the input tensor
        x = self._process_input(x)
        n = x.shape[0]

        # Expand the class token to the full batch
        batch_class_token = self.class_token.expand(n, -1, -1)
        x = torch.cat([batch_class_token, x], dim=1)

        x = self.encoder(x)

        # Either the CLS token or the global average can be passed to the MLP
        x = x.mean(dim = 1) if self.pool == 'mean' else x[:, 0]
        x = self.heads(x)
        return x

class Encoder(nn.Module):
    """Transformer Model Encoder for sequence to sequence translation."""
    def __init__(
        self,
        seq_length: int,
        num_layers: int,
        num_heads: int,
        hidden_dim: int,
        mlp_dim: int,
        dropout: float,
        attention_dropout: float,
        norm_layer: Callable[..., torch.nn.Module] = partial(nn.LayerNorm, eps=1e-6),
    ):
        super().__init__()
        # Note that batch_size is on the first dim because
        # we have batch_first=True in nn.MultiAttention() by default
        self.pos_embedding = nn.Parameter(torch.empty(
            1, seq_length, hidden_dim).normal_(std=0.02))  # from BERT
```

```
        self.dropout = nn.Dropout(dropout)
        layers: OrderedDict[str, nn.Module] = OrderedDict()
        for i in range(num_layers):
            layers[f"encoder_layer_{i}"] = EncoderBlock(
                num_heads,
                hidden_dim,
                mlp_dim,
                dropout,
                attention_dropout,
                norm_layer,
            )
        self.layers = nn.Sequential(layers)
        self.ln = norm_layer(hidden_dim)

    def forward(self, input: torch.Tensor):
        torch._assert(input.dim() == 3,
                f"Expected (batch_size, seq_length, hidden_dim) got {input.shape}")
        input = input + self.pos_embedding
        return self.ln(self.layers(self.dropout(input)))

class EncoderBlock(nn.Module):
    """Transformer encoder block."""
    def __init__(
        self,
        num_heads: int,
        hidden_dim: int,
        mlp_dim: int,
        dropout: float,
        attention_dropout: float,
        norm_layer: Callable[..., torch.nn.Module] = partial(nn.LayerNorm, eps=1e-6),
    ):
        super().__init__()
        self.num_heads = num_heads

        # Attention block
        self.ln_1 = norm_layer(hidden_dim)
        self.self_attention = nn.MultiheadAttention(hidden_dim,
                            num_heads, dropout=attention_dropout, batch_first=True)
        self.dropout = nn.Dropout(dropout)

        # MLP block
        self.ln_2 = norm_layer(hidden_dim)
        self.mlp = MLPBlock(hidden_dim, mlp_dim, dropout)

    def forward(self, input: torch.Tensor):
        torch._assert(input.dim() == 3,
                f"Expected (batch_size, seq_length, hidden_dim) got {input.shape}")
        x = self.ln_1(input)
        x, _ = self.self_attention(query=x, key=x, value=x, need_weights=False)
        x = self.dropout(x)
        x = x + input
        y = self.ln_2(x)
        y = self.mlp(y)
        return x + y
```

B.4 Positional Encoding

The below codes represent the implementation of the positional encoding we have used for our temporal transformer.

Code listing B.4: Positional Encoding implementation.

```
class PositionalEncoding(nn.Module):
    "Implement␣the␣PE␣function."

    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2) * -(math.log(10000.0) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer("pe", pe)

    def forward(self, x):
        x = x + self.pe[:, : x.size(1)].requires_grad_(False)
        return self.dropout(x)
```

B.5 Temporal Transformer

The below codes represent the implementation of the temporal transformer utilizing attention, multi-head attention, and positional encoding presented earlier.

Code listing B.5: Temporal Transformer implementation.

