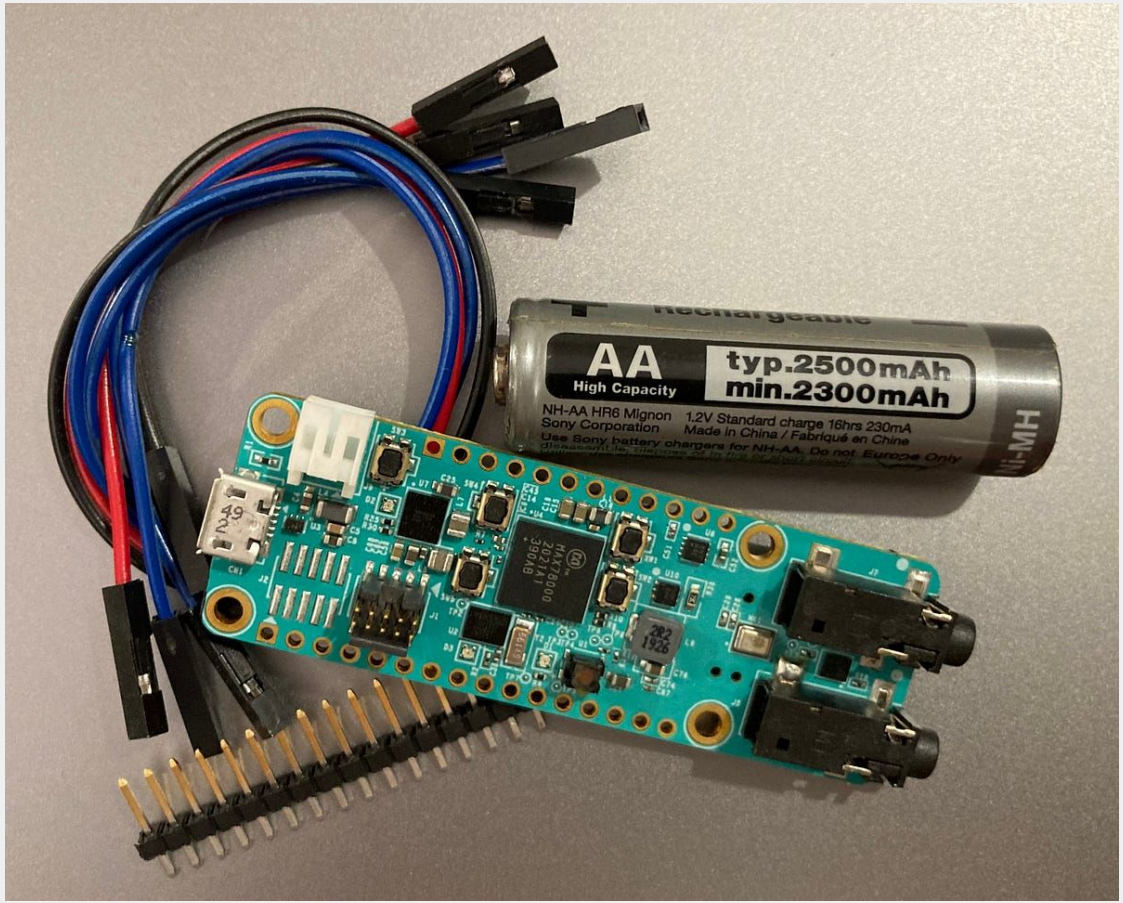
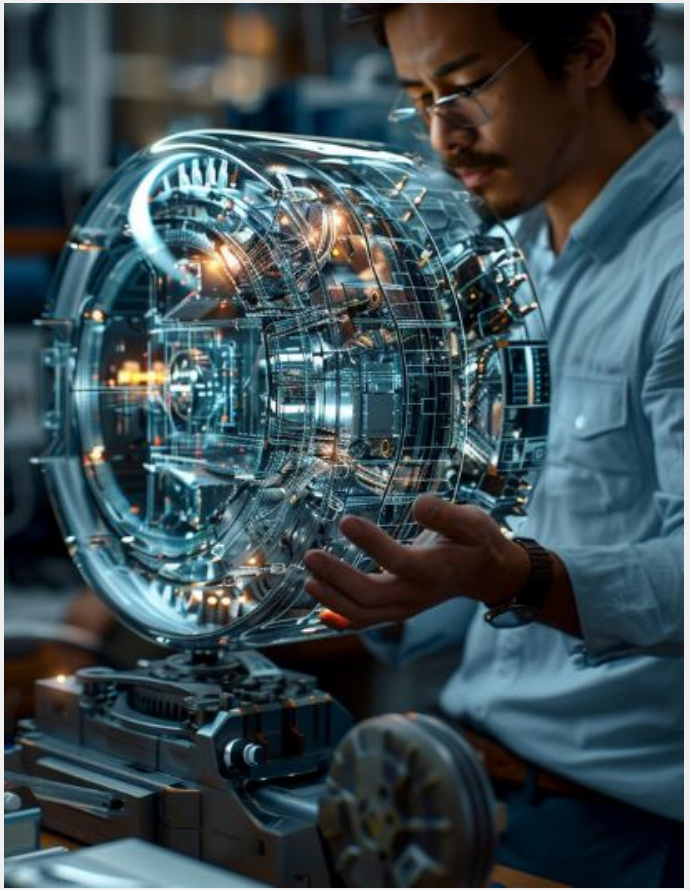
AI on a “Battery”

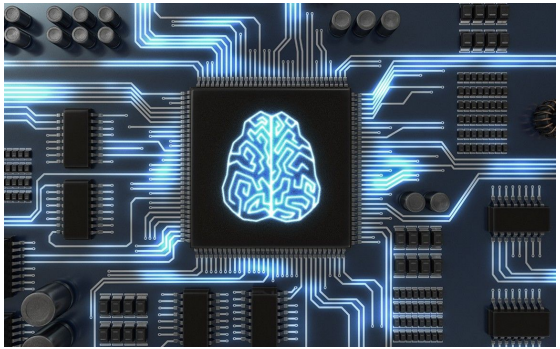


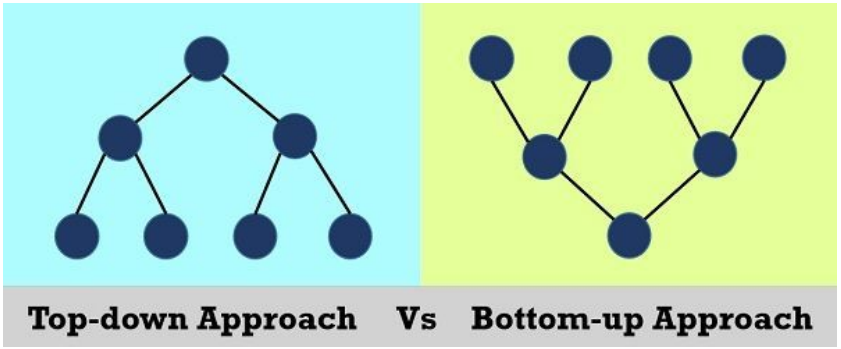


Edge Ai / Ai (on) the Edge

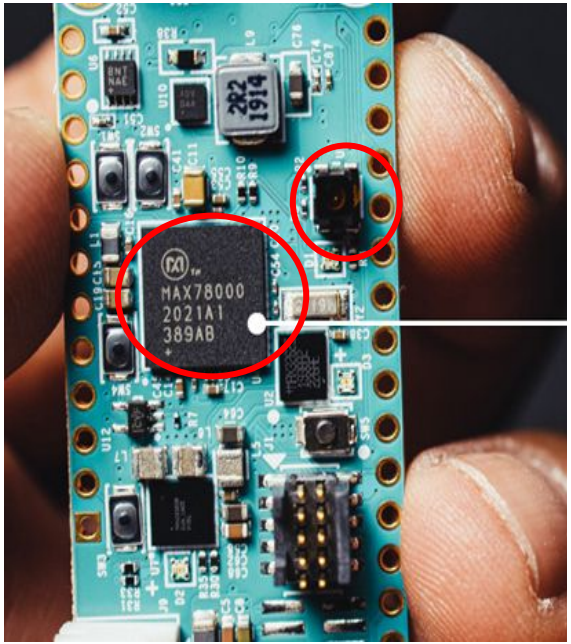
- Traditional CPUs, while versatile, struggled to handle the intensive computations
- specialized hardware accelerators designed to optimize AI processing
  - GPU - ideal for training and running neural networks
  - TPUs (Tensor Processing Units) - custom-designed chips optimized for tensor operations (google)
  - FPGAs - hardware that can be reconfigured to optimize specific tasks
  - NPUs (Neural Processing Units):
    coprocessor integrated into various devices designed specifically for AI tasks,
    providing high efficiency and performance with minimal power consumption.

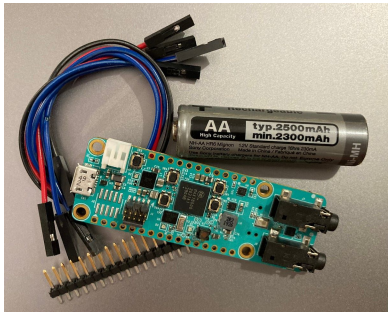
ASICs … dsp … * NLP





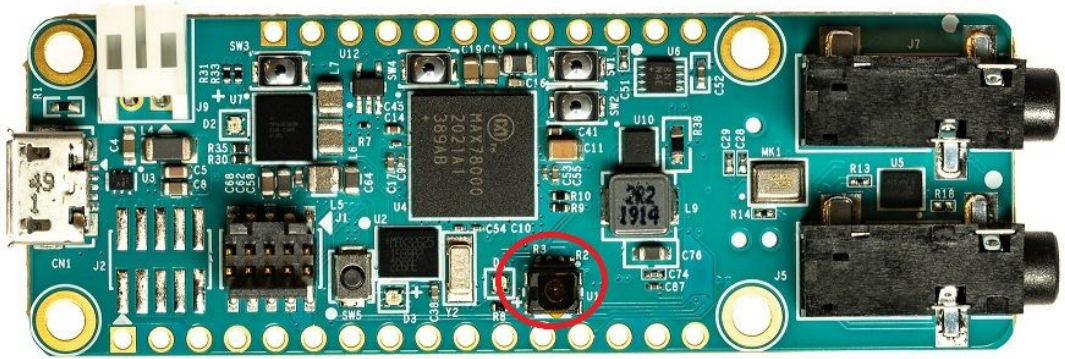
The target





“AI on a Battery”

