Yolov5 Model Deployment on TI's Edge AI Platform for Efficient Object Detection

Jacinto™ AI monthly webinar series

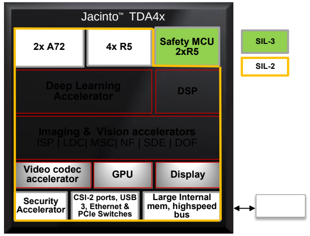
Dec 9th 2021

Webinar | Agenda

- Recap of TI’s Edge AI solution
- TI Model Zoo
- YOLOv5 model overview
- Optimizations for TI’s deep learning accelerator
- Compiling the model using open-source run time
- Model benchmarking on TI’s free cloud tool

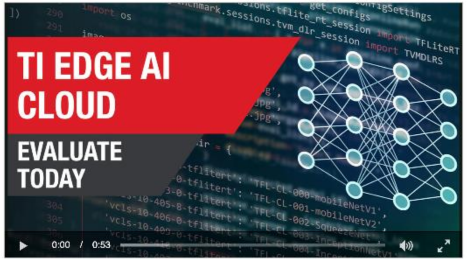
TI Edge AI | Revolutionizing applications from factories to home

Explore processors for edge AI



ti.com/edgeai

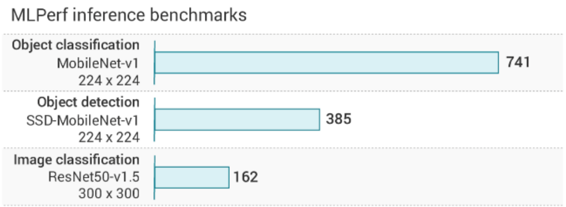
Learn with Free Cloud Tool



Build with 8 TOPS starter kit

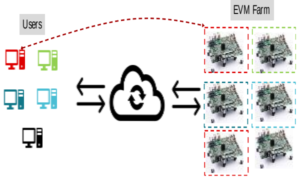
P/N: SK-TDA4VM: $199

Energy efficient AI architecture

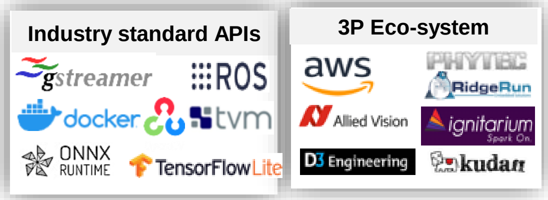


Get started for free

- Example scripts
- TI Model Zoo
- Training videos

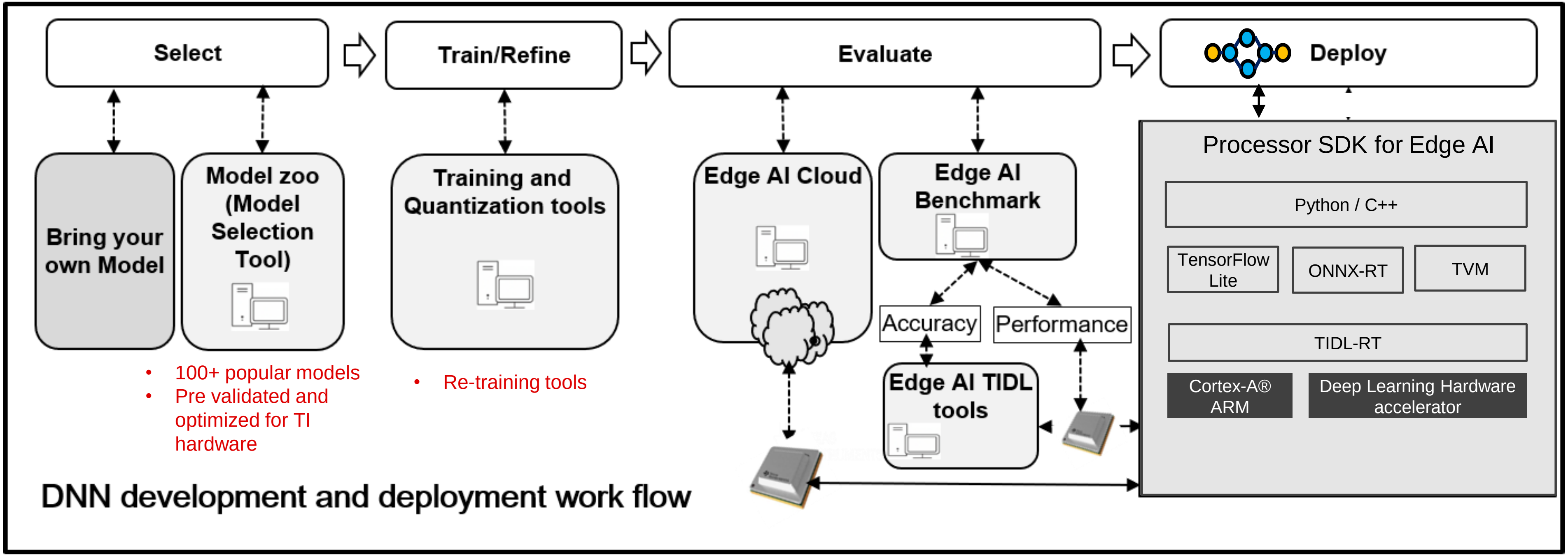


Fast Development Cycle



ti.com/edgeai for all the resources you have to get started!

Extensive tools for faster DL model development & deployment



Model Zoo | What is inside?

- Three primary Tasks:

1. Classification

Image → CNN f (x) → Single Image label (car, person,...)

2. Semantic Segmentation

Image → CNN f (x) → pixel label (car, person,...)

3. Object Detection

Image → CNN f (x) →

- Multiple ROI labels (car, person,...)
- BoundingBox locations $[(bx_1,by_1,H_1, W_1),...(bx_n,by_n,H_n, W_n)]$

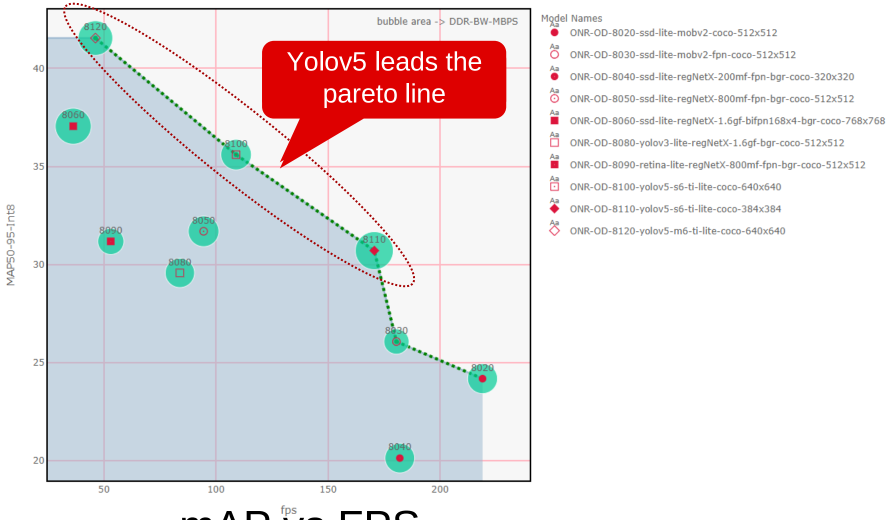
❑ All models are compiled, optimized and ready to be deployed!

