# MILESTONE REVIEW

## Team NaviGatr

**Current Navigational Technology for the Visually Impaired**

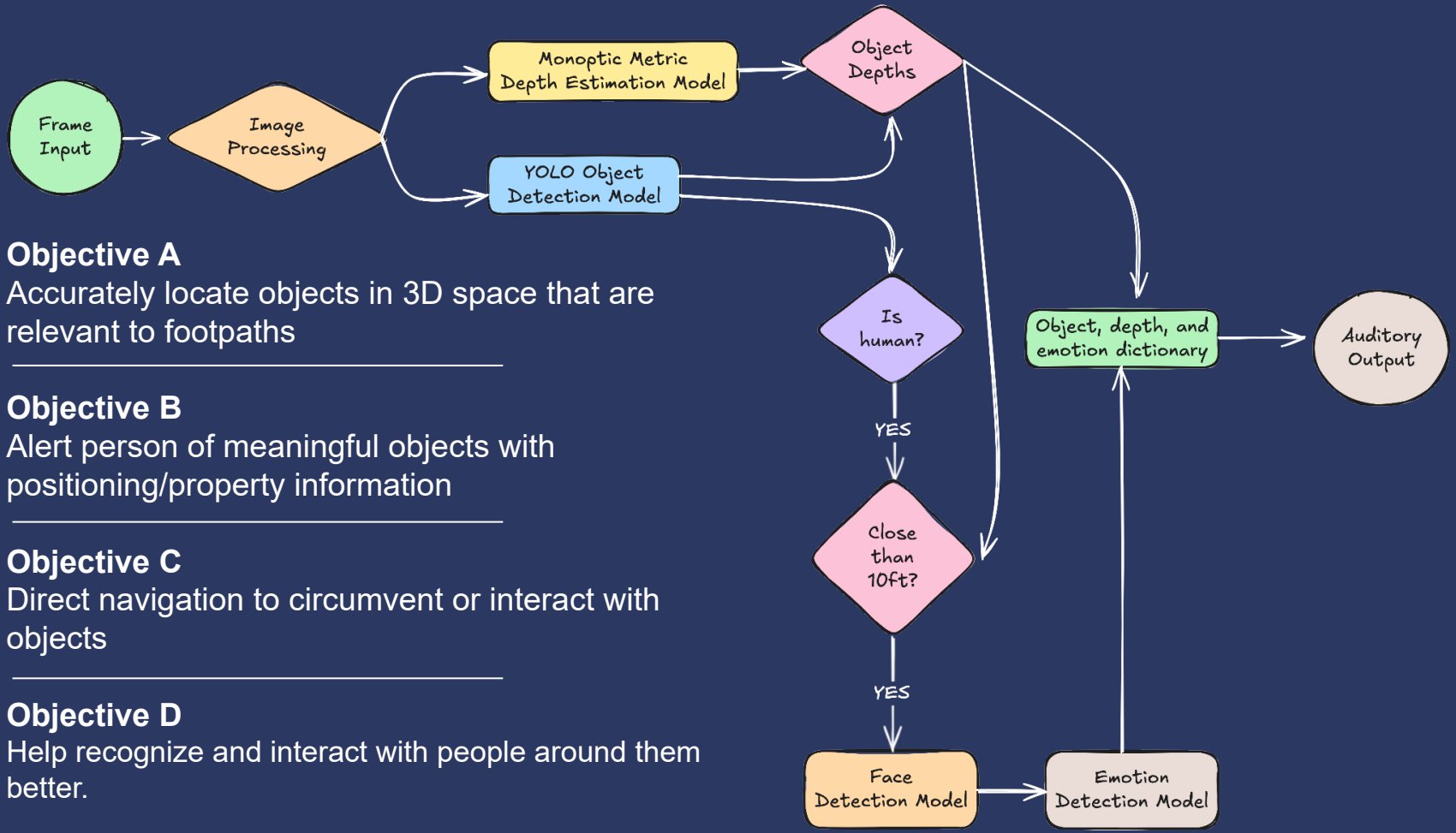
- Manual systems can require contact (cane)
- E-solutions can be bulky
- E-solutions can require internet connection
- E-solutions could require expensive stereo cameras
- E-solutions can require LiDAR or infrared modules for accuracy

**Using ML, we can create a lightweight, on-board, and inexpensive solution to navigation**

**Key Objectives**

- Objective A - Accurately locate objects in 3D space that are relevant to footpaths
- Objective B - Alert person of meaningful objects with positioning/property information
- Objective C - Direct navigation to circumvent or interact with objects
- Objective D- Help recognize and interact with people around them better.

All objectives are intended to support **zero-visual** capabilities without the use of the product.




**Objective A**
Accurately locate objects in 3D space that are relevant to footpaths

---

**Objective B**
Alert person of meaningful objects with positioning/property information

---

**Objective C**
Direct navigation to circumvent or interact with objects

---

**Objective D**
Help recognize and interact with people around them better.

# Object Detection Component

**Key considerations for choosing the right model:**

- Speed over accuracy (real-time application)

  Avg. walk speed ~1.79 [m/s] (high estimate) [7]

  Our Pi Camera captures at 30FPS for 1080p and 60FPS for 720p [8]

  Avg. stride length ~0.762 [m] (high estimate) [9]

  Minimum Model FPS (IF 2 frames/stride) = (2/0.762)(1.79) [(frames/m)(m/s)]

  = 4.698FPS

  Camera viability = (30/1.79)(0.762) [(s-frames/m-s)(m/stride)]

  = 12.77 [frames/stride] ✓

- Edge-device compatibility (Raspberry Pi4)

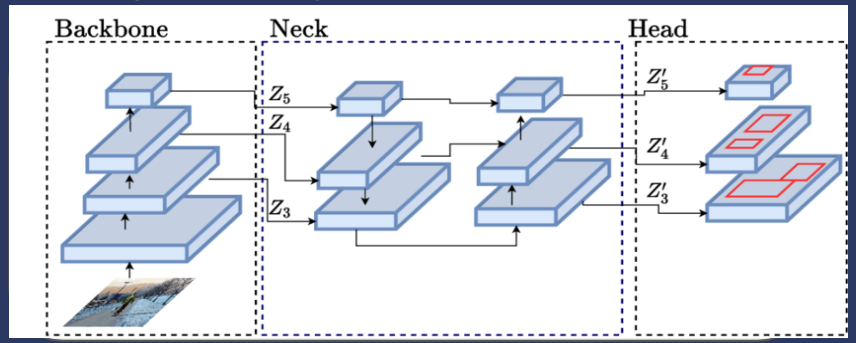
  Benchmark assessment only considers testing on the device being used

