MCA
The 31ST Mediterranean Conference on Control and Automation
June 26 – 29, 2023 | GrandResort Limassol, Cyprus

A CNN based Real-time Forest Fire Detection System for Low-power Embedded Devices

Jianlin Ye*, Stelios Ioannou*†, Panagiota Nikolaou*, Marios Raspopoulos*†

*University of Central Lancashire, Pyla, Larnaca, Cyprus

†INterdisciplinary Science Promotion & Innovative Research Exploration (INSPIRE)

jye9@uclan.ac.uk, sioannou2@uclan.ac.uk, pnikolaou1@uclan.ac.uk, mraspopoulos@uclan.ac.uk

University of Central Lancashire UCLan Cyprus

inspire INterdisciplinary Science Promotion & Innovative Research Exploration

Overview

We tried to deploy the SoTA target detection algorithm on a low computing power embedded device (Raspberry Pi 4B) to detect forest fires in real time for **small UAV** applications:

- We tried to modify the original YOLOv5 **backbone** to a lightweight network, and restructured the whole network based on the new backbone.
- We performed a **channel pruning** operation on the modified YOLO network to make the network further compact and the network structure more simplified.
- We **overclocked** the CPU of Raspberry Pi 4B at the hardware level and investigated the effect of hardware acceleration on detection frame rate.
- Experimental results show that the proposed YOLOv5 has a **higher accuracy** rate (mAP@0.5) than the original YOLOv5 on the same test set, and that the detection frame rate(FPS) has been significantly improved(1.16 FPS to 8.57 FPS).

Comparison of Single Board Computers (SBCs)

https://www.singular.com.cy/raspberry-pi-4-model-b-single-board-computer-broadcom-bcm2711-1.5-ghz-ram-8gb.html?sl=el

https://developer.nvidia.com/embedded/jetson-nano-developer-kit

After carried out a research, the mainstream single board computer/embedded platform for deploying deep learning algorithms are Raspberry Pi and Nvidia Jetson series.

Benchmarks

Model	size	objects	mAP	Jetson Nano 1479 MHz	RPi 4 64-OS 1950 MHz
NanoDet	320x320	80	20.6	26.2 FPS	13.0 FPS
NanoDet Plus	416x416	80	30.4	18.5 FPS	5.0 FPS
YoloFastestV2	352x352	80	24.1	38.4 FPS	18.8 FPS
YoloV2	416x416	20	19.2	10.1 FPS	3.0 FPS
YoloV3	352x352 tiny	20	16.6	17.7 FPS	4.4 FPS
YoloV4	416x416 tiny	80	21.7	16.1 FPS	3.4 FPS
YoloV4	608x608 full	80	45.3	1.3 FPS	0.2 FPS
YoloV5	640x640 small	80	22.5	**5.0 FPS**	1.6 FPS
YoloV6	640x640 nano	80	35.0	10.5 FPS	2.7 FPS
YoloV7	640x640 tiny	80	38.7	8.5 FPS	2.1 FPS
YoloX	416x416 nano	80	25.8	22.6 FPS	7.0 FPS
YoloX	416x416 tiny	80	32.8	11.35 FPS	2.8 FPS
YoloX	640x640 small	80	40.5	3.65 FPS	0.9 FPS

Is the Jetson nano always the better choice?

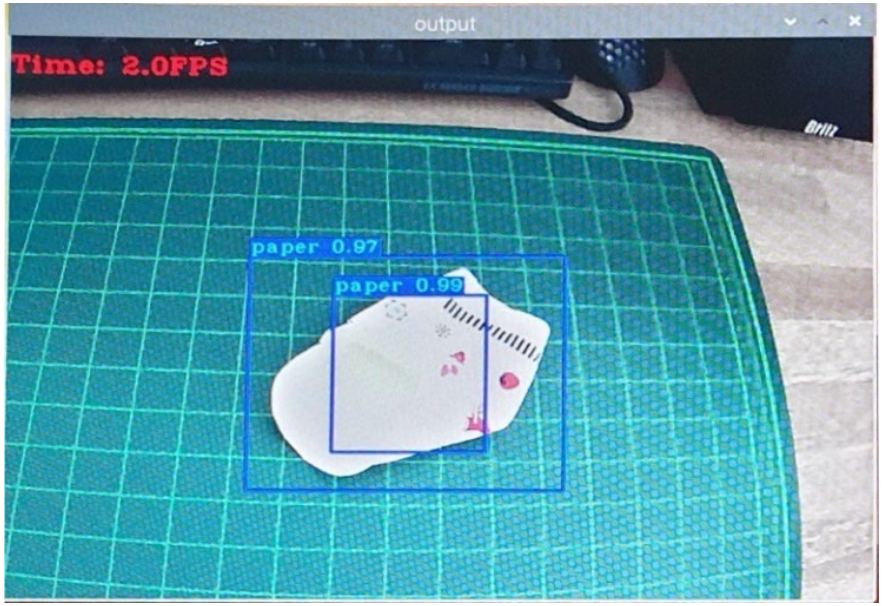
	Raspberry PI 4	Jetson Nano
Performance	13.5 GFLOPS	472 GFLOPS
CPU	Quad-core ARM CortexA72 64-bit @ 1.5 GHz	Quad-Core ARM Cortex-A57 64-bit @ 1.42 GHz
GPU	Broadcom Video Core VI (32-bit)	NVIDIA Maxwell w/ 128 CUDA cores @ 921 MHz
Memory	8 GB LPDDR4	4 GB LPDDR4 @ 1600MHz, 25.6 GB/s
Networking	Gigabit Ethernet / WiFi 802.11ac	Gigabit Ethernet / M.2 Key E
Display	2x microHDMI (up to 4Kp60)	HDMI 2.0 and eDP 1.4
USB	2x USB 3.0, 2x USB 2.0	4x USB 3.0, USB 2.0 Micro-B
Other	40-pin GPIO	40-pin GPIO
Video Encode	H264(1080p3 0)	H.264/H.265 (4Kp30)
Video Decode	H.265(4Kp60) ,H.264(1080p 60)	H.264/H.265 (4Kp60, 2x 4Kp30)
Camera	MIPI CSI port	MIPI CSI port
Storage	Micro-SD	16 GB eMMC
Power under load	2.56W-7.30W	5W-10W