❑ We have 60+ models to choose

Model Zoo | 2D Object Detection

Supported Object Detection Architectures:

- SSD
- RetinaNet
- EfficientDet
- YOLOv3
- YOLOv5
- YOLOX



mAP vs FPS

- Several example models from each family of detector are part of the modelzoo. These are pre-compiled models for easy evaluation.
- We host repositories to train these models on your own dataset as well.

YOLOv5 provides one of the best tradeoff in terms of performance and accuracy

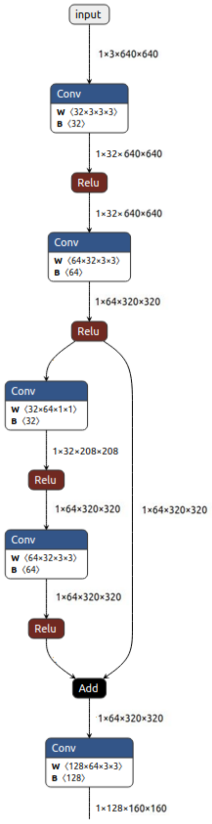
https://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/08_00_00_12/exports/docs/tidl_j7_08_00_00_10/ti_dl/docs/user_guide_html/md_tidl_fsg_meta_arch_support.html

Deep dive | YOLOv5

- Yolov5 built on top of YOLOv3 by Ultralytics with a quantum jump in accuracy.

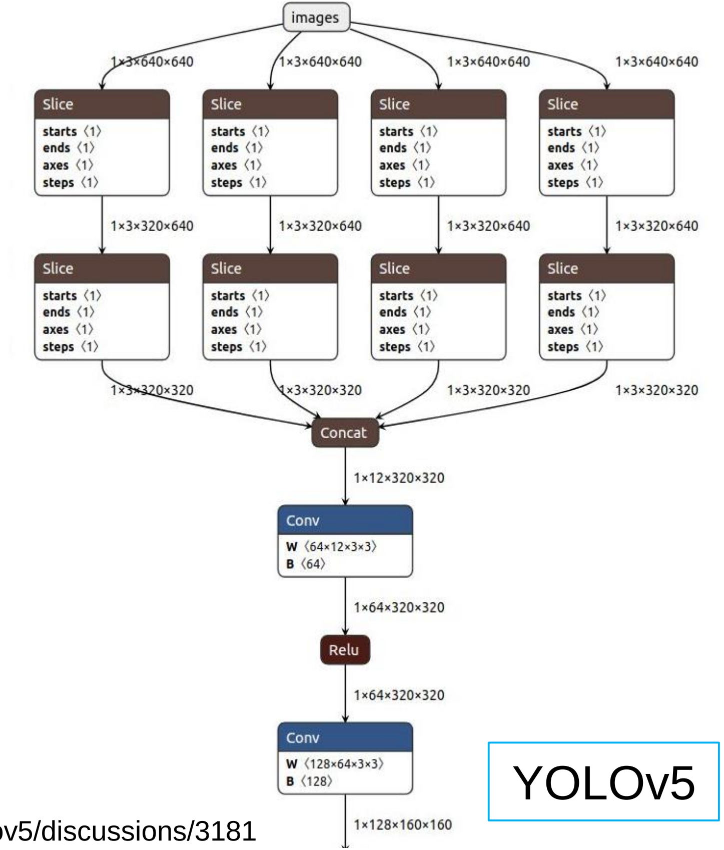
- Next, we look into the following major changes from YOLOv3 to YOLOv5:
  - Focus Layer
  - Backbone
  - Feature Fusion
  - Auto-anchor
  - Augmentation

YOLOv5 Architecture | Focus layer



- Reduces training time(~15%).
- Reduces complexity of the first few layer in YOLOV3 from 8.8GFLOPS to 1.4 GFLOPS in YOLOV5l (~7% saving of the entire n/w).

YOLOv3

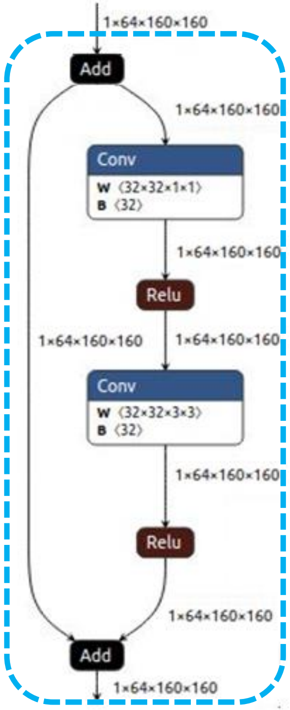


YOLOv5

Reference: https://github.com/ultralytics/yolov5/discussions/3181

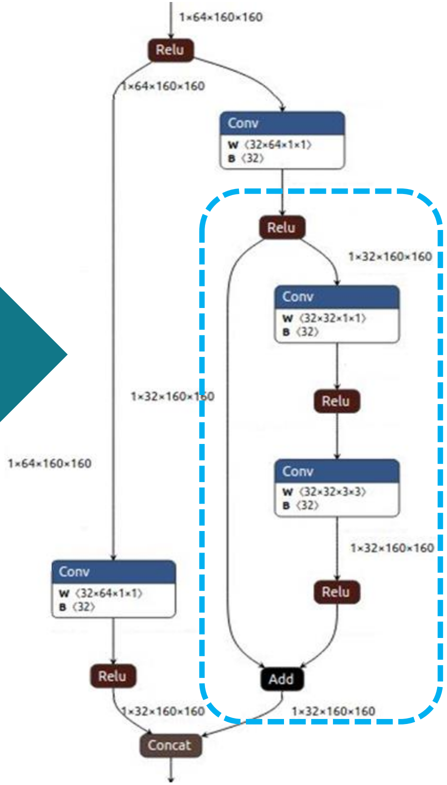
YOLOv5 Architecture | Backbone

- CSPDarknet53 is used as backbone instead of Darknet53. Results in 30% lesser FLOPS.



YOLOv3

Cross-Stage-Partial connection in YOLOV5. Number of channels for the residual block goes down by half .

