Segmenting fire from wilidfire using YOLO and Attention U-net (24-1-R-19)

Prof. Miri Weiss Cohen
Avner Ben Shlomo and Dor Filis

Braude College of Engineering, Karmiel, Israel
Avner.Ben.Shlomo@e.braude.ac.il and Dor.Filis@e.braude.ac.il

Table of Contents

Abstract. Forest fires can be caused by various factors, including extreme weather events or electrical failures, leading to property damage and loss of human life. In this project, we developed a method for segmenting live streams captured by drones in real-time, with a focus on achieving maximum accuracy while ensuring optimal performance for displaying segmentation results promptly. We implemented a model using YOLOv4 for detecting fires in each frame. YOLO provided bounding boxes indicating the fire's location, which were padded and then passed into an Attention U-Net for precise segmentation. After experimenting with various hyperparameters such as learning rate, epochs, batch size, and the binary cross-entropy loss function, we concluded that a learning rate of 1e-4, batch size of 16, and "Focal Loss" provided the best results for real-time segmentation. This project highlighted the critical role of hyperparameter tuning and model architecture, demonstrating how small adjustments can significantly impact model performance. Our final model showed great promise in identifying large forest fires in real-time, potentially aiding in the prevention of property damage and loss of human life.

Keywords: Wildfire, Real-time segmentation, YOLOv4, Attention U-Net.

1 Introduction

Throughout the world today, fires devastate entire communities, farmlands, and animal habitats due to natural causes. Fires are becoming more frequent as a result of the effects of global warming, which is the consequence of rising temperatures on an annual basis. Fires occur more frequently as temperatures continue to rise each year. These fires can persist for days or even weeks, requiring significant time and resources to contain. An early detection of a wildfire is essential for prompt intervention and the prevention of substantial damage. In particular, Strong winds and dense forest areas contribute to the rapid spread of the fire, making the timely identification of a spreading fire essential.

Fig. 1: Photo: Shutterstock, VanderWolf Images [7]

Fig. 2: Fires in the forests of Siberia / Photo: Julia Petrenko / Greenpeace [3]

The following examples illustrate significant wildfires that have caused extensive damage in the last five years (2019-2023): The Australian bushfires between 2019 and 2020, deemed an ecological disaster, destroyed 186,000 square kilometers of forests, killing 34 people and approximately 3 billion animals. 31 people were killed in wildfires in California, USA, which destroyed 16,000 square kilometers of forests and approximately 10,000 structures [3].

A majority of wildfire detection methods currently rely on human observation or technology. In spite of this, human observation has its limitations, requiring a large workforce to monitor a large forest areas. However, relying only on human observation, even with the aid of surveillance tools like binoculars, may not provide effective coverage of the entire area. Technological surveillance methods relying on sensor systems such as heat, noise, or electrical conductivity may face challenges due to background noise for audio sensors or interference for thermal sensors. Here are some approaches to detecting wildfires. **Sensor and Long-Range Camera Systems :** In order to detect wildfires from a distance, these systems use optical and thermal sensors as well as long-range cameras. They offer rapid and extensive wildfire detection, but their fixed nature limits their range, and they may not function optimally under conditions of poor visibility.
Ground-Based Sensor Systems : Sensors embedded in the ground are used to detect changes in electrical conductivity that are not related to tree falls or damage to power lines. While they are capable of identifying wildfires in remote areas based on ground conditions, they may produce false positives due to factors such as decaying trees or strong winds.
Sound-Based Technologies : By using microphones or sound sensors, these systems can detect sharp noises caused by falling trees or cracking branches. Their ability to identify specific frequencies associated with falling trees or fire noises allows them to provide immediate notification to the user. The effectiveness of these devices may, however, be compromised in noisy environments.
Floating Devices (Balloons or Spheres) : Systems such as these deploy

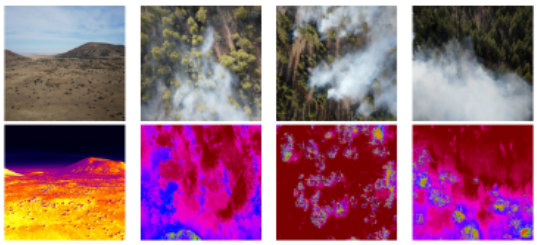


Fig. 3: FLAME 2: Fire detection and modeling - Aerial Multi-spectral image dataset [6]

floating devices equipped with sensors that detect atmospheric changes, such as pollution or smoke. Although they are capable of detecting wildfires quickly in challenging terrain, they are limited in conditions of heavy fog. Obtaining real-time data on wildfires is difficult due to their unpredictable nature. For instance,

the FLAME 2 [6] dataset includes videos and images of wildfires captured using two types of cameras: a standard video camera and an infrared camera. This dual-captured imagery enables effective comparison. The videos are aerial shots taken by drones and come with labeling indicating the presence of fire and smoke. This project proposes a method of detecting wildfires using drones, displaying real-time segmentation on video to mark the location of the fire. To address the wildfire detection challenge, we will demonstrate the accuracy of our model and provide an overview of our deep learning solution. Our solution is expected to aid in early wildfire detection, preventing significant harm to human lives and ecological damage.

Fig. 4: A drone flying over a mountain [1]

2 Literature Review

Automated fire segmentation using AI and CNN models has gained significant attention in recent studies. Below is a summary of key contributions in this area: Wang et al [14]. (2022) conducted a comparative study on various CNN-based approaches for fire segmentation in forest environments. The researchers used a large image dataset captured by UAVs, primarily from the FLAME dataset, which was divided into two types: 95.23% of images with fire in forest settings and images with visual elements that could confuse models, like fire-like colors or varying weather conditions. They evaluated several popular semantic segmentation models, including:
FCN (Fully Convolutional Networks): This model uses convolutional layers to retain the image's structure and assigns features to each pixel, resulting in a segmentation map.
U-Net: Designed for tasks like object boundary detection, U-Net features skip connections between its encoding and decoding paths, ensuring the preservation of details across different resolution levels.
PSPNet (Pyramid Scene Parsing Network): Known for its ability to handle large images, PSPNet uses Atrous Convolution and Spatial Pyramid Pooling to capture features at various scales, improving accuracy.
DeepLabV3+: A sophisticated model that also employs Atrous Convolution and pooling techniques, DeepLabV3+ combines high-resolution outputs with feature maps from different layers to enhance segmentation precision.
Their results showed that the U-Net model with a ResNet50 backbone achieved