NVIDIA's Jetson series is seen as a deployment accelerator for machine deployment. Some deep training applications of Jetson Nano developers are better evaluated than Raspberry Pi kit. [1]

Raspberry Pi 4B is 88 x 58 x 19.5mm and **46g**

NVidia Jetson nano is 164 x 107 x 42mm and **241g**

Relative Works



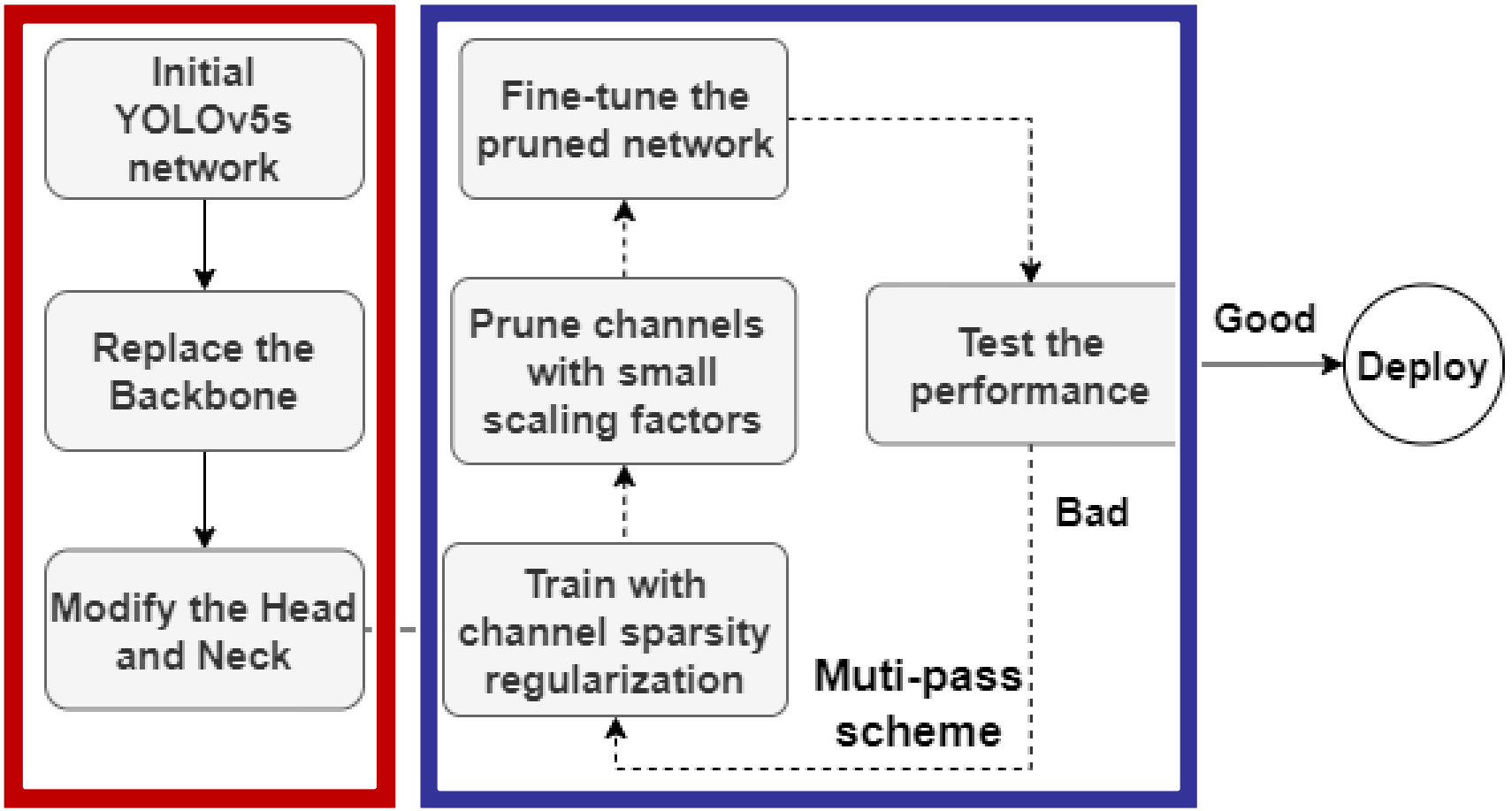
Wahyutama et al. implemented Yolov4 in Raspberry Pi and is performed at approximately 2 FPS in an actual operation scenario, resulting in an accuracy of 97–99%. Published: 21 April 2022

Electronics **2022**, 11, 1323.
https://doi.org/10.3390/electronics11091323

Gao et al. implemented Yolov5 in Raspberry Pi and the inference speed is approximately 0.5 FPS for the improved beehive detect and tracking system. Published: 27 October 2022

