Autumn Conference and School On Machine Learning “Fall into ML 2022”

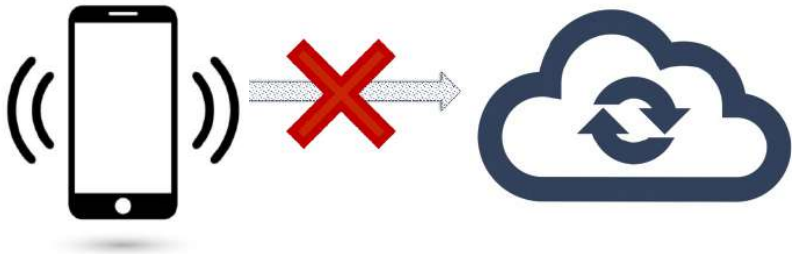
HSE University, Nizhny Novgorod, Laboratory of Algorithms and Technologies for Network Analysis

Moscow 2022

Computer vision on mobile devices

Andrey V. Savchenko
Full. Prof., Leading Researcher
Email: avsavchenko@hse.ru
URL: www.hse.ru/staff/avsavchenko

Difficulties of offline mobile AI



- **Memory!?**

Storage: 64-512 Gb, RAM: 2-12 GB on top devices

- **Computational complexity!?**

MobileNet inference time:

30-60 ms on CPU of top devices, 5-10 ms on their GPUs

vs 10-20 ms on MacBook Pro

Yes, but…

- Maximal size of binary (apk,…) files are usually limited. Typical size of .so libraries on arm64-v8a:

- TensorFlow Lite: ~ 3Mb
- TensorFlow Mobile API: ~ 30Mb
- PyTorch Lite: ~ 35-50 Mb
- OpenCV: ~18 Mb

- **Power consumption!**
- Another platform (Android, iOS). Java/Kotlin/Objective C. Where is Python and its stack?

PyTorch Lite

Sample source code

1. Convert to ptl format

```
traced_script_module = torch.jit.trace(model, example)
traced_script_module_optimized = optimize_for_mobile(traced_script_module)
traced_script_module_optimized._save_for_lite_interpreter(filename+'.ptl')
```

2. Load module

```
Module module= LiteModuleLoader.load(assetFilePath(context, MODEL_FILE));
```

3. Load and pre-process image

```
bitmap=Bitmap.createScaledBitmap(bitmap, width, height, false);
final Tensor inputTensor = TensorImageUtils.bitmapToFloat32Tensor(bitmap,
      TensorImageUtils.TORCHVISION_NORM_MEAN_RGB, TensorImageUtils.TORCHVISION_NORM_STD_RGB);
```

4. Feed image into the model

```
final Tensor outputTensor = module.forward(IValue.from(inputTensor)).toTensor();
```

5. Fetch output

```
final float[] scores = outputTensor.getDataAsFloatArray();
```

TensorFlow Lite (1)

1. Convert to tflite format

2. Load model and allocate input/output buffers

Sample source code

```
converter = tf.compat.v1.lite.TFLiteConverter.from_keras_model_file( input_model_file)
with open ('../app/src/main/assets/'+output_model_file+'.tflite', "wb") as f:
    f.write(converter.convert())
```

```
MappedByteBuffer tfliteModel= FileUtil.loadMappedFile(context,model_path);
tflite = new Interpreter(tfliteModel, new org.tensorflow.lite.gpu.GpuDelegate.Options());
tflite.allocateTensors();
int[] inputShape=tflite.getInputTensor(0).shape();
imageSizeX=inputShape[1];
imageSizeY=inputShape[2];
intValues = new int[imageSizeX * imageSizeY];
imgData =ByteBuffer.allocateDirect(imageSizeX*imageSizeY* inputShape[3]*getNumBytesPerChannel());
imgData.order(ByteOrder.nativeOrder());

int outputCount=tflite.getOutputTensorCount();
outputs=new float[outputCount][1][];
for(int i = 0; i< outputCount; ++i) {
   int[] shape=tflite.getOutputTensor(i).shape();
   int numOFFeatures = shape[1];
   outputs[i][0] = new float[numOFFeatures];
   ByteBuffer ith_output = ByteBuffer.allocateDirect( numOFFeatures* getNumBytesPerChannel());
   ith_output.order(ByteOrder.nativeOrder());
   outputMap.put(i, ith_output);
}
```

TensorFlow Lite (3)

Sample source code

3. Load and pre-process image

```
bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0, bitmap.getWidth(), bitmap.getHeight());
imgData.rewind();
int pixel = 0;
for (int i = 0; i < imageSizeX; ++i) {
   for (int j = 0; j < imageSizeY; ++j) {
      final int val = intValues[pixel++];
      imgData.putFloat((val & 0xFF) - 103.939f);
      imgData.putFloat(((val >> 8) & 0xFF) - 116.779f);
      imgData.putFloat(((val >> 16) & 0xFF) - 123.68f);
   }
}
Object[] inputs={imgData};
```