the highest accuracy (99.91%) but was relatively slow. In contrast, DeepLabV3+ with ResNet50 achieved a slightly lower accuracy (99.89%) but offered faster processing speeds, making it better suited for real-time applications.
Muksimova (2022) [8] explored real-time fire and smoke segmentation in UAV-captured images using an EfficientNetv2-based model. EfficientNetv2 is an advanced deep learning model composed of an encoder-decoder architecture with an attention gate mechanism that allows the model to focus on important areas while ignoring irrelevant information. The model was trained on a dataset of 37,526 images of wildfires and smoke, available from public sources. The study showed that this model delivered excellent accuracy and performed well in real-time fire detection, which has the potential to enhance wildfire monitoring systems.
Ghali (2023) [5] reviewed multiple models for wildfire detection, classification, and segmentation. The study evaluated different datasets, including BowFire, FLAME, and CorsicanFire, highlighting their effectiveness in various wildfire scenarios. The models were categorized by their specific tasks: classification, detection, and segmentation. For classification, EfficientNet-B5 and DenseNet201 were found to be the fastest (55.55 FPS), while in segmentation, wUUNet reached the highest frame rate (63 FPS). These results indicate the models' high potential for real-time fire detection.
Chen (2022) [4] presented a dual-camera fire detection approach using both RGB and thermal imaging. Their model processed pairs of images from the FLAME2 dataset, consisting of regular and thermal data. The model architecture involved multiple convolutional layers and a global pooling layer, achieving an accuracy of 94%. The use of thermal imaging enabled precise detection and tracking of fire in real-time, which could greatly improve the speed and accuracy of fire monitoring systems.

3 Background

In this work, our goal is to identify and precisely locate wildfires using drones equipped with cameras. Employing segmentation techniques, our objective is to accurately detect the affected areas. To achieve this, we will utilize well-known models such as YOLOv4 and Attention U-Net. Moving forward, we will provide a detailed overview of these models and their functionalities.

3.1 YOLOv4 (You Only Look Once)

Object detection is an important task in computer vision, addressing the challenge of precisely identifying and locating objects of interest in images or videos and label them with an appropriate class label. The object detection model learns to identify and locate objects by minimizing a loss function that measures the difference between predicted and ground-truth labels and bounding boxes. These models are used in applications, such as autonomous driving, surveillance,

and robotics.
In recent years, there has been significant progress in the field of object detection. object detectors are classified into two categories viz. two stage and single stage object detectors. The two stage approach in her first step predict the bounding boxes and in second stage predict the class and bounding box information. This approach is slower than other detectors but has high accuracy. One-stage object detection in one forward pass predict everything, These detectors are much faster than two-stage.
YOLO is the state of the art object detection model that utilizes deep learning, initially developed by Joseph Redmon and Ali Farhadi in 2015. Since its inception, there have been eight different versions, with the latest being YOLO V8 released in 2023. The main differences between the versions are development of new features in order to improve runtime efficiency.
YOLO, short for "You Only Look Once," is a convolutional neural network that not only predicts bounding boxes but also calculates class probabilities for all objects present within an image. As this algorithm identifies the objects and their positioning with the help of bounding boxes by looking at the image only once, hence they have named it as You Only Look Once, reflecting its efficiency, which makes it highly suitable for real-time object detection tasks. The main challenge lies in the accurate identification of multiple objects along with their exact positioning present in a single image In Yolo object detection process, the

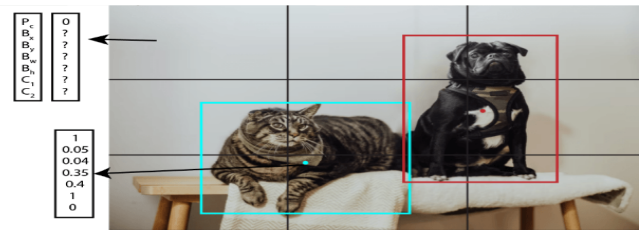


Fig. 5: The vector [15]

image is divided into S × S grid cells, each grid cell predicts B bounding boxes along with their positions and dimensions, probability of the classify. The fundamental concept behind detection of an object by any grid cell is that the center of an object should be inside that grid cell.
It is necessary to resolve the problem of detecting the same object in multiple grid cells or in multiple bounding boxes of the same grid cell. Non max suppression internally uses an important concept of Intersection over Union (IoU) which can be computed for two boxes. First, we select the box having the maximum class score. All other bounding boxes overlapped with the chosen box will be discarded having IoU is greater than some predefined threshold. We repeat these steps until there are no bounding boxes with lower confidence scores than the chosen bounding box. In YOLO v4, the processing begins with the input image, which then goes to feature extraction within the backbone. Following this, the features are passed to the neck, responsible for aggregating them to generate

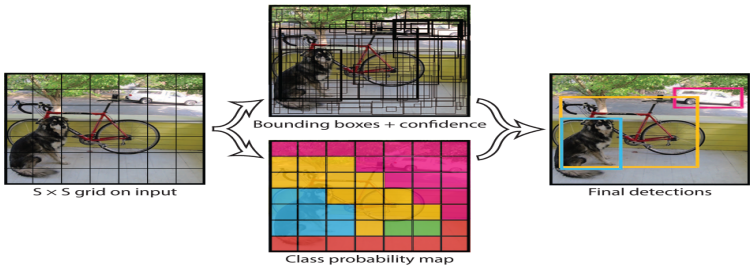


Fig. 6: The Model [10]

optimal features for predictions. Finally, the processed features are forwarded to the head, where both object detection bounding box prediction and class prediction take place. In YOLO v4, the backbone employed is the CSP Darknet, chosen for its effectiveness in feature extraction. For the neck, a combination of SPP and PAN is utilized to enhance feature aggregation, while the detection head employs the YOLO architecture, ensuring accurate and efficient predictions.

Backbone: CSPNet employs an approach where the input channels are divided

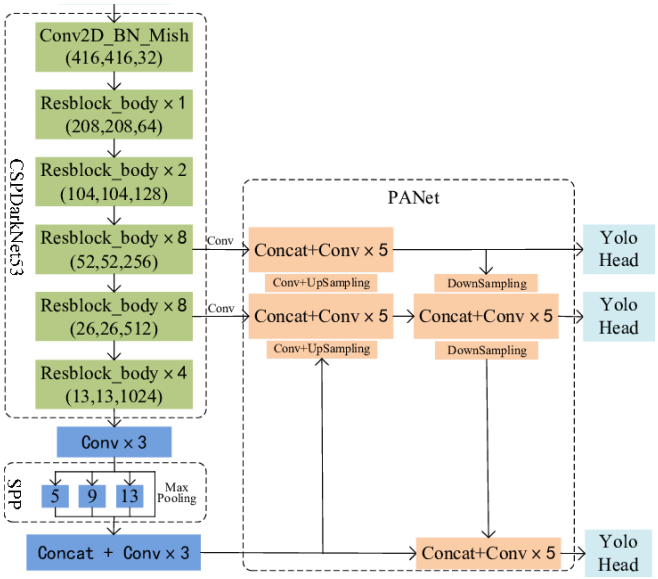


Fig. 7: Architecture of YOLOv4 [2]