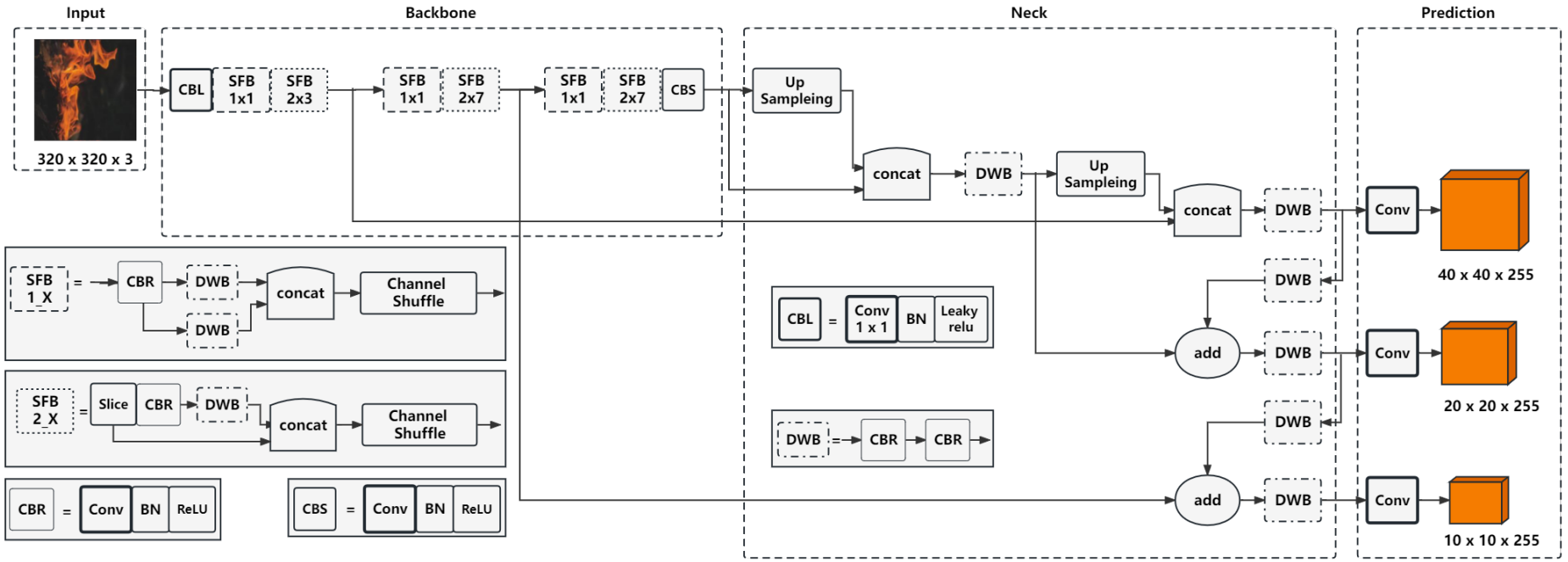
Journal of Biosystems Engineering
https://doi.org/10.1007/s42853-022-00166-6

Proposed optimization Framework



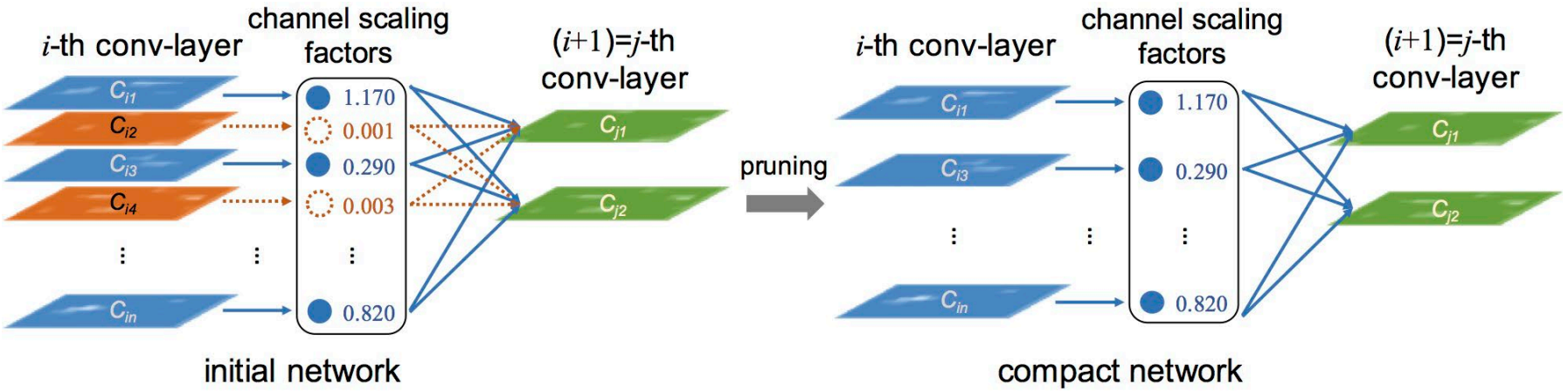
Dashed lines denote the iterative process.

Network Architecture



The architecture of the proposed network. 1) The input is a 320 × 320 three-channel RGB image. 2) The backbone of the proposed network is ShuffleNetV2, which can reduces the amount of cache space occupied and increases the inference speed. 3) The Neck network part uses a FPN + PAN architecture, with channel pruning of the Head in order to optimise memory access and usage.

