UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Symbolic and Evolutionary Artificial Intelligence

Real-time fire and smoke detection with Yolov5

Project Documentation

TEAM MEMBERS:
Michelangelo Martorana
Mirco Ramo

Academic Year: 2021/2022

Contents

1 — Introduction

Fire Hazard is a serious threat for wild forests and national parks: uncontrolled wildfires can be deadly for native animals and human beings, they can destroy millions of hectares of woods and they also release a huge amount of CO_2 and other toxic gasses while destroying precious sources of oxygen. Firefighters are often notified too late about accidents, when they are not able anymore to avoid the disaster, and they are usually required to put their life in danger.

With the recent advance of aerial vehicles, including self-driving UAVs, together with the stunning evolution of computer vision tasks, it is now possible to predict and early detect wild fires, so that to constantly keep under control forests and woods, and notifying on time the rescuers. A key element to do it is smoke: quickly detecting smoke sources could help recognizing new born fires even in the absence of flames. Moreover, when possible, thermal cameras can be exploited to "see beyond" trees and bushes, allowing the system to detect also hidden fires in denser forests. Accuracy and recall are not the only important metrics of this scenario: final system must be fast and lightweight, in order to guarantee **real-time** detection, even for **high resolutions** and **high FPS rates**, in **small and embedded** hardware, like NanoGPUs on UAVs.

In this work, we discuss about two methods for fire detection: on one hand we will present fire vs non-fire classification, highlighting potentialities and limits; on the other hand, we will test an object-detection approach to jointly individuate in a real-time fashion fire and smoke, both from camera pictures and from thermal heatmaps. Eventually, we provide a short recap of the performance achieved by the system when tested on a NVIDIA NanoGPU.

1.1 Related work

Wildfire detection is an active and trendy topic for worldwide research efforts. However, as far as we know, this is the first attempt to train a real-time detector working either on normal or thermal images, or on jointly both. [1] and [2] were some early methods respectively for fire detection and fire segmentation through Computer Vision; they introduced a feature-based pipeline to detect fires starting from flicker detection and pixel colors, and then analyzing hand-crafted regions and color variations. Through their work, they proved the promisingness of Computer Vision in this field, but hand-crafted feature approaches are nowadays obsolete and suffer from false positives. More modern works like [3] showed the potentiality of Deep Learning on fire detection tasks. They achieved stunning results both on accuracy and on inference time, but they used a much older version of YOLO, firstly introduced by [4]. Moreover, they speed up computation removing "static" pixels from successive video frames: this approach could be dangerous, because in some situations, especially on long-distance images, it could be not so easy to detect fire movement. In addition, they do not use thermal cameras, nor they designed the system to work on wild forests. Finally, [5] introduced a fire detection model jointly fed with thermal camera videos and smartphone's camera. It showed to behave perfectly when very close to fire source, but it cannot go beyond 0% accuracy when further than 2 meters, which is of course unsuitable for aerial images.

2 — Dataset

This section details the sources used for the creation of the models with the aim of fire and smoke detection. The metadata, images and videos provided from the paper [6] are related to prescribed fires in a winter forest. The recreated situation provides an excellent opportunity for researchers to collect and update imagery data. The test was conducted in Arizona and managed by the local Fire Department, who burned a pile of slash in partly cloudy conditions with no wind. Furthermore, the hardware utilized to collect data are commercial drones equipped with a normal or a thermal camera. Below will be shown some examples of captured images.

Figure 1: Images taken from drones with a RGB camera

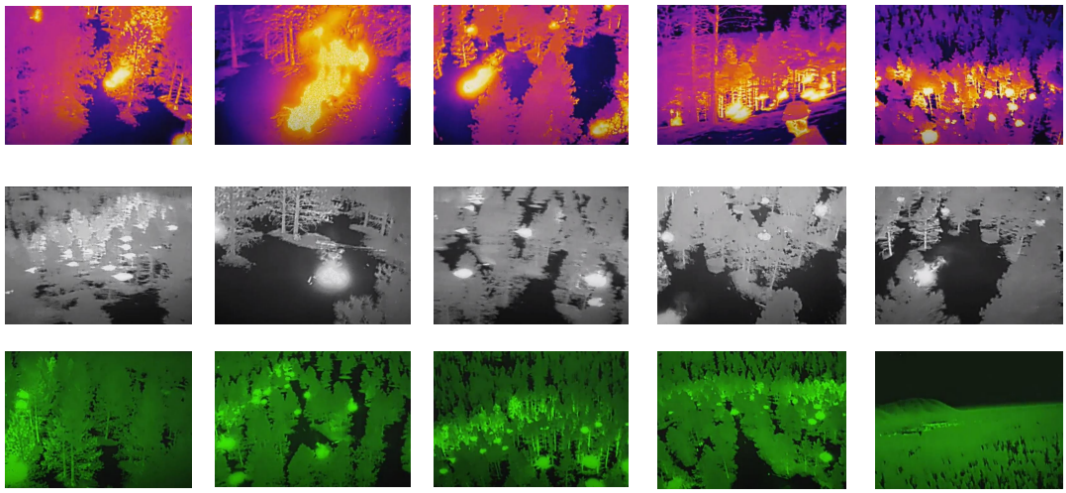

Figure 2: Images taken from drones with thermal camera with different filters

All the videos available for our work were captured with a frame rate of about 30 FPS. For this reason, a function to extract frames from a video, given a desired frame rate, was included in the video manager notebook. Before extracting the frame rate, it is necessary to convert the video in .mp4 format to utilize MoviePy library. The data lake provided by IEEE is detailed in Table 1

As training set for the model working on raw images, we utilized the set 7 provided from the paper in union with other two data sets taken from Kaggle, with the same goal of Fire vs Non-Fire classification. They are:

Num	Type	Camera	Size	Class. usage	Detection Usage
1	Video	Zenmuse	966s	-	-
2	Video	Zenmuse	399s	-	RGB Visual test
3	Video	WhiteHot FLIR	89s	-	Thermal Train/-Val/Test
4	Video	GreenHot FLIR	305s	-	Thermal Train/-Val/Test
5	Video	Fusion FLIR	1500s	-	Thermal Train/-Val/Test/Visual test
6	Video	Phantom	1020s	-	RGB Train/-Val/Test
7	Frames(from 1)	Zenmuse	39375	Train/Val	RGB Train/-Val/Test
8	Frames(from 6)	Phantom	8617	Test	-

Table 1: Available data set information and utilization.