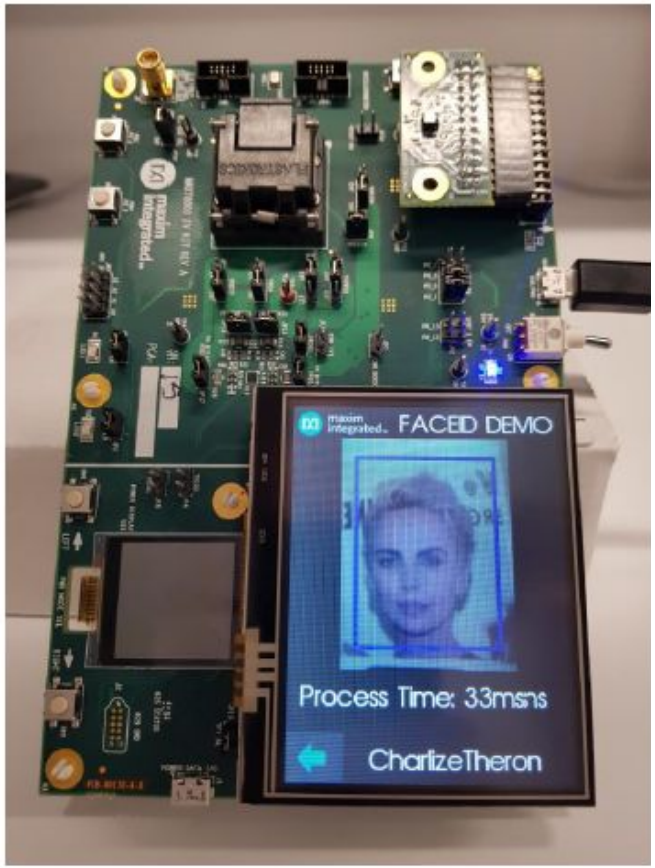
MAX78000FTHR
Camera module:
Himax HM01B0
320 x 320 pixels

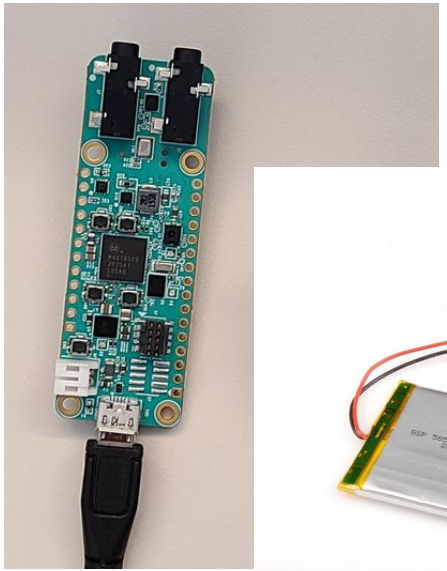


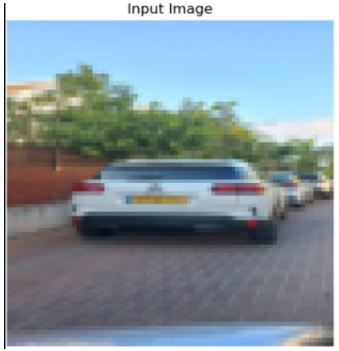
Make sure the protective film on the lens is removed!

- **MCU Core**: ARM Cortex-M4 with FPU.
- **Memory**: **512KB Flash, 128KB SRAM**.
- **Energy Efficiency**: **Ultra-low-power** design, optimized for battery-powered devices.
- **Power Consumption**: Typically around 1 mW when running inference.
- **AI Performance**: Capable of running AI models like CNNs efficiently on the edge.
  - **TOPS (Tera Operations Per Second)**: Approximately 0.1 TOPS (100 GOPS).

(some) application







segmentation

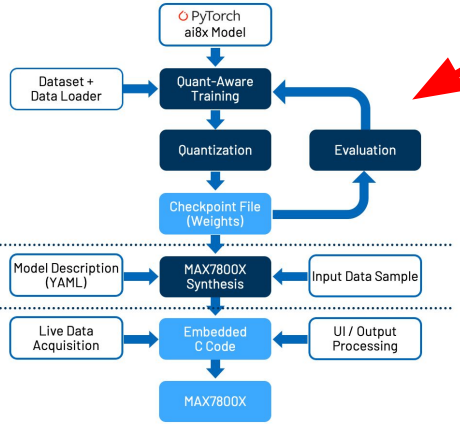
...objective

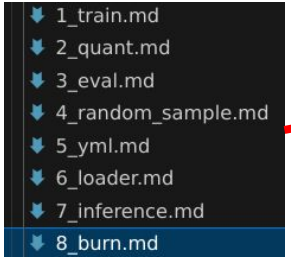
Python on dev host

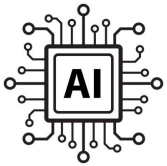
```
/data/tinyai/maxCNN/ai8x-trai                                                    --wd 0 --compress policies/s
chedule-cifar-nas.yaml --model ai85nascifarnet --dataset CIFAR10 --device MAX78000 --batch-size 100 --print-freq 100 --validation-split 0 --use-bias --qat-policy policies/
qat_policy_late_cifar.yaml --confus
ion

 "args": [
            "--deterministic",
            // "--epochs", "300",
            "--epochs", "1",
            "--optimizer", "Adam",
            "--lr", "0.001",
            "--wd", "0",
            "--compress", "policies/schedule-cifar-nas.yaml",
            "--model", "ai85nascifarnet",
            "--dataset", "CIFAR10",
            "--device", "MAX78000",
            "--batch-size", "100",
```

Python model for ia8x

```
class
    ""
    Si
    ""
    def __init__(
        self,
        num_classes=10,
        num_channels=3,
        dimensions=(32, 32),  # pylint: disable=unused-argument
        bias=False,
        **kwargs
    ):
        super().__init__()

        self.conv1_1 = ai8x.FusedConv2dBNReLU(num_channe                    , padding=1,
                                              bias=bias,                    e', **kwargs)
        self.conv1_2 = ai8x.FusedConv2dBNReLU(64, 32, 1,                    0,
                                              bias=bias,                    e', **kwargs)
        self.conv1_3 = ai8x.FusedConv2dBNReLU(32, 64, 3,                    1,
                                              bias=bias,                    e', **kwargs)
        self.conv2_1 = ai8x.FusedMaxPoolConv2dBNReLU(64,                    adding=1,
                                                     bias=bias, batchnorm='NoAffine', **kwargs)
        self.conv2_2 = ai8x.FusedConv2dBNReLU(32, 64, 1, stride=1, padding=0,
                                              bias=bias, batchnorm='NoAffine', **kwargs)
        self.conv3_1 = ai8x.FusedMaxPoolConv2dBNReLU(64, 128, 3, stride=1, padding=1,
                                                     bias=bias, batchnorm='NoAffine', **kwargs)
        self.conv3_2 = ai8x.FusedConv2dBNReLU(128, 128, 1, stride=1, padding=0,
                                              bias=bias, batchnorm='NoAffine', **kwargs)
        self.conv4_1 = ai8x.FusedMaxPoolConv2dBNReLU(128, 64, 3, stride=1, padding=1,
                                                     bias=bias, batchnorm='NoAffine', **kwargs)
```