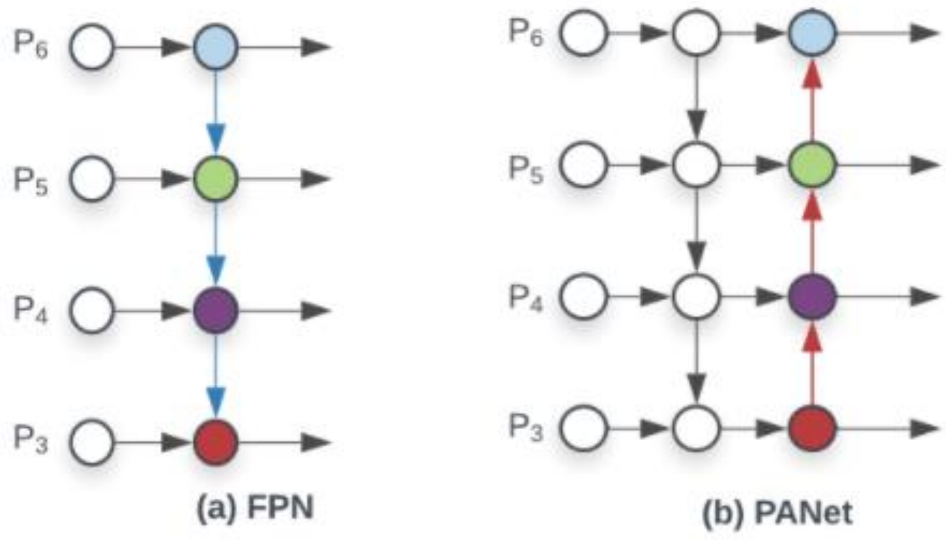


YOLOv5

Reference: https://github.com/WongKinYiu/CrossStagePartialNetworks

YOLOv5 Architecture | Feature fusion

- Feature fusion is used to fuse features from multiple spatial levels of a deep network**.**
- Path Aggregation Network **(**PANet) is used instead of Feature Pyramid Network (FPN) of YOLOv3.



YOLOv3

YOLOv5

Note: For input resolution of 640x640, P_3 represents feature level 3 with resolution $\frac{640}{2^3} = 80$

YOLOv5 Architecture | Other highlights

- **Auto anchor:** Learning anchor boxes based on the distribution of bounding boxes in the custom dataset with K-means and genetic learning algorithms. Eliminate need for hand-designed anchors.
- **Augmentation:** Mosaic augmentation among many others.
- No Image-net pre-trained weights. Trains from scratch.

YOLOv5 vs YOLOv3 | Complexity Reduction

Model	Complexity Info (GFLOPS)	mAP	Comments
YOLOv3	156	34.0	YOLOv3 complexity @ 640x640
YOLOv5L(Backbone)	-45		CSPDarknet53 reduces complexity by 30%
YOLOv5L(Focus Layer)	-7		
YOLOv5L(Feature Fusion)	+11		FPN vs PANet
YOLOv5L	(156-45-7+11) = 115	48.8	Overall complexity reduction is ~25%
YOLOv5m	51.3	45.2	depth_multiple = 0.67, width_multiple = 0.75
YOLOv5s	17.0	37.2	depth_multiple = 0.33, width_multiple = 0.5

YOLOv5L reduces complexity by ~25% and improves accuracy by ~43%

Defining YOLOv5-ti-lite | Optimizations for TI’s deep learning accelerator

- This is in a spirit to deriving Efficient-net-Lite from Efficient-net with the intention of making it embedded friendly.
  https://blog.tensorflow.org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite.html

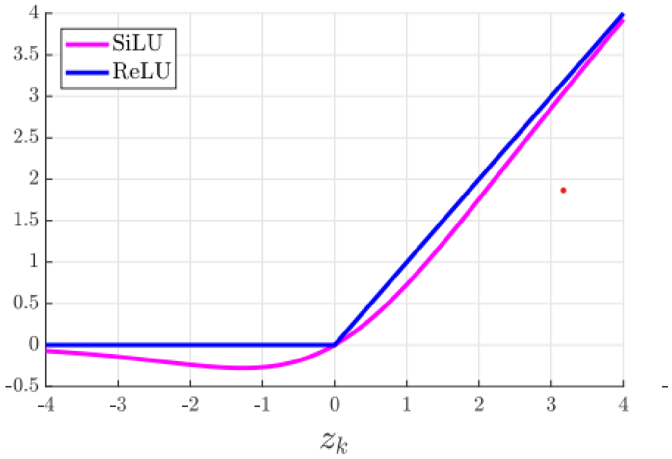
- These changes are focused on replacing an existing layers that **are not embedded or quantization friendly** with layers of similar functionality. E.g.
  - Replace activations like SiLU, hswish, leaky ReLU with ReLU.
  - Remove Squeeze and Excitation layers.
  - Perform down-sampling using Convolution or max-pool instead of spatial slicing.
  - Replace max-pool of higher receptive field with serially connected max-pool of lower receptive field.

- Apart from YOLOv5, TI model zoo has similar optimized model for MobileNetv3, EfficientNet, YOLOX, YOLOv3 and so on.

Refer https://github.com/TexasInstruments/edgeai-modelzoo for TI optimized models

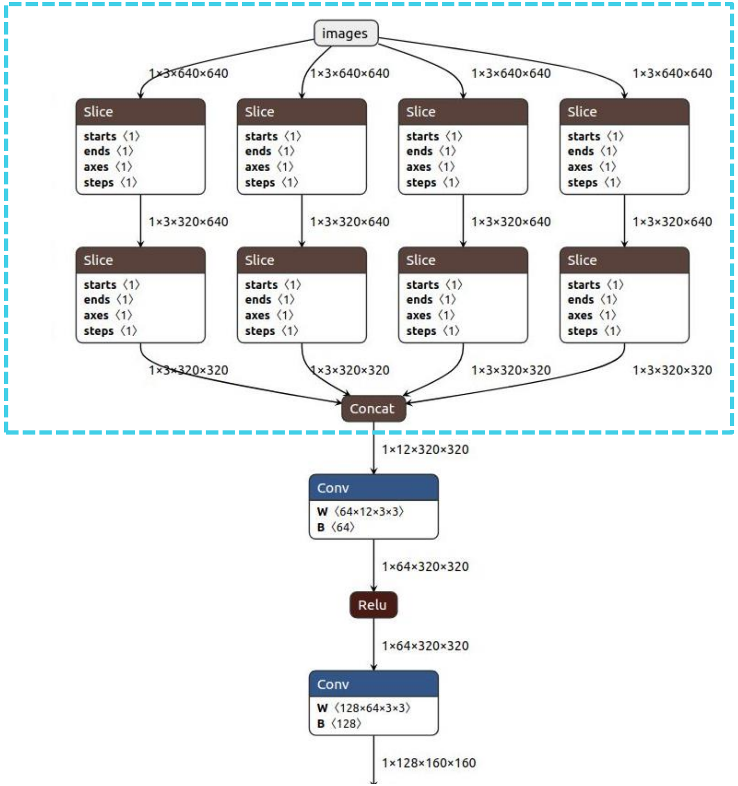
Yolov5-ti-lite definition | Activation, Image size

- SiLU is defined as **x∗sigmoid(x).** This non-linearity is not embedded friendly.
- Replaced SiLU non-linearity with ReLU for better acceleration.