Network Pruning

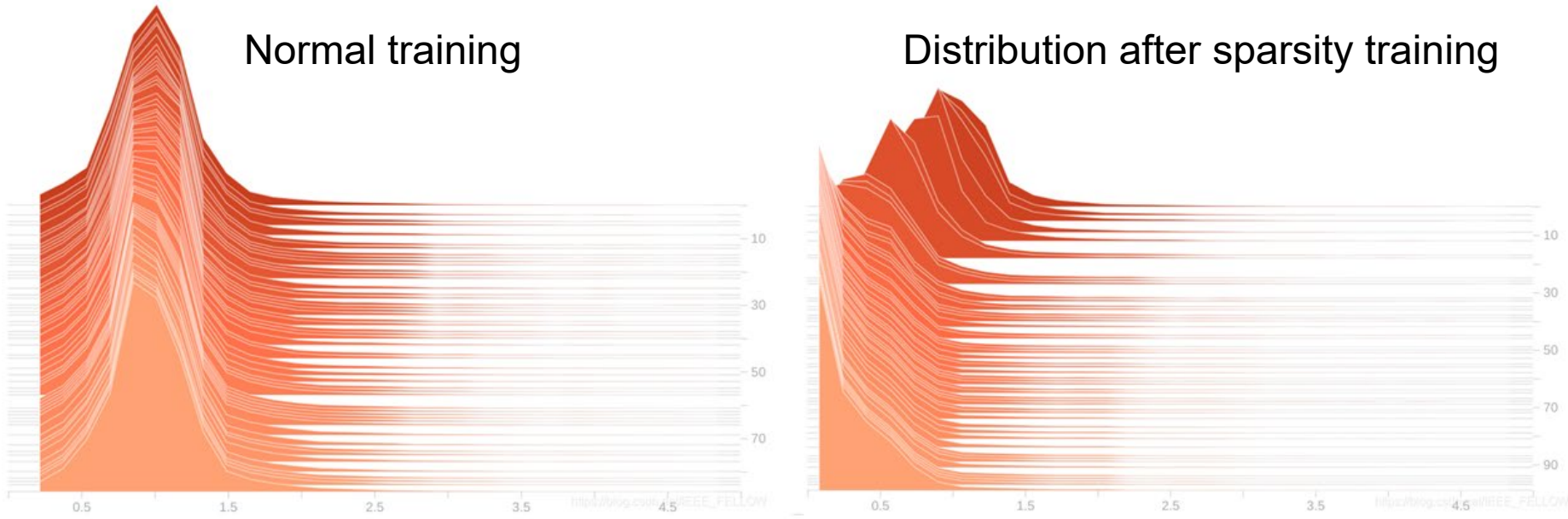


Liu, Zhuang, et al. "Learning efficient convolutional networks through network slimming." *Proceedings of the IEEE international conference on computer vision*. 2017.

We associate a scaling factor (reused from batch normalization layers) with each channel in convolutional layers. **Sparsity regularization** is imposed on these scaling factors during training to automatically identify **unimportant** channels.

The channels with small scaling factor values (in orange color) will be pruned (left side). After pruning, we obtain compact models (right side), which are then fine-tuned to achieve comparable (or **even higher**) accuracy as normally trained full network.

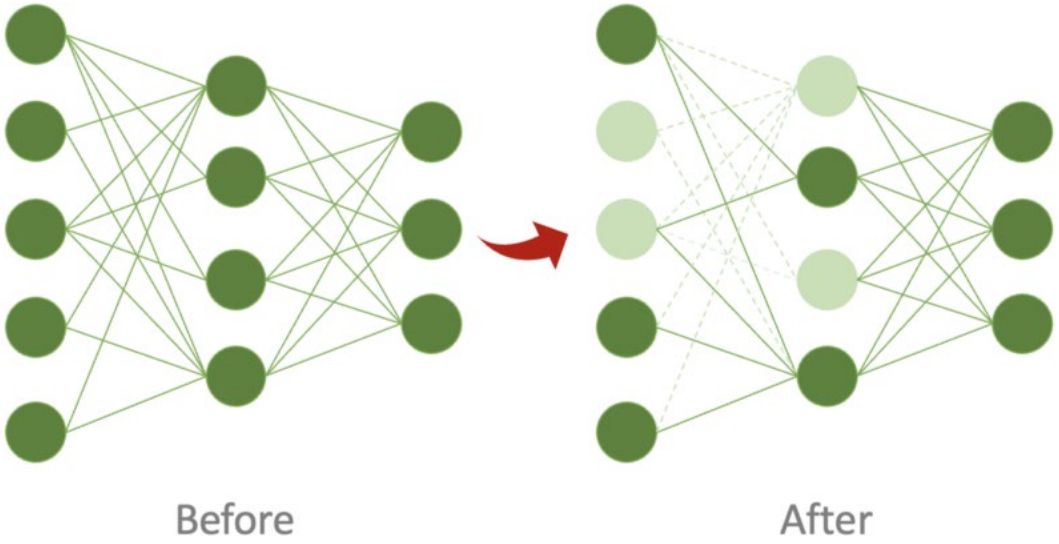
Sparse training and pruning preparation



(a) (b)

- It uses the scaling factors of the BN layer and associates the scaling factors with each channel in the convolutional layer.
- A sparse regularization is applied to these scale factors during training so that the unimportant scale factor is approximated to **zero**, thus automatically identifying the unimportant channels.
- By pruning the orange channels with a scale factor close to 0, we can obtain the compact network, which is the channel with a larger scale factor.

Pruning process and Fine-tuning of the pruned model



- After obtaining the sparse trained model, the next step is to prune out the channels with γ going to 0. This method is based on the structured pruning of the channels, and the accuracy of the pruning will generally be reduced. We can fine-tune the finetune of the compact network to improve its accuracy, so that it is comparable to the normal training network, or even more accurate.

- The reduction in the accuracy after pruning can be **fine-tuned** to recover. When the pruned model is able to relearn the neural network parameters based on the current network structure, it is the fine-tuned for training and this restores the detection accuracy of the model and improves the mapping effect.