- Supporting Documentation

  Must be open-source, provides adequate demonstration for deployment, and readily readable

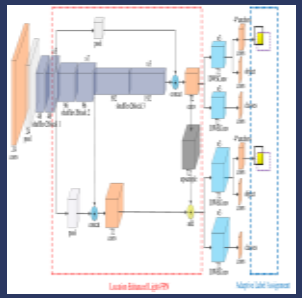
# Object Detection Component

Common high–level architecture for a model designed for object detection:

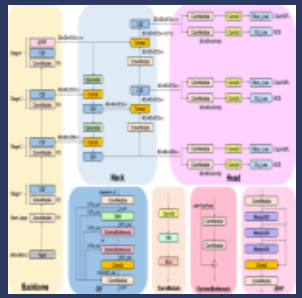



[2]

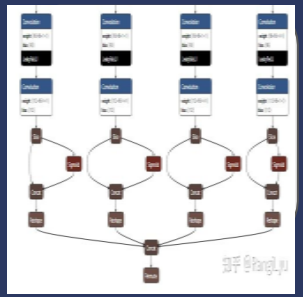
Models with this architecture: Possible choices



[3]

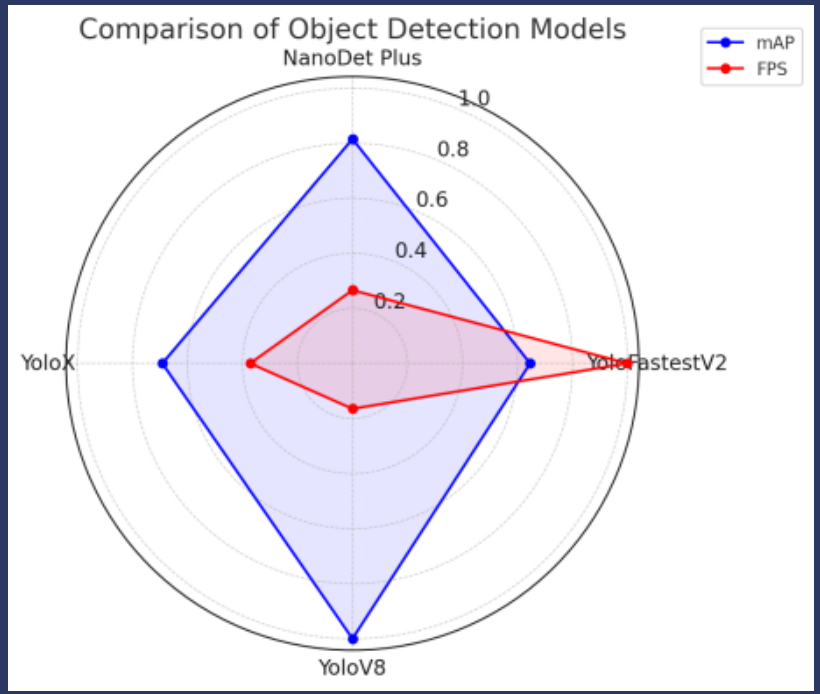


[4]



[5]

Comparison of possible models:




[6]

# Object Detection Component

**Key features:**

- FCOS-style (grid-cells instead of per-pixel)
- ShuffleNetV2 (ultra lightweight and fast)

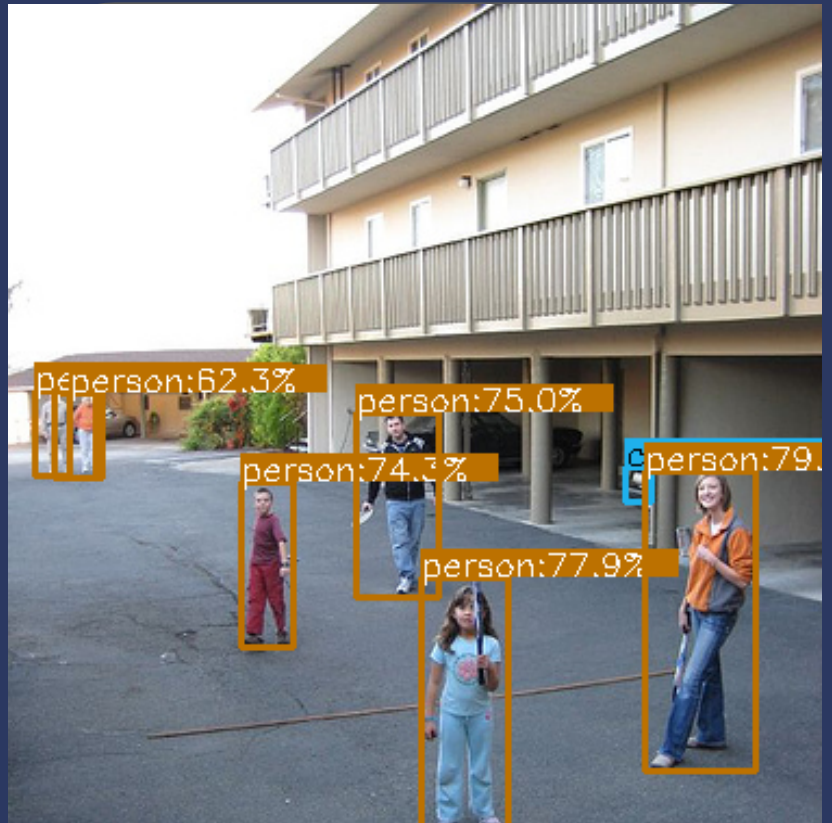
**Key issues found:**

- Boxes in foreground cause cluttering
- Classification confidence variability

**Key steps forward:**

- Upon integration, a performance assessment is necessary
  - Avg. speed to process single frame
  - Accuracy to detect same frame, many runs
  - Accuracy to detect different frames, many runs
  - Latency in providing service
- Output processing to yield only relevant information

Current Test Results:



# Depth Sensing Component

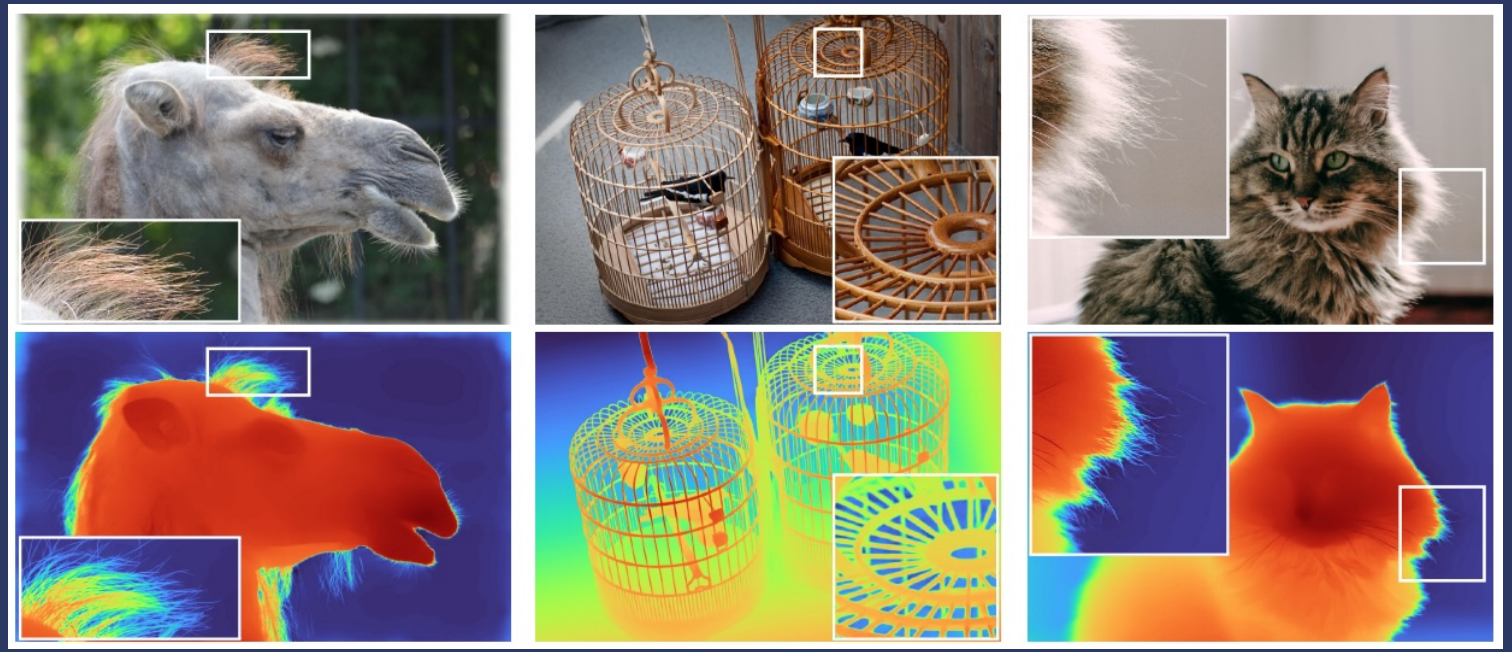
How can we learn the depth of each pixel in a 2D image (i.e. 2D → 3D)?
**Monocular Depth Estimation**

**Issues**

- Previous approaches focused on relative depth between pixels[7]
- Some required metadata about the camera for accurate depth estimation[8]
- Different models varied widely in their speed[7]
- Some approaches require specific training data to the environment they will perform in[9]

# Depth Sensing Component

Our Solution? **Apple's Depth Pro**

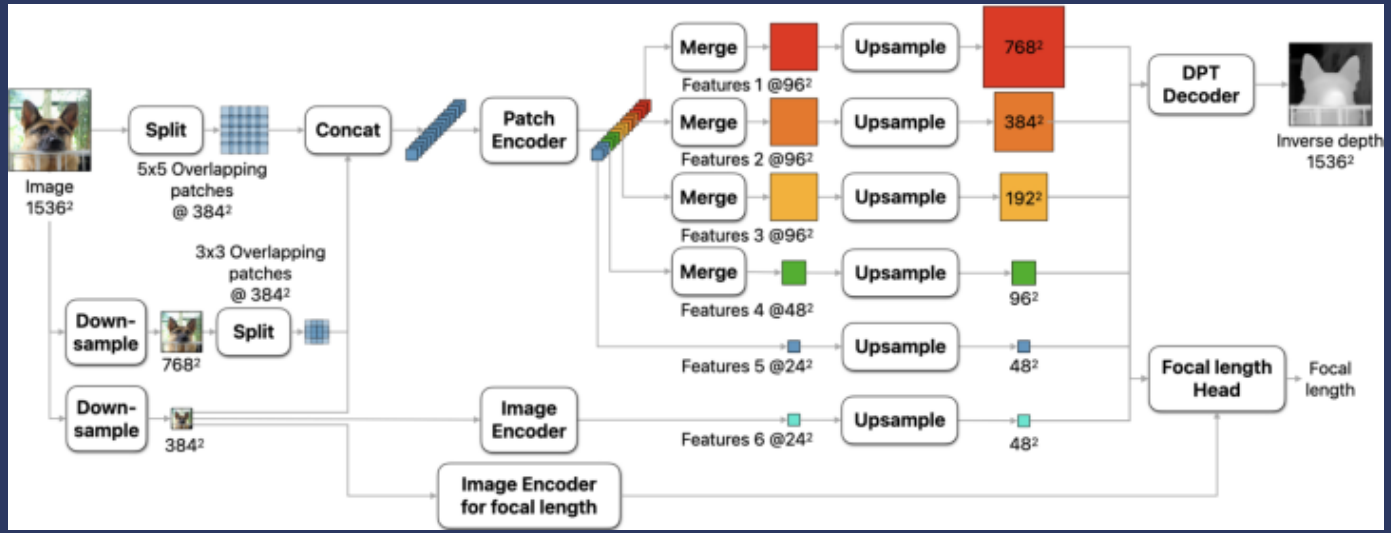


Multiple images with color gradient depth maps generated by Depth Pro [8]