4. Feed image into the model

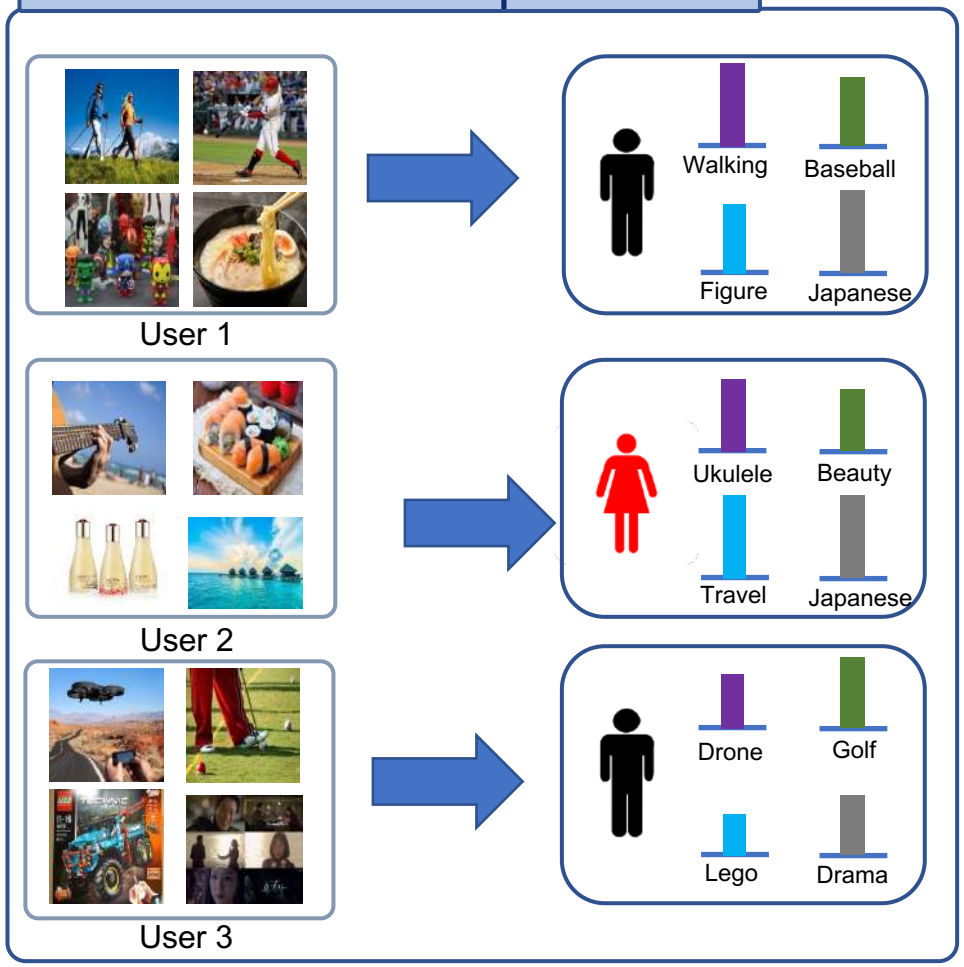
```
tflite.runForMultipleInputsOutputs(inputs, outputMap);
```

5. Fetch output

```
 for(int i = 0; i< outputs.length; ++i) {
    ByteBuffer ith_output=(ByteBuffer)outputMap.get(i);
    ith_output.rewind();
    int len=outputs[i][0].length;
    for(int j=0;j<len;++j){
       outputs[i][0][j]=ith_output.getFloat();
    }
    ith_output.rewind();
 }
```

Example project:
Visual preferences prediction on mobile devices

Generate user profile



Recommendations

Shop : Japanese (Sushi)

Product : Beauty (SKII)

Content : Drama (Hwayugi)

...