into two halves. One half serves as the input to the computation block, while the other half is concatenated with the block's output afterward. This computation block can be a DenseBlock. Unlike traditional architectures, where layers are connected sequentially, DenseBlock ensures that every layer is connected to all others. In this setup, half of the features go through the dense block, while the remaining half bypasses it, feeding directly into the transition layer. Notably, in CSPDarknet, the activation function has been replaced with Mish, a novel acti-

vation function known for its smoother gradients and improved performance.
Neck: The neck plays an important role in collecting information from different levels of feature maps coming from the backbone and merging them effectively for detection purposes. It's crucial for small object detection, as it combines spatial information required for their accurate identification. As we propagate through the network and ascend to higher levels, we gain more semantic information, which is also essential for detecting small objects. What FPN (Feature Pyramid Network) accomplishes is to combine the high-level feature maps with lower-level ones, generating a new set of feature maps. PAN (Path Aggregation Network) extends upon FPN, introducing an additional branch of the bottom-up path. This innovation shortens the information path. Additionally, SPP (Spatial Pyramid Pooling) is integrated on top of the backbone to extract more significant context features. In YOLOv4, instead of a polling layer, an SPP layer is employed after the final convolutional layer. This involves dividing the final feature map into a fixed number of cells and performing pooling for each cell, resulting in a fixed-length feature vector.

3.2 Attention U-Net

In order to mark the wildfires in the project, we decided to use the Attention U-Net model to segment the wildfires. The U-Net model is well-known for its excellent segmentation results, and when combined with Attention, it performs even better. Now, let's explain what segmentation is, how the U-Net architecture works, and how we can integrate Attention blocks to enhance its performance even further.
Segmentation: There are tasks in deep learning that require us to classify objects based on their context within an image. Segmentation is one method for accomplishing this. In segmentation, an image is divided into regions and colored according to their color. An object is represented by a region, while a class is represented by a color. In our project, we intend to highlight the fire in one color and the forest in another so that the appropriate classification can be displayed. Pixels with similar features are grouped together by segmentation. We will first pass the image through filters in order to understand its features and then we will enter them into clusters based on the common denominator. As a first step, we pass the image through filters to identify its features, find commonalities, and then classify it according to these features. This presentation will provide an overview of the U-Net model for segmentation and demonstrate how it works.

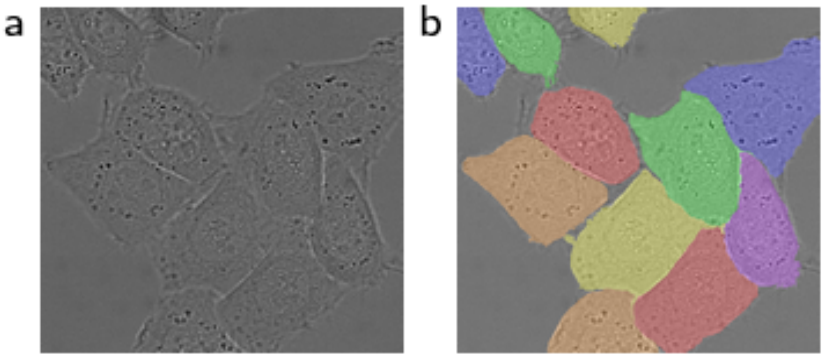


Fig. 8: Segmentation using U-net [11]

U-Net: The U-Net architecture was first introduced by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015 [11]
At first, it was developed to address the segmentation problem in the medical field, but was quickly adopted for a number of other applications. It consists of an encoder, a decoder, and a bottleneck. In the following sections, we will discuss each of these components in greater detail.

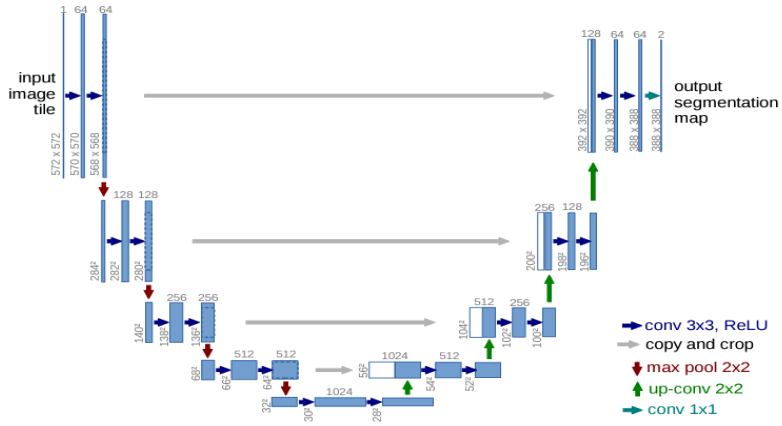


Fig. 9: U-net architecture[11]

The encoder is responsible for extracting the features from the image. In order to accomplish this, it uses downsampling. This can be viewed as a descending process. During each stage, we receive data in the form of dimensions (w,h,d). A convolution of 3X3 with ReLU activation function is performed twice in each such block. Two recipients receive the result. The first is to a skip connection, which we will explain later, and the second is to a max pool of 2X2, which will be the input for the next stage. By reducing the width and length of the image and increasing the depth of the filters in each block, we reduce the dimensions

of the image. The U-Net designer can decide how many downsampling blocks he wants to add based on the resolution we enter as our initial input. The number of these blocks was 4 in the original paper that introduced the U-Net, when the input images were (572,572,1) and the output of the last block was (32,32,512). After the downsampling, the bottleneck is actually what connects the upsampling and the downsampling. Convolution of 3X3 with the ReLU activation function is performed twice here, followed by upsampling.
The final step is the decoder, also known as upsampling. At this stage, we also divide into blocks. There are two UPConv 2X2 operations in each block. Essentially, it enlarges the low-resolution image. The second operation is the skip connection. Parallel to each upsampling block, there is a downsampling block with the same dimensions. The actual process involves concatenating all the filters we obtained during downsampling with the UPConv result, thereby doubling our features. It is this characteristic that distinguishes the U-Net from other models. A convolution of 3X3 is then performed twice with an activation function and sent to the next block. As a result of downsampling, the number of blocks will always be symmetrical. Lastly, before we output the image to the output, we add another convolution layer of 1X1. By using this layer, we reduce the depth of the features while maintaining the height and width of the image, and thus we output the image in exactly the same dimensions as the input image.
The architecture has demonstrated good performance since it was presented in 2015, beating the sliding-window convolutional network at the same time with a warping error of 0.0003529 and a random error of 0.0382.