# Depth Sensing Component

## Depth Pro Architecture

- Utilizes vision transformers, not CNNs
- Splits image into patches at different scales
- Learns different types of features at different scales
- Uses a Dense Prediction Transformer for depth prediction
- Predicts the camera's focal length in parallel for metric estimation




Depth Pro Architecture Model [7]

# Depth Sensing Component

## Depth Pro Performance?

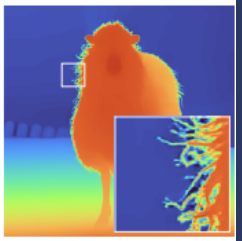
| Method | Booster | ETH3D | Middlebury | NuScenes | Sintel | Sun-RGBD | Avg. Rank ↓ |
|---|---|---|---|---|---|---|---|
| DepthAnything (Yang et al., 2024a) | 52.3 | 9.3 | 39.3 | 35.4 | 6.9 | 85.0 | 5.7 |
| DepthAnything v2 (Yang et al., 2024b) | 59.5 | 36.3 | 37.2 | 17.7 | 5.9 | 72.4 | 5.8 |
| Metric3D (Yin et al., 2023) | 4.7 | 34.2 | 13.6 | 64.4 | 17.3 | 16.9 | 5.8 |
| Metric3D v2 (Hu et al., 2024) | 39.4 | 87.7 | 29.9 | 82.6 | 38.3 | 75.6 | 3.7 |
| PatchFusion (Li et al., 2024a) | 22.6 | 51.8 | 49.9 | 20.4 | 14.0 | 53.6 | 5.2 |
| UniDepth (Piccinelli et al., 2024) | 27.6 | 25.3 | 31.9 | **83.6** | 16.5 | **95.8** | 4.2 |
| ZeroDepth (Guizilini et al., 2023) | OOM | OOM | 46.5 | 64.3 | 12.9 | OOM | 4.6 |
| ZoeDepth (Bhat et al., 2023) | 21.6 | 34.2 | 53.8 | 28.1 | 7.8 | 85.7 | 5.3 |
| **Depth Pro (Ours)** | 46.6 | 41.5 | **60.5** | 49.1 | **40.0** | 89.0 | **2.5** |

| | Method | Sintel F1 ↑ | Spring F1 ↑ | iBims F1 ↑ | AM R ↑ | P3M R ↑ | DIS R ↑ |
|---|---|---|---|---|---|---|---|
| Absolute | DPT (Ranftl et al., 2021) | 0.181 | 0.029 | 0.113 | 0.055 | 0.075 | 0.018 |
| | Metric3D (Yin et al., 2023) | 0.037 | 0.000 | 0.055 | 0.003 | 0.003 | 0.001 |
| | Metric3D v2 (Hu et al., 2024) | 0.321 | 0.024 | 0.096 | 0.024 | 0.013 | 0.006 |
| | ZoeDepth (Bhat et al., 2023) | 0.027 | 0.001 | 0.035 | 0.008 | 0.004 | 0.002 |
| | PatchFusion (Li et al., 2024a) | 0.312 | 0.032 | 0.134 | 0.061 | 0.109 | 0.068 |
| | UniDepth (Piccinelli et al., 2024) | 0.316 | 0.000 | 0.039 | 0.001 | 0.003 | 0.000 |
| Rel. | DepthAnything (Yang et al., 2024a) | 0.261 | 0.045 | 0.127 | 0.058 | 0.094 | 0.023 |
| | DepthAnything v2 (Yang et al., 2024b) | 0.228 | 0.056 | 0.111 | 0.107 | 0.131 | 0.056 |
| | Marigold (Ke et al., 2024) | 0.068 | 0.032 | 0.149 | 0.064 | 0.101 | 0.049 |
| | **Depth Pro (Ours)** | **0.409** | **0.079** | **0.176** | **0.173** | **0.168** | **0.077** |

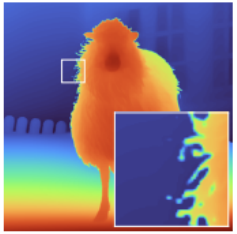
| | DDDP | | FiveK | | PPR10K | | RAISE | | SPAQ | | ZOOM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\delta_{25\%}$ | $\delta_{50\%}$ | $\delta_{25\%}$ | $\delta_{50\%}$ | $\delta_{25\%}$ | $\delta_{50\%}$ | $\delta_{25\%}$ | $\delta_{50\%}$ | $\delta_{25\%}$ | $\delta_{50\%}$ | $\delta_{25\%}$ | $\delta_{50\%}$ |
| UniDepth (Piccinelli et al., 2024) | 6.8 | 40.3 | 24.8 | 56.2 | 13.8 | 44.2 | 35.4 | 74.8 | 44.2 | 77.4 | 20.4 | 45.4 |
| SPEC (Kocabas et al., 2021) | 14.6 | 46.3 | 30.2 | 56.6 | 34.6 | 67.0 | 49.2 | 78.6 | 50.0 | 82.2 | 23.2 | 43.6 |
| im2pcl (Baradad & Torralba, 2020) | 7.3 | 29.6 | 28.0 | 60.0 | 24.2 | 61.4 | 51.8 | 75.2 | 26.6 | 55.0 | 22.4 | 42.8 |
| **Depth Pro (Ours)** | **66.9** | **85.8** | **74.2** | **92.4** | **64.6** | **88.8** | **84.2** | **96.4** | **68.4** | **85.2** | **69.8** | **91.6** |

Image

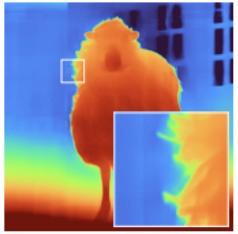
Alpha Matte



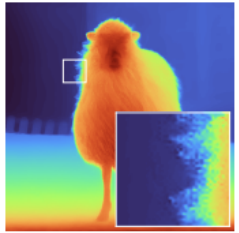
Depth Pro (Ours)