Funded by SAMSUNG

Profile of user's interests

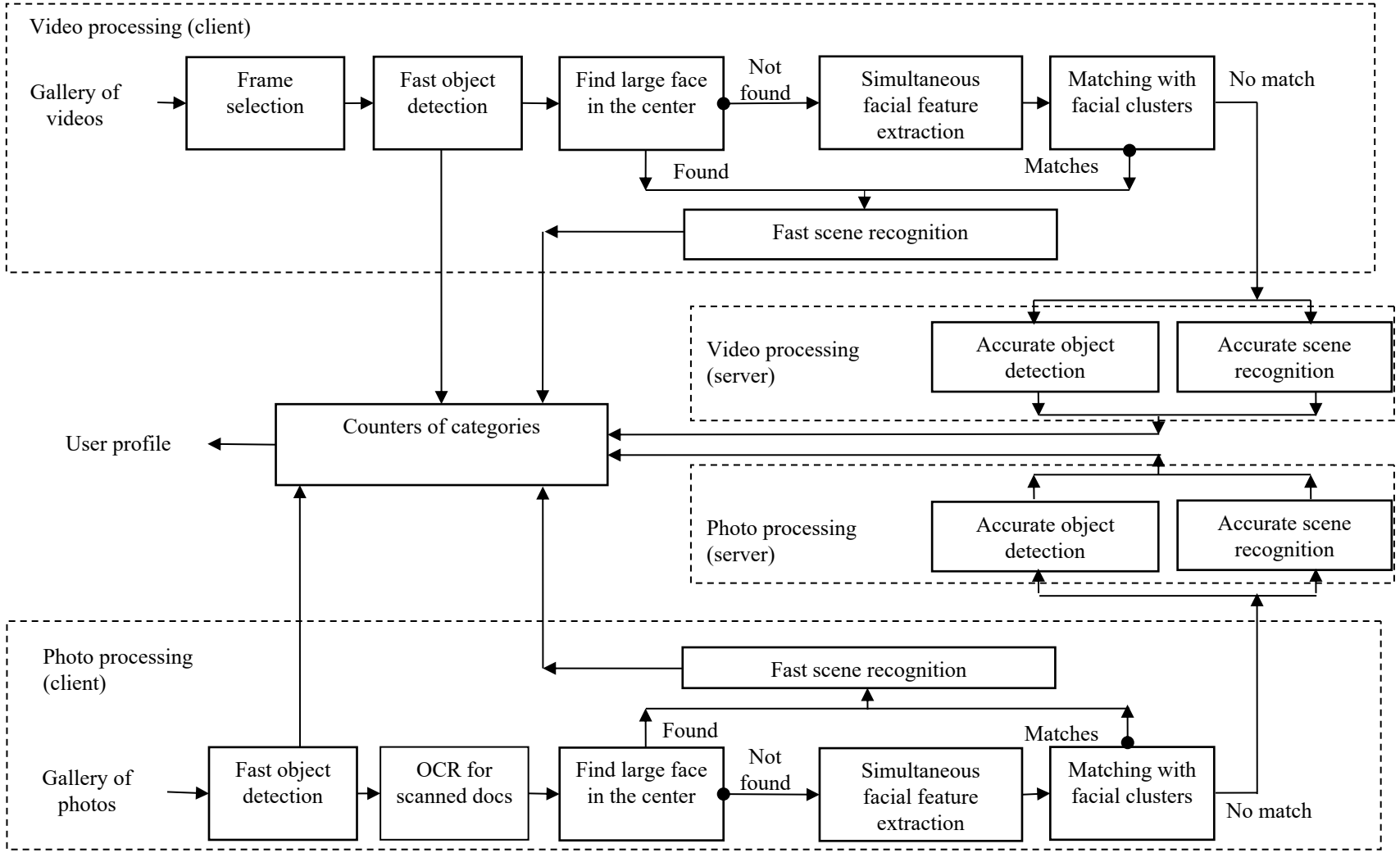
Hobbies	Food code	Pets
Restaurant	Junk food	Dog
Beach	Health - salad and etc.	Cat
Tracks	Sandwich	Fish
Bar	Meat	Horse

Sports	Household Income	Vacation
Fishing	Age	Ski
Golf	Gender	Museum
Diving	Car types	Tracks
Tennis	household	Monuments