Hardware Acceleration

Clock (MHz)	Overvoltage	Vcore	Max temp. (°C | °F)	Power (Watt)	Preformance increase	Remarks
0	0	0.8625		1.5		RPi 4 shut down
200	0	0.8625		1.75		RPi 4 min working clock
600	0	0.8625		2.8		RPi 4 running idle
1500	0	0.8625	82 | 180	7		Factory settings
1600	1	0.8875	80 | 176	7.6	6.6 %	
1700	2	0.9125	78 | 172	8.3	13.3 %	
1800	3	0.9375	77 | 170	8.9	20 %	
1900	4	0.9625	75 | 167	9.5	26.6 %	
2000	6	1.0125	72 | 162	11	33.3 %	
2100	6	1.0125	72 | 162	11	40 %	
	7	1.0375	56 | 132	11.7		no improvement
	8	1.0625	50 | 122	12.3		no improvement

- In order to get the best performance out of the algorithms, the RPi4B CPU was overclocked to **2.0 GHz** (maximum of 2140 MHz).
- As the RPi is normally used with a CPU with its NEON-ARM instructions, the GPU was not overclocked for this study and its default frequency, 500 MHz was used (maximum of 650 MHz).
- To avoid overheating of the RPi platform, automatic over-voltage adjustment and dynamic clock frequency were used.

Dataset

The dataset was randomly divided into three independent and equally distributed sets:

(i) the **Training set**, containing 83% of the images (20,255);

(ii) the **Validation set**, containing 13% of the images (3,148);

(iii) the **Test set**, containing 4% of the images (977).

Some examples of the dataset: (a) and (b) show image-based data augmentation techniques to expand the dataset. (c) and (d) show respective ground-truth bounding boxes. (e) and (f) represent images that are prone to false detection. (g) is a ground-based image of the forest fire and (h) is an aerial view of the fire.

Comparison of models‘ performances on Prediction

TABLE I
PERFORMANCE METRICS ON THE TEST SET

Network	mAP@0.50 (%)	AP@0.50 smoke (%)	AP@0.50 fire (%)	F_1 score	avg IoU (%)
YOLOv5	82.93	91.17	74.86	0.83	73.51
Proposed	92.54	96.35	88.71	0.85	68.34

The optimised YOLOv5 outperforms the original YOLOv5 across most of the evaluation metrics.

However, the location of objects detected by the original YOLOv5 network is on average more accurate in terms of IoU experimental values. This is because the original YOLOv5 retains more convolutional layers than the optimised network

Analysis of the model pruning results

TABLE II
PARAMETER COMPARISON OF THE PROPOSED DETECTION MODEL UNDER DIFFERENT PRUNING RATES.

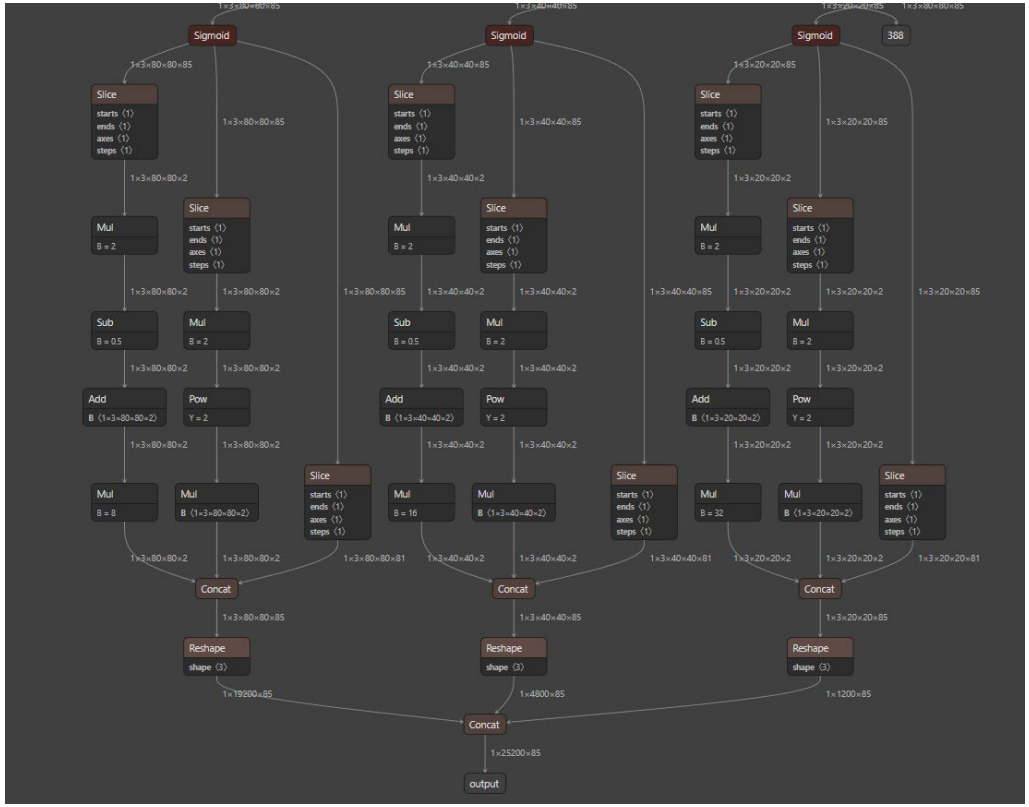
Pruning Rate(%)	mAP@0.5(%)	Parameters/10^6	Model Size(MB)
Baseline	93.7	7.02	14.1
80	82.3	0.89	1.95
70	89.7	1.54	3.34
60	91.2	2.30	4.64
50	93.2	2.85	5.66

Parameter comparison of the proposed detection models for different channel pruning rates are tabulated on Tab. II, According to Tab. II, all 4-evaluation metrics were reduced for different channel pruning rates. After performing fine-tuning training, the mAP recovered to 84.87%, 92.5%, 92.65% and 93.41% respectively.