- FIRE Dataset, that are outdoor-fire images and non-fire images conceived for computer vision tasks [7]
- Fire Detection Dataset, another labelled data set composed of google images [8]

These data sets are used to enlarge the training/validation/test sets for the task of fire detection. Regarding the model trained with thermal images, the sets are generated from the three video provided by the paper [6], namely 3,4 and 5. In these videos we exploited the same function cited previously to extract 1 FPS in order to have:

- 152 images for 3-WhiteHot
- 312 images for 4-GreenHot
- 1185 images for 5-ThermalFusion

The WhiteHot and GreenHot videos are utilized entirely for the task of frame extraction. Instead, the 5-ThermalFusion video with the duration of 24 minutes is divided in [0:16,21:24] for the task of frame extraction and [16:21] left for the final Demo.

In particular, following the frame extraction, we sub-sampled the data set in order to prevent the variance of images to be too low, otherwise supernumerary images would be useless for training tasks.

3 — Fire vs Non-fire Classification

In this section, we describe the first application we built for the fire detection task: it consist of Deep Neural Networks trained to classify input images as containing fire ("Fire" class) or not showing it ("Non-fire" class). In the past, researchers tried to use RGB methods to detect smoke or fire [9, 10]. However those methods are definitely not free of errors, since they could confuse fire with the sun or smoke with clouds. On the other hand, a DNN deepens and processes image features at a higher level of abstraction, distinguishing much better fire and smoke, even when they are partially occluded.

3.1 Data set preparation

From the previously discussed data set, we considered a subset of the "Fire vs Non-fire" labelled images mixed with Kaggle ones, and we used them to feed our neural networks. The original data set already provided the train-test splitting of the images, so we decided to shuffle and randomly divide the training set directory, using then 80% of it as actual training samples, and the remaining 20% as validation images. The resulting learning set sizes are:

- 2375 images for training
- 593 images for validation
- 8617 images for testing

This splitting is unusually unbalanced: typically, 60 to 80% of the available images are used for training, while here the majority belongs to the test set. This choice is justified by three reasons:

1. As already explained in 2, the original training set was composed by more than 30k labelled frames extracted from a video, but since they were very similar one another, we decided to subsample them to **speed-up training** and to **increase independence** between different objects. On the contrary, we kept all the test images to have a more accurate final evaluation

2. In presence of i.i.d data, it could be beneficial to take some test images and move them in the training set. This could be very dangerous in our case: also the provided test set is composed by contiguous frames extracted from a recording, so **test samples are highly correlated** one-another; sampling some of those images to add them to the training set would highly increase correlation between the two sets, leading to extraordinary high but **meaningless** accuracy values.

3. In order to have a fair comparison between our results and the ones achieved in the original paper [11], we should test model performance on the same test set, and considering it as unknown till the evaluation phase.

On the following subsection, we briefly introduce the DNN models we used, showing the achieved classification accuracy.

3.2 Inceptionv3

The first model we used to tackle the classification task is Inception-v3 [12]. This architecture, introduced by Szegedy et al. at Google, is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset, and it is the result of a coherent aggregation of multiple ideas developed by researchers over the years.

The model itself is made up of symmetric and asymmetric building blocks, which are connected "in parallel": the idea is not to fix the architecture and the hyperparameters apriori, but to let the model exploit the layer that "help" the more at each stage.

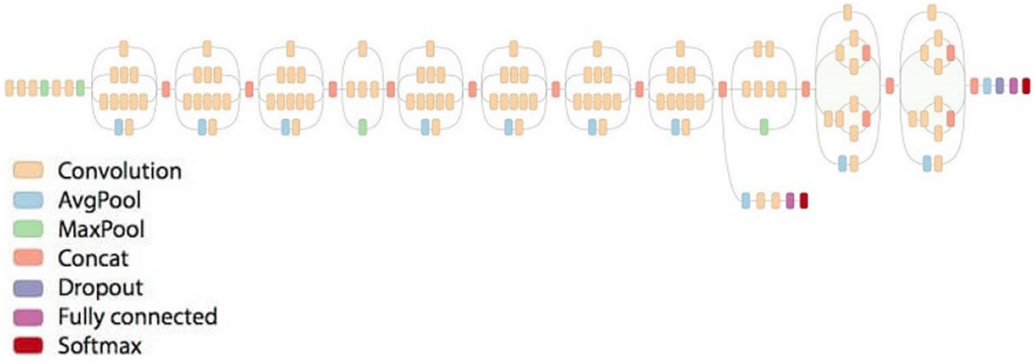

Figure 3: Inception v3 First Proposal

Figure 3 shows a simplified representation of Inception-v3 architecture. On the first configuration, we froze the backbone inception model, appending on top of it a GlobalAveragePooling layer, followed by a 2 fully connected layers regulated by a fixed dropout rate. The training phase ran for 20 epochs, with fixed learning rate of 1e-3 applied to Adam optimizer, and batch size of 64.

After that, we unfroze some final layer and we fine-tuned the resulting network to adapt it to this specific task. Training phase lasted 10 epochs, using the same hyperparameters.

Model	**Train(%)**	**Val(%)**	**Test(%)**
Inception+FCs	87	**82**	61
Fine-tuned Inception	**96**	76	**63**

Table 2: Training, validation and test accuracy of Inception-v3.

Table 2 summarizes accuracy values achieved with Inception-v3 on both the configurations. Note that the high correlation between training and validation data, together with the moderated incorrelation between training and test data, leads to very different values in accuracy, especially between validation and test results. This phenomenon was evident also in the original paper [11], and it will characterize all the following configurations we are going to try. All in all, Inception-v3 performance is not very good, and even if this network can evaluate 8617 images in just 39 seconds (leading to a throughput of 220 FPS in a real-time application, that is remarkable), accuracy test values are definitely poor.