DepthAnything v2



PatchFusion



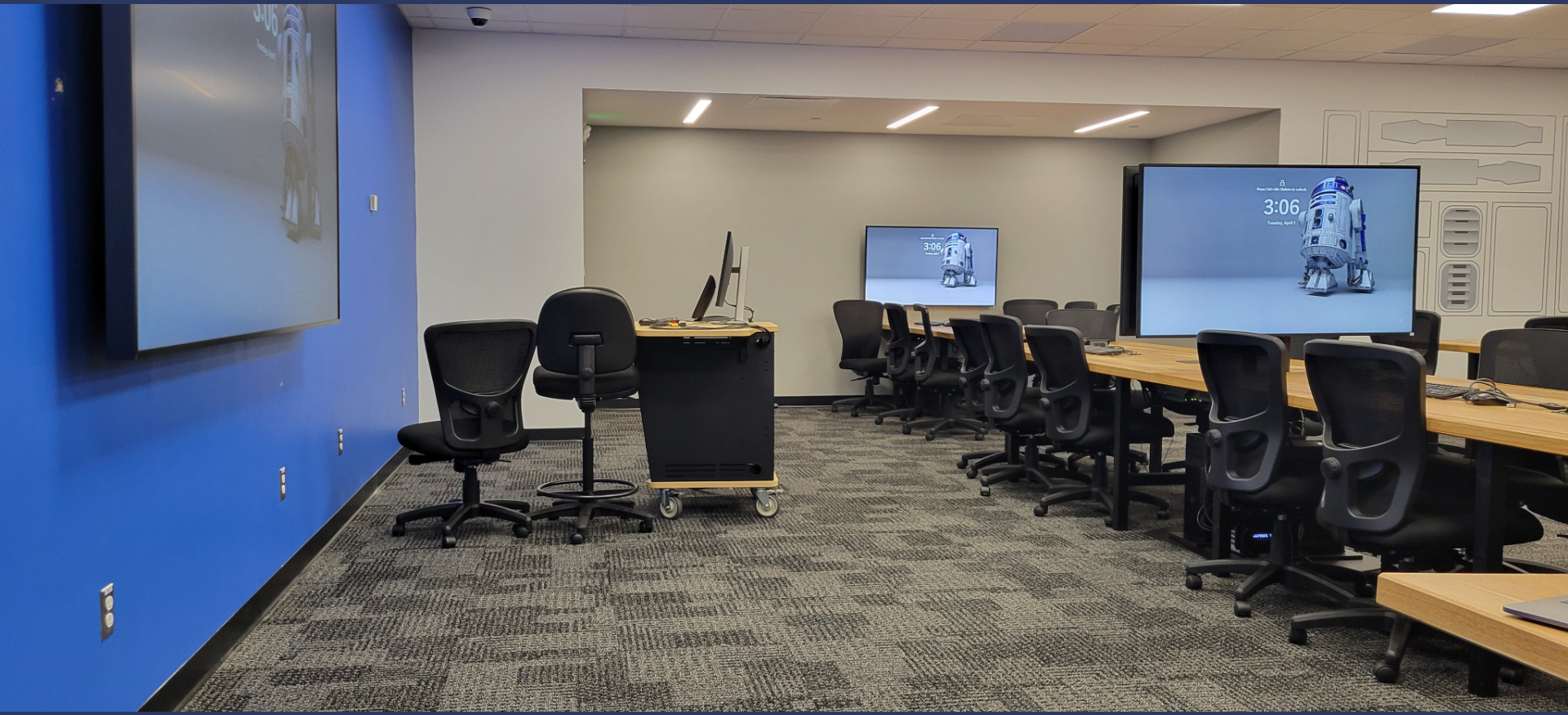
Marigold

Statistic Tables and Boundary Demonstrations [7]

# Depth Sensing Component

**Potential Issues and Solutions:**

1. Ignoring near-field non-obstacles (i.e. the floor)
    a. Use gradient vectors to identify the floor
    b. Limit detection to object dataset
2. Distinguishing obstacle from background (bounding boxes)
    a. Focus on box center
    b. Nearest pixel
    c. Masking with averaging
3. Viewing angle difficulties & initial orientation
    a. Wider camera
    b. Multi-scan mode




# Image of Our Classroom

# Normalized Depth Map of Our Classroom

# Emotion Detection

Introduction

- Emotion Detection models have existed for a while and are used in many current scenarios:
  - Driver monitoring systems
  - Psychology Studies
  - Human-computer interaction
  - Customer service Tracking
- Our Goal: We are trying to use these models and integrate them into a project that can aid a blind person interact better with humans as they navigate the world.

Current Models and datasets

- Traditional CNN models
  - FERNet, VGGNet, ResNet
    - These were popular research models and work on datasets like FER-2013
- Mobile/Edge-Optmized Models
  - MobileNet, EfficientNet or SqueezeNet
    - These models are more efficient and lightweight. Great for small embedded systems

# Comparing Emotion Models

FER images

Comparing Models and Takeaways.

- The two models I decided to train and compare are the efficientNet and a model that I trained based on the FER Dataset. Here are some key differences

| FER2013(Custom CNN Model) | EfficientNet(Pretrained model) |
|---|---|
| Dataset is mainly greyscale images (48X48px) | Model trained on coloured images(248X248px) |
| Shallow, custom CNN(2-4 layers) | Deep Model with compound scaling |
| Model is small, <5MB | 4 times bigger at 20MB |
| Accuracy is low due to age, 65-70 % | Accuracy is better due to modern tech, 75-85% |
| Good for greyscale webcam and close-range | Better for varied lighting, RGB and distances |

Training the custom model took a while, even on my RTX3060. Runtime was faster than efficientNet, however: 8 s instead of 26 s