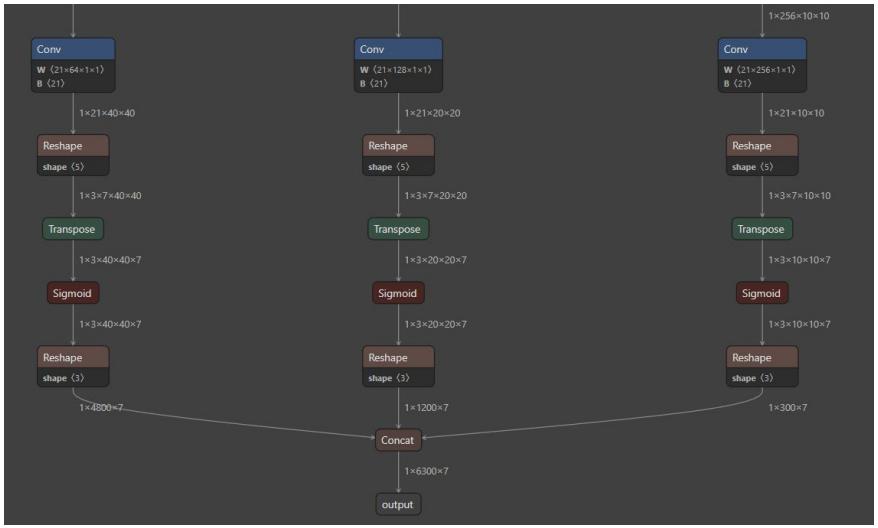
Fine-tuning revealed that the channel pruning rate of **70%** achieved the best balance between accuracy and inference speed, which resulted in a better model compression with less loss of average accuracy.

Network Structure Comparison

The pruning rules refer to the design guidelines of ShuffleNet v2.



Original Yolov5 Head



Optimized Yolov5 Head

Overclocking Performance Comparison

TABLE III
PERFORMANCE METRICS ON THE COMPARISON EXPERIMENT

Network	Clock Speed (GHz)	Power (W)	CPU Usage (%)	CPU Temp ($^{\circ}C$)	FPS
YOLOv5	0.6	5.70	93	47	0.41
YOLOv5	1.8	7.70	95	58	1.06
YOLOv5	2.0	8.10	97	67	1.16
Proposed	0.6	5.20	60	43	3.02
Proposed	1.8	7.02	58	47	7.23
Proposed	2.0	7.28	64	50	8.57

Performance of the Raspberry Pi 4B at Different Overclocked Frequencies

Detection results from the Comparison Experiment

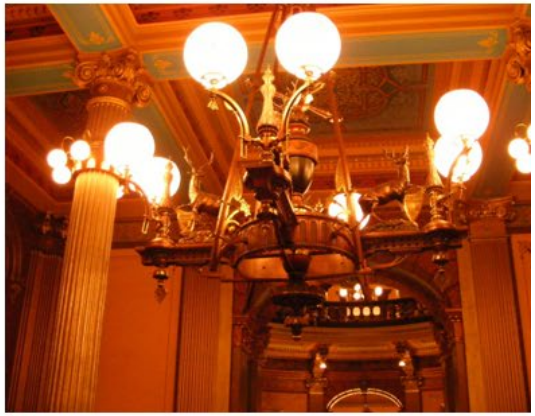
(a)

(b)

(c)

(d)

(e)



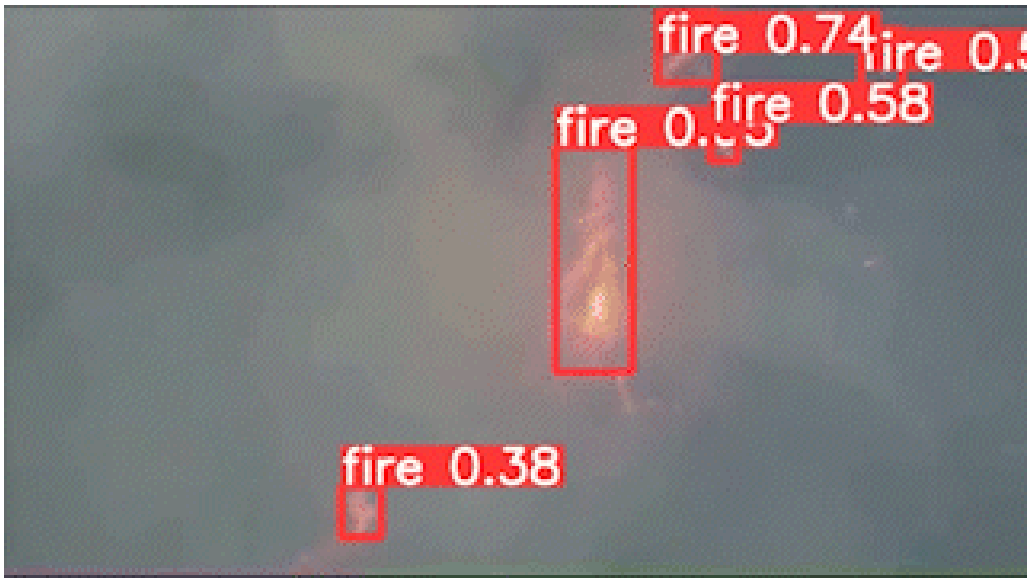
(f)

The detection results show that the proposed YOLOv5 can mark out more areas where flames and smoke are present in the pictures and there are no false detections in pictures with flame-like objects.

Detection Results on Raspberry Pi Platform

Evaluation on the Raspberry Pi platform: the detection result of the image obtained in real time from the web camera, with the live detection frame rate in the top left corner of the detection screen.

Video input Results for testing the model performance



A video of the California fires [2] was obtained from the internet and the results of flame and smoke detection using the optimized model show that the model can still detect the location of the flames even in a smoke-filled scene and mark the location of the flames when the camera is zoomed out.

Conclusion

- The optimized YOLOv5 produced the highest mAP of 92.5% compared to the ordinary YOLOv5s model and could detect at **7-9 FPS** on the RPi-4B.
- The optimized model use **35%** less CPU usage than the original YOLOv5.
- The reduced CPU usage also translated to **25%** reduction in CPU temperature.
- The deployment approach in this study reduces the difficulty of deploying the deep-learning fire detection model on edge devices.