Rank	Group name	Warping Error	Rand Error	Pixel Error
	** human values **	0.000005	0.0021	0.0010
1.	u-net	**0.000353**	0.0382	0.0611
2.	DIVE-SCI	0.000355	0.0305	0.0584
3.	IDSIA [1]	0.000420	0.0504	0.0613
4.	DIVE	0.000430	0.0545	**0.0582**
⋮				
10.	IDSIA-SCI	0.000653	**0.0189**	0.1027

Fig. 10: U-net result [11]

Attention: In essence, attention consists of focusing on what interests us and ignoring what does not. As part of the U-Net training process, attention is used to emphasize relevant activations. Furthermore, it reduces processing time by not referring to irrelevant information and allows the network to be more generalized. Attention can be classified into two types as follows [9]:
The method of hard attention involves dividing the image into smaller portions. Due to the small area covered by the model each time, the model is incapable of differentiating, and this requires strong learning that cannot be achieved using

backpropagation. A network can either focus on something or not; there is no middle ground.
In soft attention, areas in the image that are more relevant are given a higher weight than those that are unrelated. Backpropagation is possible due to the weight method. Backpropagation allows us to learn from our network, to update the weights continuously and to become more accurate, allowing us to pay more attention to relevant areas.
So how does the Attention block look? It has 2 inputs: the first is X, the second is G (gating signal). First, we will pass G through a convolution of 1X1 and a certain number of filters that we want to get immediately. We will explain how to determine the number of filters for input X. We will perform a convolution of 2X2 to reduce the dimensions of the length and width of the image by half and leave the number of filters the same. The number of filters of X is the number of filters we want G to have after the convolution. This is because the next step will be to connect them with the weights, which actually helps us to emphasize the large weights. Then we will pass them through the ReLU activation function. Then the result will pass through a convolution layer of 1X1 with a number of filters of 1, which will reduce all the depth of the filters to 1. Then we will pass them through a sigmoid activation function that will limit the values of the weights between 0 and 1. We will perform upsampling to bring the result to the same dimensions as the input X. This is in order to multiply the layers between them, and thus we will get the original dimensions of X and will not interfere with the flow of the network in which we implement the block.

Attention U-Net: Let us now see how we can combine the U-Net architec-

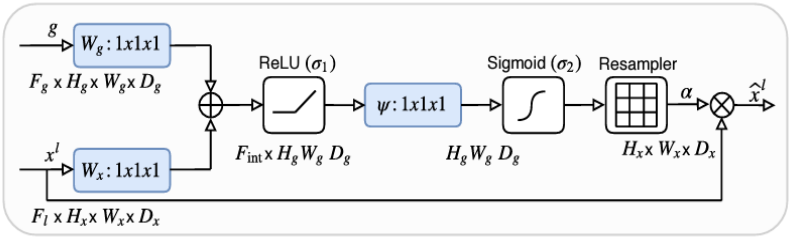


Fig. 11: Attention block [9]

ture and the Attention block. As described above, the skip connections in the original U-Net pass filter results from downsampling to upsampling. Because of the symmetric structure, after every two convolution layers in downsampling, we receive a result that is sent to the parallel upsampling process. Here, we use the skip connection as input X instead of directly concatenating to the parallel block, and input G comes from one layer below in upsampling since we are doing upsampling to it, and we concatenate the result exactly as we did with the skip connection in the original U-Net. In the paper that published the new architecture in 2018, [9] you can see a significant improvement in the performance of the Attention U-Net when compared to the original U-Net.

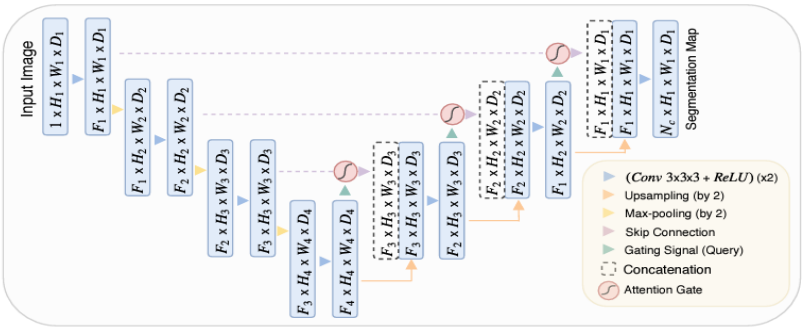


Fig. 12: Attention U-Net architecture[9]

4 Proposed approach

The model we present is designed to detect forest fires in real time. Our research aims for the model's performance to be both fast and accurate in marking fire in video footage efficiently. We will utilize a dataset called FLAME2, containing images and videos captured during a forest fire in Arizona in 2021. Each pair within the dataset consists of an image and video in either RGB or IR format. The dataset is labeled as "Fire/NoFire" to facilitate model evaluation and training. Throughout our research, we will explore various hyperparameters to optimize the model's performance. We will compare training loss versus validation loss, or training accuracy versus validation accuracy over epochs, to ensure accuracy while preventing overfitting. Our model will be constructed by combining two established architectures. Firstly, we will use YOLOv4 to swiftly identify fire locations within images. Subsequently, we will transfer the identified regions, termed as the "head" of YOLOv4, to the Attention U-Net model for segmentation. This approach ensures both speed and accuracy in real-time fire detection.

4.1 Model architecture

The architecture of our proposed model consists of integrating the YOLOv4 framework to identify fire, followed by segmenting the result using the Attention U-Net, as explained earlier in our research. YOLOv4 excels at swiftly and efficiently detecting objects in images, requiring no internal architectural modifications. We will input video frames as received from a drone. The output of YOLOv4 provides a vector containing parameters defining a bounding box around detected objects, along with confidence scores. We will assess the model's detection performance and establish a threshold percentage for detection, beyond which we transfer the identified object to the Attention U-Net model. Each YOLOv4 detection will yield a bounding box, which we'll extract from the original image and pass to the subsequent model. Given potential variations in box sizes, we'll standardize boundaries for input into Attention U-Net. Regarding the Attention U-Net, we'll maintain its structure, featuring three downsampling

blocks, a bottleneck block, and three upsampling blocks. For each upsampling block, we'll incorporate an Attention block, merge its output with the model's, restore it to the original frame, and deliver the segmented image to the user. We anticipate that this model will provide highly accurate results with swift identification and segmentation, rendering it suitable for real-time systems.