3.3 EfficientNetB4

Because of the limited prediction power of Inception-v3 on our task, together with the need of a faster network, led us to the choice of EfficientNetB4. EfficientNet was originally introduced by Tan, Mingxing and Le, Quoc in [13], and it quickly reached the state-of-the-art in the ImageNet classification task, without needing any additional training data, and requiring many parameters less. Still today, the most performing networks, like Meta Pseudo-labels, are based on EfficientNet. [14] Figure 4 shows how EfficientNet reached state-of-the-art accuracy using much less parameters, when originally introduced.

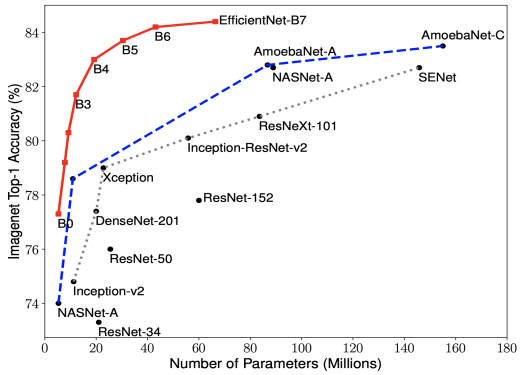

Figure 4: EfficientNet performance when originally introduced

The core idea of EfficientNet is to cleverly scale a basic architecture, both on depth, width and resolution, according to the trade-off between accuracy and numbers of parameters we are looking for. More formally, defined as d model's depth, w model's width and r the resolution, instead of applying a trial-and-error approach to optimize those hyper-parameters, a *scaling compound factor* ϕ is defined, thus setting:

- $d = \alpha^\phi$
- $w = \beta^\phi$
- $r = \gamma^\phi$
- $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$
- $\alpha, \beta, \gamma \geq 1$

Base architecture instead is shown on Figure 5

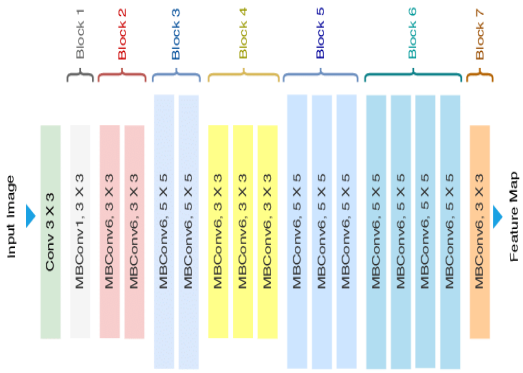

Figure 5: EfficientNet base architecture

Considering again Figure 4, we decided to adopt the B4 version of EfficientNet, since it could be the best trade-off between accuracy, which is a key requirement for critical tasks like fire detection, and numbers of parameters, which impacts on training and inference time. For our tasks, we trained the following configurations:

1. Backbone EfficientNetB4 (frozen) + MLP, made of 2 FC layers respectively of 2048 and 1024 neurons and regulated by dropout.
2. Backbone EfficientNetB4 (frozen) + smaller MLP, made of 2 FC layers respectively of 512 and 256 neurons and regulated by dropout.
3. Fine tuning of the previous model unfreezing EfficientNet's block 7.
4. Configuration 2, but introducing class weights. Loss is weighted by the frequency of the class: in this way we prevent the model to always output the majority class, and they are computed as $\frac{n}{c*n_c}$where n is the number of training samples, c is the number of classes and n_c is the number of sampling belonging to the considered class.

Model	**Train(%)**	**Val(%)**	**Test(%)**
EfficientNetB4 + MLP	**99.2**	97.0	61.4
EfficientNetB4 + smaller MLP	98.0	**98.0**	62.3
Fine-tuned EfficientNetB4	98.7	97.3	62.6
FT EfficientNetB4 with class weights	99.1	97.1	**64.2**

Table 3: Training, validation and test accuracy of EfficientNetB4.

Table 3 summarizes accuracy values achieved with EfficientNetB4. Once again, we notice the sharp difference between training/validation and test accuracy values. The introduction of EfficientNet was of no help, performance are still poor and far from the ones reached by [11] using Xception, which actually are not so good as well. Throughput is 82 FPS considering only inference time, that is acceptable for our application.

Generally speaking, the imbalance, low variability, and special purpose of this data set seem to limit DNNs' ability to generalize and to actually distinguish fire: adding some external images helps a bit, but accuracy levels are far from being acceptable.

For this reason, we decided to change the approach, and to develop a real-time object detector, trained on a self-labelled data set, to specifically localize fire sources and smoke columns, instead of signaling only fire presence without specifying the location and/or the entity. On the next sections, we will describe the efforts to implement such a network.

4 — Fire and smoke detection using Yolov5

In this section we deepen the choices we took to implement a real-time fire and smoke detector, from the data set preparation to the final test and result, detailing pros and limits of this approach.

As backbone detector, we selected Yolov5: Yolo (You Only Look Once) is a state-of-the-art object detection network trained on COCO data set [15]. Yolo was originally introduced by introduced by Redmon et al in [4]as an open-source project, but he decided to drop it after Yolov3, worried about the social impact his creation could have. In 2020, Bochkovskiy resumed the project releasing Yolov4 [16], and in 2021 Ultralytics claimed the release of Yolov5. The community criticised a lot this statement, since neither any paper about Yolov5 has been released, nor it exists a fair and deep comparison between it and previous models. However, due to the ability of the network to guarantee quite good accuracy, together with a very high throughput, we consider it as absolutely suitable for our purposes. Figure 6 better shows Yolov5 performance on Coco data set, compared to the state-of-the-art, double-stage neural network EfficientDet by Google [17].