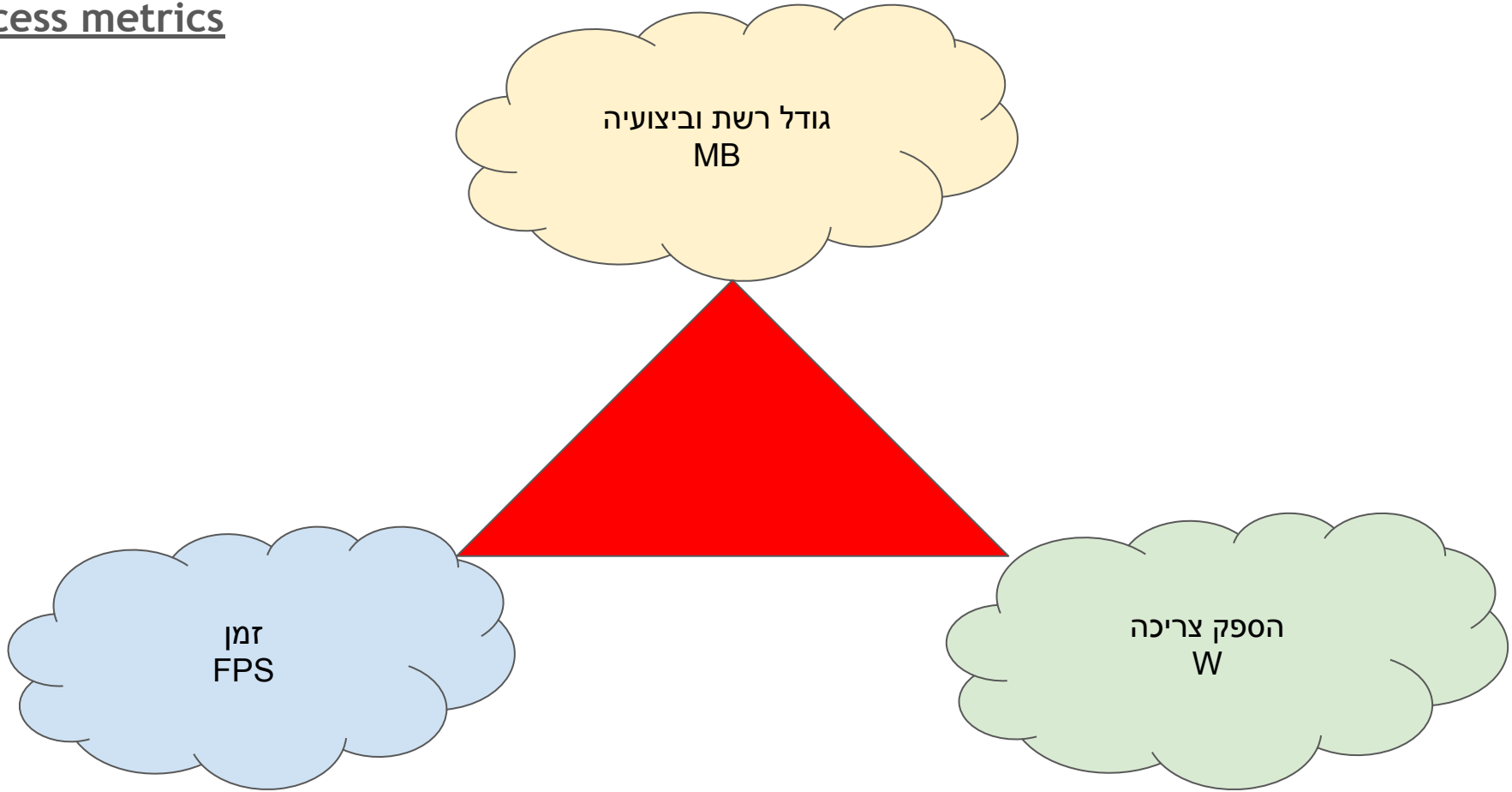


C for ARM

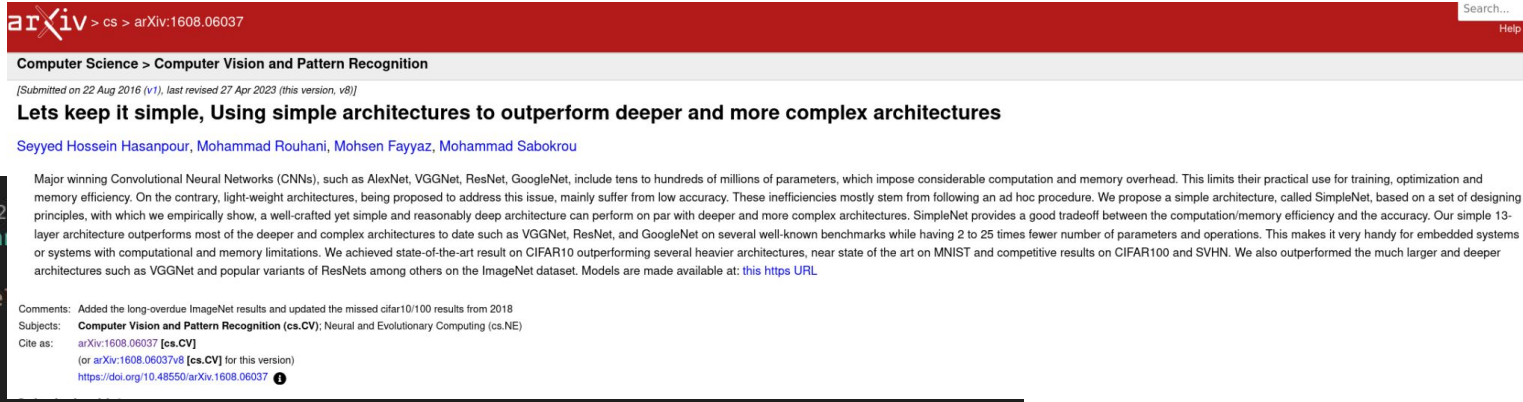
```
ar> C m
:ation l
? t ml_data[CNN_NUM_OUTPUTS];
: ml_softmax[CNN_NUM_OUTPUTS];

( layer(void)

((uint32_t *) ml_data);
l7p14_q15((const q31_t *) ml_data, CNN_NUM_OUTPUTS, ml_softmax);
```

```
int main(void)
{
  int i;
  int digs, tens;

  MXC_ICC_Enable(MXC_ICC0); // Enable cache

  // Switch to 100 MHz clock
  MXC_SYS_Clock_Select(MXC_SYS_CLOCK_IPO);
  SystemCoreClockUpdate();

  printf("Waiting...\n");

  // DO NOT DELETE THIS LINE:
  MXC_Delay(SEC(2)); // Let debugger interrupt if needed

  // Enable peripheral, enable CNN interrupt, turn on CNN clock
  // CNN clock: APB (50 MHz) div 1
  cnn_enable(MXC_S_GCR_PCLKDIV_CNNCLKSEL_PCLK, MXC_S_GCR_PCLKDIV_CNNCLKDIV_DIV1);
```

Success metrics



Starting point:



arXiv > cs > arXiv:1608.06037

Search... Help

Computer Science > Computer Vision and Pattern Recognition

[Submitted on 22 Aug 2016 (v1), last revised 27 Apr 2023 (this version, v8)]

Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures

Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, Mohammad Sabokrou

Major winning Convolutional Neural Networks (CNNs), such as AlexNet, VGGNet, ResNet, GoogleNet, include tens to hundreds of millions of parameters, which impose considerable computation and memory overhead. This limits their practical use for training, optimization and memory efficiency. On the contrary, light-weight architectures, being proposed to address this issue, mainly suffer from low accuracy. These inefficiencies mostly stem from following an ad hoc procedure. We propose a simple architecture, called SimpleNet, based on a set of designing principles, with which we empirically show, a well-crafted yet simple and reasonably deep architecture can perform on par with deeper and more complex architectures. SimpleNet provides a good tradeoff between the computation/memory efficiency and the accuracy. Our simple 13-layer architecture outperforms most of the deeper and complex architectures to date such as VGGNet, ResNet, and GoogleNet on several well-known benchmarks while having 2 to 25 times fewer number of parameters and operations. This makes it very handy for embedded systems or systems with computational and memory limitations. We achieved state-of-the-art result on CIFAR10 outperforming several heavier architectures, near state of the art on MNIST and competitive results on CIFAR100 and SVHN. We also outperformed the much larger and deeper architectures such as VGGNet and popular variants of ResNets among others on the ImageNet dataset. Models are made available at: this https URL

Comments: Added the long-overdue ImageNet results and updated the missed cifar10/100 results from 2018
Subjects: **Computer Vision and Pattern Recognition (cs.CV)**; Neural and Evolutionary Computing (cs.NE)
Cite as: arXiv:1608.06037 **[cs.CV]**
(or arXiv:1608.06037v8 **[cs.CV]** for this version)
https://doi.org/10.48550/arXiv.1608.06037

```
20
    Gorkem Ulkar, 3 years ago | 2
21  class AI85SimpleNet(n
22      """
23      SimpleNet v1 Mode
24      """
25      def __init__(
26              self,
27              num_classes=100,
28              num_channels=3,
29              dimensions=(32, 32),  # pylint: disable=unused-argument
30              bias=False,
31              **kwargs
32      ):
33          super().__init__()
34
35          self.conv1 = ai8x.FusedConv2dBNReLU(num_channels, 16, 3, stride=1, padding=1, bias=bias,
36                                              **kwargs)
37          self.conv2 = ai8x.FusedConv2dBNReLU(16, 20, 3, stride=1, padding=1, bias=bias, **kwargs)
38          self.conv3 = ai8x.FusedConv2dBNReLU(20, 20, 3, stride=1, padding=1, bias=bias, **kwargs)
39          self.conv4 = ai8x.FusedConv2dBNReLU(20, 20, 3, stride=1, padding=1, bias=bias, **kwargs)
40          self.conv5 = ai8x.FusedMaxPoolConv2dBNReLU(20, 20, 3, pool_size=2, pool_stride=2,
41                                                     stride=1, padding=1, bias=bias, **kwargs)
42          self.conv6 = ai8x.FusedConv2dBNReLU(20, 20, 3, stride=1, padding=1, bias=bias, **kwargs)
```

Data set : CamVid_sXX_YY

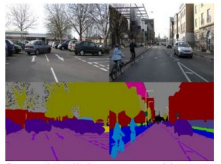
```
train_loaderl len =  369
val_loader len =  100
test_loader len =  232
```

CamVid (Cambridge-driving Labeled Video Database)

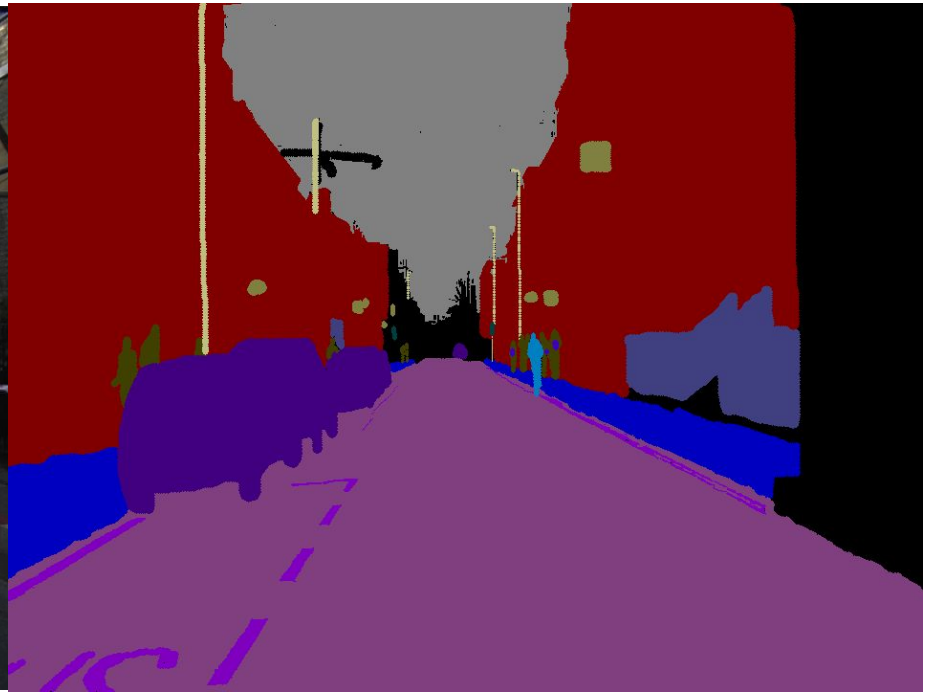
Edit

Introduced by Gabriel J. Brostow et al. in Semantic object classes in video: A high-definition ground truth database

CamVid (**Cambridge-driving Labeled Video Database**) is a road/driving scene understanding database which was originally captured as five video sequences with a 960×720 resolution camera mounted on the dashboard of a car. Those sequences were sampled (four of them at 1 fps and one at 15 fps) adding up to 701 frames. Those stills were manually annotated with 32 classes: void, building, wall, tree, vegetation, fence, sidewalk, parking block, column/pole, traffic cone, bridge, sign, miscellaneous text, traffic light, sky, tunnel, archway, road, road shoulder, lane markings (driving), lane markings (non-driving), animal, pedestrian, child, cart luggage, bicyclist, motorcycle, car, SUV/pickup/truck, truck/bus, train, and other moving object



Source: http://mi.eng.cam.ac.uk/resea...