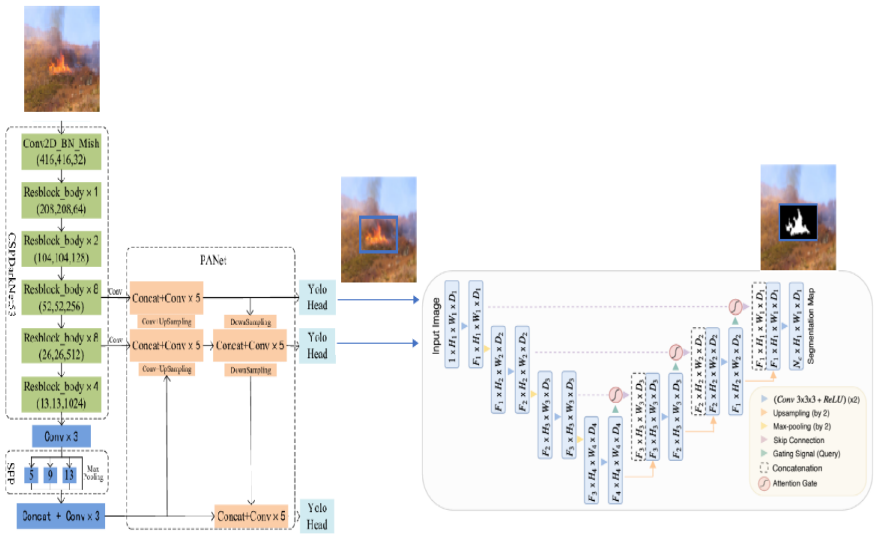


Fig. 13: Proposed Model

4.2 Hyper parameter

To train our model effectively, we need to define and optimize hyperparameters, which significantly influence both the accuracy and speed of the model. These parameters include Learning Rate, Epochs, Batch Size, Loss Function, and Evaluation Metric. Throughout our research, we'll explore and set values for these parameters to align with our desired results and performance goals. Adjusting these parameters will also help prevent overfitting, ensuring the model generalizes well to new fire detection scenarios beyond the training dataset.For each parameter, we'll explore a range of values and select those that result in the best performance according to our evaluation metrics. By systematically adjusting these hyperparameters, we aim to optimize the model's accuracy, speed, and generalization ability for real-time fire detection.

Learning Rate is a crucial hyperparameter that determines the step size taken during the optimization process. A lower learning rate increases the likelihood

of reaching the optimal point but slows down progress, while a higher learning rate speeds up training but risks overshooting the optimum.
Finding the right balance is essential, ensuring efficient progress without sacrificing accuracy. In our case, we'll set the learning rate within the range of $10^{-6} \leq x \leq 10^{-5}$, considering the optimization function ADAM.
This range allows us to explore both slower and faster learning rates, facilitating the discovery of an optimal value. By experimenting within this range, we aim to strike a balance between training speed and accuracy, ultimately guiding the model towards effective real-time fire detection.
Epochs represents a complete pass of the entire training dataset through the model during the training phase. It's crucial to determine the number of epochs for training, balancing between underfitting and overfitting.
An undertrained model may result in inaccurate predictions and suboptimal performance if there are too few epochs. Conversely, an excessively high number of epochs can cause overfitting, where the model memorizes the training data too well and fails to generalize to new data.
In our research, we'll explore the model's behavior across a range of epochs, specifically from 50 to 80. After each set number of epochs, we'll evaluate the model's performance and analyze its learning progress. By monitoring metrics such as training and validation loss, as well as accuracy, we'll assess how well the model is learning and whether additional epochs are beneficial.
This iterative approach allows us to determine the optimal balance between training efficiency and model generalization. Based on the observed results, we'll decide the final number of epochs that best aligns with our goals for accurate and efficient real-time fire detection.
Batch size is a critical parameter that influences how many samples are processed in each iteration during model training. A smaller batch size leads to more frequent weight updates but may introduce instability, while a larger batch size results in slower convergence but greater stability.
In our research, we'll explore batch sizes of 32, 64, and 128. Each option presents trade-offs between training speed and stability. Additionally, we'll leverage cloud technology to harness advanced hardware capabilities, enabling efficient computation without the need for physical infrastructure. This approach ensures we can effectively train our model while optimizing resource utilization and computational efficiency.
Loss functions: Dice Score is a metric that shows us how similar two datasets are. For example, in our project, we want to determine how closely the segmentation provided by our model matches the actual data mask. The Dice Score ranges from 0 (indicating no overlap or proximity) to 1 (representing perfect overlap). We can calculate the Dice Score using the following formula:

$$DS = \frac{2 \times \text{number of common elements}}{\text{number of elements in set A} + \text{number of elements in set B}}$$

For our project, we have chosen the Sparse Categorical Cross-Entropy loss function. This loss function is used when dealing with multi-class classification tasks, where the target variable represents classes as integers. It helps the model by penalizing incorrect predictions across multiple classes. The Sparse Categorical Cross-Entropy function takes two parameters. The first parameter, Y, represents the true class label as an integer, while the second parameter, P, represents the predicted probability distribution over all possible classes. Instead of feeding a one-hot encoded vector, Y is simply an integer indicating the correct class, which reduces memory usage for large datasets. The function computes the negative logarithm of the predicted probability corresponding to the correct class. A low predicted probability results in a higher loss, penalizing the model for its mistake, while a high probability reduces the loss, rewarding the model for accurate predictions. This encourages the model to increase the probability for the correct class over time.

The formula for Sparse Categorical Cross-Entropy is:

$$\mathrm{SCE}(y, p) = -\log(p_y)$$

Where y is the true class label, and p_y is the predicted probability of the correct class y.

5 Dataset

As part of the research process, we searched for two types of datasets. The first is for training the YOLOV4 system. This dataset needs to contain images, some of which have fire or flames, and some without any fire at all, in order to train the model. Each image must have an accompanying text file with the same name, containing the locations of the corners of the bounding box around the object and the class to which the object belongs. We chose to use a dataset called D-fire [13], which contains more than 21,000 images.

Number of images		**Number of bounding boxes**	
Category	**# Images**	**Class**	**# Bounding boxes**
Only fire	1,164	Fire	14,692
Only smoke	5,867	Smoke	11,865
Fire and smoke	4,658		
None	9,838		

Table 1: Number of images and bounding boxes by category

The second type of dataset is for training the Attention U-Net model. This dataset should contain images of forest fires to train the model. We will use masks - a mask is an image that colors each pixel according to the class it belongs to in the original image. For example, if the pixel is part of a fire, we will color it red, while if it's part of a tree, it will be colored green. This way, we can train the model to segment the image and learn the patterns and features we're looking for. For this purpose, we used a dataset called FLAME [12], which contains 2,003 images and 2,003 masks.