```
class Transformer(nn.Module):
    """
    A standard transformer architecture.
    """

    def __init__(self, encoder, src_embed):
        super(Transformer, self).__init__()
        self.encoder = encoder
        self.src_embed = src_embed

    def forward(self, src, src_mask):
        "Take␣in␣and␣process␣masked␣src␣and␣target␣sequences."
        return self.encode(src, src_mask)

    def encode(self, src, src_mask):
        return self.encoder(src, src_mask)

class Encoder(nn.Module):
    "Core␣encoder␣is␣a␣stack␣of␣N␣layers"
```

```
    def __init__(self, layer, N):
        super(Encoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, mask):
        "Pass the input (and mask) through each layer in turn."
        for layer in self.layers:
            x = layer(x, mask)
        return self.norm(x)

class SublayerConnection(nn.Module):
    """
    A residual connection followed by a layer norm.
    Note for code simplicity the norm is first as opposed to last.
    """

    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm = LayerNorm(size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, sublayer):
        "Apply residual connection to any sublayer with the same size."
        return x + self.dropout(sublayer(self.norm(x)))

class EncoderLayer(nn.Module):
    "Encoder is made up of self-attn and feed forward (defined below)"

    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 2)
        self.size = size

    def forward(self, x, mask):
        "Follow Figure 1 (left) for connections."
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
        return self.sublayer[1](x, self.feed_forward)

class PositionwiseFeedForward(nn.Module):
    "Implements FFN equation."

    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(self.w_1(x).relu()))

class Embeddings(nn.Module):
    def __init__(self, d_model, vocab):
        super(Embeddings, self).__init__()
        self.lut = nn.Embedding(vocab, d_model)
        self.d_model = d_model

    def forward(self, x):
```

```
        return self.lut(x) * math.sqrt(self.d_model)
```

B.6 Reproduction of the ResNet model

The below codes import the trained ResNet model and execute a multi-label classification on a provided video snippet. It also generates a report as a diagram using *extract-summery* script added in B.7.

Code listing B.6: Load the ResNet model and perform multi-label classification.

```
"""
Created on Thu Oct 06 2022
@author: Azad
"""
import glob
import os
import argparse
import extract_summary

def parse_args():
    parser = argparse.ArgumentParser(description= "Run␣the␣multi-label␣video
␣␣␣␣␣␣␣␣␣␣␣␣classification␣with␣additional␣configuration␣&␣output␣options.")
    parser.add_argument("--input_type", default="", choices=["video",
            "plot_label"], help= "Specify␣the␣input␣type␣as␣video␣or
␣␣␣␣␣␣␣␣␣␣␣␣plot_label␣for␣drawing␣from␣existing␣results.")
    parser.add_argument("--input_path", default="", help=
            "Path␣to␣the␣folder␣contains␣all␣the␣videos.")
    parser.add_argument("--output_path", default="", help=
            "Specify␣the␣path␣to␣the␣output␣folder.")
    args = parser.parse_args()
    return args

if __name__ == "__main__":
    args = parse_args()

    if args.output_path == "":
        args.output_path = os.path.join(args.input_path, "output")

    if args.input_type == "video":
        if args.input_path == "":
            raise ValueError("Must␣specify␣the␣'--input_path'␣to␣the␣video␣folder.")
        for file_path in glob.glob(args.input_path+"/*.mp4"):
            snippet_name = os.path.splitext(os.path.basename(file_path))[0]
            snippet_csv_path = os.path.join(args.output_path, f"{snippet_name}.csv")
            extract_summary.classify_video(file_path, snippet_csv_path)
            extract_summary.plot_labels(file_path, snippet_csv_path)
    elif args.input_type == "plot_label":
        if args.input_path == "":
            raise ValueError("Must␣specify␣the␣'--input_path'␣to␣the␣video␣folder.")
        for file_path in glob.glob(args.input_path+"/*.mp4"):
            snippet_name = os.path.splitext(os.path.basename(file_path))[0]
            snippet_csv_path = os.path.join(args.output_path, f"{snippet_name}.csv")
            extract_summary.plot_labels(file_path, snippet_csv_path)
    else:
        raise ValueError("Must␣specify␣the␣'--input_type'␣as␣image␣or␣video")
```

Code listing B.7: Additional resource to reproduce the ResNet.