Starting point:

1. **Input Preparation Layers:**
   - **prep0, prep1, prep2**: These initial layers sequentially transform the input image (3 channels) through a series of convolutional operations:
     - **Conv2d + BatchNorm + ReLU**: Convert the input to higher dimensions (64 -> 64 -> 32 channels).
2. **Encoder Path:**
   - **enc1**: Applies convolution to reduce dimensions (32 to 8 channels).
   - **enc2**: MaxPooling followed by convolution (8 to 28 channels).
   - **enc3**: Another MaxPooling and convolution step (28 to 56 channels).
3. **Bottleneck Layer:**
   - **bneck**: Central part of the network, combining MaxPooling and convolution (56 to 112 channels).
4. **Decoder Path:**
   - **upconv3**: Upsampling (deconvolution) layer increasing dimensions (112 to 56 channels).
   - **dec3**: Combines features from encoder (concatenation) and applies convolution (56 channels).
   - **upconv2**: Further upsampling (56 to 28 channels).
   - **dec2**: Applies convolution after concatenation (28 channels).
   - **upconv1**: Final upsampling layer (28 to 8 channels).
   - **dec1**: Final set of convolutional operations after concatenation (16 to 48 channels).
   - **dec0**: Final convolution operations (48 to 64 channels).
5. **Final Convolution Layers:**
   - **conv_p1, conv_p2, conv_p3**: Further refine the output with convolutional layers (64 channels each).
   - **conv**: Final layer, converting the refined output to the final desired output (64 to 256 channels).

fold

the final number of output channels is calculated as `num_final_channels = num_classes * fold_ratio * fold_ratio`.

This allows the model to provide a detailed and high-dimensional feature representation, which is beneficial for tasks requiring fine-grained segmentation.

Param size

Pythorch summary

```
  )
  (enc1): Sequential(
    (0): Conv2d(32, 8, kernel_size=(3, 3), stride=(1, 1), padd
    (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
    (2): ReLU(inplace=True)
  )
  (enc2): Sequential(
    (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilatio
    (1): Conv2d(8, 28, kernel_size=(3, 3), stride=(1, 1), padd
    (2): BatchNorm2d(28, eps=1e-05, momentum=0.1, affine=True,
...
Forward/backward pass size (MB): 76.73
Params size (MB): 1.06
Estimated Total Size (MB): 77.86
----------------------------------------------------------------
```

```
Forward/backward pass size (MB):

    This is the amount of memory required to store the intermediate
    activations (outputs of each layer) and gradients during the forward and
    backward passes of the model. This memory is needed for computing
    gradients during the backpropagation step.


Params size (MB):

    This is the memory required to store the parameters (weights and biases)
    of the model. This size is computed based on the total number of
    parameters and their data type (usually float32).

Estimated Total Size:

    total memory needed to train the model
```

```
# Calculate number of parameters
total_params, trainable_params = count_parameters(model)
print(f"Trainable parameters: {trainable_params}")
```

```
def dtype_to_bytes(dtype):
    dtype_bytes = {
        torch.float32: 4,
        torch.float64: 8,
        torch.float16: 2,
        torch.int32: 4,
        torch.int64: 8
        # Add other data types if needed
    }
```

```
check_unique_param_sizes(model)
```

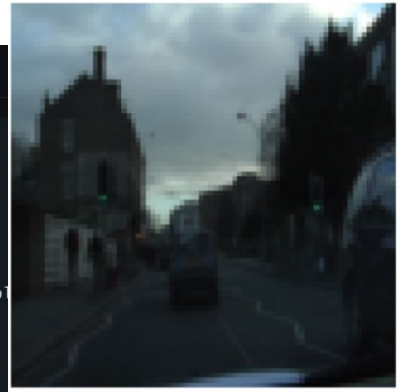
Size: 4 bytes, Count: 66

```
  size_of_each_param_in_bytes=4
  total_size_in_bytes=total_params * size_of_each_param_in_bytes
  total_size_in_megabytes = total_size_in_bytes / (1024*1024)

  print(f"total_size_in_megabytes = {total_size_in_megabytes}")
✓ 0.0s
total_size_in_megabytes = 1.0598602294921875
```

1try:

c33

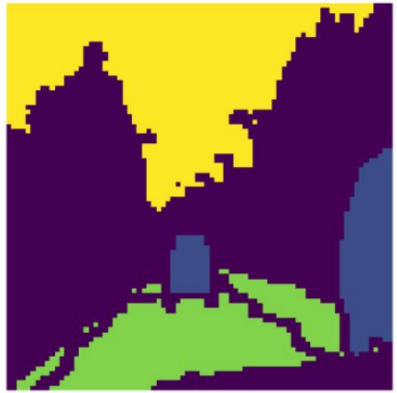
```
{
    'name': 'CamVid_s352_c33',
    'input': (48, 88, 88),
    'output': ('None', 'Animal', 'Archway', 'Bicyclist', 'Bridge', 'Building', 'Car',
               'CartLuggagePram', 'Child', 'Column_Pole', 'Fence', 'LaneMkgsDriv',
               'LaneMkgsNonDriv', 'Misc_Text', 'MotorcycleScooter', 'OtherMoving',
               'ParkingBlock', 'Pedestrian', 'Road', 'RoadShoulder', 'Sidewalk', 'SignSymbol',
               'Sky', 'SUVPickupTruck', 'TrafficCone', 'TrafficLight', 'Train', 'Tree',
               'Truck_Bus', 'Tunnel', 'VegetationMisc', 'Void', 'Wall'),
    'loader': camvid_get_datasets_s352_c33,
    'fold_ratio': 4,
},
```



c3

```
{
    'name': 'CamVid_s80_c3',  # 3 classes
    'input': (3, 80, 80),
    'output': (0, 1, 2),
    'weight': (1, 1, 1),
    'loader': camvid_g
},
```

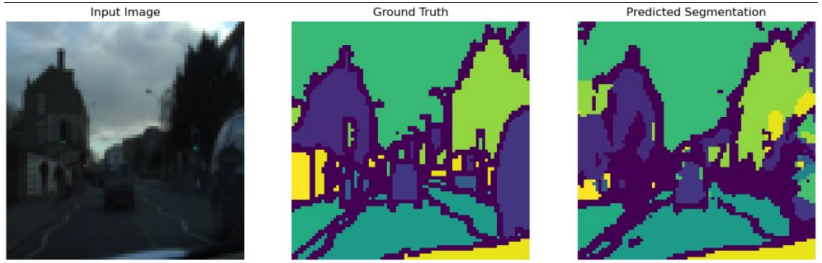
```
# 23        TrafficCone    0    0   64
# 24      TrafficLight     0   64   64
# 25             Train   192   64  128
# 26              Tree   128  128    0
# 27         Truck_Bus   192  128  192
# 28            Tunnel    64    0   64
# 29     VegetationMisc   192  192    0
# 30              Void     0    0    0
# 31              Wall    64  192    0

filter_label = [5,21,17] # car sky road
```

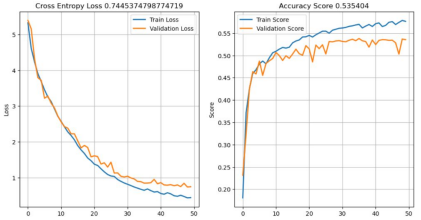


1try:

UNetLarge CamVid_80_c33

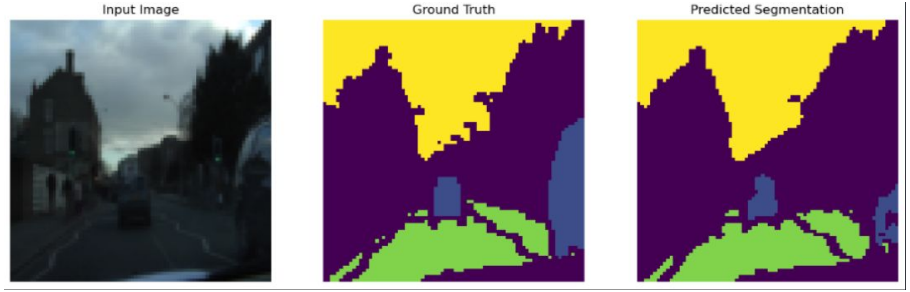
```
trainning   : EPOCHS_TO_RUN=50 ; batch=8
model       : file=./unet_model_large_x80_c33_e50_A.pth
input data  : train_len =  369 ; test_len=100 ; val_len=232
img data    : IMAGE_SIZE=(80, 80)
model       : NUM_CLASS=33 ; NUM_CHANN=3 ; FOLD_RATIO=4
score       : val_score=0.535404 ; loss_score 0.7445374798774719
Trainable parameters: 308924
parameters_size_in_megabytes : 1.1784515380859375
```

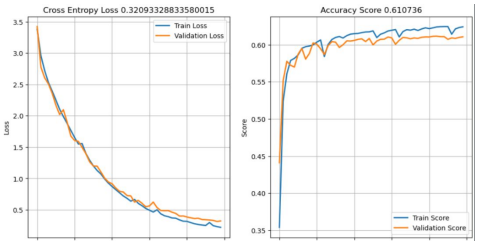




UNetLarge CamVid_80_c33

```
trainning   : EPOCHS_TO_RUN=50 ; batch=8
model       : file=./unet_model_large_x80_c3_e50_B.pth
input data  : train_len =  369 ; test_len=100 ; val_len=232
img data    : IMAGE_SIZE=(80, 80)
model       : NUM_CLASS=4 ; NUM_CHANN=3 ; FOLD_RATIO=4
score       : val_score=0.610736 ; loss_score 0.32093328833580015
Trainable parameters: 277836
parameters_size_in_megabytes : 1.0598602294921875
```