Public photos

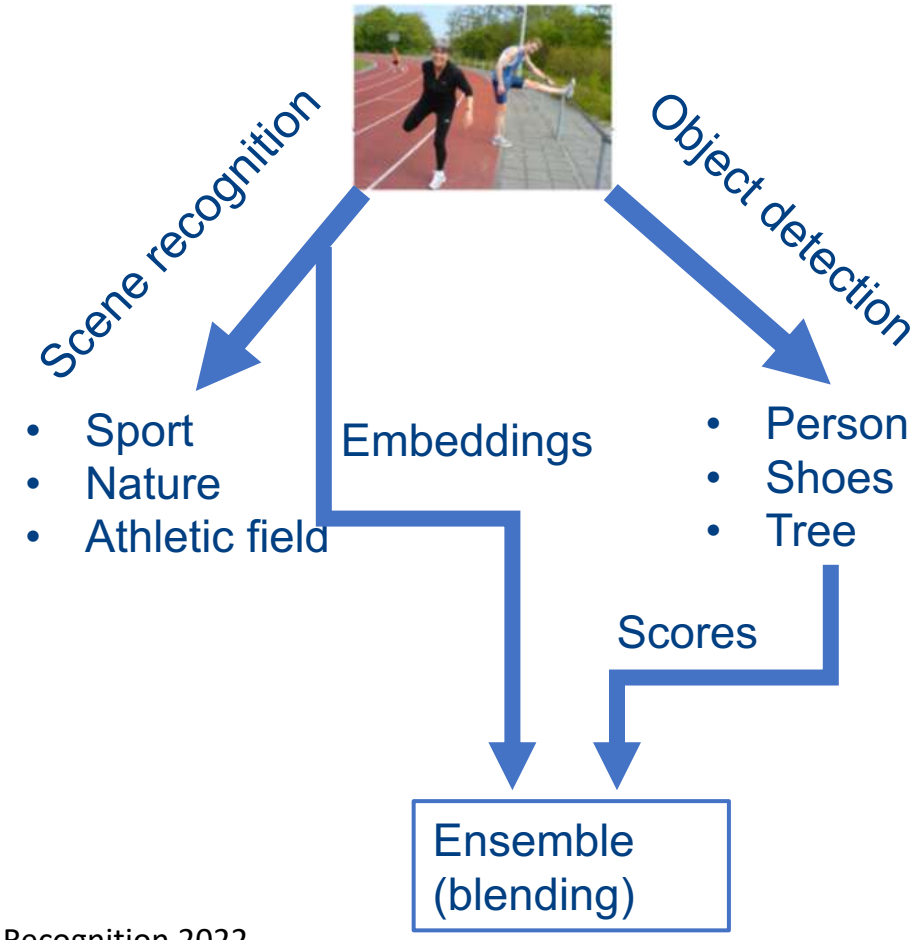
Private photos

Pipeline



Savchenko et al, Pattern Recognition 2022

Visual representations of photos

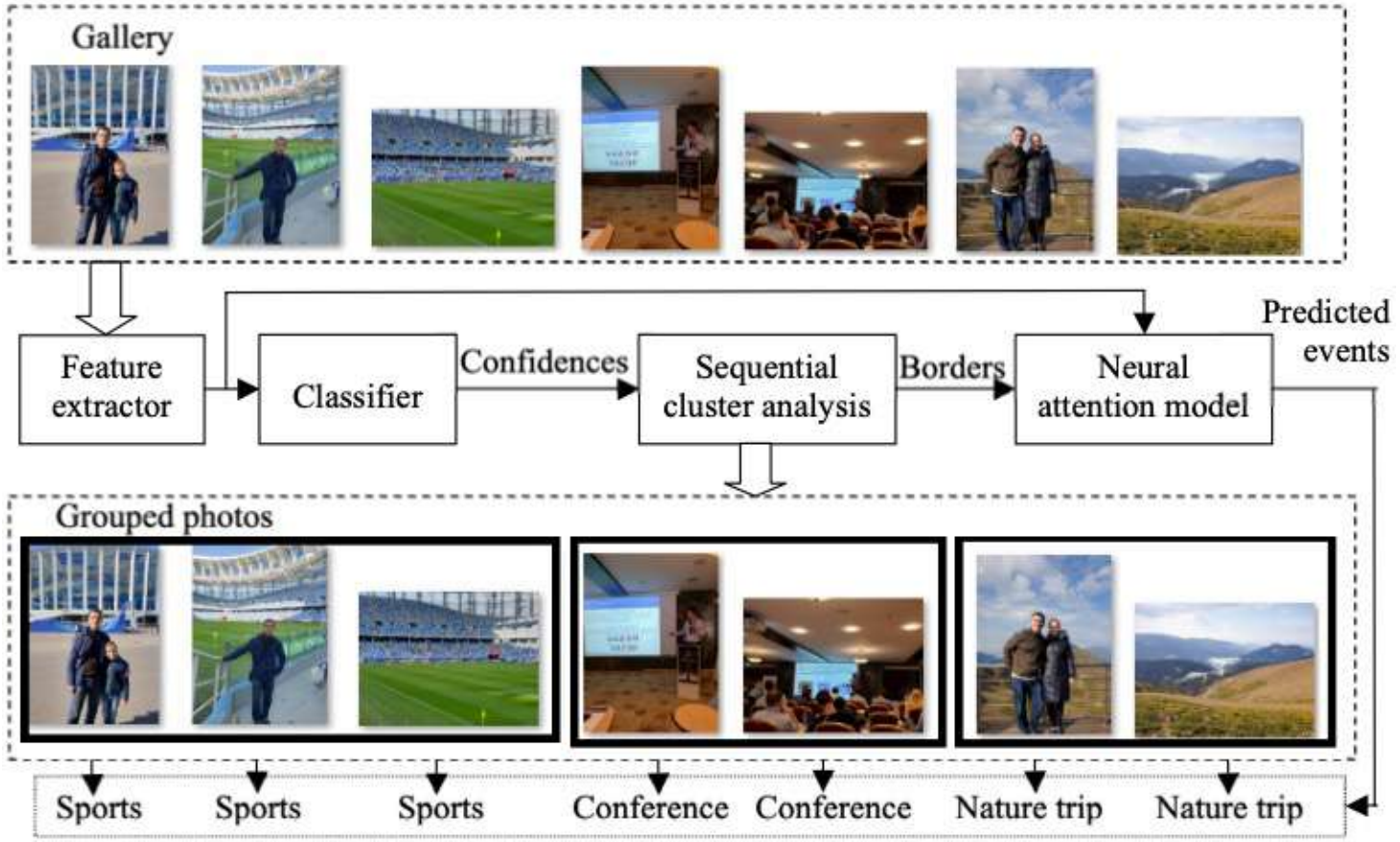


Savchenko et al, Pattern Recognition 2022

Initialize lists $T_f := [], T_o := []$
for each training image **do**
  Compute features $\mathbf{f}$ by feeding the image into a scene CNN
  Append $\mathbf{f}$ to the list T_f
  Compute maximal confidences of detected objects $\mathbf{o}$ by feeding the image into object detector
  Append $\mathbf{o}$ to the list T_o
end for
Train classifiers $\mathcal{C}_f, \mathcal{C}_o$ using T_f and T_o
for each validation image **do**
  Compute scene features $\mathbf{f}$
  Predict C-dimensional confidence scores $\mathbf{cs}_f$ using classifier $\mathcal{C}_f$
  Compute maximal confidences of detected objects $\mathbf{o}$
  Predict C-dimensional confidences $\mathbf{cs}_o$ using $\mathcal{C}_o$
end for
Assign $\alpha^* := 0$
for all possible weights $w_f \in [0, 1]$ **do**