```
def classify(image):
    # onnxruntime inference
    sess = rt.InferenceSession(MODEL_FILENAME, providers=['CPUExecutionProvider'])
    # Get the input name and shape of the model
    input_name = sess.get_inputs()[0].name
    h,w = sess.get_inputs()[0].shape[2:]
    blob = cv2.dnn.blobFromImage(image=image, size=(w,h), swapRB=False, crop=False)
    start = time.time()
    #Running the session by passing in the input data of the model
    out = sess.run(None, {input_name: blob})
    end = time.time()
    inference_time = end - start
    scores = list(out[1][0].values())
    return scores

def classify_video(video_url,csv_file):
    cap = cv2.VideoCapture(video_url)
    TOTAL_NUM_OF_FRAMES = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    classification = {}
    for i in range(0,TOTAL_NUM_OF_FRAMES, 1):
        cap.set(cv2.CAP_PROP_POS_FRAMES, i)
        frame = cap.read()[1]
        outputs = classify(frame)
        value_dict = {}
        for key, value in zip(labels, outputs):
            value_dict[key] = value
        classification[i]= value_dict

    df = pd.DataFrame.from_dict(classification, orient='index')
    df.to_csv(csv_file)
    cap.release()

def plot_labels(video_url, csv_file):
    cap = cv2.VideoCapture(video_url)
    FPS = cap.get(cv2.CAP_PROP_FPS)
    label_classified = pd.read_csv(csv_file)
    np_labels = label_classified.to_numpy()
    fig, ax = plt.subplots(1,1,figsize=(20,5.5))
    img = ax.imshow(np_labels[:,1:].transpose(),aspect=12,interpolation = 'none')
    ax.set_yticks([0,1,2,3,4,5,6,7,8])
    x_ticks = np.arange(0, len(np_labels[:,1]), len(np_labels[:,1])/10)
    ax.set_xticks(x_ticks)
    xtick_labels=(x_ticks/FPS)
    ax.set_xticklabels([str(round(float(label), 2)) for label in xtick_labels],
                       fontsize=22)
    ax.set_yticklabels(['anode','bilge_keel','corrosion','defect','marine_growth',
                        'over_board_valve','paint_peel','propeller',
                        'sea_chest_grating'], fontsize=30)
    plt.tight_layout()
    ax.set_xlabel('Time␣[s]', fontsize=30)
    ax.set_title('Multi-label␣Video␣Classification', fontsize=30)
    fig.savefig(os.path.splitext(csv_file)[0] + '.png')
    plt.close(fig)
```

B.7 Implementation of Multi-label Image Classifiers

The below codes combined implementations of all the ViT image-based models we have produced in our works.

Code listing B.8: Implementation of ViT image models.

```
"""
Created on Wednesday Feb 02 2023
@author: Azad Md Abulkalam

The script creates and trains a multi-label vision transformer for multi-label
image classification.
"""
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import pandas as pd
import torchvision
from torchvision import models, transforms
import matplotlib.pyplot as plt
import time
import os
import argparse
from torch.utils.tensorboard import SummaryWriter

from datasets import CustomDataset
from multilabel_metric import MultMetric
import inference
import utils

classes = ['anode', 'bilge_keel', 'corrosion', 'defect', 'marine_growth',
    'over_board_valve', 'paint_peel', 'propeller', 'sea_chest_grating']

def getClass(name="LIACI"):
    if name=="LIACI":
        return 9
    elif name=="COCO":
        return 80
    elif name=="IMAGENET":
        return 1000
    else: # default LIACI
        return 9