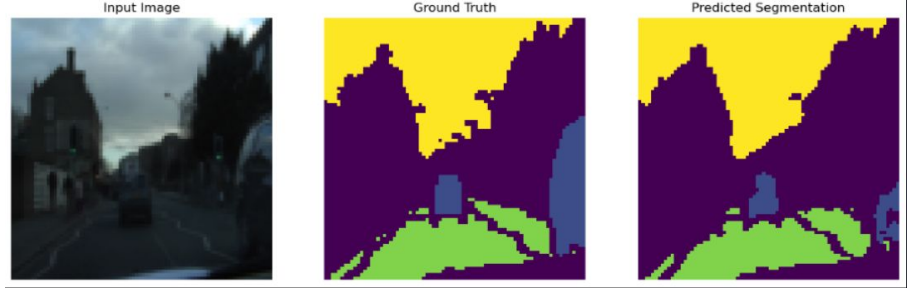


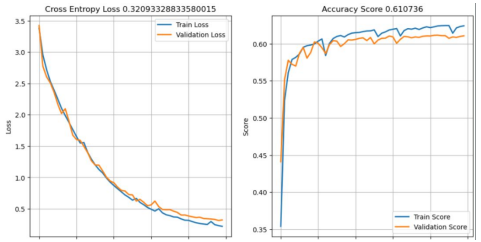


2try:

UNetLarge CamVid_80_c33

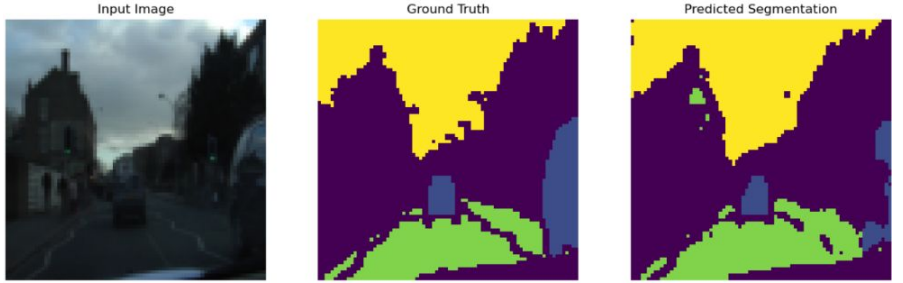
```
trainning   : EPOCHS_TO_RUN=50 ; batch=8
model       : file=./unet_model_large_x80_c3_e50_B.pth
input data  : train_len =  369 ; test_len=100 ; val_len=232
img data    : IMAGE_SIZE=(80, 80)
model       : NUM_CLASS=4 ; NUM_CHANN=3 ; FOLD_RATIO=4
score       : val_score=0.610736 ; loss_score 0.32093328833580015
Trainable parameters: 277836
parameters_size_in_megabytes : 1.0598602294921875
```

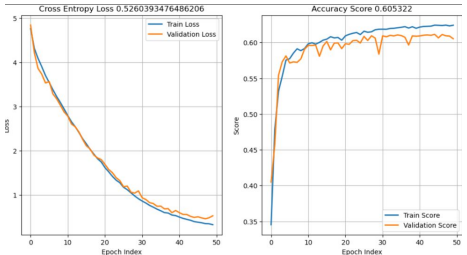




UNetLarge CamVid_80_c3

```
trainning   : EPOCHS_TO_RUN=50 ; batch=8
model       : file=./unet_model_large_x80_c3_e50_C.pth
input data  : train_len =  369 ; test_len=100 ; val_len=232
img data    : IMAGE_SIZE=(80, 80)
model       : NUM_CLASS=4 ; NUM_CHANN=3 ; FOLD_RATIO=8
score       : val_score=0.605322 ; loss_score 0.5260393476486206
Trainable parameters: 290700
parameters_size_in_megabytes : 1.1089324951171875
```





3try:

B

Lets change the network...

1. **Initial Layers (prep0, prep1, prep2):**
   - These layers prepare the input image by applying a series of depthwise separable convolutions, batch normalization, and ReLU activations. They transform the input channels (3 for RGB) to intermediate feature representations (3 -> 32 -> 32 -> 16 channels).
2. **Encoder Path (enc1, enc2, enc3):**
   - The encoder progressively reduces the spatial dimensions while increasing the feature depth:
     - **enc1:** Converts 16 channels to 8 through depthwise separable convolutions.
     - **enc2:** Downsamples (via max pooling) and then increases features from 8 to 16 channels.
     - **enc3:** Further downsampling and feature increase from 16 to 32 channels.
3. **Bottleneck Layer (bneck):**
   - Acts as the central layer, reducing spatial dimensions further while capturing complex features (32 to 64 channels).
4. **Decoder Path (upconv3, dec3, upconv2, dec2, upconv1, dec1, dec0):**
   - The decoder upsamples the feature maps and combines them with corresponding encoder features through concatenation:
     - **upconv3 & dec3:** Upsamples from 64 to 32 channels and combines with encoder features.
     - **upconv2 & dec2:** Further upsamples and processes features from 32 to 16 channels.
     - **upconv1 & dec1:** Final upsampling steps, combining and processing features down to 8 channels and then up to 16 channels.
     - **dec0:** Final set of convolutional operations, increasing features from 16 to 32 channels.
5. **Final Convolution Layers (conv_p1, conv_p2, conv_p3, conv):**
   - These layers refine the output further and prepare the final feature map:
     - The final convolution layer outputs 64 channels, converting the processed features to the final desired output.

The number of output channels in the final layer is fixed at 64. provide a detailed feature space, allowing the network to capture intricate details necessary for accurate image segmentation.

diffs

```
print_model_summary(model, input_size=(3, IMAGE_SIZE[0], IMAGE_SIZE[1]))
```

[20] ✓ 0.0s Python

```
UNetLarge(
  (prep0): Sequential(
    (0): Conv2d(3, 64, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (prep1): Sequential(
    (0): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (prep2): Sequential(
    (0): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (enc1): Sequential(
    (0): Conv2d(32, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (enc2): Sequential(
    (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (1): Conv2d(8, 28, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (2): BatchNorm2d(28, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU(inplace=True)
```

A

```
    summary(model, input_size=input_size)

print_model_summary(model, input_size=(3, IMAGE_SIZE[0], IMAGE_SIZE[1]))
```

[21] ✓ 0.0s

```
UNetSmall(
  (prep0): Sequential(
    (0): Conv2d(3, 3, kernel_size=(1, 1), stride=(1, 1), groups=3)
    (1): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(3, 32, kernel_size=(1, 1), stride=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (prep1): Sequential(
    (0): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1), groups=32)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
  )
  (prep2): Sequential(
    (0): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1), groups=32)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1))
    (4): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
  )
```

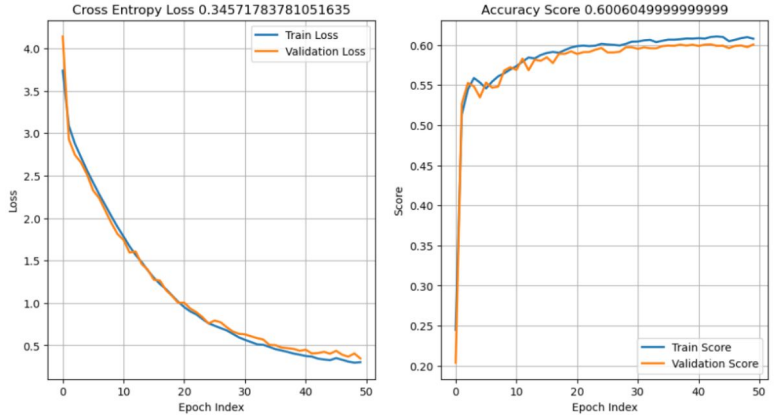
B

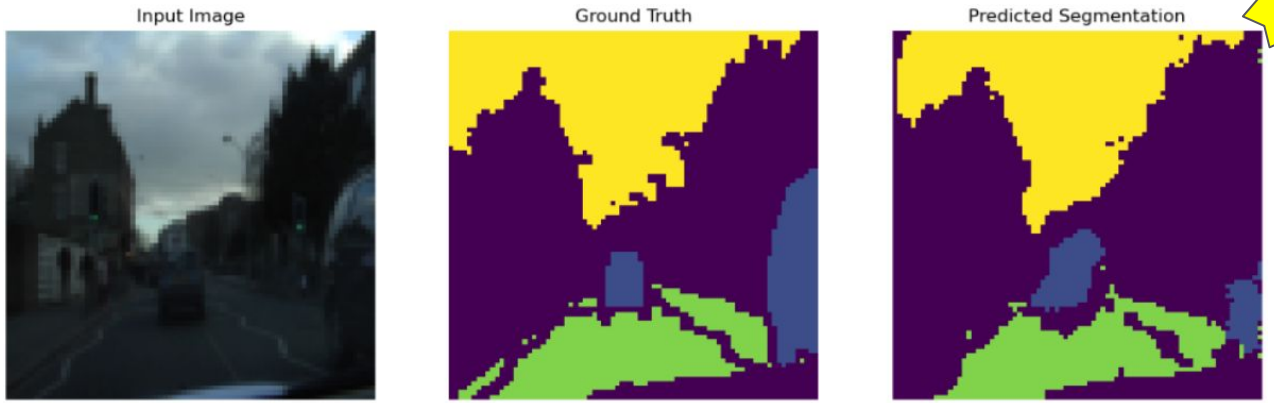
Lets change the network...

UNetSmall CamVid_80_c3

```
trainning    : EPOCHS_TO_RUN=50 ; batch=8
model        : file=./unet_model_large_x80_c3_e50_D.pth
input data   : train_len =  369 ; test_len=100 ; val_len=232
img data     : IMAGE_SIZE=(80, 80)
model        : NUM_CLASS=4 ; NUM_CHANN=3 ; FOLD_RATIO=4
score        : val_score=0.6006049999999999 ; loss_score 0.34571783781051635
Trainable parameters: 41436
parameters_size_in_megabytes : 0.1580657958984375
```

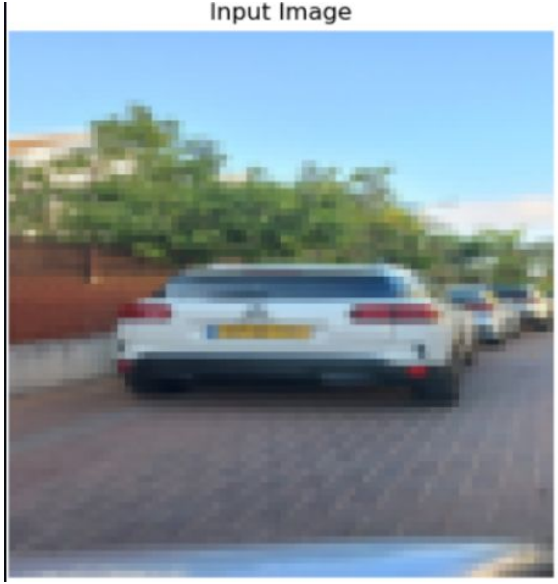
B



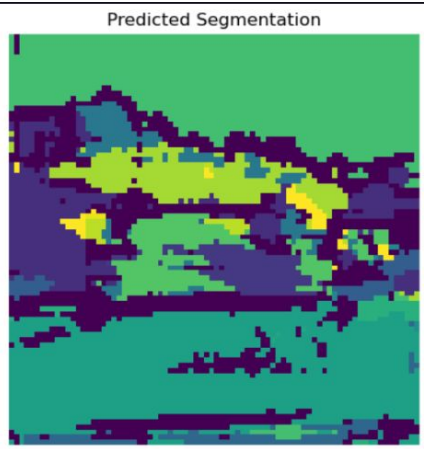


B

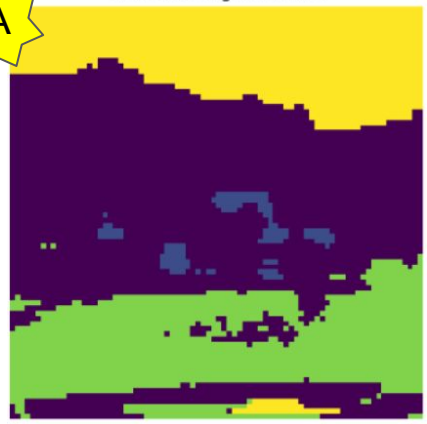
(my) Blind test...