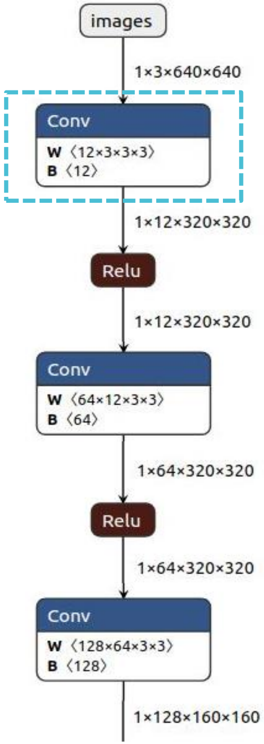
- Variable size inference was replaced with **fixed size inference**.

Yolov5-ti-lite definition | Focus layer



Replace Focus Layer with a light-weight embedded-friendly convolution.

Focus layer is a data manipulation layer and hence not efficient on a compute-focused accelerator.

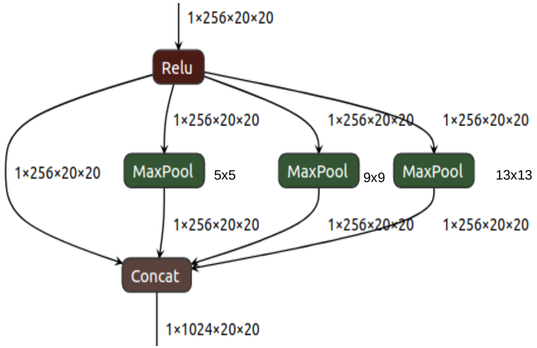


Yolov5-ti-lite definition | Spatial Pyramid Pooling

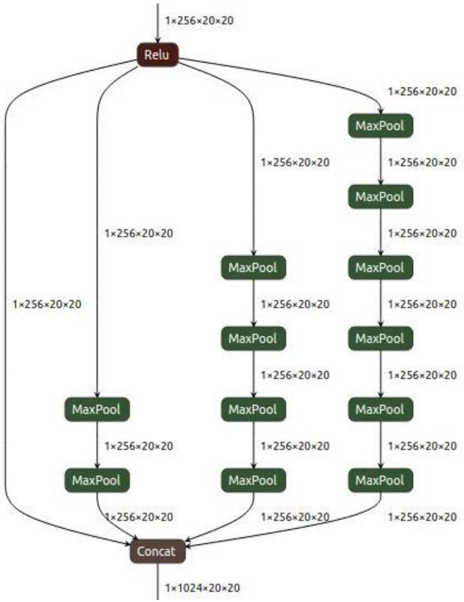
- Spatial Pyramid Pooling (SPP) is used by modern networks to increase the effective receptive field. In YOLOv5, this module is implemented using max-pool with large kernel (k=13,s=1)

- In YOLOv5-ti-lite, unsupported Max-pools inside SPP module are realized with TIDL supported max-pool layers.
  - E.g. max-pool(k=13,s=1) is replaced with six serially connected max-pool (k=3,s=1). They are **functionally same**.

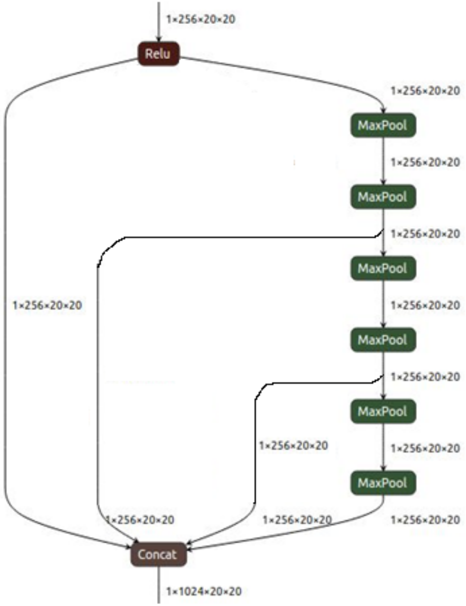
Yolov5-ti-lite definition | Spatial Pyramid Pooling



SPP module in YOLOV5.
Four parallel branches :
{
Original i/p,
K=5,s=1,
K=9,s=1,
K=13,s=1
}



Max-pools with larger receptive fields are implemented by max-pool with (k=3,s=1)

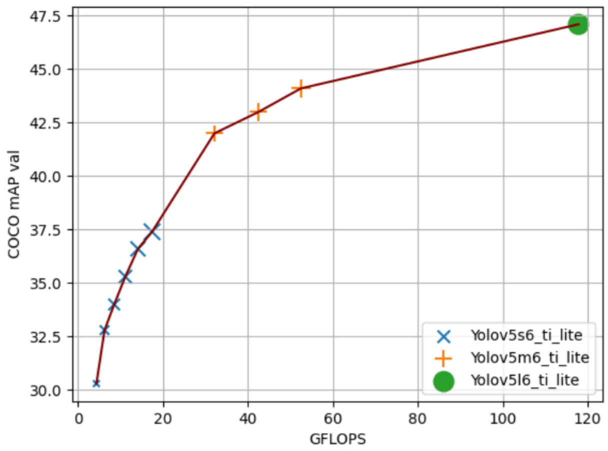


Further optimized version.

Model Training

Training | YOLOv5-ti-lite

- Forked from the official YOLOv5 repository, edgeai-yolov5 contains all the changes that we have described until now.
- Train a suitable YOLOv5-ti-lite model using this repo on your own dataset.
- Pretrained checkpoints are hosted to reproduce all the listed results.



Dataset	Model Name	Input Size	GFLOPS	AP[0.5:0.95]%	AP50%	Notes
COCO	Yolov5s6_ti_lite_640	640x640	**17.48**	37.4	56.0	
COCO	Yolov5s6_ti_lite_576	576x576	**14.16**	36.6	55.7	(Train@ 640, val@576)
COCO	Yolov5s6_ti_lite_512	512x512	**11.18**	35.3	54.3	(Train@ 640, val@512)
COCO	Yolov5s6_ti_lite_448	448x448	**8.56**	34.0	52.3	(Train@ 640, val@448)
COCO	Yolov5s6_ti_lite_384	384x384	**6.30**	32.8	51.2	(Train@ 384, val@384)
COCO	Yolov5s6_ti_lite_320	320x320	**4.38**	30.3	47.6	(Train@ 384, val@320)
COCO	Yolov5m6_ti_lite_640	640x640	**52.5**	44.1	62.9	
COCO	Yolov5m6_ti_lite_576	576x576	**42.52**	43.0	61.9	(Train@ 640, val@576)
COCO	Yolov5m6_ti_lite_512	512x512	**32.16**	42.0	60.5	(Train@ 640, val@512)
COCO	Yolov5l6_ti_lite_640	640x640	**117.84**	47.1	65.6	This model is fintuned from the official ckpt for 100 epochs

Yolov5s-ti-lite | Impact on Accuracy

Model config	PyTorch mAP/AP50
Official model	36.7/55.4
SiLU replaced with ReLU	34.9/53.7
ReLU + conv_focus (Definition of yolov5-ti-lite)	35.0/54.4

~2% drop occurs from ReLU to SiLU activation function.