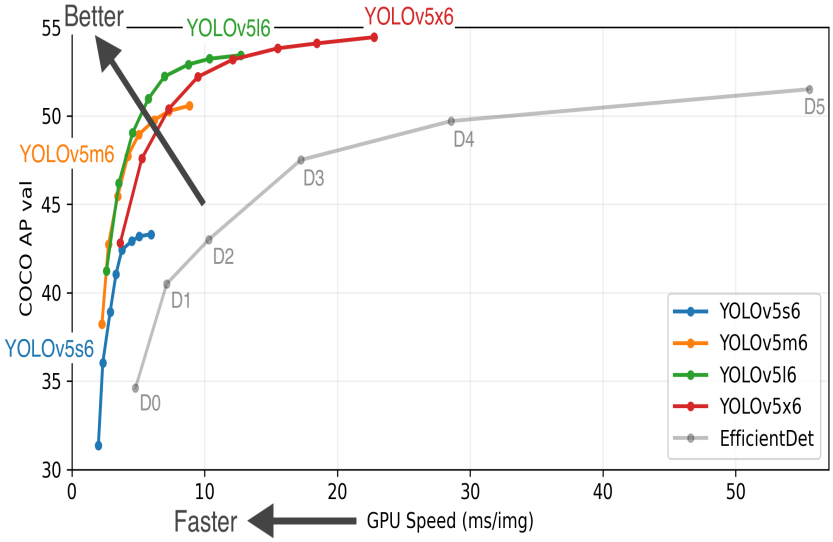

Figure 6: Yolov5 performance on Coco

Yolo proved to be much faster and much more indicated to real-time detection tasks with respect to traditional detection networks, and it that's mainly because it is a **single stage** network: frame object detection is tackled as a regression problem, allowing the network to jointly predict class labels and bounding boxes with a single pass. Yolo divides input image in a SxS grid, which is responsible to predict at most B bounding boxes, whose confidence is computed as the predicted IoU with the ground truth bounding box. Moreover, for each cell, the network outputs C conditional class probabilities, and eventually computes final mixed predictions. Figure 7 represents the backbone behind Yolo.

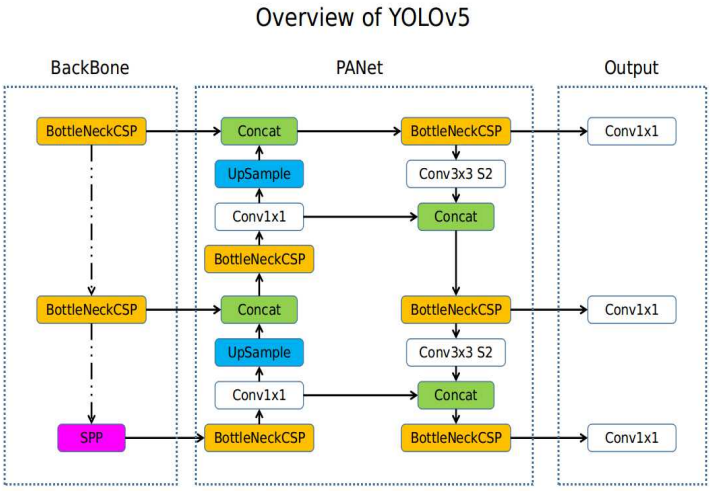

Figure 7: Yolov5 architecture

4.1 Data set preparation

In this part of the project we decided to introduce thermal images provided from the different cameras utilized during data collection phase. Tanks to the double nature of input images, we can now generate two models for the task of fire detection., which will be trained on two different data set.

In particular, one training set is composed by frames extracted from the thermal infrared videos available in [11], following the splitting explained in the section 2, while the other one is composed by the same raw images utilized also in the classification chapter, plus some manually extracted frames from the raw video taken with the Phantom camera, reaching this way 3960 .jpg files.

The advantages to utilize these two model is to combine the great ability of thermal chambers to find smoke and fire from thermal camera with the utilization of RGB images. The optimal situation would be to have 2 cameras that provide data in real time to an Ensemble Model. The images utilized during training are labelled with [18] that is an application specifically conceived to create bounding boxes over the objects on which you want to perform the detection, outputting then a .txt file that contains the class of the object and the coordinates of the bounding boxes in terms of percentages distances from borders, expressed in Yolo Format. The two data collections labelled have been saved as the "normal images" data set and "thermal images" data set.

4.2 Yolov5 on raw images

Once the learning sets have been generated, we split them into training, validation and test subsets. Due to the reduced number of available labelled images, we decided to allocate a large portion of them to the training set, specifically 80% of the totality, splitting then the left portion into 2 sets of 75% of the cardinality for validation and only 25% for test, which is translated in absolute numbers as:

- 3168 images for training
- 594 for validation
- 198 for test

Test images are very few, but this is not a concern since the true testing phase will be conducted at the end on the stream video. We tested 2 different sizes of Yolov5, namely Yolov5m and Yolov5l; those are the models that, looking at Figure 6, we believed to have the best trade-off between accuracy and throughput. We configured those models to work on our data set, and we loaded them with the weights achieved after the pretraining on Microsoft Coco[15].

4.2.1 Yolov5l on raw images

Yolov5l (l stands for "large") is the second biggest version of Yolov5, it counts about 46.5 millions of parameters, and Ultralytics claims 67.3 of mean Average Precision on Coco Dataset, with 10.1 ms of inference time on GPU. We trained this model for 50 epochs, using Adam optimizer[19] with learning rate=1e-3, and we retained the best model, i.e., the one that reached the best validation mAP. To speed-up the training phase, we enabled the NVIDIA Apex extension for PyTorch, which consists on a set of modular and independent utilities to streamline **mixed precision** computation, along with **distributed training** over multiple GPU cards, when possible.

4.2.2 Yolov5m on raw images

Yolov5m (m stands for "medium") is the third biggest version of Yolov5, it counts about 21.2 millions of parameters, and Ultralytics claims 64.1 of mean Average Precision on Coco Dataset, with 18.2 ms of inference time on GPU. We trained this model with the same hyper-parameters of the previous one.

Data set	Model	Class	Precision	Recall	mAP@0.5
Validation	Yolov5l	Fire	**0.656**	0.627	**0.629**
		Smoke	0.614	**0.496**	0.510
		All	**0.635**	0.562	**0.569**
	Yolov5m	Fire	0.651	**0.645**	0.624
		Smoke	**0.616**	0.469	**0.512**
		All	0.633	**0.577**	0.568
Test	Yolov5l	Fire	**0.689**	0.663	**0.688**
		Smoke	**0.679**	**0.530**	**0.569**
		All	**0.684**	**0.596**	**0.628**
	Yolov5m	Fire	0.656	**0.665**	0.649
		Smoke	0.672	0.516	0.535
		All	0.664	0.591	0.592

Table 4: Yolov5 performance comparison on raw images

Table 4 shows the outcome of the training of both Yolov5l and Yolov5m, comparing values on Precision, Recall and Mean Average Precision at 0.5 of IoU. Yolov5l seems to be slightly better, but not so much: validation measures are almost the same, while it seems to outperform Yolov5m on the test set, despite the very narrow difference. On the other hand, Yolov5m is much faster: it took **3.6ms** of inference time per image, against the 5.5ms of Yolov5l, and for this reason we decided to adopt Yolov5m as final model for our fire and smoke detector on raw images, trading off some accuracy for a high improvement in speed.