Fig. 14: FLAME dataset

6 Research Process

We began the project by conducting an in-depth study of the models we wanted to explore. The first was YOLOV4, which we decided to use for fire detection and bounding box marking. The second model was Attention U-Net, which we would use to segment the objects identified by YOLOV4. After understanding these models, we searched for the most efficient way to implement them and chose to do so in Python using the Google Colab environment. This provides support for the libraries we wanted to work with, streamlining our work process. Another advantage of this working environment is easy access to datasets from Google Cloud (we uploaded both datasets to the cloud), facilitating work from any location and any computer.
Training the model requires very high hardware resources that we don't have on our personal computers. Google Colab gives us the option to use very powerful hardware through cloud computing. We used a Google Colab Pro Plus account, which provided us with computing power of 83.5 GB RAM, 40 GB GPU RAM, and a 235.7 GB disk, giving us the resources to train the model in a reasonable time without exceeding computational memory. After choosing the work environment, we began developing the YOLOV4 model. We researched and found that we needed to use the model developed by AlexeyAB, who developed dark-

net. We had to clone the Git repository, make changes to the Makefile to train the model, and then compile it.
Next, we prepared the dataset for training. We uploaded it to a Google Drive folder and divided it into two folders for training and testing. Each folder contains both image files and the text files we'll use to train the model. We used the glob library to pull all the file addresses. We defined the config file according to the classes we want to identify, in our case, two: smoke and fire. We then started the training process. The model achieved an average Intersection over Union (IOU) score of about 0.74, demonstrating its ability to accurately predict object locations, with losses primarily driven by IOU and class prediction errors.

(a) Prediction using OpenCV library (b) Prediction using Darknet library

Fig. 15: Results of YOLO predictions: (a) using OpenCV library, (b) using Darknet library.

After finishing the training of the YOLOV4 model, we moved on to implementing Attention U-Net. First, we built a simple U-Net model as we had researched, containing four down-sampling blocks and four up-sampling blocks. We trained the model to see if it could learn, and then we moved on to implement the Attention gates that sit between the skip connection and the end of the up-sampling block. We researched the implementation of the gate through papers and incorporated it into the model's architecture.
After building the model, we uploaded the dataset to the cloud. As mentioned, it contains 2003 images and 2003 masks, which need to be split into training and testing sets. For the testing process, we took 20% of the images, with the rest for training. After choosing the split, we prepared the dataset for the model. Each image and corresponding mask went through the following steps: First, we loaded the image from the cloud and encoded it in the appropriate format (JPEG for images, PNG for masks) and selected the number of channels in the image (3 channels for the image and 1 for the mask).
Next, the image and mask underwent resizing because the model accepts a uniform image size. Initially, we chose an image size of 256x256 and performed the resizing using a resize function that employs the NEAREST NEIGHBOR method to reduce the image to the desired size. The next step is augmentation.

Because images can be similar, we want to challenge our model to learn more accurately. So we change the image as follows: with an 80% chance we flip it on the horizontal axis, 80% chance we flip it on the vertical axis, and with a 70% chance we rotate it on its axis. Of course, we do this process for both the mask and the image because the comparison needs to be consistent.
Later, we refined our approach to image resizing. We examined the dataset and saw that most fire objects could fit into a 320x320 bounding box. Our solution to this problem was divided into three cases: In the simple case, if the image is 320x320, we do nothing and move to the next stage. If it's larger on one of the sides by 320 pixels, we maintain the image's proportions and shrink it to fit the dimensions, padding with zeros (black pixels) if necessary, which won't affect the model's learning process. If both sides are smaller than 320, we pad until we get the appropriate size.
The final step is to normalize the image by dividing the pixel values by 255 to bring the pixel values between 0 and 1, which helps improve training efficiency. This is the last stage in preparing the dataset. The next step is choosing the batch size, dividing the images and masks into batches for both training and testing.
After making all the preparations, we could start the training and research. After a few attempts to train the model, the model did not give us the results we were looking for. This caused us to look at our dataset. When we opened the images alongside the masks, we saw that in all the images, most of the image itself is without fire, and the parts containing fire are small relative to the image. This prevents our model from learning effectively because when the model looks for important pixels, it finds few of them, and they become irrelevant for the model's computation.

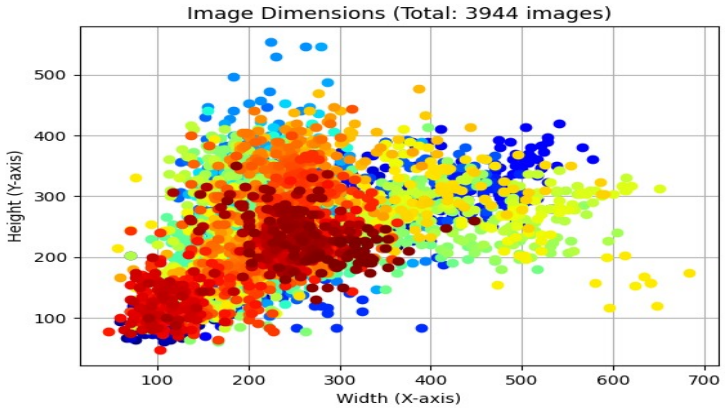


Fig. 16: fire detection bbox on FLAME dataset

Therefore, the idea came up to take the dataset and cut out only the parts where fire appears. This way, the images will have a majority of meaningful pixels, allowing us to train the model. After preparing a new dataset of all the cropped images and masks containing only the parts with fire, we performed a statistical analysis of the new dataset. The reason is that the Attention unet model knows how to accept an input image of a fixed size, and now we have a dataset with different sizes for images and masks.
From here, we needed to analyze what image size we can say most images fit. As can be seen in FIG 24, we can fit most images under a size of 320x320, thus not distorting the image proportions. If one of the sides of the image is smaller than 320, we pad it with black color (pixel value 0), which won't affect our calculation. But then we encountered a problem of what to do if the image size is larger than 320 on one side, and there are quite a few of these. So we decided on a solution of preserving image proportions and resizing proportionally so that the larger side would be 320 and the smaller side would be padded with black.
By doing this, we not only increased the dataset to 3944 images, but we also managed to bring the model to a state where we could train it and get results that are starting to show progress. We researched the model training according to the following hyperparameters: learning curve and number of epochs. During the research process, we discovered that the maximum batch size we could train the model with is 32 because above 32, the GPU memory exceeds and the model training crashes, so we couldn't check other values.
We chose the activation function to be ADAM and the loss function to be sparse categorical crossentropy. The metric we chose to evaluate the model is accuracy.
Now we'll detail the research process. We chose to train the model with the following parameters: The first was 30 epochs with a learning rate of 0.00001. After training the model, we looked at the graphs. The first graph was accuracy versus number of epochs, and the second graph was loss versus epochs. The graphs showed us that the validation wasn't converging well, so we decided to increase the number of epochs to 60 with the same learning rate, and we saw that we could stop around 40 epochs where the validation converges.
The next step was to check if at 40 epochs with different learning rates we could achieve better segmentation results. We chose to start with a learning rate of 0.000001 and achieved relatively good results, while with a learning rate of 0.0001, the results were really not good.
Now that we have two trained models, we started to develop the connection between them. The connection will be made so that the original image is loaded into the YOLO model, it outputs the bounding boxes, we feed those images within the box boundaries into the Attention U-Net model, which will segment the information in the image. We'll stitch the segmented image back onto the original image and return an image where the fire is colored uniformly so we can notice where the fire area is.
The process of building the final model is as follows: First, we load our two models from the weight files we trained. We wrote a main function called detect_and_segment that receives a frame or image and returns the image with