ONNX Export

ONNX Export | YOLOv5-ti-lite

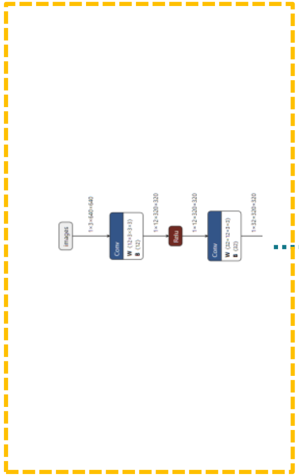
- In order to deploy the model in EVM, we need to export the entire model. This support is added in edgeai-yolov5 repository.

- Export the ONNX model by running the following command:

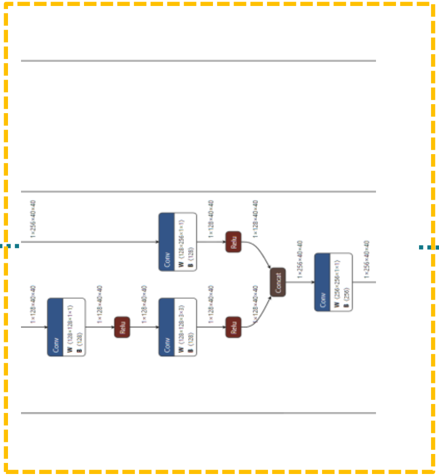
```
python export.py --weights   pretrained_models/yolov5s6_640_ti_lite/weights/best.pt --img 640 --batch 1 --simplify --export-nms --opset 11  # export at 640x640 with batch size 1
```

- This gives a complete ONNX model that can be offloaded fully in our DL accelerator.

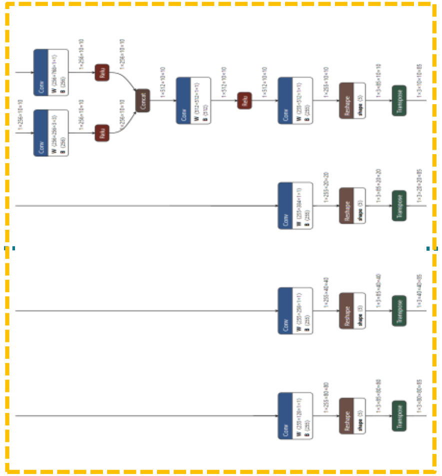
ONNX Export | YOLOv5-ti-lite



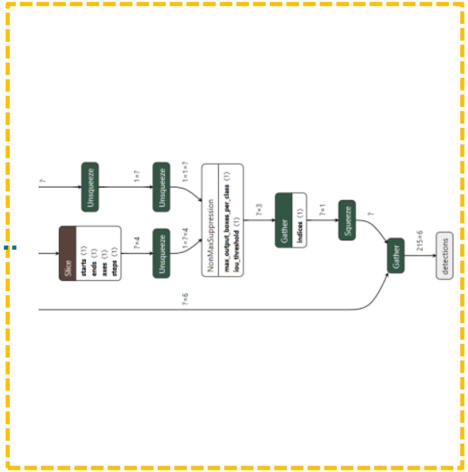
Encoder



Feature Fusion



Detection Heads



Detection Layer

For an OD model, detection layer consists of layers after the last convolution that includes box-decoding, confidence computation and NMS.

Model Compilation and Deployment

OD Model Compilation | TI Deep Learning

- Similar to classification and segmentation models, this model needs to be compiled before deployment.

- An important step of compiling an OD model is defining prototxt.

- Prototxt file contains all relevant information of the detection layer.

- Our training repo edgeai-yolov5 contains multiple sample prototxt for reference.
  - https://github.com/TexasInstruments/edgeai-yolov5/tree/master/pretrained_models/models

OD Model Compilation | Defining Prototxt

This section of the prototxt is same for YOLOv3, YOLOv5 and YOLOX

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 640
  in_height: 640
 # 9,11,  21,19,  17,41,  43,32,  39,70,  86,64,  65,131,  134,130,  120,265,  282,180,
  yolo_param {
    input: "370"
    anchor_width: 9
    anchor_width: 21
    anchor_width: 17
    anchor_height: 11
    anchor_height: 19
    anchor_height: 41
  }
  yolo_param {
    input: "680"
    anchor_width: 43
    anchor_width: 39
    anchor_width: 86
    anchor_height: 32
    anchor_height: 70
    anchor_height: 64
  }
  yolo_param {
    input: "990"
    anchor_width: 65
    anchor_width: 134
    anchor_width: 120
    anchor_height: 131
    anchor_height: 130
    anchor_height: 265
  }
  yolo_param {
    input: "1300"
    anchor_width: 282
    anchor_width: 247
    anchor_width: 512
    anchor_height: 180
    anchor_height: 354
    anchor_height: 387
  }
  detection_output_param {
    num_classes: 80
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
      top_k: 200
    }
    code_type: CODE_TYPE_YOLO_V5
    keep_top_k: 200
    confidence_threshold: 0.3
  }
  output: "detections"
}
```

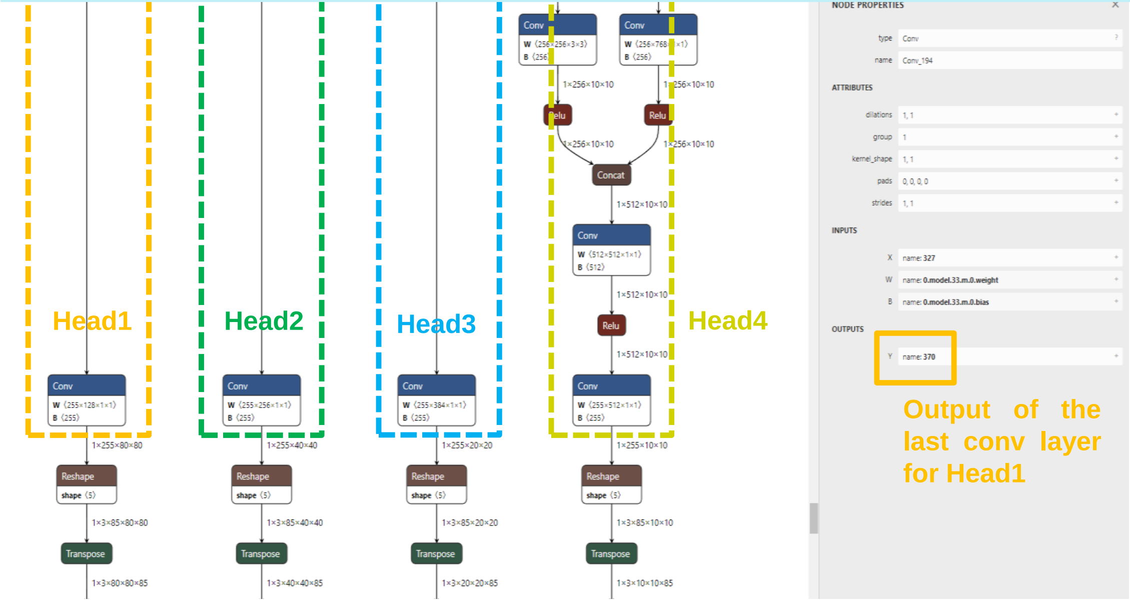
Head1

Head2

Head3

Head4

Prototxt file for YOLOv5s6 at input resolution of 640.



“input” in yolo_param will change across model and even depending on the Pytorch or ONNX version

OD Model Compilation | Updating Anchor Dimensions