4.3 Yolov5 on thermal camera images

The images that compose the thermal images data set, as cited in the 2 Data set chapter, are the images extracted from WhiteHot, GreenHot and ThermalFusion videos. As in the raw images the learning set are divided in training, validation and test with the same ratio for a total of:

- 1275 images for training
- 239 for validation
- 80 for test

As models was chose Yolo5l and was configured to work with our data set. The model is loaded with the weights achieved after the pretraining on Coco.

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95: 100
all	239	399	0.826	0.813	0.849	0.506
Fire	239	330	0.891	0.891	0.926	0.598
Smoke	239	69	0.76	0.734	0.772	0.414

Table 5: Yolov5l performance on thermal test set.

4.4 Real-time simulation

A key requirement of the project is the ability of the system to detect smoke and fire in real time. To satisfy this, we installed a local deployment of CUDA, that is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU). The GPU enabled is a GeForce GTX 1660 Super with 6 GiB of dedicated RAM.

The CUDA toolkit has been installed in Conda environment.

To achieve the real time detection task, it was also necessary to modify the run() function of detect.py. To enable live detection is required to change a show_result flag that print the results, in terms of object detection, frame by frame.

4.5 Ensemble outputs

In this section we discuss some theoretical principles that could be exploited in order to ensemble the predictions of the 2 trained models, to combine the ability of the thermal camera to detect also small and hidden fire principles, together with the higher precision on smoke detection achieved by RGB images.

Figure 8: Yolov5 predictions on similar frames

In Figure 8 we showed the output of 2 similar frames, taken from the 2 kinds of cameras. Having no corresponding video images from the data set, we seek for the most similar examples we have. In order to exploit both real-time videos, we combine the predictions of both models, according to a self-designed policy that we present in the current subsection. Given output bounding boxes of two temporally overlapping frames, we compute 3 squares corresponding to the 2 single image bounding boxes, adding them their intersection if not empty. Then, the confidence of each of those squares is recomputed starting from original confidence values, weighted by hand-defined values that we asses to be the most suitable ones for our use case. In particular, due to the different abilities of the models to detect different fire and smoke features, focusing on different aspects of the inputs, we decided to implement 2 different policies for fire and smoke detection.

Figure 9: Example of overlapping bounding boxes relative to fire predictions

Figure 9 shows an example of the policy implementation relative to fire detection. The dark red square resembles a possible prediction of the model 4.2, while the pink square is related to the 4.3 one.

Defining $conf_R$ as the confidence value on the bounding box outputted by the RGB model, and $conf_T$ the confidence value by the Thermal One, the final confidence of each bounding box is given by $conf = 0.1 * conf_R + 0.9 * conf_T$. Referring to Figure 9 as example gratia, assuming 0.55 of confidence value for the dark red bounding box, and 0.93 for the pink one (as it were for the example Figure 8), the confidence of the single squares will be:

- $conf(DarkRed(RGB\_only)) = 0.1 * 0.55 + 0.9 * 0 = 0.055$
- $conf(Pink(Thermal\_only)) = 0.1 * 0 + 0.9 * 0.93 = 0.837$
- $conf(Grey(intersection)) = 0.1 * 0.55 + 0.9 * 0.93 = 0.892$

Figure 10 instead deals with the policy relative to smoke hazard. The dark blue square resembles a possible prediction of the model 4.2, while the turquoise square is related to the 4.3 one.

Figure 10: Example of overlapping bounding boxes relative to smoke predictions

This time, to take into consideration the ability of the raw image camera to bound more precisely the smoke, the final confidence of each bounding box is given by $conf = 0.7 * conf_R + 0.3 * conf_T$. Referring to Figure 10 and assuming the same values of the previous example, the confidence of the single squares will be:

- $conf(DarkBlue(RGB\_only)) = 0.7 * 0.55 + 0.3 * 0 = 0.385$
- $conf(Turquoise(Thermal\_only)) = 0.7 * 0 + 0.3 * 0.93 = 0.279$
- $conf(Grey(intersection)) = 0.7 * 0.55 + 0.3 * 0.93 = 0.664$

Of course, not every bounding box is returned, but each of them will be filtered by the confidence value passed to the detect function. For instance, assuming a Threshold value of $thresh\_conf = 0.3$, only the Pink, the Dark Blue, the Turquoise and the 2 Grey squares will be drawn.

Unfortunately, the lack of overlapping test frames prevented us from coding and validating an implementation of the actual ensemble of the predictions: this task, together with a finer-grained estimation of the weights network, is scheduled for future work, when we will possess such data.

5 — Visual results

In this section we deepen the ability of the model under analysis to detect smoke and fire on the wild, presenting some visual results to explain the points of strength and weakness of the network itself.

Given the performance and capability analysis conducted on the previous 2 sections, we decided to focus our attention only on the most promising approach, that is the fire and smoke detection using Yolov5.

In 4.4 we discussed how we equipped our Yolo model with a new function that allows it to work as real-time detector: independently from the input nature, that can be a series of images as like a set of camera frames, a formatted video or a web stream (like a YouTube video), outputting a series of labelled images.

As explanatory example, Figure 11 shows a screenshot taken while the real-time detect command was running: as we can see, the model analyzes and predicts bounding boxes on each single frame of a video: the prompt prints the execution state of the running detection, while an external window shows the last labelled frame.

Figure 11: Screenshot of a real-time detection instance

Once done it, we tested it on unlabelled data, to have a visual feedback of the recall of the model in detecting fire and smoke on previously unseen video streams, and, in the meanwhile, to avoid false alarms, thus evaluating the precision of it. As we explained in the 2 section, and more in detail in the 1, we allocated the videos with ID 2 and 5[16:21] as visual demo inputs, respectively for the RGB and for the thermal camera: in the following paragraphs we report some interesting results of the developed models on those test frames, proving model strength but also analyzing the error trend.