Questions

Reference

1. P. Howard, “The Cost of Carbon Project - Flammable Planet: Wildfires and the Social Cost of Carbon”, Institute of Policy Integrity, NY University, School of Law, 2014.
2. https://medium.com/@anil.ozenn/jetson-nano-vs-raspberry-pi%CC%87-4-b1f6fbf5a00e
3. https://www.youtube.com/watch?v=0k7ipkU6gHw&ab_channel=CBS8SanDiego

Replacement of the Backbone Network

- Computational complexity and parameter storage have a **negative impact** on the speed of CNN networks which can be extremely slow when running on computational- and power constrained devices.
- The current state-of-the-art lightweight network ShuffleNetV2 is an improved version based on ShuffleNet. ShuffleNetV2 is faster and more accurate than most other networks for the same complexity of inference. Hence, it is ideal for replacing the ordinary YOLOv5 backbone network.

The selection process of the Backbone

We try to replace the backbone feature extraction network with the lighter ShuffleNetV2 network to achieve a lightweight network model that balances speed and accuracy.

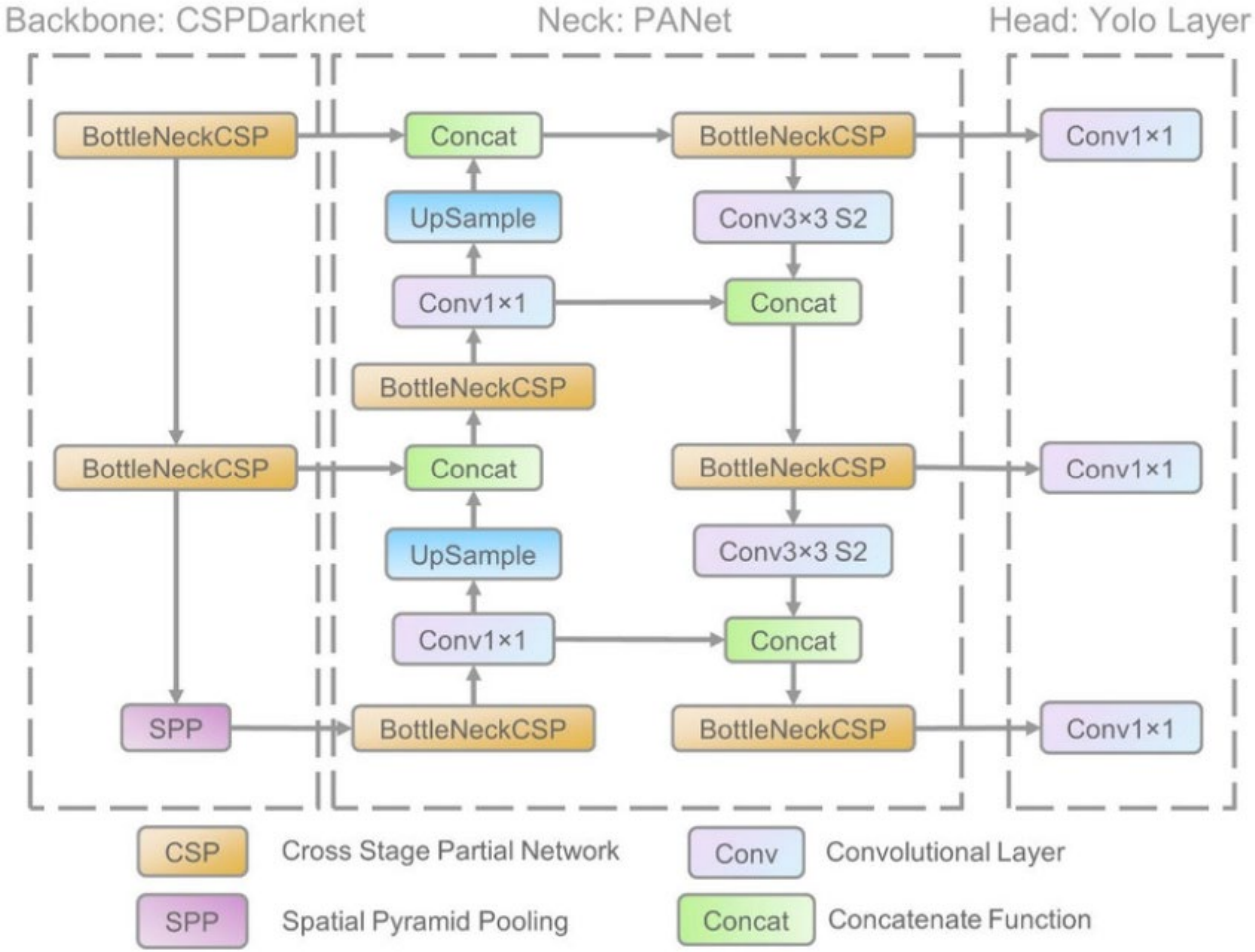
Backbone	mAP
cspdarknet	-
shufflenetv2	-2.1%
mobilenetv2	-3.8%
ghostnet	-7.2%

https://blog.csdn.net/hexiao260/article/details/124915149

This blog documents the process of deploying the target detection algorithm to an embedded device (jetson nano) and some modification strategies to lighten and improve the accuracy of the YOLOv4 algorithm.

ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design https://arxiv.org/abs/1807.11164

YOLOv5 Algorithm



As the current state-of-the-art deep learning target detection algorithm, YOLOv5, has gathered a large number of tricks, but there is still room for improvement and enhancement, and different improvement methods can be used for the detection difficulties of specific application scenarios.