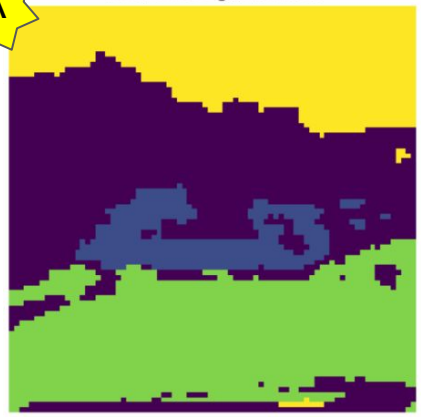
A



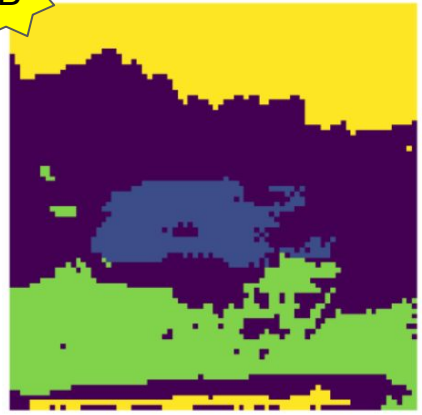
A



A



B

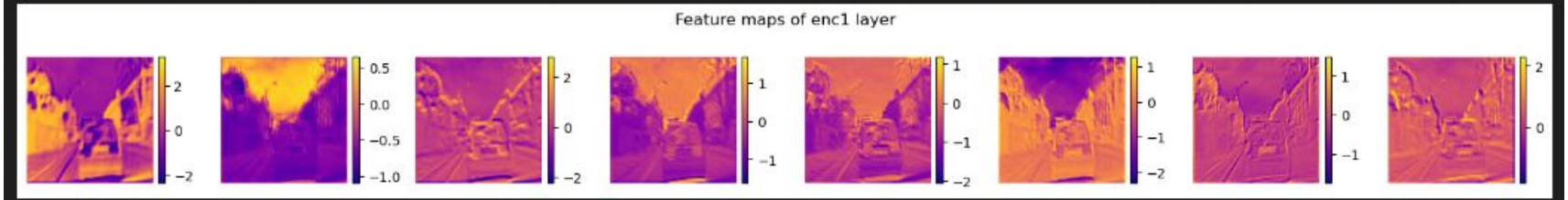


hooks

```
visualize_single_layer(feature_maps[model.enc1[0]])
```

✓ 1.5s Python

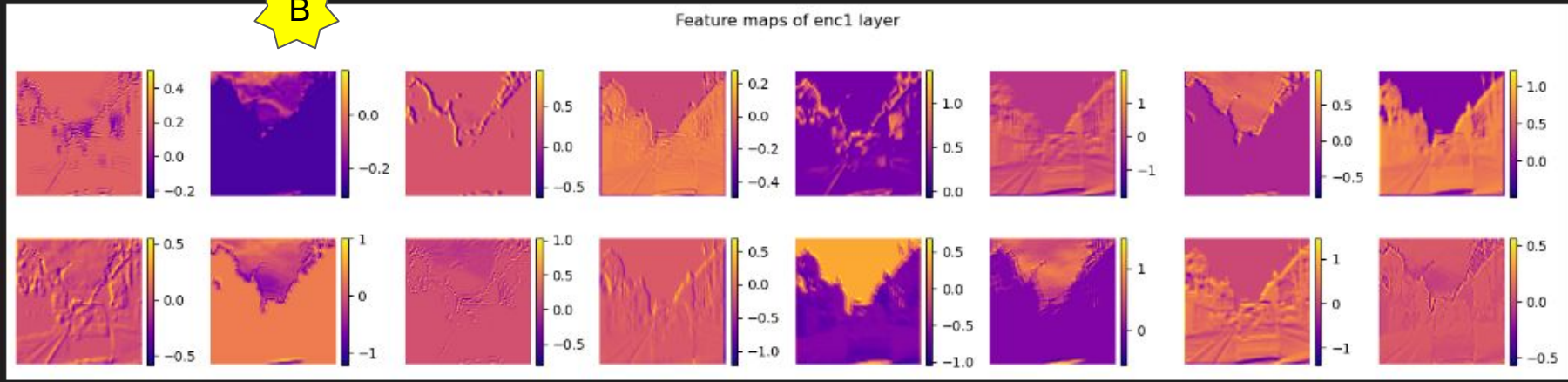
A




```
visualize_single_layer(feature_maps[model.enc1[0]])
```

✓ 1.1s Python

B



Make the magic

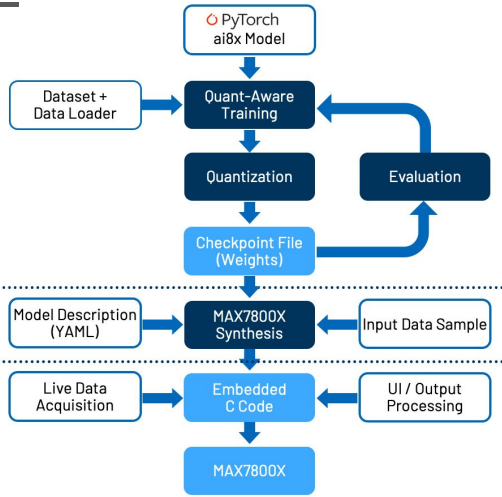
```
(ai8x-training)python train.py --deterministic --epochs 50
 --optimizer Adam --lr 0.001 --wd 0
--model ai85unetlarge --out-fold-ratio 4 --use-bias
 --dataset CamVid_s352_c3 --device MAX78000
--batch-size 16 --qat-policy policies/qat_policy_
camvid.yaml --compress policies/schedule-camvid.yaml
 --validation-split 0 --compiler-mode none;
```

```
(ai8x-synthesis) python quantize.py
../test2_unet/best.pth.tar
../test2_unet/best-q.pth.tar --device MAX78000 -v
```

① Training
ai8x-training

② Synthesis
ai8x-synthesis

③ Deployment
Embedded SDK




```
(ai8x-training) python train.py --deterministic --model ai85unetlarge --out-fold-ratio 4 --dataset CamVid_s352_c3
--device MAX78000 --qat-policy policies/qat_policy_camvid.yaml --use-bias --evaluate -8 --exp-load-weights-from
./test2_unet/best-fakept-q.pth.tar--save-sample 10 --compiler-mode none ;
(ai8x-synthesize) python ai8xize.py --verbose --test-dir ../test2_unet/demo --prefix ai85-unet --checkpoint-file
../test2_unet/best-fakept-q.pth.tar --config-file ../test2_unet/camvid-unet-large-fakept.yaml --device MAX78000
--overlap-data --mlator --no-unload --max-checklines 8192 --new-kernel-loader --overwrite
(sdk)>make
(Burn)openocd -s /data/tinyai/maxCNN/sdk/Tools/OpenOCD/scripts -f interface/cmsis-dap.cfg -f target/max78000.cfg -c
"program build/max78000.elf verify reset exit" ; > minicom;
```

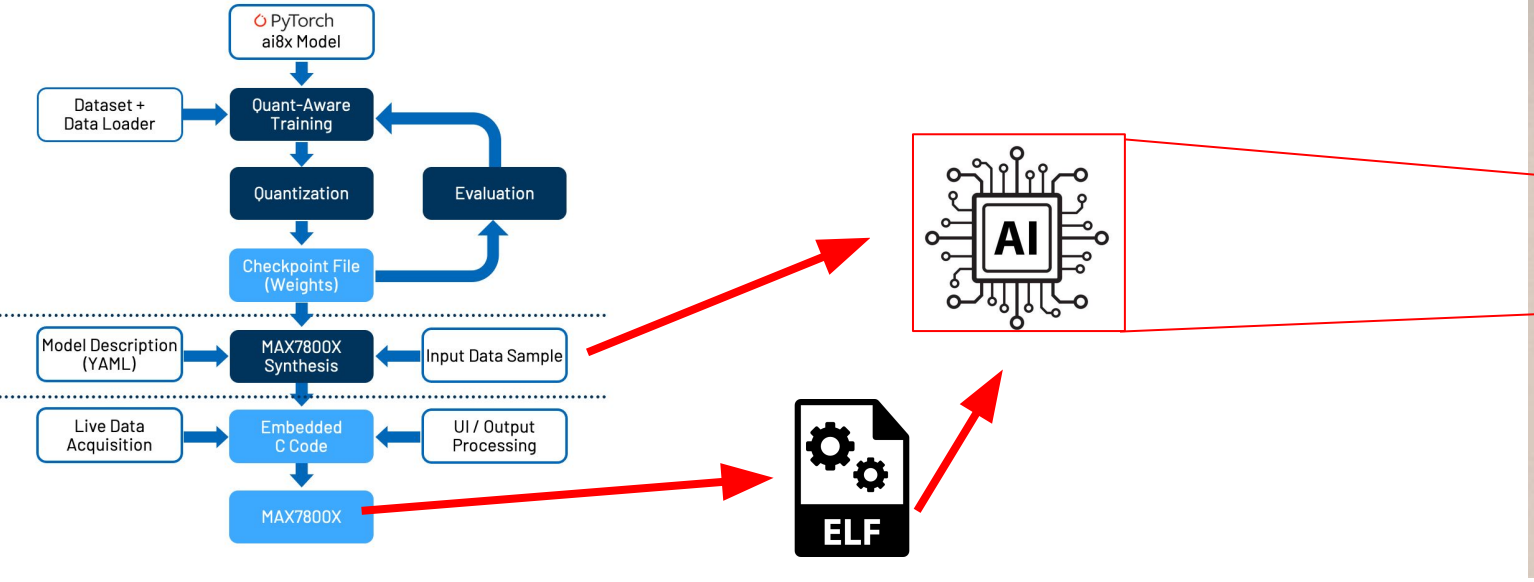
Verify:

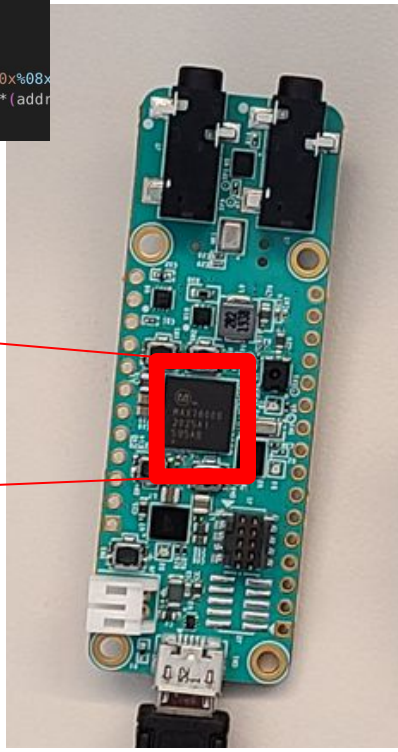
main

```
101     // Enable peripheral, enable CNN interrupt, turn on CNN
102     // CNN cl
103     cnn_enabl                    NCLKSEL_PCLK, MXC_S_GCR
104
105     printf("\                  ***\n");
106
107     cnn_init(); // Bring state machine into consistent stat
108     cnn_load_weights(); // Load kernels
109     cnn_load_bias();
110     cnn_configure(); // Configure state machine
111     load_input(); // Load data input
112     cnn_start(); // Start CNN processing
113
114     SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk; // SLEEPDEEP=0
115     while (cnn_time == 0) __WFI(); // Wait for CNN
116
117     if (check_output() != CNN_OK)
118         fail();
119     softmax_layer();
120
121     printf("\n*** PASS ***\n\n");
122
```

check_output

```
48
49      memcpy32((uint3
50  }
51
52  // Delete this func                    Smith, 2
53  static const uint32_t sample_output[] = SAMPLE_OUTPUT;
54  int check_output(void)
55  {
56      int i;
57      uint32_t mask, len;
58      volatile uint32_t *addr;
59      const uint32_t *ptr = sample_output;
60
61      while ((addr = (volatile uint32_t *)*ptr++) != 0) {
62          mask = *ptr++;
63          len = *ptr++;
64          for (i = 0; i < len; i++)
65              if ((*addr++ & mask) != *ptr++) {
66                  printf("Data mismatch (%d/%d) at address 0x%08x
67                         i + 1, len, addr - 1, *(ptr - 1), *(addr
68                  return CNN_FAIL;
69              }
```

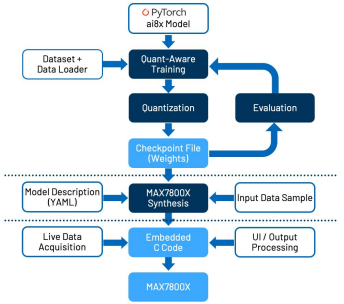




train

A

```
2024-07-09 06:50:24,177 - ==> Best [Top1: 80.296   Params: 278176 on epoch: 48]
2024-07-09 06:50:24,177 - Saving checkpoint to: logs/2024.07.09-063349/qat_checkpo
2024-07-09 06:50:24,228 - --- test ---------------------
2024-07-09 06:50:24,228 - 600 samples (16 per mini-batch)
2024-07-09 06:50:26,400 - Test: [   10/   38]    Loss 0.675823    Top1 79.704277
2024-07-09 06:50:28,607 - Test: [   20/   38]    Loss 0.673408    Top1 79.728353
2024-07-09 06:50:30,235 - Test: [   30/   38]    Loss 0.689856    Top1 78.895035
2024-07-09 06:50:31,204 - Test: [   38/   38]    Loss 0.694540    Top1 78.687414
2024-07-09 06:50:31,331 - ==> Top1: 78.687    Loss: 0.695
```



B

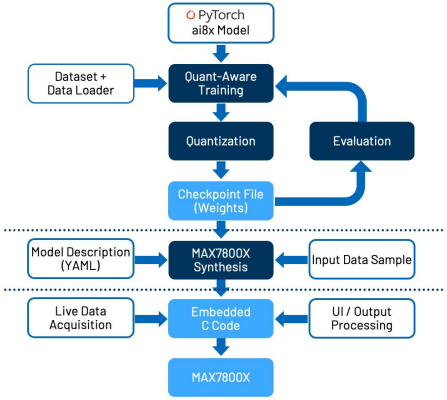
```
==> Best [Top1: 77.720   Params: 40504 on epoch: 45]
Saving checkpoint to: logs/2024.07.13-062820/qat_checkpoint.pth.tar
--- test ---------------------
600 samples (16 per mini-batch)
Test: [   10/   38]    Loss 0.863710    Top1 69.603670
Test: [   20/   38]    Loss 0.864482    Top1 69.490563
Test: [   30/   38]    Loss 0.871459    Top1 69.198025
Test: [   38/   38]    Loss 0.882090    Top1 68.698626
```

ensemble

quant

- 32-bit floating-point numbers to 8-bit integers

Reduces Model Size / Speeds Up Inference / Lowers Power Consumption
While
…retain the model's accuracy as much as possible….



A

```
3.3M  qat_best.pth.tar
3.4M  best-q.pth.tar
```

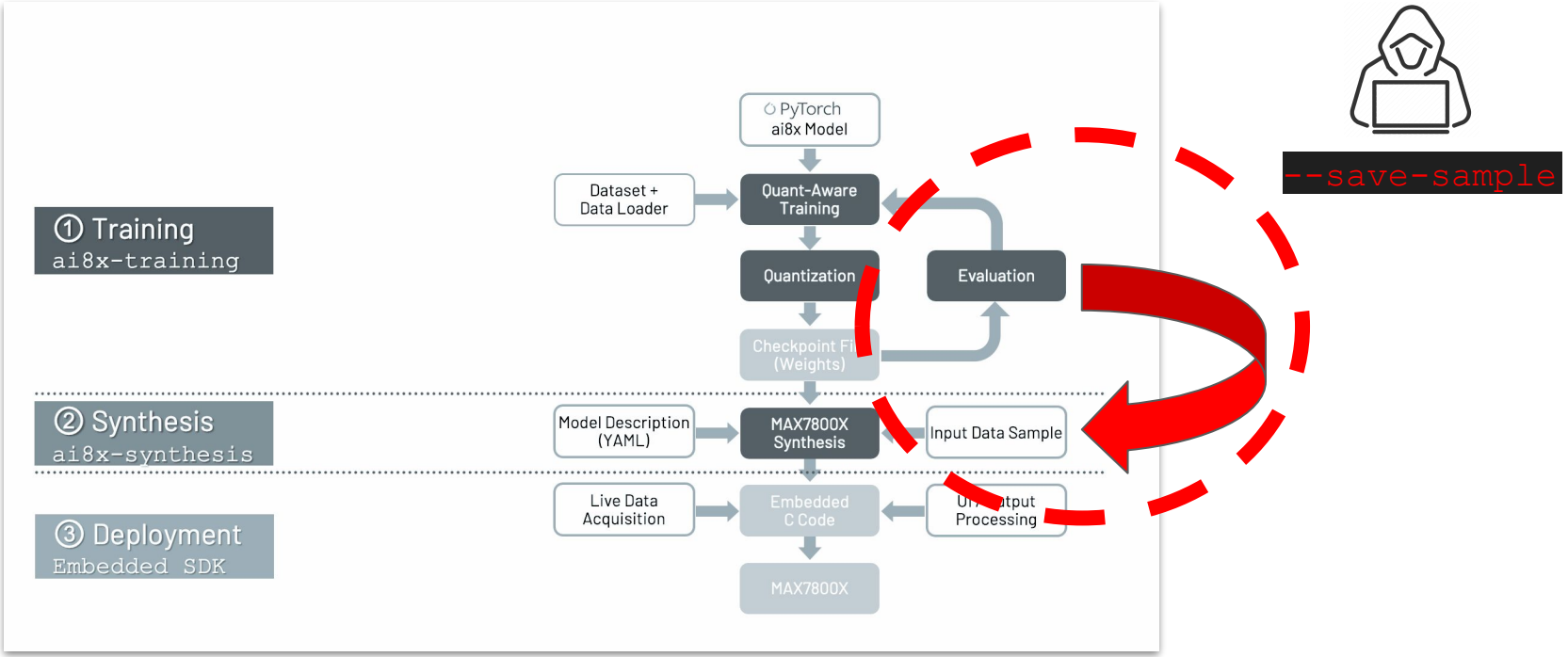
B

```
├─ [674K]  qat_best.pth.tar
├─ [655K]  best-q.pth.tar
```

~x5

eval + sample

```
Dataset sizes:
        test=600
--- test --------------------
600 samples (256 per mini-batch)
==> Saving sample at index 10 to sample_camvid_s352_c3.npy
[1]    215756 killed     python train.py --deterministic --model myunetsmall --out-fold-ratio 4
(ai8x-training) (base) ➜  ai8x-training git:(develop) ✗
```



build

#include camra.h

(A) 3.4Mb

A

```
RESOURCE USAGE
Weight memory: 281,312 bytes out of 442,368 bytes total (63.6%)
Bias memory:   908 bytes out of 2,048 bytes total (44.3%)
```

```
├── ai85-unet.launch
├── cnn.c
├── cnn.h
├── log.txt
├── main.c
├── Makefile
├── project.mk
├── sampledata.h
├── sampleoutput.h
├── softmax.c
└── weights.h
```

```
if (check_output() != CNN_OK)
    fail();
softmax_layer();

printf("\n*** PASS ***\n\n");
```