```
yolo_param {
    input: "370"
    anchor_width: 9
    anchor_width: 21
    anchor_width: 18
    anchor_height: 12
    anchor_height: 18
    anchor_height: 41
 }
```

Head1

```
yolo_param {
    input: "680"
    anchor_width: 44
    anchor_width: 37
    anchor_width: 84
    anchor_height: 32
    anchor_height: 71
    anchor_height: 61
 }
```

Head2

```
yolo_param {
    input: "990"
    anchor_width: 59
    anchor_width: 124
    anchor_width: 122
    anchor_height: 138
    anchor_height: 111
    anchor_height: 237
 }
```

Head3

```
yolo_param {
    input: "1300"
    anchor_width: 278
    anchor_width: 237
    anchor_width: 473
    anchor_height: 170
    anchor_height: 360
    anchor_height: 406
 }
```

Head4

- Each head contains information of anchor dimensions for that head.
- In YOLOv5, anchor dimensions are optimized for “Best Possible Recall” based on dataset and input resolution.
- Anchor dimensions need to be updated for any change in dataset or resolution .
- Next, we discuss several example prototxt.
- Given below is a snippet of anchor evolution in YOLOv5.

```
autoanchor: Analyzing anchors... anchors/target = 4.77, Best Possible Recall (BPR) = 0.9685. Attempting to improve anchors, please wait...
autoanchor: WARNING: Extremely small objects found. 3910 of 849942 labels are < 3 pixels in size.
autoanchor: Running kmeans for 12 anchors on 849898 points...
autoanchor: thr=0.25: 0.9867 best possible recall, 5.17 anchors past thr
autoanchor: n=12, img_size=640, metric_all=0.266/0.684-mean/best, past_thr=0.467-mean: 15,18,  37,37,  41,79,  98,57,  77,138,  179,107,
119,230,  283,195,  195,355,  486,222,  342,482,  584,421
autoanchor: Evolving anchors with Genetic Algorithm: fitness = 0.7164: 100%|                    | 1000/1000 [03:30<00:00,  4.75it/s]
autoanchor: thr=0.25: 0.9963 best possible recall, 5.58 anchors past thr
autoanchor: n=12, img_size=640, metric_all=0.282/0.717-mean/best, past_thr=0.468-mean: 9,12,  21,18,  18,41,  44,32,  37,71,  84,61,  59,1
38,  124,111,  122,237,  278,170,  237,360,  473,406
autoanchor: New anchors saved to model. Update model *.yaml to use these anchors in the future.
```

OD Model compilation | Defining Prototxt

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 640
  in_height: 640
 # 9,11,  21,19,  17,41,  43,32,  39,70,  86,64,  65,131,  134,130,  120,265,  282,180,  247,354,  512,387
 yolo_param {
    input: "370"
    anchor_width: 9
    anchor_width: 21
    anchor_width: 17
    anchor_height: 11
    anchor_height: 19
    anchor_height: 41
  }
 yolo_param {
    input: "680"
    anchor_width: 43
    anchor_width: 39
    anchor_width: 86
    anchor_height: 32
    anchor_height: 70
    anchor_height: 64
  }
 yolo_param {
    input: "990"
    anchor_width: 65
    anchor_width: 134
    anchor_width: 120
    anchor_height: 131
    anchor_height: 130
    anchor_height: 265
  }
 yolo_param {
    input: "1300"
    anchor_width: 282
    anchor_width: 247
    anchor_width: 512
    anchor_height: 180
    anchor_height: 354
    anchor_height: 387
  }
  detection_output_param {
    num_classes: 80
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
      top_k: 200
    }
    code_type: CODE_TYPE_YOLO_V5
    keep_top_k: 200
    confidence_threshold: 0.3
  }
  output: "detections"
}
```

- Information related to NMS and box decoding are stored under s detection_output_param.
- These parameters must be matched against the training framework for expected outcome.
- **nms_threshold:** Determines the overlap beyond which boxes are ignored
- **top_k :** Number of boxes that goes for NMS.
- **keep_top_k:** Maximum number of boxes that are retained after NMS.
- **confidence_threshold:** Boxes with confidence greater than this threshold are only retained.
- For accuracy reproduction, confidence_threshold is set to a low value (E.g. 0.01)
- For all practical purposes, confidence_threshold can be set to a high value like 0.3.

Prototxt Example | Input resolutions

- Dataset and the model definition are same.
- “Auto-anchor” algorithm tunes the anchor dimensions to the input resolution.

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 640
  in_height: 640
 # 9,11,  21,19,  17,41,  43,32,  39,70,  86,64,  65,131,  134,130,  120,265,  282,180,  247,354,  512,387
 yolo_param {
    input: "370"
    anchor_width: 9
    anchor_width: 21
    anchor_width: 17
    anchor_height: 11
    anchor_height: 19
    anchor_height: 41
  }
 yolo_param {
    input: "680"
    anchor_width: 43
    anchor_width: 39
    anchor_width: 86
    anchor_height: 32
    anchor_height: 70
    anchor_height: 64
  }
 yolo_param {
    input: "990"
    anchor_width: 65
    anchor_width: 134
    anchor_width: 120
    anchor_height: 131
    anchor_height: 130
    anchor_height: 265
  }
 yolo_param {
    input: "1300"
    anchor_width: 282
    anchor_width: 247
    anchor_width: 512
    anchor_height: 180
    anchor_height: 354
    anchor_height: 387
  }
  detection_output_param {
    num_classes: 80
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
      top_k: 200
    }
    code_type: CODE_TYPE_YOLO_V5
```

YOLOv5s6_640

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 384
  in_height: 384
 # 5,7,  12,11,  10,24,  25,19,  21,40,  46,32,  33,74,  69,61,  70,119,  147,96,  126,196,  292,224
 yolo_param {
    input: "370"
    anchor_width: 5
    anchor_width: 12
    anchor_width: 10
    anchor_height: 7
    anchor_height: 11
    anchor_height: 24
  }
 yolo_param {
    input: "680"
    anchor_width: 25
    anchor_width: 21
    anchor_width: 46
    anchor_height: 19
    anchor_height: 40
    anchor_height: 32
  }
 yolo_param {
    input: "990"
    anchor_width: 33
    anchor_width: 69
    anchor_width: 70
    anchor_height: 74
    anchor_height: 61
    anchor_height: 119
  }
 yolo_param {
    input: "1300"
    anchor_width: 147
    anchor_width: 126
    anchor_width: 292
    anchor_height: 96
    anchor_height: 196
    anchor_height: 224
  }
  detection_output_param {
    num_classes: 80
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
      top_k: 200
    }
    code_type: CODE_TYPE_YOLO_V5
```