The next section will focus on how we made improvements to YOLOv5 in detail.

https://www.analyticsvidhya.com/blog/2021/12/how-to-use-yolo-v5-object-detection-algorithm-for-custom-object-detection-an-example-use-case/

Change the backbone (1/2)

```
class ShuffleNetV2_InvertedResidual(nn.Module):
    def __init__(
            self,
            inp: int,
            oup: int,
            stride: int
    ) -> None:
        super(ShuffleNetV2_InvertedResidual, self).__init__()

        if not (1 <= stride <= 3):
            raise ValueError('illegal stride value')
        self.stride = stride

        branch_features = oup // 2

        if self.stride > 1:
            self.branch1 = nn.Sequential(
                self.depthwise_conv(inp, inp, kernel_size=3, stride=self.stride, padding=1),
                nn.BatchNorm2d(inp),
                nn.Conv2d(inp, branch_features, kernel_size=1, stride=1, padding=0, bias=False),
                nn.BatchNorm2d(branch_features),
                nn.ReLU(inplace=True),
            )
        else:
            self.branch1 = nn.Sequential()

        self.branch2 = nn.Sequential(
            nn.Conv2d(inp if (self.stride > 1) else branch_features,
                      branch_features, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(branch_features),
            nn.ReLU(inplace=True),
            self.depthwise_conv(branch_features, branch_features, kernel_size=3, stride=self.stride, padding=1),
            nn.BatchNorm2d(branch_features),
            nn.Conv2d(branch_features, branch_features, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(branch_features),
            nn.ReLU(inplace=True),
        )
```

First step is to modify common.py and add the ShuffleNetV2 module.

Change the backbone (2/2)

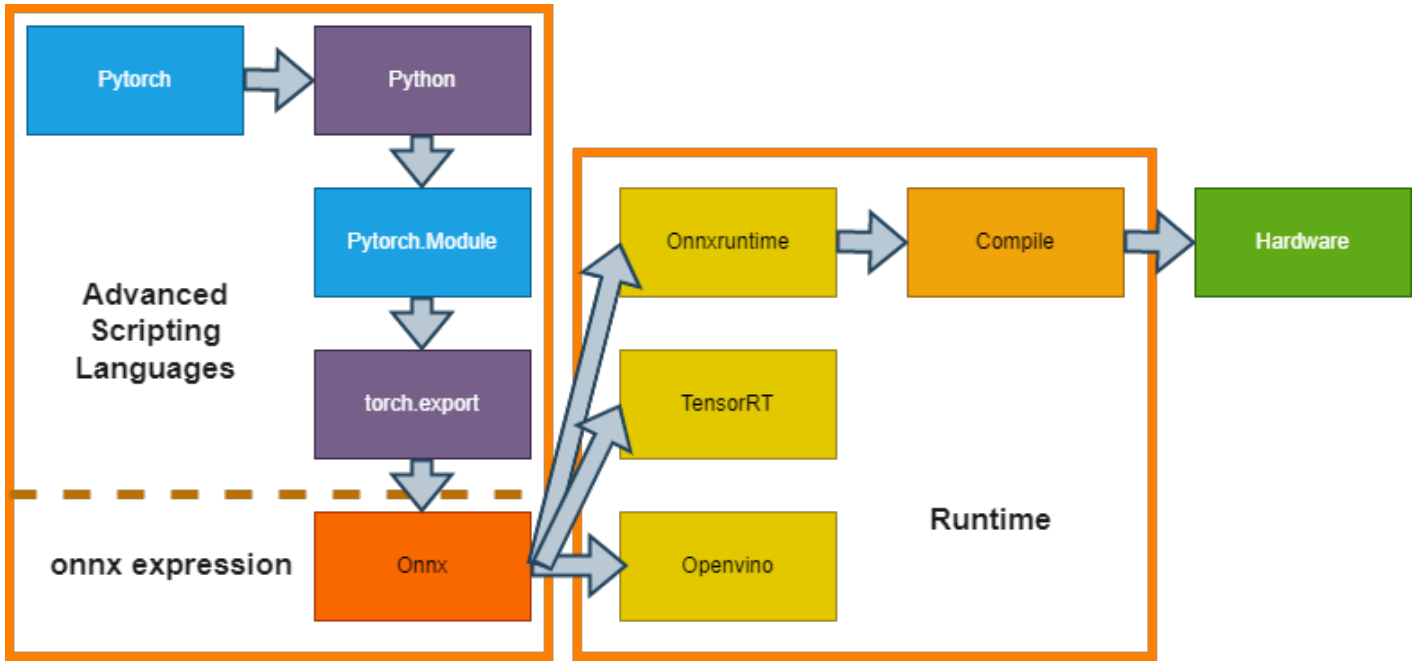
Step 2: Register the module ShuffleNetV2 in yolo.py.

If m in [Conv,MobileNetV3_InvertedResidual,ShuffleNetV2_InvertedResidual]

Step 3: Modify the yaml file

```yaml
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],    # 0-P2/4
   [-1, 1, ShuffleNetV2_InvertedResidual, [128, 2]], # 1-P3/8
   [-1, 3, ShuffleNetV2_InvertedResidual, [128, 1]], # 2
   [-1, 1, ShuffleNetV2_InvertedResidual, [256, 2]], # 3-P4/16
   [-1, 7, ShuffleNetV2_InvertedResidual, [256, 1]], # 4
   [-1, 1, ShuffleNetV2_InvertedResidual, [512, 2]], # 5-P5/32
   [-1, 3, ShuffleNetV2_InvertedResidual, [512, 1]], # 6
  ]
```

Deployment on Raspberry Pi Platform



- To achieve running deep learning algorithms on Raspberry Pi in real time, we transform the PyTorch model obtained after the training of the improved Yolov5 algorithm into the ONNX cross-frame model intermediate expression model.
- The quantized model is then run in the onnxruntime framework for testing the performance of the fire detection algorithm.
- After completing the evaluation, the onnxruntime framework and the completed quantization model are deployed to the Raspberry Pi 4B to run the fire detection algorithm.