the segmentation. The order of operations is as follows: First, we implemented a function called detect_objects, a function that takes the YOLOV4 model and returns two lists. The first is the bounding boxes it detected in the image, and the second is the confidence percentage of the model that it detected fire. We had to decide from what threshold of confidence percentage we keep the bounding box. After a few attempts, we saw that above 20% confidence, it returns most fire sources in the image.
After receiving the bounding boxes, we take them and cut the coordinates from the original image into a list of images. The next step will be to take each image from the list and segment it, but as in the training process, we'll need to preprocess the image so we can input it into the Attention U-Net model. Image processing will be to convert the image from BGR format to RGB, do the same resizing process as we did in the training process, and we'll need to expand the image dimension so it has the batch space. Then the model will predict the segmentation for the image.
The segmented image will undergo further processing before we return it to the original image because we only want the pixels colored as fire, so we'll look for them in the segmented image and color them accordingly in the original image. The last step will be to receive the list of images after segmentation and paste them in the appropriate locations in the large image. There's also an option to draw the bounding boxes on the original image (optional).
After we finished everything, we ran the results on an image and saw that the results did not provide a satisfactory outcome in terms of segmentation. So we thought about how we could teach the model better. After research, we decided to increase our dataset using augmentation with the mosaic method. This method takes 4 random images from the dataset, flips and rotates them as we detailed, and then creates a new image containing 4 images connected in a 2x2 matrix. This way, the model can learn more. This method gave us a new angle for learning and provided different and better results.

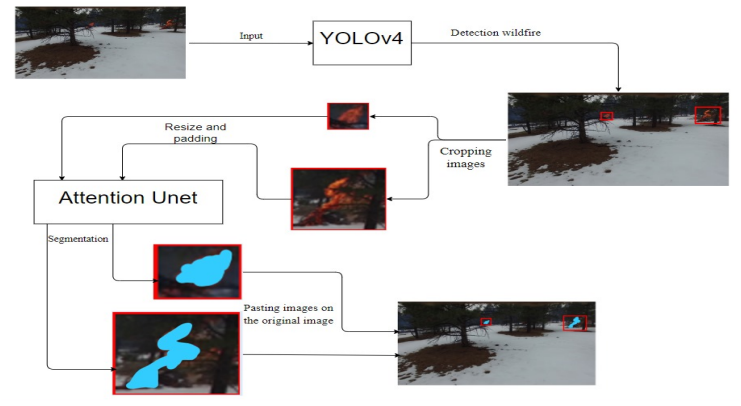


Fig. 17: Flow chart of the system

7 Results

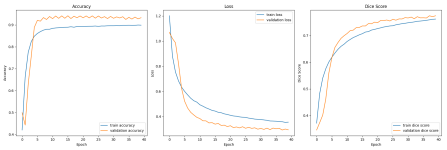
To compare the results we obtained, we divided them into two cases. The first case is a comparison of the graphs we got from training the Attention UNET. The graphs, as we mentioned, show accuracy versus epochs, loss versus epochs, and Dice Score versus epochs. In the second case, we compare the results of the complete model.
A change in the learning curve helps us find the optimal learning point more quickly, covering less ground, or more slowly, covering more ground. We studied several cases of learning curves, both high and low. The results can be seen in the attached table 2, which compares the model's accuracy and Dice Score, along with the hyperparameters we used.
The model was initially trained for 60 epochs with a learning curve of 10^{-5}. We examined the results using the graphs we already mentioned accuracy versus epochs, loss versus epochs, and Dice Score versus epochs. In the accuracy graph of the first training, we saw that it did not change much around 40 epochs, with accuracy around 0.85. Similarly, the third graph of the Dice Score, which is supposed to indicate how well our predictions match the masks, also didn't change much around 40 epochs, staying at around 0.8. So, we decided to conduct two more experiments: the first was to increase the learning rate, and the second was to decrease it and reduce the number of epochs to 40 to avoid overfitting.

Dataset	**Optimizer**	**Learning Rate**	**Epochs**	**Train accuracy**	**Validation accuracy**	**Train dice score**	**Validation dice score**
Mosaic	Adam	10^{-6}	70	84 %	86 %	74 %	79 %
Original	Adam	10^{-6}	40	86 %	92 %	74 %	75 %
Original	Adam	10^{-5}	60	97 %	85 %	91 %	81 %
Original	Adam	10^{-4}	40	89 %	91 %	81 %	82 %

Table 2: Performance of Attention unet with various configurations

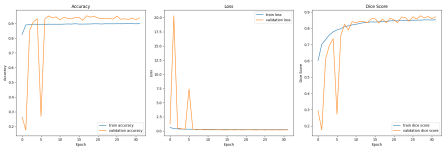
In the first experiment, with a learning rate of 10^{-4}, the segmentation results were not visually good the fire segmentation was too broad and covered a larger area than the actual fire. Additionally, the learning process was unstable. The accuracy stabilized at 0.9, and the Dice Score stabilized at 0.8, which did not reflect the reality and indicated clear overfitting.
In the second experiment, we reduced the learning rate to 10^{-6}. Here, the results were better. The segmentation visually matched the fire more closely and was similar to the mask, and the graphs showed a more stable learning process. The accuracy stabilized at 0.87, and the Dice Score stabilized at 0.72.
After examining the existing models visually in terms of segmentation, we decided to try a different approach with the dataset. We took the existing images and created a mosaic of four images into one, as we detailed earlier. We trained the model with the mosaic dataset using a learning rate of 10^{-6} for 70 epochs.