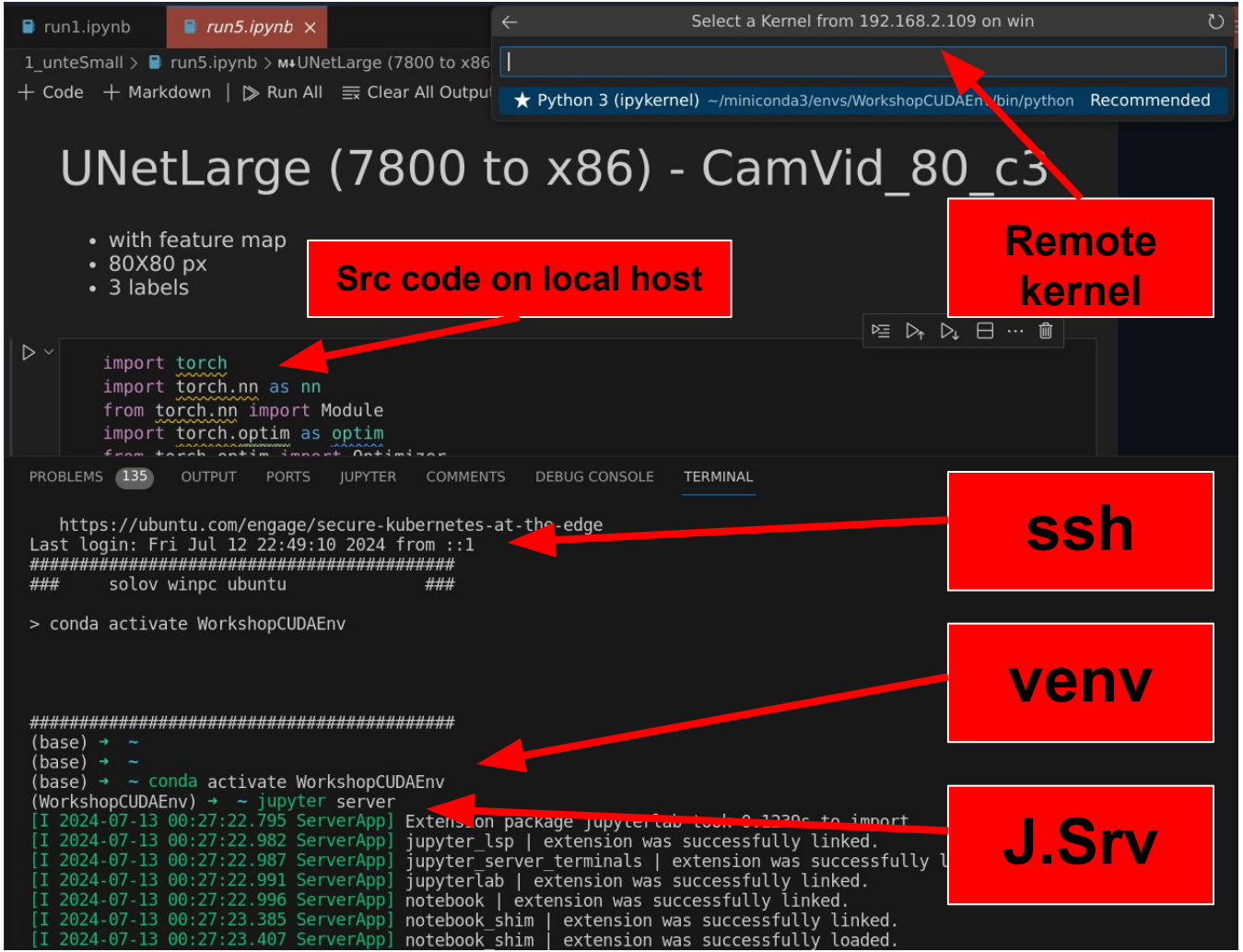
~x5

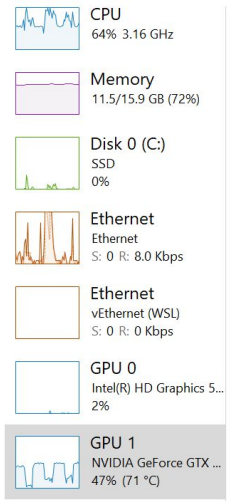
B

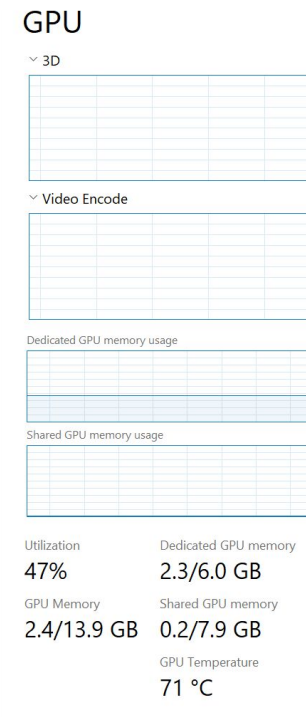
(B) 655 K

├── max78000.elf

devEnv


run1.ipynb
run5.ipynb
1_unteSmall > run5.ipynb > UNetLarge (7800 to x86
+ Code + Markdown | Run All Clear All Outpu
Select a Kernel from 192.168.2.109 on win
Python 3 (ipykernel) ~/miniconda3/envs/WorkshopCUDAEn/bin/python Recommended
UNetLarge (7800 to x86) - CamVid_80_c3
with feature map
80X80 px
3 labels
Src code on local host
Remote kernel
import torch
import torch.nn as nn
from torch.nn import Module
import torch.optim as optim
PROBLEMS 135 OUTPUT PORTS JUPYTER COMMENTS DEBUG CONSOLE TERMINAL
https://ubuntu.com/engage/secure-kubernetes-at-the-edge
Last login: Fri Jul 12 22:49:10 2024 from ::1
solov winpc ubuntu
> conda activate WorkshopCUDAEnv
ssh
venv
(base) ~
(base) ~
(base) ~ conda activate WorkshopCUDAEnv
(WorkshopCUDAEnv) ~ jupyter server
[I 2024-07-13 00:27:22.795 ServerApp] Extension package jupyterlab took 0.1339s to import
[I 2024-07-13 00:27:22.982 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-07-13 00:27:22.987 ServerApp] jupyter_server_terminals | extension was successfully l
[I 2024-07-13 00:27:22.991 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-07-13 00:27:22.996 ServerApp] notebook | extension was successfully linked.
[I 2024-07-13 00:27:23.385 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-07-13 00:27:23.407 ServerApp] notebook_shim | extension was successfully loaded.
J.Srv


CPU
64% 3.16 GHz
Memory
11.5/15.9 GB (72%)
Disk 0 (C:)
SSD
0%
Ethernet
Ethernet
S: 0 R: 8.0 Kbps
Ethernet
vEthernet (WSL)
S: 0 R: 0 Kbps
GPU 0
Intel(R) HD Graphics 5...
2%
GPU 1
NVIDIA GeForce GTX ...
47% (71 °C)


GPU
3D
Video Encode
Dedicated GPU memory usage
Shared GPU memory usage
Utilization
47%
Dedicated GPU memory
2.3/6.0 GB
GPU Memory
2.4/13.9 GB
Shared GPU memory
0.2/7.9 GB
GPU Temperature
71 °C

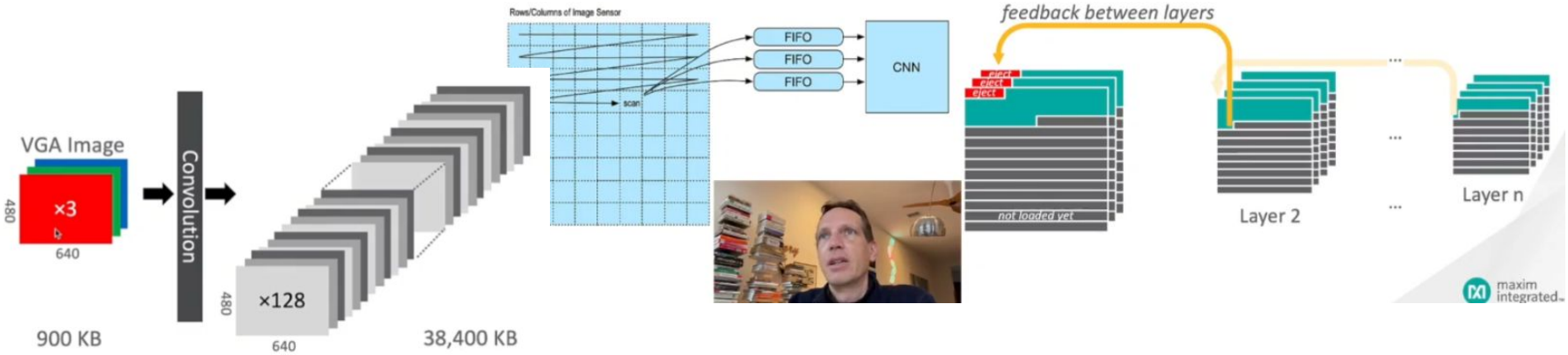
Next?

- Embedded app (camera / sd card / debug port)
- Measure power / frame rate
- Larger/multiple networks
  - mobileNet with Unet

1. Model Pruning (Structured/Unstructured -close to zero)
2. Quantization-Aware Training (during training to achieve higher accuracy)
3. Teacher-Student (smaller network to mimic the outputs of a larger)
4. Tensor Decomposition
5. Group Convolutions (limit connections between input and output)
6. NAS techniques (automatically find a more efficient network architecture)
7. Weight Sharing (share weights across different layers)
8. Regularization Techniques (L1/Droput)
9. Custom Loss Functions (penalize model size)

Streaming

- Streaming allows input data dimensions that exceed the available per-channel data memory in the accelerator
  - > On MAX78000: anything bigger than 181×181
- Natural when input data is acquired from an image sensor that scans row-by-row



Resolution	Reference	Memory for 128 ch. (max. intermediate)
32×32	CIFAR-10	128 KB
160×160	Intel Movidius	3,200 KB
224×224	ImageNet	6,272 KB
320×240	QVGA	9,600 KB
640×480	VGA	38,400 KB