  for each validation image **do**
    Compute confidences $[cs_1, ..., cs_C] := w_f \mathbf{cs}_f + (1 - w_f)\mathbf{cs}_o$
    Perform blending to predict the best class $c^* := \underset{c \in \{1,...,C\}}{\operatorname{argmax}}\, cs_c$
  end for
  Compute accuracy α using predictions for all validation images
  if $\alpha^* < \alpha$ **then**
    Assign $\alpha^* := \alpha, w_f^* := w_f$
  end if
end for
return classifiers $\mathcal{C}_f, \mathcal{C}_o$ and their weights $w_f^*, (1 - w_f^*)$

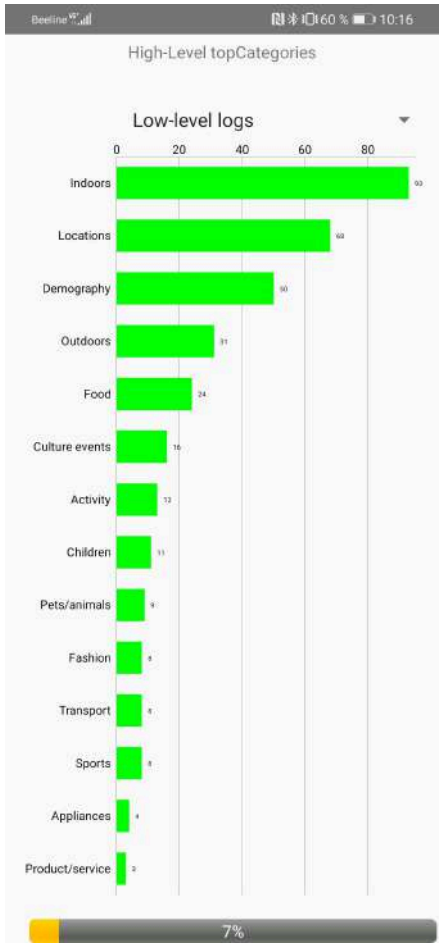
Processing of a sequence of photos

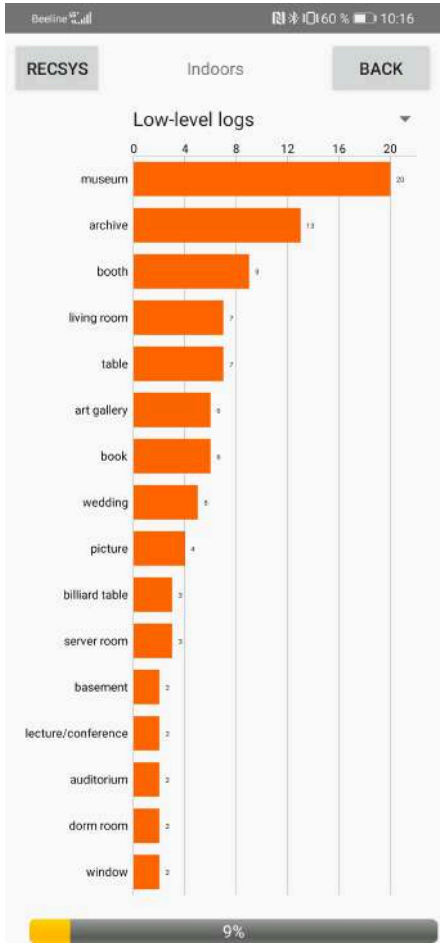
1. Combine consecutive photos by using distance $\rho(p_t, p_{t-1})$ between L_2-normed classifier's scores for CNN embeddings of neighbor photos
2. Learn distance threshold by random permutation of albums from the training set
3. Prior art (Wang, Y. et al., BMVC17) requires the input image set to represent one album. Our method automatically assigns album's borders from a gallery
4. Neural attention for final decision-making