# Data augmentation and normalization for training
# Just normalization for validation
def get_transform(train, model):
    data_transforms = {
        'LIACI': {
            'train': utils.Compose([
                #transforms.RandomResizedCrop(224), # resizing
                utils.GuidedCrop(label=0, p=0.5), #order of the transformation
                #is important here #label in here refers to the category that
                #should be cropped
                transforms.Resize((224, 224)),
                transforms.RandomHorizontalFlip(),
                transforms.RandomApply(torch.nn.ModuleList([
```

```
                transforms.GaussianBlur(kernel_size=(5, 9), sigma=(0.1, 5))]),
                                                                    p=0.5),
                transforms.RandomApply(torch.nn.ModuleList([transforms.AugMix()]),
                                                                    p=0.5),
                transforms.ToTensor(),
                transforms.Normalize([0.3485, 0.3699, 0.3520],
                                                [0.2495, 0.2446, 0.2062])
                ]),
            'val': transforms.Compose([
                transforms.Resize((224, 224)),
                #transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize([0.3485, 0.3699, 0.3520],
                                                [0.2495, 0.2446, 0.2062])
                ]),
            },
        'COCO': {
            'train': transforms.Compose([
                #transforms.RandomResizedCrop(224), # resizing
                transforms.Resize((224, 224)),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
                ]),
            'val': transforms.Compose([
                transforms.Resize((224, 224)),
                #transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
                ]),
            },
    }

    if train:
        return data_transforms[model]['train']
    else:
        return data_transforms[model]['val']



def parse_args():
    parser = argparse.ArgumentParser(description =
                        "Creates␣and␣trains␣a␣multi-label␣vision␣transformer.")
    parser.add_argument("--root_dir", default="", help=
            "Specify␣the␣path␣to␣the␣root␣directory␣of␣the␣dataset␣folder.")
    parser.add_argument("--model", default="COCO", choices=["COCO", "LIACI"],
                    help="Specify␣the␣model␣name␣for␣compatible␣architecture.")
    parser.add_argument("--weight", default="", choices=["IMAGENET","COCO",
    "LIACI"], help="Specify␣the␣pretrained␣weight␣to␣start␣the␣training␣from.")
    parser.add_argument("--checkpoint", default="", help=
                    "Path␣tp␣the␣saved␣model␣if␣pretrained␣weight␣is␣used.")
    parser.add_argument("--batch_size", type=int, default=16, help=
                                    "Define␣the␣batch␣size,␣otherwise␣16.")
    parser.add_argument("--n_worker", type=int, default=4, help=
                "Define␣the␣number␣of␣worker␣for␣multiprocessing,␣otherwise␣4.")
    parser.add_argument("--n_epoch", type=int, default=10, help=
                    "Define␣the␣number␣of␣epochs␣for␣training,␣otherwise␣10.")
    parser.add_argument("--runs_n", type=int, default=100, help=
                            "Define␣the␣experiment␣number,␣otherwise␣100.")
    parser.add_argument("--partial_train", action="store_true", help=
```

```
                                    "If␣partial␣finetuning␣is␣desireabled.")
    args = parser.parse_args()
    return args


def train_model(args, model, dataloaders, dataset_sizes, criterion, optimizer,
                                            scheduler, device, num_epochs=25):
    since = time.time()
    print(f'The␣device␣is␣being␣used␣for␣training:␣{device}')
    model = model.to(device)

    #tensorboard logs
    run_folder = os.path.join("runs", f"runs{args.runs_n}")
    model_folder = os.path.join("model_zoo", f"runs{args.runs_n}")

    # threshold for computing running accuracy
    metric = MultMetric(threshold=0.5)

    if args.model == "LIACI":
        best_model_path = os.path.join(f"{model_folder}",
        f"{'partial_finetuned'␣if␣args.partial_train␣else␣'fully_finetuned'}",
        f"{args.weight}_pretrain", "best_model")
        checkpoint_path = os.path.join(f"{model_folder}",
        f"{'partial_finetuned'␣if␣args.partial_train␣else␣'fully_finetuned'}",
        f"{args.weight}_pretrain", "checkpoint")
        log_path = os.path.join(f"{run_folder}",
        f"{'partial_finetuned'␣if␣args.partial_train␣else␣'fully_finetuned'}",
        f"{args.weight}_pretrain")
    elif args.model == "COCO":
        best_model_path = os.path.join(f"{model_folder}",
                "pretrain_models",
                f"{args.model}",
                "best_model")
        checkpoint_path = os.path.join(f"{model_folder}",
                "pretrain_models",
                f"{args.model}",
                "checkpoint")
        log_path = os.path.join(f"{run_folder}",
                "pretrained_models",
                f"{args.weight}_pretrain")