# Emotion Detection Issues and Pivots

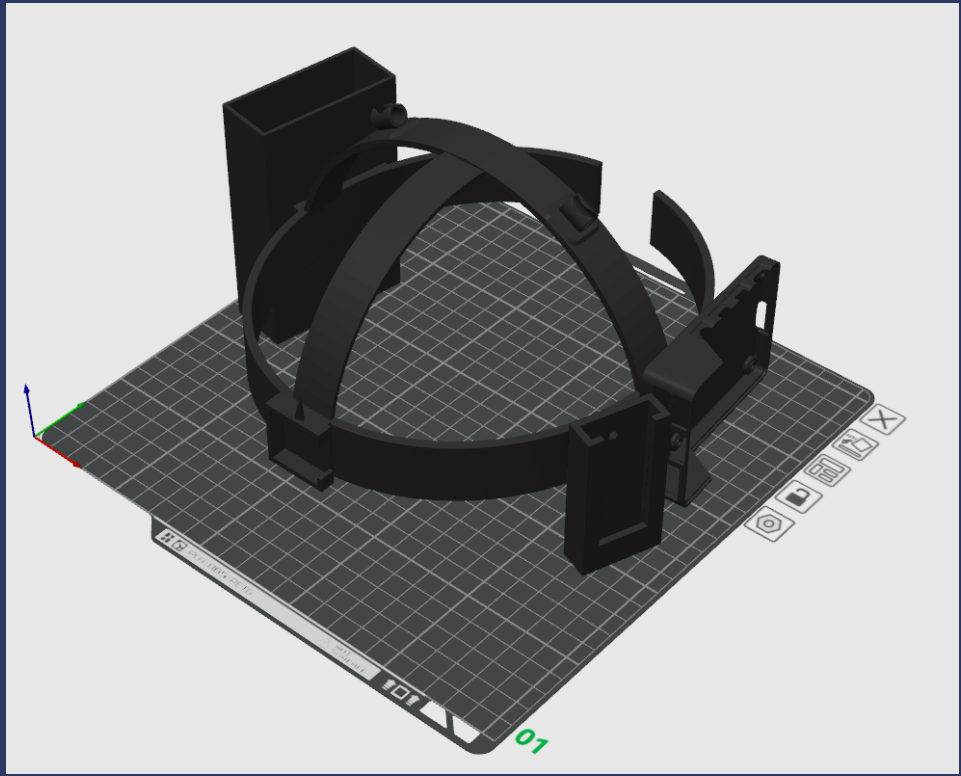
Custom FER2013 CNN Model

- This was my originally chosen model due to how light this was.
- This can be solely run on the raspberry pi 4 without a TPU
- Issues
  - Our Webcam footage is RGB and high resolution. This means we need to downscale to 48X48px which is very low considering the image might have a big FOV
  - The predicted emotions were not accurate, probably because the image didnt have a zoomed in enough photo

Finalized Model and necessary adjustments

- The final chosen model was efficientNet. Here are considerations:
  - trained better than the two layer CNN
  - Accepts RGB images that are 256X256px, meaning a zoomed out image still might give a good output
- Important hardware changes
  - The raspberry pi 4 now needed a TPU so it can run a heavier model as the same framerate. We chose to use a Coral TPU by google
- Important software changes.
  - Frames from video need to run a face detection model so it can be zoomed in before conversion to 256 pxs