5.1 Visual results on RGB images

In the following, we provide some example image of the visual results we achieved applying the RGB Yolov5m model upon the test video 2.

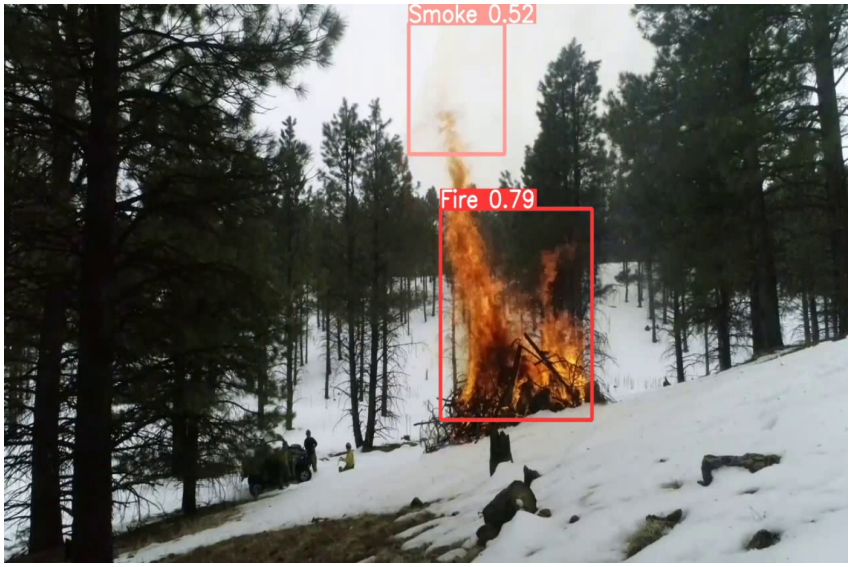

Figure 12: Screenshot of the detection on a random frame

Figure 12 represents the detection applied to a random frame of the video. As we can see, the model behaves very well, detecting with good confidence the fire column, along with the related produced smoke, and does not produce any false positives.

Figure 13: Two examples of good detection by the RGB model

Figure 13 shows the potentiality of the model. On the sub-figure on the left, we see an example of the high precision and recall it can achieve, even in conditions of invisible fire: the two smoke columns are perfectly detected and bounded, near the truth source. On the sub-figure on the right instead, we notice how much it is capable of detecting even occluded fire and smoke, allowing it to be suitable also for denser forests.

On one hand the presented examples prove the recall level of the model, on the other hand Figure 14 illustrates the precision of it: when the drone frames a cloud bank: no alarm is sent out, since the network correctly distinguished humidity from smoke.

Figure 14: No smoke alarm is sent out when the camera frames clouds

On the contrary. the model seems to struggles whenever the camera records images containing smoke which is in an unusual position. In the Figure 15 an example is reported: the network cannot detect the smoke that is present in the left hand side of the image, on the south-west with respect to the fire. On the contrary, it tends to bound a region where smoke is usually present, i.e. above the fire, even though it is not actually there this time. Testing it on similar images, we can state that it is not a systematic error, nor the wrong bounding boxes have high confidence, but it represent a point of weakness of the trained detector.

Figure 15: An example of failed smoke detection

5.2 Visual results on Thermal images

In this subsection we complement the presentation of model's performance, showing the visual results we achieved feeding the model with thermal images, both taken from the test set and from the training one: our aim is to show the outcome of the predictions in all the thermal filters we had, however as explained in the section 2, we are forced to use the totality of WhiteHot and GreenHot frames as training samples, thus in order to show the results on those filters,

we are obliged to run detection task on that set. We are aware that this is not a truly fair prove, and that the results are going to appear obviously better than test ones, but as we are going to show, given the stunning performance of this model on Thermal Fusion test set, we are very confident that it would behave equally great if executed on wild WhiteHot and GreenHot images.

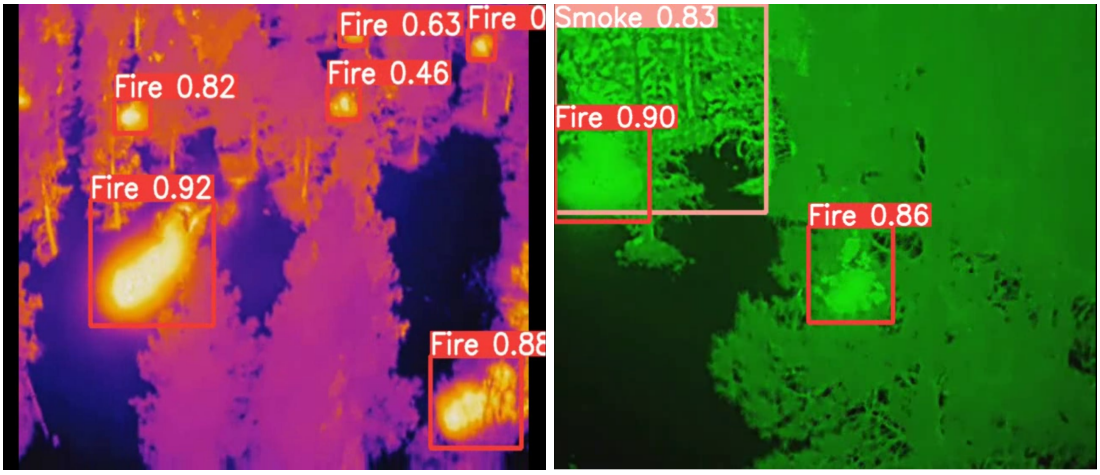

Figure 16: Screenshot of the detection on random frames

Figure 16 represents the result of the detection of smoke and fire, executed on a random sample of the Thermal Fusion test portion of the video. As we can see, also this model behaves very well, catching with high confidence every fire and smoke source.