    if not os.path.exists(best_model_path):
        os.makedirs(best_model_path)
    if not os.path.exists(checkpoint_path):
        os.makedirs(checkpoint_path)
    if not os.path.exists(log_path):
        os.makedirs(log_path)

    #best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    running_eval = {}

    # inspection images
    ins_imgs, ins_labels = next(iter(dataloaders['val']))

    writer = SummaryWriter(log_dir=log_path)
    ins_imgs = ins_imgs.to(device)
    #writer.add_graph(model, ins_imgs)

    for epoch in range(num_epochs):
```

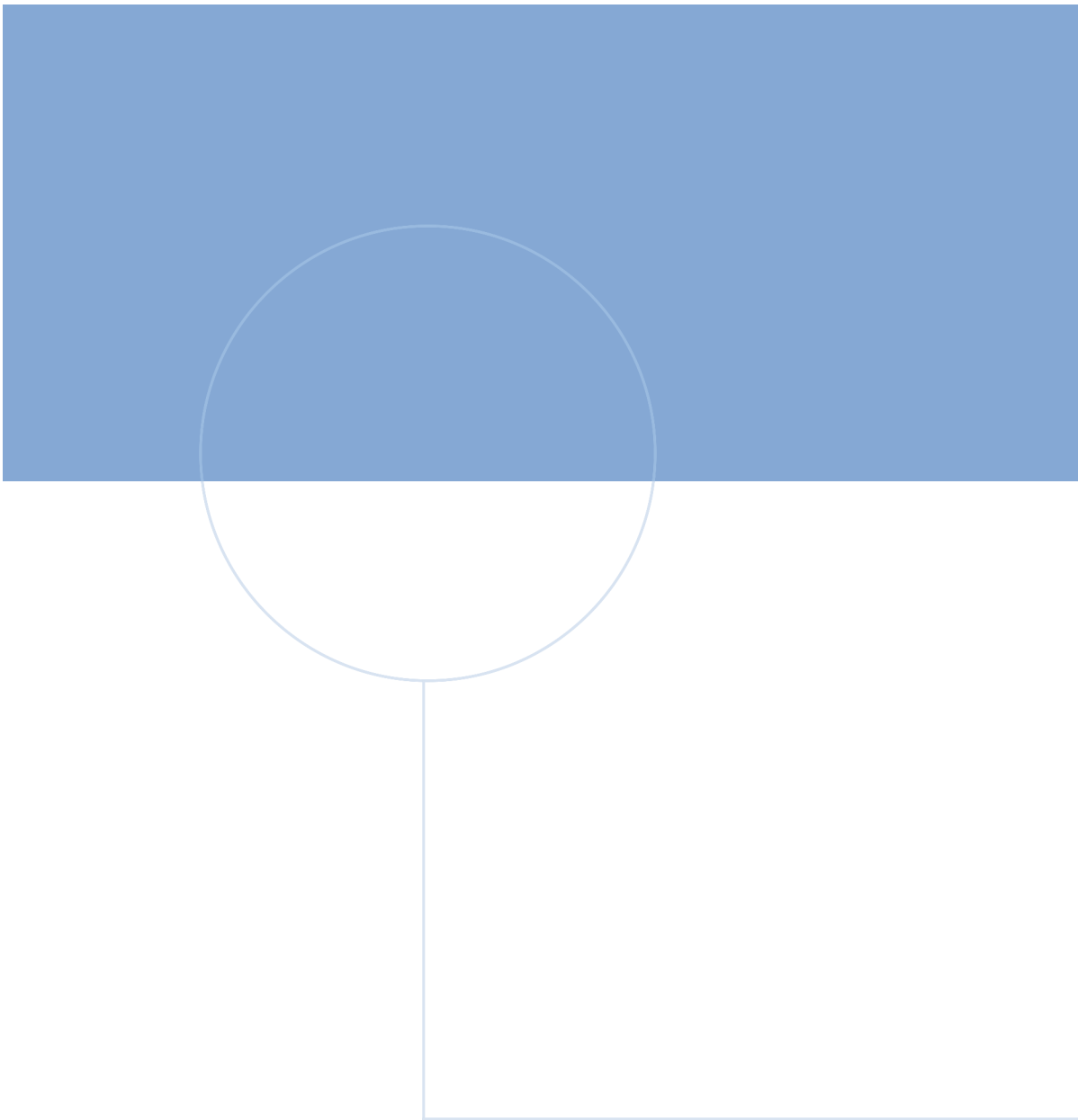
```
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_eval['loss'] = 0.0
            running_eval['accuracy'] = 0.0
            running_eval['precision'] = 0.0
            running_eval['f1-score'] = 0.0
            #running_eval['mAP'] = 0.0
            running_eval['recall'] = 0.0
            for cat in classes:
                running_eval[cat] = {
                    'precision': 0.0,
                    'recall': 0.0,
                    'f1-score': 0.0
                }