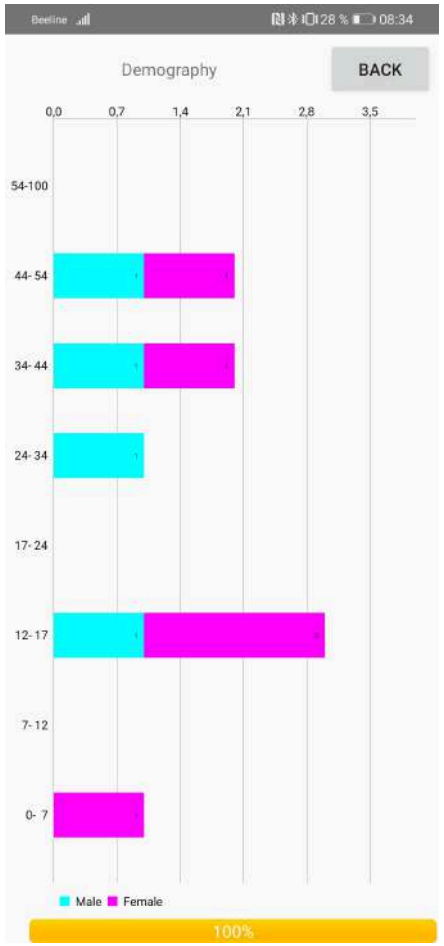


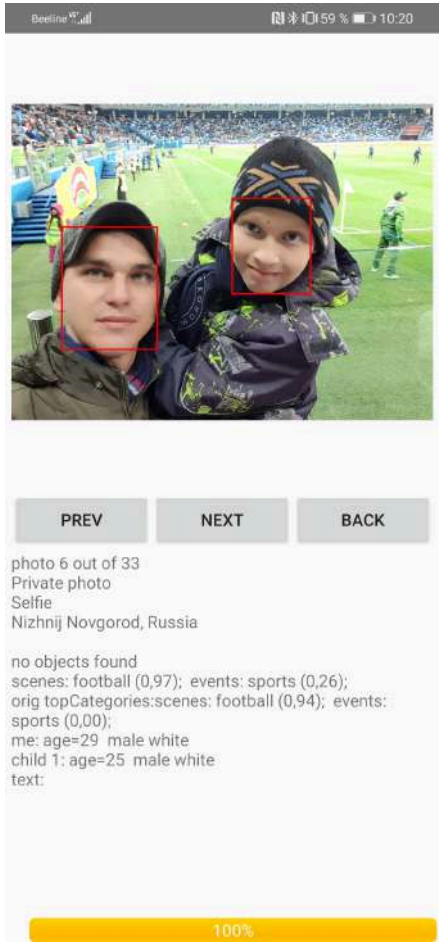
Savchenko et al, IJCNN 2020

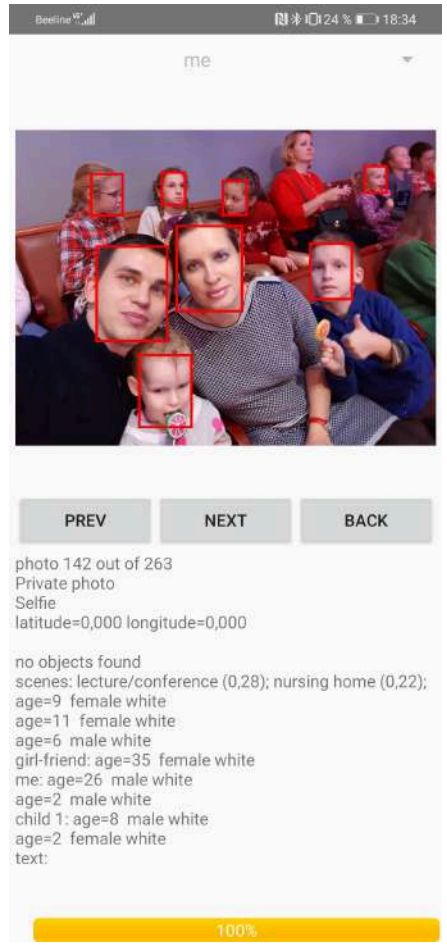
Mobile demo application