YOLOv5s6_384

Prototxt comparison | Different models

- Anchor dimensions remain same for the same dataset and input resolution.
- Since the model definition has changed, “input” to yolo_param has to be updated as shown below.

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 640
  in_height: 640
 # 9,11,  21,19,  17,41,  43,32,  39,70,  86,64,  65,131,  134,130,  120,265,  282,180,  247,354,  512,387
 yolo_param {
    input: "498"
    anchor_width: 9
    anchor_width: 21
    anchor_width: 17
    anchor_height: 11
    anchor_height: 19
    anchor_height: 41
  }
 yolo_param {
    input: "808"
    anchor_width: 43
    anchor_width: 39
    anchor_width: 86
    anchor_height: 32
    anchor_height: 70
    anchor_height: 64
  }
 yolo_param {
    input: "1118"
    anchor_width: 65
    anchor_width: 134
    anchor_width: 120
    anchor_height: 131
    anchor_height: 130
    anchor_height: 265
  }
 yolo_param {
    input: "1428"
    anchor_width: 282
    anchor_width: 247
    anchor_width: 512
    anchor_height: 180
    anchor_height: 354
    anchor_height: 387
  }

  detection_output_param {
    num_classes: 80
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
```

YOLOv5s6_640

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 640
  in_height: 640
 # 9,11,  21,19,  17,41,  43,32,  39,70,  86,64,  65,131,  134,130,  120,265,  282,180,  247,354,  512,387
 yolo_param {
    input: "370"
    anchor_width: 9
    anchor_width: 21
    anchor_width: 17
    anchor_height: 11
    anchor_height: 19
    anchor_height: 41
  }
 yolo_param {
    input: "680"
    anchor_width: 43
    anchor_width: 39
    anchor_width: 86
    anchor_height: 32
    anchor_height: 70
    anchor_height: 64
  }
 yolo_param {
    input: "990"
    anchor_width: 65
    anchor_width: 134
    anchor_width: 120
    anchor_height: 131
    anchor_height: 130
    anchor_height: 265
  }
 yolo_param {
    input: "1300"
    anchor_width: 282
    anchor_width: 247
    anchor_width: 512
    anchor_height: 180
    anchor_height: 354
    anchor_height: 387
  }

  detection_output_param {
    num_classes: 80
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
```

YOLOv5m6_640

Prototxt Example | Across dataset

- "Auto-anchor" algorithm tunes the anchor dimensions for different dataset.
- Wider-face prevalently has smaller objects than COCO. Hence, resultant anchor dimensions are much smaller.
- "num_classes " is set to 1.

COCO

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 640
  in_height: 640
 yolo_param {
    input: "370"
    anchor_width: 9
    anchor_width: 21
    anchor_width: 17
    anchor_height: 11
    anchor_height: 19
    anchor_height: 41
  }
 yolo_param {
    input: "680"
    anchor_width: 43
    anchor_width: 39
    anchor_width: 86
    anchor_height: 32
    anchor_height: 70
    anchor_height: 64
  }
 yolo_param {
    input: "990"
    anchor_width: 65
    anchor_width: 134
    anchor_width: 120
    anchor_height: 131
    anchor_height: 130
    anchor_height: 265
  }
 yolo_param {
    input: "1300"
    anchor_width: 282
    anchor_width: 247
    anchor_width: 512
    anchor_height: 180
    anchor_height: 354
    anchor_height: 387
  }

  detection_output_param {
    num_classes: 80
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
      top_k: 200
```

Wider-face

```
name: "yolo_v3"
tidl_yolo {
  name: "yolo_v3"
  in_width: 640
  in_height: 640
 yolo_param {
    input: "370"
    anchor_width: 4
    anchor_width: 6
    anchor_width: 8
    anchor_height: 5
    anchor_height: 7
    anchor_height: 10
  }
 yolo_param {
    input: "432"
    anchor_width: 11
    anchor_width: 15
    anchor_width: 22
    anchor_height: 14
    anchor_height: 19
    anchor_height: 26
  }
 yolo_param {
    input: "494"
    anchor_width: 30
    anchor_width: 42
    anchor_width: 63
    anchor_height: 38
    anchor_height: 58
    anchor_height: 89
  }
 yolo_param {
    input: "556"
    anchor_width: 111
    anchor_width: 167
    anchor_width: 276
    anchor_height: 134
    anchor_height: 222
    anchor_height: 335
  }

  detection_output_param {
    num_classes: 1
    share_location: true
    background_label_id: -1
    nms_param {
      nms_threshold: 0.65
      top_k: 200
```

Model Compilation | Quantization

- Floating-point inference are not cost and power-efficient so we need to quantize the model to Fixed-point.
- Fixed point 8-bit quantization results in reduction in accuracy.
- We explore various options to bridge the gap between float and fixed point model close to 1%.
- Given below are the quantization options used for YOLOv5.

Name	Description	Default values
tensor_bits	Number of bits for TIDL tensor and weights - 8/16	8
accuracy_level	0 - basic calibration, 1 - higher accuracy(advanced bias calibration), 9 - user defined [^3]	1
advanced_options:calibration_frames	Number of frames to be used for calibration - min 10 frames recommended	50
advanced_options:calibration_iterations	Number of bias calibration iterations	50
advanced_options:output_feature_16bit_names_list	List of names of the layers (comma separated string) as in the original model whose feature/activation output user wants to be in 16 bit	“last conv layers, first conv layer“

YOLOv5s6-ti-lite | Advanced calibration