            # Iterate over data.
            #iter = 0
            for inputs, labels in dataloaders[phase]:
                #iter += 1
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    #_, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_eval['loss'] += loss.item() * inputs.size(0)
                runningMet = metric.metrics(outputs, labels)
                running_eval['accuracy'] += runningMet['accuracy'] * inputs.size(0)
                running_eval['precision'] += runningMet['report']['samples avg']
                                                ['precision'] * inputs.size(0)
                running_eval['f1-score'] += runningMet['report']['samples avg']
                                                ['f1-score'] * inputs.size(0)
                running_eval['recall'] += runningMet['report']['samples avg']
                                                ['recall'] * inputs.size(0)

                for cat in classes:
                    running_eval[cat]['precision'] += runningMet['report'][cat]
                                                ['precision'] * inputs.size(0)
                    running_eval[cat]['recall'] += runningMet['report'][cat]
```

```
                                        ['recall'] * inputs.size(0)
            running_eval[cat]['f1-score'] += runningMet['report'][cat]
                                        ['f1-score'] * inputs.size(0)


    epoch_report = {}
    epoch_report['loss'] = running_eval['loss'] /
                                        dataset_sizes[phase]
    epoch_report['accuracy'] = running_eval['accuracy'] /
                                        dataset_sizes[phase]
    epoch_report['precision'] = running_eval['precision'] /
                                        dataset_sizes[phase]
    epoch_report['f1-score'] = running_eval['f1-score'] /
                                        dataset_sizes[phase]
    epoch_report['recall'] = running_eval['recall'] /
                                        dataset_sizes[phase]
    for cat in classes:
        epoch_report[cat] = {
            'precision': running_eval[cat]['precision'] /
                                        dataset_sizes[phase],
            'recall': running_eval[cat]['recall'] /
                                        dataset_sizes[phase],
            'f1-score': running_eval[cat]['f1-score'] /
                                        dataset_sizes[phase]
        }

    # if phase == 'train':
    #     scheduler.step()
    # step should be called after validate() for ReduceLROnPlateau
    if phase == 'val':
        scheduler.step(epoch_report['loss'])