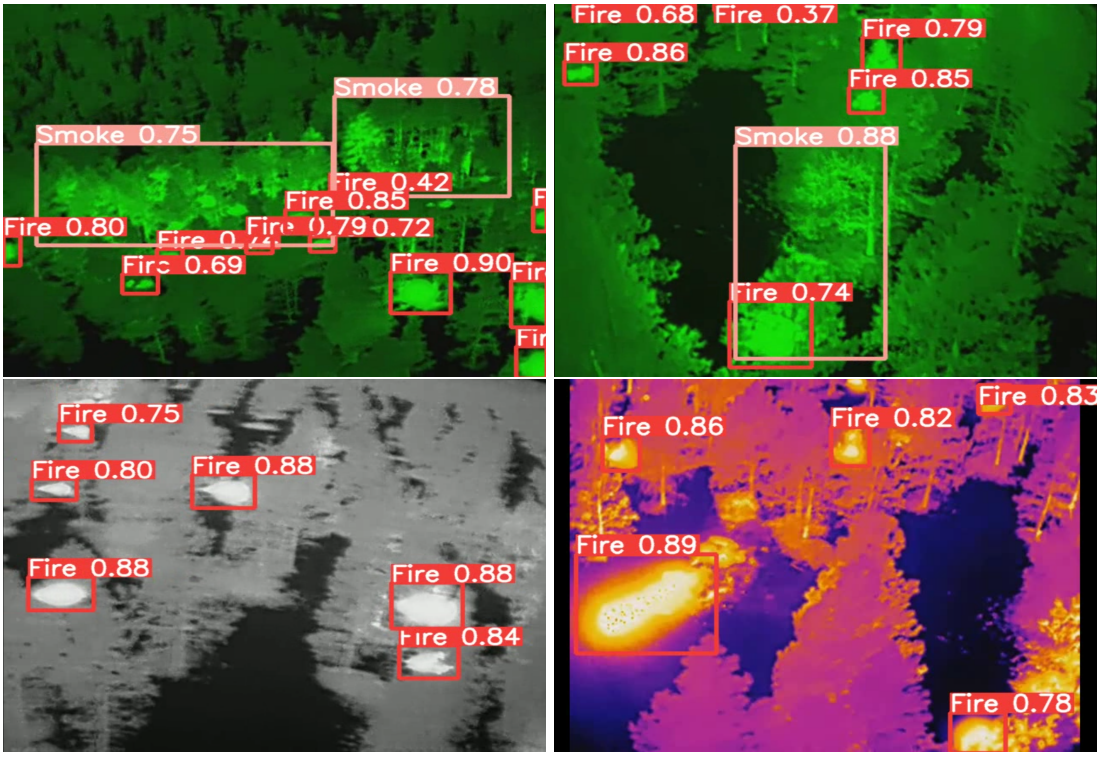

Figure 17: Examples of good detection by the Thermal model

Figure 13 shows the huge added value that a thermal camera gives to the system: thanks to its ability to detect heat sources in any visibility condition, this model easily recognizes small and/or occluded fires, even when flames are not visible at all, and independently from

the particular thermal filter. Yolov5l on thermal images detects and bounds every single fire instance, and does it with pretty high confidence. Its recall is stunning, but also the precision is very good: in presence of other sources of heat like people or cars, it does not produce any false positive, and as well as previous model it does not throw any alarm when no fire or smoke is present in the scene. On the contrary, given the thermal nature of the input, the model never mistakes clouds with smoke, as illustrated in Figure 18.

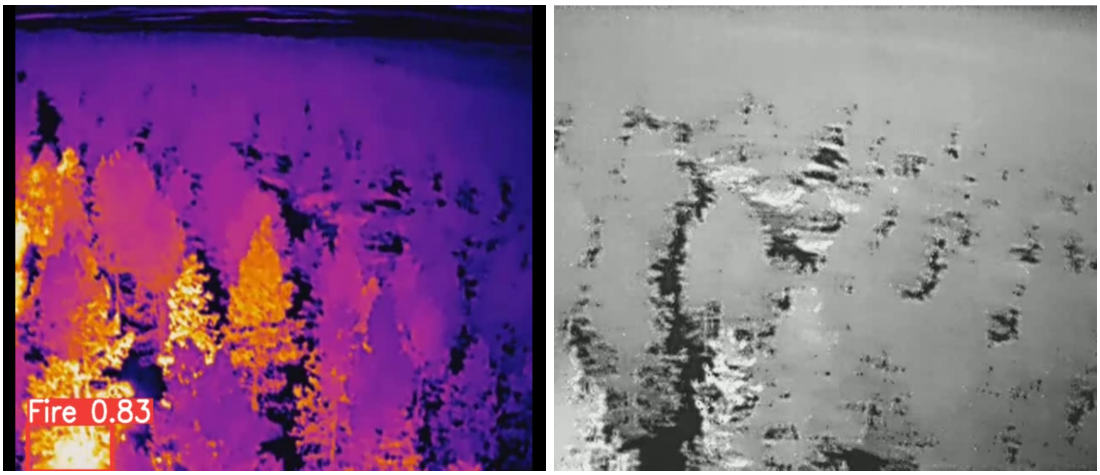

Figure 18: No smoke alarm is sent out when the camera frames clouds

The only point of weakness of this network seems to be smoke detection when many fire sources are present in the scene: in this case, as we can see from Figure 19, all the fires have been caught with high confidence, but the model does not signal the presence of smoke. This is not a problem of the model but a limitation of the thermal camera itself: the hot zone around a live fire is much wider than the smoke's one, so thermal cameras tend to highlight almost the totality of the scene; in such cases it is impossible even for humans to distinguish fire from any generic zone of hot air.