# Proposed v1 Product

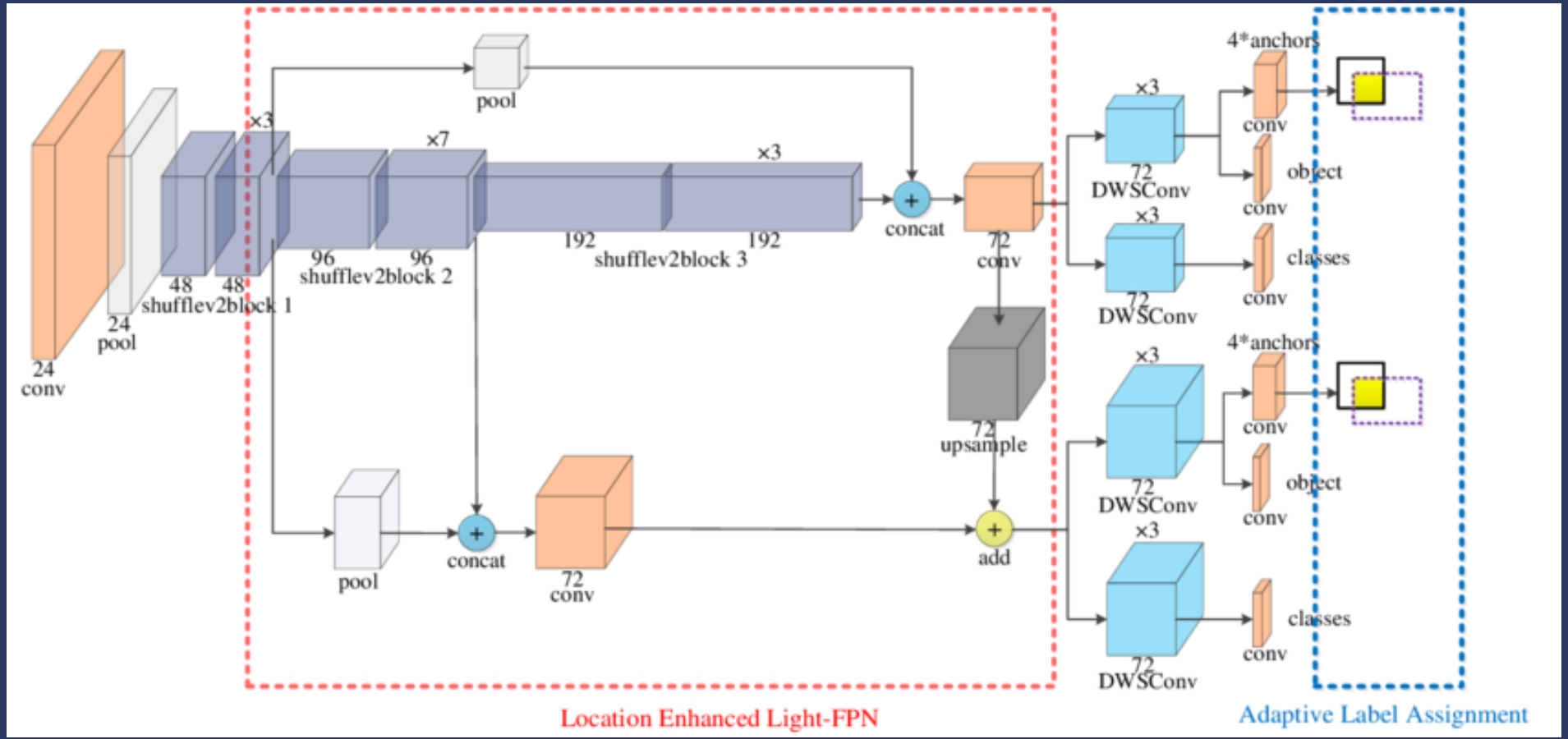


Components in the image

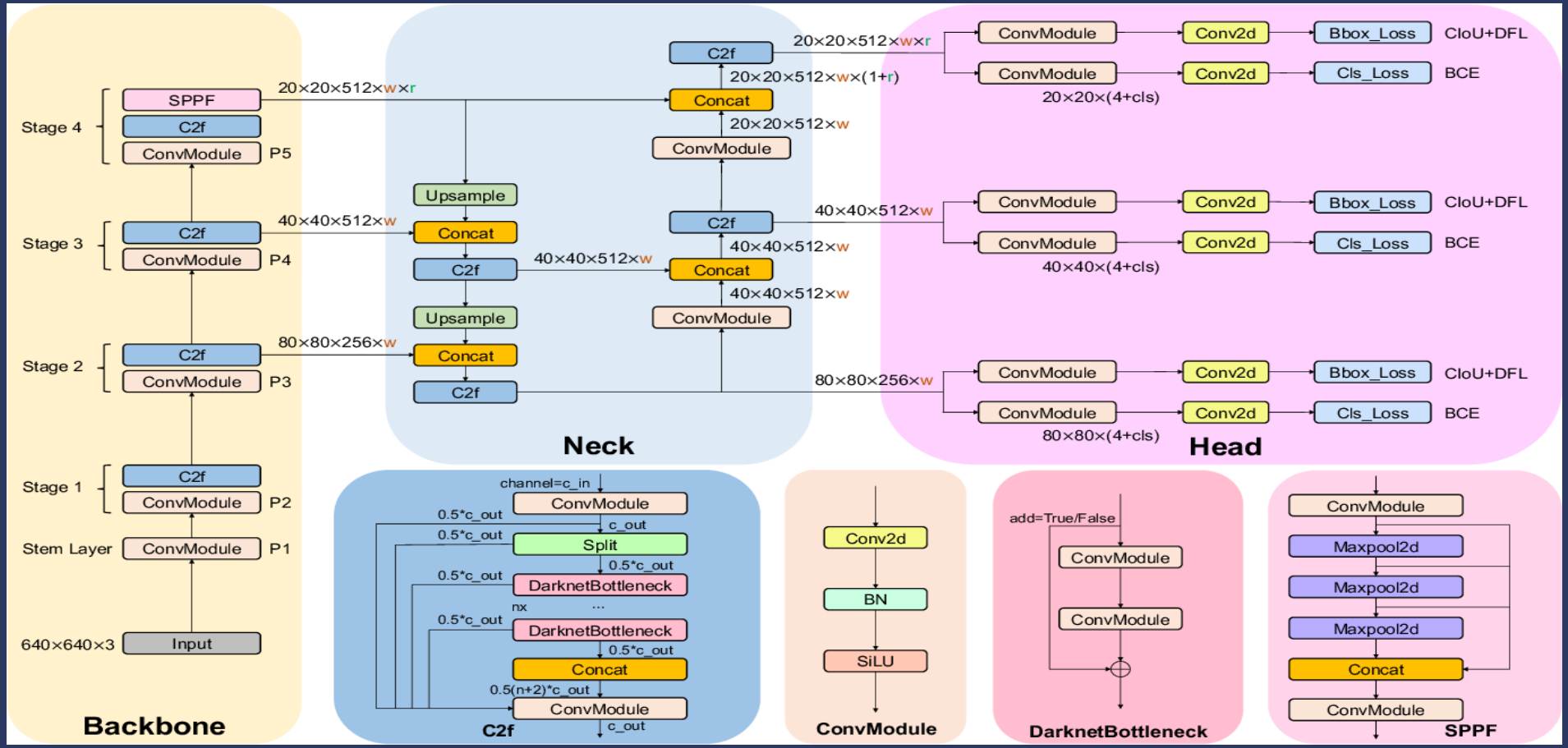
1. The big rectangular box on the far side of the image is a holder for a powerbank
2. The rectangular attachment which is further of the two on the close side of the screen is a holder for a raspberry pi
3. The closest attachment is a holder for the Coral TPU