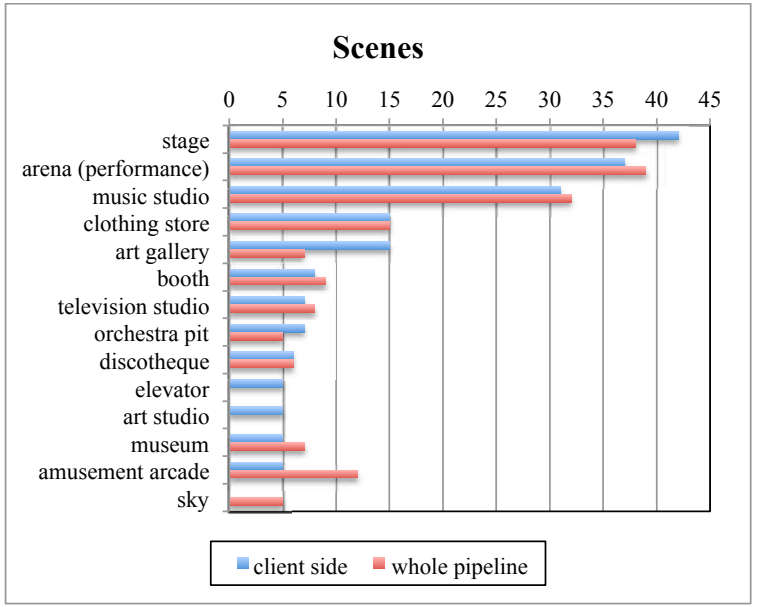




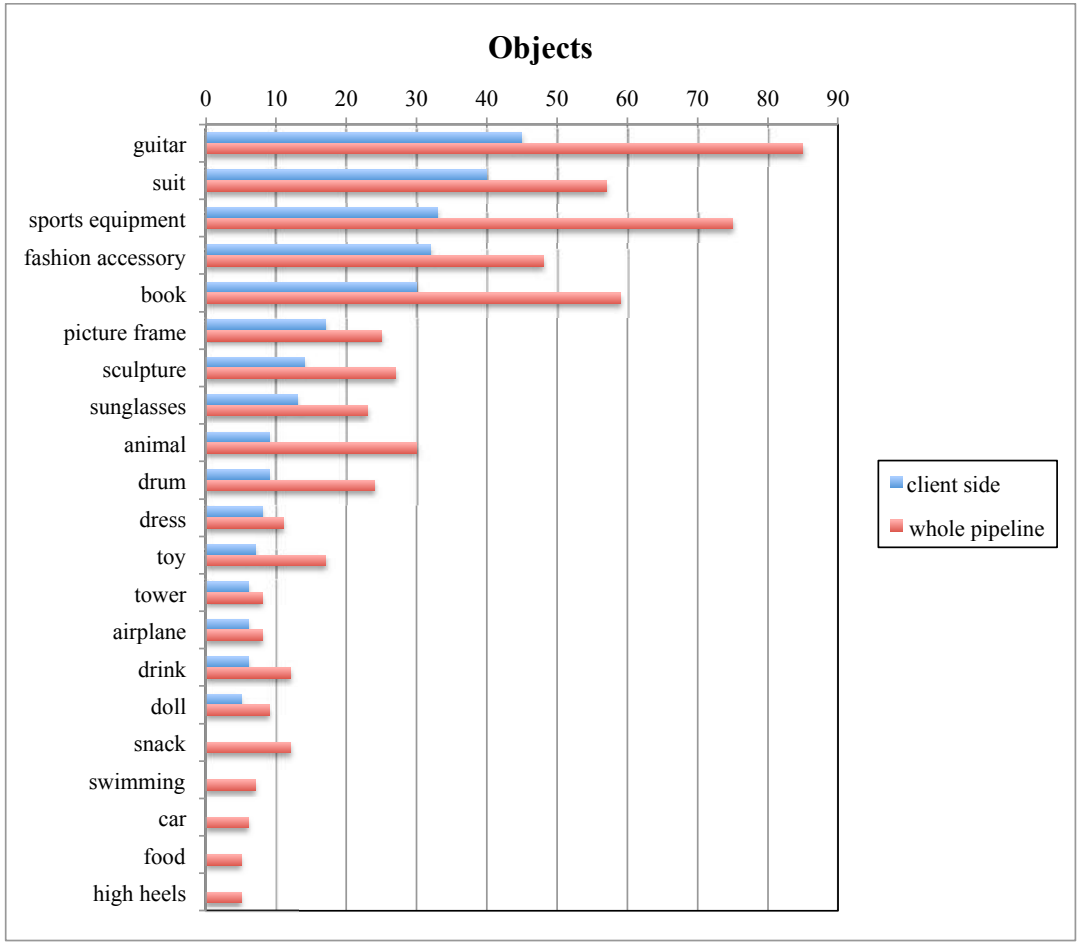
https://drive.google.com/drive/folders/1rQkJZifq_89pu0sT_UnYXziuxutTpNEN