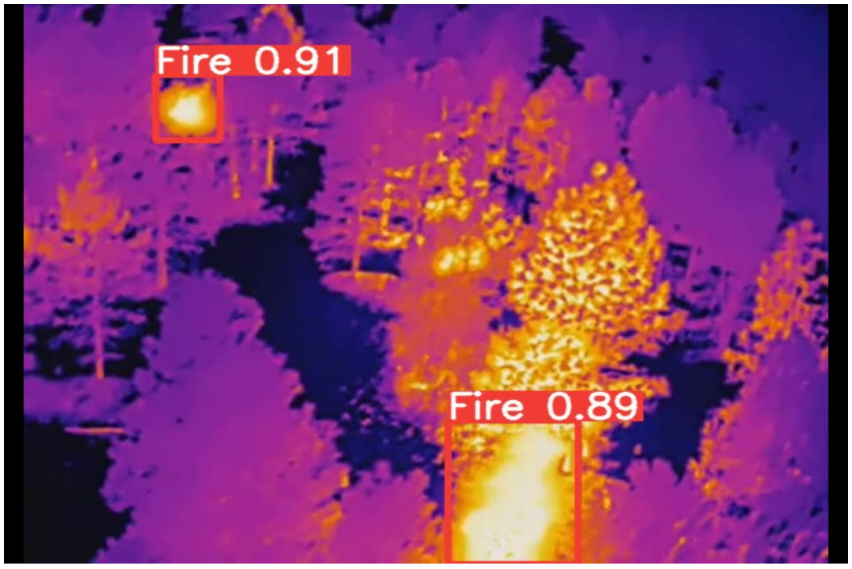

Figure 19: An example of failed smoke detection

6 — Conclusion

In this work, we presented a new approach to tackle fire and smoke detection on wild forests, using Computer Vision techniques to develop a detector network that could make use both of raw RGB images and Thermal ones, captured using embeddable cameras of any resolution.

Once we found a suitable mix of data sets, we processed it to create the learning sets, as summarized in the Table 1. For both tasks, we operated a subsampling of the training images in order to reduce the correlation between frames, and we added external data sources to increase the variance. All the object detection input pictures, both RGB and thermal ones, have been manually annotated by the authors, drawing bounding boxes and obtaining corresponding Yolo format labels.

After that, we tackled the Fire vs Non-Fire classification problem. Testing some configurations of Inception-v3 and EfficientNetB4 against the proposed data set, we discovered that, independently from the backbone model we choose, it is very hard to achieve an acceptable accuracy rate. For this reason, and for the fact that this approach does not allow us to exploit thermal images, in spite of the excellent results of the networks in the prediction time, we decided to discard them and to adopt a paradigm shift towards object detection.

We then moved to discuss the approaches we adopted and the models we trained to obtain a real-time smoke and fire detector starting from Yolov5. We presented the trade-off choices we had to evaluate and the motivation of the selections, analyzing then the training and testing results of the two models. We pointed out the great accuracy improvement we can achieve with this approach, which also takes care of prediction time, exploits thermal images and can exactly localize fire and smoke positions. We then modified the prediction function of the models to adapt it to work with real-time data. Given those 2 networks, we designed a policy to ensemble outputs that, leveraging the respective points of strength, could make the resulting model improve its detection ability.

To conclude, we showed some visual results of our detectors on same example images, highlighting the ability to catch all the fire sources represented in the pictures with high confidence, especially on thermal cameras. The high recall is coupled with a very good precision, allowing it not to signal false positives; however they expose some limitations, like the inability of the RGB-camera trained model to correctly bound the smoke when in unusual positions, or the struggle of the other one to detect smoke when there are multiple fire sources.

6.1 Future work

As future work, we propose to investigate more the ensemble between the two models. In the 4.5 subsection we introduced a possible policy to combine predictions, that consists on recomputing confidence level of the bounding boxes taking into consideration the intersections and the ability of each type of network to better detect smoke rather than fire. However, this approach needs to take as input synchronized and corresponding raw-thermal image pairs, which is possible nowadays but, as far as we know, not yet present on a public domain data set. The lack of this kind of data prevented us from implementing and testing the policy: as possible improvement of the current system, asserted the presence of the aforementioned images, would allow us also to experiment other joining strategies. Examples of them could be:

- ensemble the two models at a higher level of abstraction by combining deep layers
- consider the confidence weights as input to a multi-objective GA, to find the values that best trade-off mAP, precision, recall, prediction time
- choose a primary objective, like mAP, and optimize the weights through an ANFIS fuzzy network.

References

[1] Xiaojun Qi and Jessica Ebert. A computer vision based method for fire detection in color videos. *International journal of imaging*, 2(S09):22–34, 2009.

[2] Turgay Celik and Kai-Kuang Ma. Computer vision based fire detection in color images. In *2008 IEEE Conference on Soft Computing in Industrial Applications*, pages 258–263. IEEE, 2008.

[3] Hao Wu, Deyang Wu, and Jinsong Zhao. An intelligent fire detection approach through cameras based on computer vision methods. *Process Safety and Environmental Protection*, 127:245–256, 2019.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[5] Yun-Cheol Nam and Yunyoung Nam. A low-cost fire detection system using a thermal camera. *KSII Transactions on Internet and Information Systems (TIIS)*, 12(3):1301–1314, 2018.

[6] Alireza Shamsoshoara, Fatemeh Afghah, Abolfazl Razi, Liming Zheng, Peter Fulé, and Erik Blasch. The flame dataset: Aerial imagery pile burn detection using drones (uavs), 2020.

[7] FIRE Dataset outdoor-fire images and non-fire images for computer vision tasks. `https://www.kaggle.com/datasets/phylake1337/fire-dataset`.

[8] Fire Detection Dataset. `https://www.kaggle.com/datasets/atulyakumar98/test-dataset`.

[9] Malik Mohamed Umar, Liyanage C. De Silva, Muhammad Saifullah Abu Bakar, and Mohamad Iskandar Petra. State of the art of smoke and fire detection using image processing. *International Journal of Signal and Imaging Systems Engineering*, 10(1-2):22–30, 2017.

[10] Turgay Çelik and Hasan Demirel. Fire detection in video sequences using a generic color model. *Fire Safety Journal*, 44(2):147–158, 2009.

[11] Alireza Shamsoshoara, Fatemeh Afghah, Abolfazl Razi, Liming Zheng, Peter Z Fulé, and Erik Blasch. Aerial imagery pile burn detection using deep learning: the flame dataset. *Computer Networks*, 193:108001, 2021.

[12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[13] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[14] Imagenet benchmark. `https://paperswithcode.com/sota/image-classification-on-imagenet`. Accessed: 2012-05-24.

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision – ECCV 2014*. Springer International Publishing, 2014.

[16] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[17] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[18] YoloLabel application to label images used in yolo. `https://github.com/developer0hye/Yolo_Label`.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.