The learning process was not very stable, but the visual segmentation results were good. The model's accuracy stabilized at 0.84, and its Dice Score at 0.74. Of all the model results, we were most satisfied with the last one using the mosaic method, as it was the only one we did not suspect of overfitting.



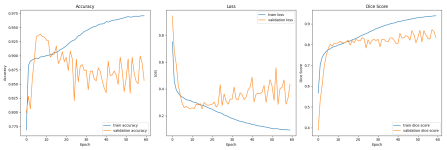
(a) 40 epochs, learning rate = 10^{-6}

(b) 40 epochs, learning rate = 10^{-6} result



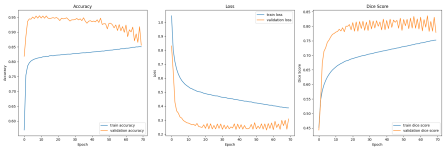
(c) 40 epochs, learning rate = 10^{-4}

(d) 40 epochs, learning rate = 10^{-4} result



(e) 60 epochs, learning rate = 10^{-5}

(f) 60 epochs, learning rate = 10^{-5} result



(g) 70 epochs, learning rate = 10^{-6}

(h) 70 epochs, learning rate = 10^{-6} result

Fig. 18: Comparison of training results and segmentation performance with different learning rates and epochs

(a) Original Image

(b) mosaic ,learning rate = 10^{-6}

(c) learning rate = 10^{-6}

(d) learning rate = 10^{-5}

(e) learning rate = 10^{-4}

Fig. 19: Comparison of original image and U-Net model segmentation results with varying learning rates and epochs

(a) Original Image

(b) mosaic ,learning rate = 10^{-6}

(c) learning rate = 10^{-6}

(d) learning rate = 10^{-5}

(e) learning rate = 10^{-4}

Fig. 20: Comparison of original image and U-Net model segmentation results with varying learning rates epochs and Dice Score

In our comparative analysis of fire detection models, we examined the original image alongside outputs from various models with different hyperparameters (as shown in Figures 19 and 20). The YOLOV4 model, represented by red bounding boxes, demonstrated consistent performance across images, successfully detecting all fires in Figure 19. However, it struggled with partially obscured and distant, smaller fires in Figure 20, revealing potential areas for improvement. The segmentation results, visualized in blue, showed that the model trained for 70 epochs using mosaic data augmentation significantly outperformed other models, accurately isolating fire regions. Other models struggled with precise segmentation, often producing less accurate or overextended boundaries. In con-

clusion, while the YOLOV4 model exhibited robust fire detection capabilities, particularly in straightforward scenarios, the segmentation model with mosaic data augmentation provided superior accuracy in delineating fire regions. This underscores the importance of optimizing both model parameters and data augmentation strategies to enhance detection performance and efficiency.

8 Conclusions

Our research focused on finding a real-time segmentation solution by combining two models for fast segmentation and detection. We studied the YOLOV4 and Attention UNet models, exploring their architectures and how they work for detection and segmentation tasks. We also examined how adjusting their hyperparameters affects the training process. We discovered that combining these two models results in strong detection and segmentation performance. However, we noticed an issue with our dataset: the ground truth masks didn't accurately outline the fire, but instead highlighted broader areas, which confused the model and made segmentation harder since it wasn't just focusing on the fire patterns but also on surrounding objects. Additionally, we found that the model's learning improved when we focused the training on relevant parts of the images, meaning we cropped the important sections and trained on those, which significantly boosted the model's performance. Throughout our research, we monitored the learning curve, adjusted the number of epochs, and selected the 'sparse categorical cross-Entropy' loss function to optimize the learning process.

References

1. The best forestry drones for forest management (2021), https://www.irisonboard.com/best-drones-for-forest-management/
2. , M., Ji, K., Xiong, B., Zhang, L., Sijia, F., Kuang, G.: Light-yolov4: An edge-device oriented target detection method for remote sensing images. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **PP**, 1–1 (10 2021). https://doi.org/10.1109/JSTARS.2021.3120009
3. Ashkenazi, S.: Not only the corona virus: ten climatic disasters in 2020 that will be hard to forget (2021), https://www.globes.co.il/news/article.aspx?did=1001355530
4. Chen, X., Hopkins, B., Wang, H., O'Neill, L., Afghah, F., Razi, A., Fulé, P., Coen, J., Rowell, E., Watts, A.: Wildland fire detection and monitoring using a drone-collected rgb/ir image dataset. IEEE Access **10**, 121301–121317 (2022)
5. Ghali, R., Akhloufi, M.A.: Deep learning approaches for wildland fires remote sensing: Classification, detection, and segmentation. Remote Sensing **15**(7), 1821 (2023)
6. Hopkins, B., O'Neill, L., Afghah, F., Razi, A., Rowell, E., Watts, A., Fule, P., Coen, J.: Flame 2: Fire detection and modeling: Aerial multi-spectral image dataset (2022). https://doi.org/10.21227/swyw-6j78, https://dx.doi.org/10.21227/swyw-6j78
7. Klein, A.: The 10 biggest natural disasters in the last hundred (and a little) years (2022), https://www.globes.co.il/news/sparticle.aspx?did=1001416041

8. Muksimova, S., Mardieva, S., Cho, Y.I.: Deep encoder–decoder network-based wildfire segmentation using drone images in real-time. Remote Sensing **14**(24), 6302 (2022)
9. Oktay, O., Schlemper, J., Folgoc, L.L., Lee, M., Heinrich, M., Misawa, K., Mori, K., McDonagh, S., Hammerla, N.Y., Kainz, B., et al.: Attention u-net: Learning where to look for the pancreas. arXiv preprint arXiv:1804.03999 (2018)
10. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
11. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18. pp. 234–241. Springer (2015)
12. Shamsoshoara, A., Afghah, F., Razi, A., Zheng, L., Fulé, P., Blasch, E.: The flame dataset: Aerial imagery pile burn detection using drones (uavs) (2020). `https://doi.org/10.21227/qad6-r683`, `https://dx.doi.org/10.21227/qad6-r683`
13. de Venâncio, P.V.A., Lisboa, A.C., Barbosa, A.V.: An automatic fire detection system based on deep convolutional neural networks for low-power, resource-constrained devices. Neural Computing and Applications **34**(18), 15349–15368 (2022)
14. Wang, Z., Peng, T., Lu, Z.: Comparative research on forest fire image segmentation algorithms based on fully convolutional neural networks. Forests **13**(7), 1133 (2022)
15. Zvornicanin, E.: What is yolo algorithm? (2023), `https://www.baeldung.com/cs/yolo-algorithm`