    #logs metrics to tensorboard
    writer.add_scalar(f"Loss/{phase}", epoch_report['loss'],
                                        epoch)
    writer.add_scalar(f"Accuracy/{phase}", epoch_report['accuracy'],
                                        epoch)
    writer.add_scalar(f"Precision/{phase}", epoch_report['precision'],
                                        epoch)
    writer.add_scalar(f"Recall/{phase}", epoch_report['recall'],
                                        epoch)
    writer.add_scalar(f"F1/{phase}", epoch_report['f1-score'], epoch)


    writer.add_scalars(f"Classes/Precision/{phase}", {cat: epoch_report[cat]
                                ['precision'] for cat in classes}, epoch)
    writer.add_scalars(f"Classes/Recall/{phase}", {cat: epoch_report[cat]
                                ['recall'] for cat in classes}, epoch)
    writer.add_scalars(f"Classes/F1/{phase}", {cat: epoch_report[cat]
                                ['f1-score'] for cat in classes}, epoch)

    #logs one batch of images from the epoch
    #logs model confidence on each class labels during every epoch
    if phase== 'val':
        _, probs = inference.images_to_probs(model, ins_imgs)
        for i in range(probs.shape[0]):
            writer.add_scalars(f"Images/{i}", {cat:
                    probs[i][j] for j, cat in enumerate(classes)}, epoch)

    #logs visual images along with confidence and ground truth
```

```
            #at every certain number of epochs
            if epoch%10==0 and phase == 'val':
                # create grid of images
                img_grid = inference.plot_classes_preds(args.model, model, ins_imgs,
                ins_labels, os.path.join(args.root_dir, "train", "labels",
                                                        "categories.csv"))
                writer.add_image(f"{epoch}/images", img_grid)



            # saving the training checkpoint
            if phase == 'train':
                torch.save({
                    'epoch': epoch,
                    'model_state_dict': model.state_dict(),
                    'optimizer_state_dict': optimizer.state_dict(),
                    'loss': loss,
                    }, os.path.join(checkpoint_path,
                                    f"mvit_last_cp_{args.model}.pt"))


            if phase == 'val' and epoch_report['accuracy'] > best_acc:
                best_acc = epoch_report['accuracy']
                #best_model_wts = copy.deepcopy(model.state_dict())
                # saving the best model
                torch.save({
                    'epoch': epoch,
                    'model_state_dict': model.state_dict(),
                    'optimizer_state_dict': optimizer.state_dict(),
                    'loss': loss,
                    }, os.path.join(best_model_path, f"mvit_{args.model}.pt"))

        print()
    writer.flush()
    writer.close()
    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s
                                                                    for {args.model}')
    print(f'Best val Acc: {best_acc:4f}')
    print(f'Best model is saveed in {best_model_path}')
    print(f'Last checkpoint is saveed in {checkpoint_path}')


'''
To load either LIACI or COCO dataset
root_dir: path to the root directory of the dataset
name: 'LIACI' or 'COCO'

Returns a dict containing dataloader and dataset size
'''
def loadDataset(args):#root_dir, batch_size, n_workers, name="LIACI"):
    dataset = {}

    dataset['name'] = args.model
    #Dataset & Dataloader from COCO folder -----------------------------#
    train_dataset = CustomDataset(args.root_dir, train=True, transforms=
                                get_transform(train=True, model=args.model))
    val_dataset = CustomDataset(args.root_dir, val=True, transforms=
                                get_transform(train=False, model=args.model))
```

```
    # Dataloader for train dataset
    train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=
                        args.batch_size, shuffle=True, num_workers=args.n_worker)
    # Dataloader for train dataset
    val_dataloader = torch.utils.data.DataLoader(val_dataset, batch_size=
                        args.batch_size, shuffle=True, num_workers=args.n_worker)

    data_loaders = {
        'train': train_dataloader,
        'val': val_dataloader
    }
    dataset_sizes = {
        'train': len(train_dataset),
        'val': len(val_dataset)
    }
    dataset['dataloader'] = data_loaders
    dataset['size'] = dataset_sizes