```
compile_options = {
    'tidl_tools_path' : os.environ['TIDL_TOOLS_PATH'],
    'artifacts_folder' : output_dir,
    'tensor_bits' : 8,
    'accuracy_level' : 1,
    'advanced_options:calibration_frames' : len(calib_images),
    'advanced_options:calibration_iterations' : 50 # used if accuracy_level = 1
    'advanced_options:output_feature_16bit_names_list': '370, 680, 990, 1300',
    'object_detection:meta_arch_type': 6,
    'object_detection:meta_layers_names_list': f'./yolov5s6_640_ti_lite_metaarch.prototxt',
}
```

Mixed precision Quantization: Precision for the last conv layers of the model are set to 16 bits.

OD specific compilation options

Model config	Tidl-f32 mAP/AP50	Tidl-q16 mAP/AP50	Tidl-q8 mAP/AP50	Tidl-Quant mAP/AP50 (first layers in 16 bits)	Tidl-Quant mAP/AP50 (last layers in 16 bits)	Tidl-Quant mAP/AP50 (first and last layers in 16 bits)
Yolov5s6_ti_lite_640	37.1/56.2	37.1/56.2 (23.52mS)	33.1/53.9 (8.55mS)	31.3/52.8 (9.23mS)	36.0/55.5 (9.13mS)	36.0/55.5 (9.82mS)

- ❑ With mixed precision, we are able to bridge the gap with floating point to ~ 1%.
- ❑ All YOLOv5 models perform well with only the last layer in 16 bit or both the first and last layer in 16 bit.

YOLOv5s6-ti-lite | Edgeai-benchmark

- ❑ Another Easy and preferred way of compiling a model is to use “edgeai-benchmark”.
- ❑ All required pre-processing and post-processing to reproduce the reported accuracy has been taken care of.
- ❑ This repo can be used to compile a model and then use it for deployment.
- ❑ Given below is the config for YOLOv5s6-640-ti-lite:

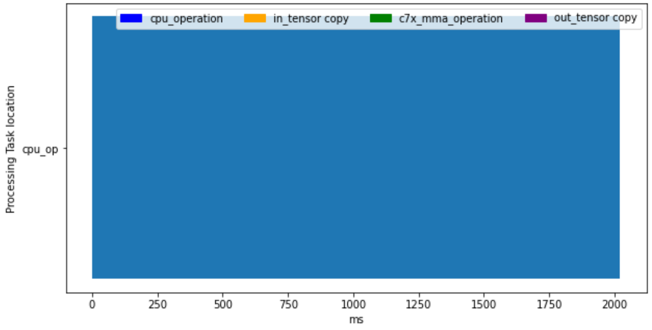
```
'od-8100':utils.dict_update(common_cfg,
    preprocess=preproc_transforms.get_transform_onnx(640, 640,  resize_with_pad=True, mean=(0.0, 0.0, 0.0), scale=(0.003921568627, 0.003921568627, 0.003921568627), backend='cv2', pad_color=[114,114,114]),
    session=onnx_session_type(**common_session_cfg,
        runtime_options=utils.dict_update(settings.runtime_options_onnx_np2(),
                            {'object_detection:meta_arch_type': 6,
                             'object_detection:meta_layers_names_list':f'../edgeai-yolov5/pretrained_models/models/yolov5s6_640_ti_lite/weights/yolov5s6_640_ti_lite_metaarch.prototxt',
                             'advanced_options:output_feature_16bit_names_list':'370, 680, 990, 1300'
                             }),
        model_path=f'../edgeai-yolov5/pretrained_models/models/yolov5s6_640_ti_lite/weights/yolov5s6_640_ti_lite_37p4_56p0.onnx'),
    postprocess=postproc_transforms.get_transform_detection_yolov5_onnx(squeeze_axis=None, normalized_detections=False, resize_with_pad=True, formatter=postprocess.DetectionBoxSL2BoxLS()), #TODO: check this
    metric=dict(label_offset_pred=datasets.coco_det_label_offset_80to90(label_offset=1)),
    model_info=dict(metric_reference={'accuracy_ap[.5:.95]%':37.4})
),
```


https://github.com/TexasInstruments/edgeai-benchmark/blob/master/configs/detection.py

YOLOv5 | Latency estimation

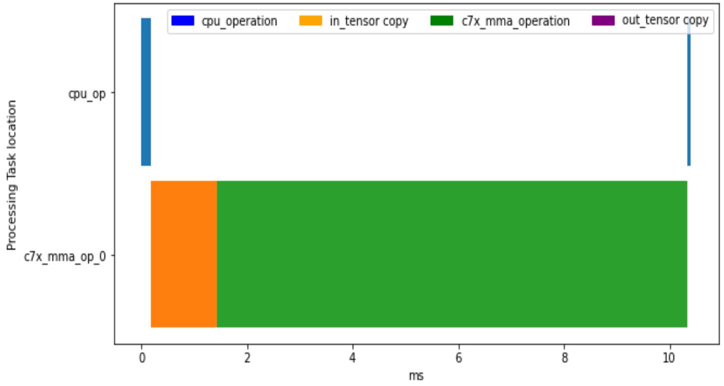
- We try to estimate latency of the original YOLOv5s6 model without any optimization.
- Estimate shows:
  - Focus Layer (ARM): 7.6mS
  - SPP module (ARM): 4.1mS
  - SiLU Activation (Estimate): 4.2 mS
  - Rest of the model(Accelerated): 8mS
  - Total latency: (7.6+4.1+4.2+8) = 23.9mS
  - FPS : (1000/23.9) = 41.84
  - Our optimized model is ~**2.6x faster** with negligible drop in accuracy.

YOLOv5-ti-lite | Cloud Evaluation




```
Statistics :
 Inferences Per Second    :    0.49 fps
 Inference Time Per Image : 2020.71 ms
 DDR BW Per Image         :    0.00 MB
```

- ❑ YOLOv5s6-ti-lite_640
- ❑ 0.49 fps AI processing (ONLY ARM)




```
SoC: J721E/DRA829/TDA4VM
 OPP:
  Cortex-A72 @2GHZ
  DSP C7x-MMA @1GHZ
  DDR @4266 MT/s

ONR-OD-8100-yolov5-s6-ti-lite-coco-640x640 :
 Inferences Per Second    :  109.59 fps
 Inference Time Per Image :    9.13 ms
```

- ❑ YOLOv5s6-ti-lite_640
- ❑ 109.6 fps AI processing (TIDL acceleration)

CPU is involved only in the beginning for small amount of time (Blue)

Conclusion

- YOLOv5-ti-lite is an optimized version of YOLOv5 that can be fully accelerated in our hardware accelerator.

- These models are part of model-zoo
  - Ready for deployment in **TI Edge AI** cloud tool and the Starter-kit EVM.

- Our training repository **"edgeai-yolov5"** can be used to train an optimized model on your own dataset.
  - Models trained here can be easily compiled and deployed.

- **"edgeai-benchmark"** can be used to compile and benchmark your trained model.

Call to action

- You can use Edge AI Cloud tool today to run the examples that we showed in this webinar

- Future topics
  - Several topics are planned: AIOT with AWS, AI-BOX, AI based 3D Lidar processing for edge AI and Robotics, Pose estimation and Robotics
  - Let us know any specific topics are of interest

- Contact TI for support (e2e.ti.com)
  - Please also let us know any specific topics you want us to cover in the future webinars

ti.com/edgeai