# References

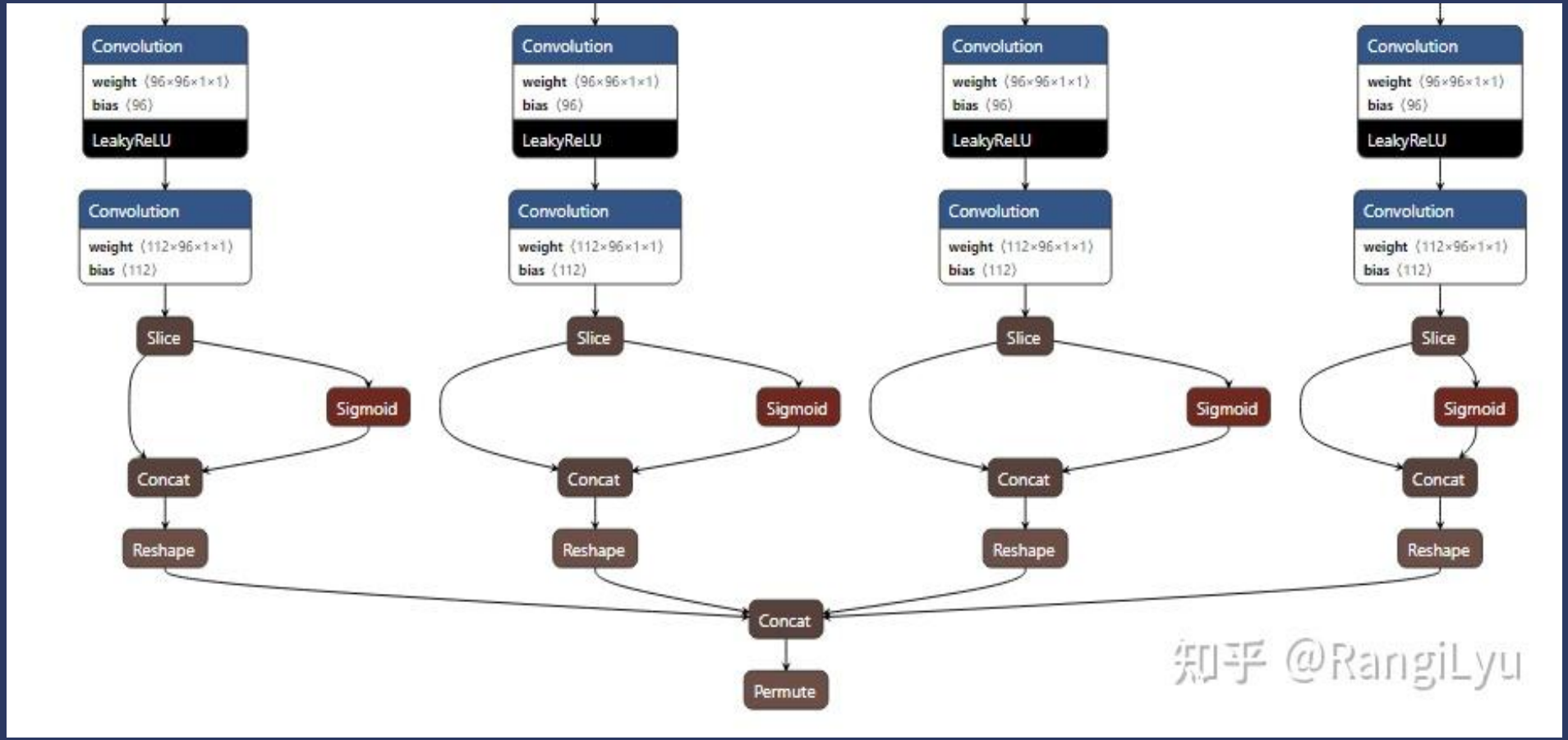

1. Messaoudi MD, Menelas BJ, Mcheick H. Review of Navigation Assistive Tools and Technologies for the Visually Impaired. Sensors (Basel). 2022 Oct 17;22(20):7888. doi: 10.3390/s22207888. PMID: 36298237; PMCID: PMC9606951.
2. Sunkara, Raja & Luo, Thomas. (2023). YOGA: Deep Object Detection in the Wild with Lightweight Feature Learning and Multiscale Attention. 10.48550/arXiv.2307.05945.
3. Zhang, Hairong & Xu, Dongsheng & Cheng, Dayu & Meng, Xiaoliang & Xu, Geng & Liu, Wei & Wang, Teng. (2023). An Improved Lightweight Yolo-Fastest V2 for Engineering Vehicle Recognition Fusing Location Enhancement and Adaptive Label Assignment. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. PP. 1-13. 10.1109/JSTARS.2023.3249216.
4. Ju, Rui-Yang & Cai, Weiming. (2023). Fracture detection in pediatric wrist trauma X-ray images using YOLOv8 algorithm. Scientific Reports. 13. 10.1038/s41598-023-47460-7.
5. RangiLyu, “NanoDet-Plus,” GitHub, Sep. 03, 2022. https://github.com/RangiLyu/nanodet
6. Qengineering, “GitHub - Qengineering/YoloX-ncnn-Jetson-Nano: YoloX for a Jetson Nano using ncnn.,” GitHub, 2021. https://github.com/Qengineering/YoloX-ncnn-Jetson-Nano?tab=readme-ov-file (accessed Apr. 07, 2025).
7. PatricioGonzalezVivo, “The State of the Art of Depth Estimation from Single Images,” Medium, Jan. 17, 2024. https://medium.com/@patriciogv/the-state-of-the-art-of-depth-estimation-from-single-images-9e245d51a315
8. A. Bochkovskii et al., “Depth Pro: Sharp Monocular Metric Depth in Less Than a Second,” arXiv.org, 2024. https://arxiv.org/abs/2410.02073
9. Jaykumaran, “Depth Pro Explained: Sharp, Fast Monocular Metric Depth Estimation,” LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow with code, & tutorials, Jan. 21, 2025. https://learnopencv.com/depth-pro-monocular-metric-depth/#depthpro-model-architecture (accessed Apr. 07, 2025).


24
conv
24
pool
48
48
shufflev2block 1
×3
96
96
shufflev2block 2
×7
192
192
shufflev2block 3
×3
pool
concat
72
conv
72
upsample
pool
concat
72
conv
add
×3
72
DWSConv
4*anchors
conv
object
conv
×3
72
DWSConv
conv
classes
conv
×3
72
DWSConv
4*anchors
conv
object
conv
×3
72
DWSConv
conv
classes
conv
Location Enhanced Light-FPN
Adaptive Label Assignment

[3]


Backbone
Stage 4
SPPF
C2f
ConvModule
P5
Stage 3
C2f
ConvModule
P4
Stage 2
C2f
ConvModule
P3
Stage 1
C2f
ConvModule
P2
Stem Layer
ConvModule
P1
640×640×3
Input
20×20×512×w×r
40×40×512×w
80×80×256×w
Neck
Upsample
Concat
C2f
Upsample
Concat
C2f
C2f
20×20×512×w×r
20×20×512×w×(1+r)
Concat
20×20×512×w
ConvModule
40×40×512×w
40×40×512×w
Concat
40×40×512×w
ConvModule
Head
ConvModule
Conv2d
Bbox_Loss
CIoU+DFL
ConvModule
Conv2d
Cls_Loss
BCE
20×20×(4+cls)
ConvModule
Conv2d
Bbox_Loss
CIoU+DFL
ConvModule
Conv2d
Cls_Loss
BCE
40×40×(4+cls)
ConvModule
Conv2d
Bbox_Loss
CIoU+DFL
ConvModule
Conv2d
Cls_Loss
BCE
80×80×(4+cls)
C2f
channel=c_in
ConvModule
0.5*c_out
c_out
0.5*c_out
Split
0.5*c_out
0.5*c_out
DarknetBottleneck
nx
...
0.5*c_out
DarknetBottleneck
0.5*c_out
Concat
0.5(n+2)*c_out
ConvModule
c_out
Conv2d
BN
SiLU
ConvModule
add=True/False
ConvModule
ConvModule
DarknetBottleneck
ConvModule
Maxpool2d
Maxpool2d
Maxpool2d
Concat
ConvModule
SPPF

[4]


Convolution
weight (96×96×1×1)
bias (96)
LeakyReLU
Convolution
weight (112×96×1×1)
bias (112)
Slice
Sigmoid
Concat
Reshape
Concat
Permute
知乎 @RangiLyu

[5]