    # debugging with subsets of the dataset
    train_indices = torch.randperm(dataset_sizes["train"]).tolist()
    val_indices = torch.randperm(dataset_sizes["val"]).tolist()

    dataset_train = torch.utils.data.Subset(train_dataset, train_indices[:100])
    dataset_val = torch.utils.data.Subset(val_dataset, val_indices[:20])
    sub_train_dataloader = torch.utils.data.DataLoader(dataset_train, batch_size=
                        args.batch_size, shuffle=True, num_workers=args.n_worker)
    sub_val_dataloader = torch.utils.data.DataLoader(dataset_val, batch_size=
                        args.batch_size, shuffle=True, num_workers=args.n_worker)
    data_loaders = {
        'train': sub_train_dataloader,
        'val': sub_val_dataloader
    }
    dataset_sizes = {
        'train': len(dataset_train),
        'val': len(dataset_val)
    }
    dataset['dataloader'] = data_loaders
    dataset['size'] = dataset_sizes
    return dataset

'''
To select a model either for LIACI or COCO dataset
name: 'LIACI' or 'COCO'

Returns a pretrained ViT (16) model with setting the head according
to LIACI (9 classes) or COCO (80 classes)
'''
def selectModel(args):
    # declaring the base model
    if args.weight == "IMAGENET":
        model = models.vit_b_16(weights=models.ViT_B_16_Weights.DEFAULT)
    else:
        model = models.vit_b_16()
        # default number of classes
        num_ftrs = model.heads.head.in_features
        model.heads.head = nn.Linear(num_ftrs, getClass(args.weight))
        # loading weight from the given checkpoint
        if (os.path.exists(args.checkpoint)):
            checkpoint = torch.load(args.checkpoint)
            model.load_state_dict(checkpoint['model_state_dict'])
```

```
        else:
            raise Exception("Checkpoint␣doesn't␣exist!!!")

    if args.model != args.weight:
        num_ftrs = model.heads.head.in_features
        model.heads.head = nn.Linear(num_ftrs, getClass(args.model))

    # if partial training is one
    if args.partial_train:
        for name, param in model.named_parameters():
            if 'head' in name:
                param.requires_grad = True
            else:
                param.requires_grad = False
    return model


if __name__ == '__main__':
    args = parse_args()

    # load the prefered dataset: either 'COCO' or 'LIACI'
    dataset = loadDataset(args)
    print(f"Name␣of␣the␣dataset:␣{dataset['name']}")
    print(f'Train␣images:␣{dataset["size"]["train"]}')
    print(f'Validation␣images:␣{dataset["size"]["val"]}')
    #---------------------- Some Debugging ----------------------------#


    #ViT finetune and Training -----------------------------#
    mvit = selectModel(args)
    # The loss funtion should be binary cross entropy for
    #multi-label classification problem
    criterion = nn.BCEWithLogitsLoss()
    if not args.partial_train:
        # Observe that all parameters are being optimized
        optimizer_ft = optim.SGD(mvit.parameters(), lr=0.001, momentum=0.9)
        #optimizer_ft = optim.Adam(mvit.parameters(), lr=0.001,
        #weight_decay=0.3)
    else:
        # Observe that only parameters of final layer are being optimized
        optimizer_ft = optim.SGD(mvit.heads.parameters(), lr=0.001, momentum=0.9)
        #optimizer_ft = optim.Adam(mvit.heads.parameters(), lr=0.001,
        #weight_decay=0.3)
    # Decay LR by a factor of 0.1 every 7 epochs
    #exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=50, gamma=0.1)
    exp_lr_scheduler = lr_scheduler.ReduceLROnPlateau(optimizer_ft, mode='min',
                                                      factor=0.1)
    # set the device
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    mvit_ft = train_model(args=args, model=mvit, dataloaders=dataset['dataloader'],
        dataset_sizes=dataset['size'], criterion=criterion, optimizer=optimizer_ft,
        scheduler=exp_lr_scheduler, device=device, num_epochs=args.n_epoch)
```



NTNU
Norwegian University of
Science and Technology

SINTEF