Example: The Rolling Stone profile

Scene recognition



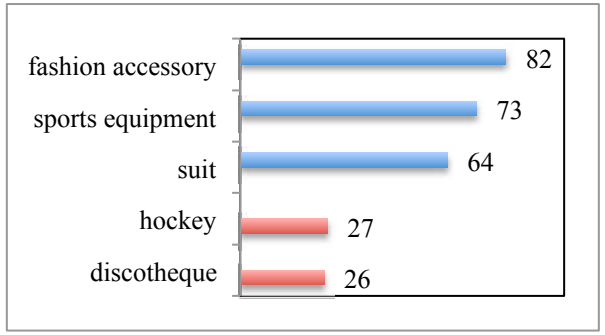
Object detection



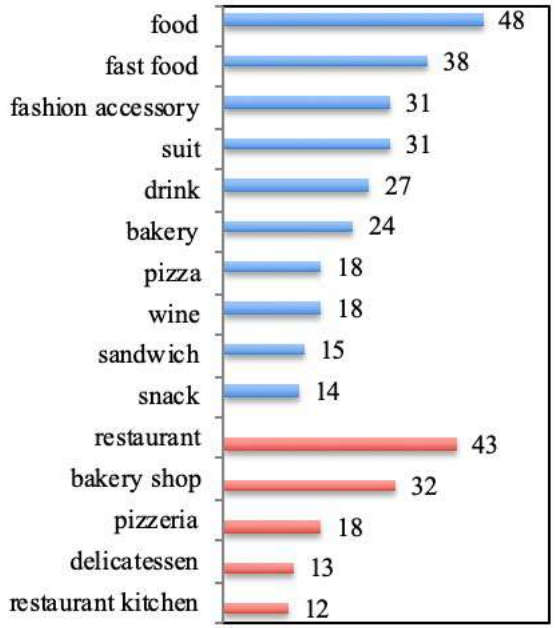
User profiles from a social net

Alex Ovechkin

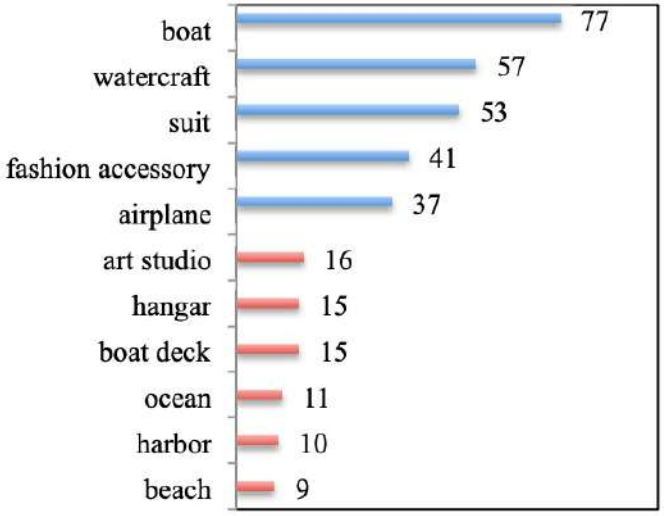
...



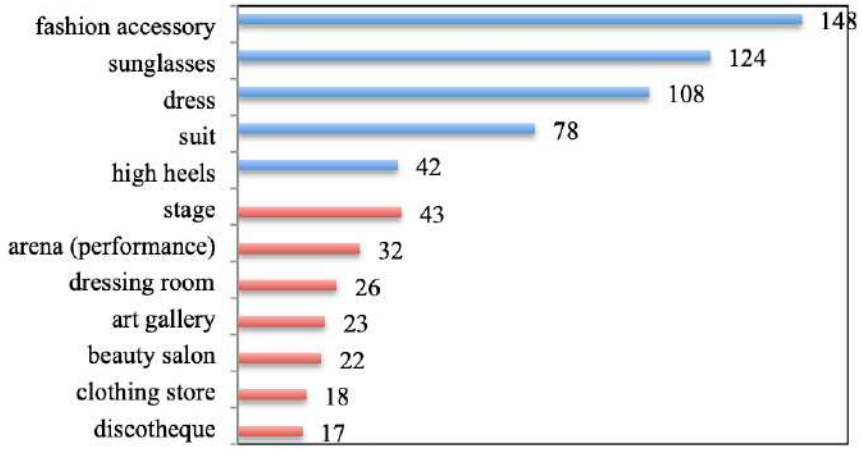
Gordon Ramsay



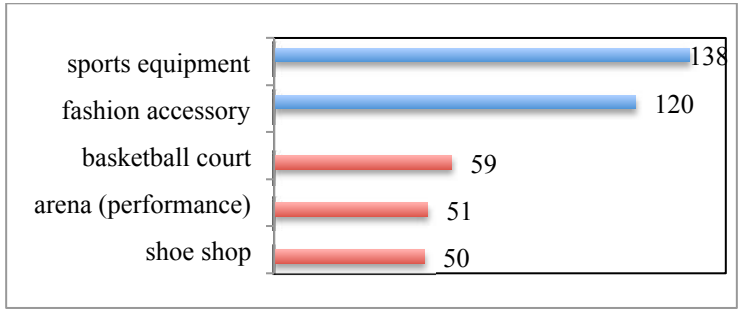
Fedor Konyukhov



Beyonce



LeBron James



Neural networks for mobile devices

Deep neural networks for image recognition

Model	Top-1 Accuracy	# Params	Depth	Inference time (ms) (CPU)
VGG16	71.3%	138.4M	16	69.5
ResNet50	74.9%	25.6M	107	58.2
ResNet152	76.6%	60.4M	311	127.4
InceptionV3	77.9%	23.9M	189	42.2
InceptionResNetV2	80.3%	55.9M	449	130.2
DenseNet121	75.0%	8.1M	242	77.1
DenseNet201	77.3%	20.2M	402	127.2
NASNetMobile	74.4%	5.3M	389	27.0
NASNetLarge	82.5%	88.9M	533	344.5

Deep models are accurate but have many parameters:
long training and inference!

https://keras.io/applications/

Depthwise separable convolution

Convolution

256 kernels 5x5x3, 8x8 shifts. Totally, 256x3x5x5x8x8 = 1,228,800 multiplications

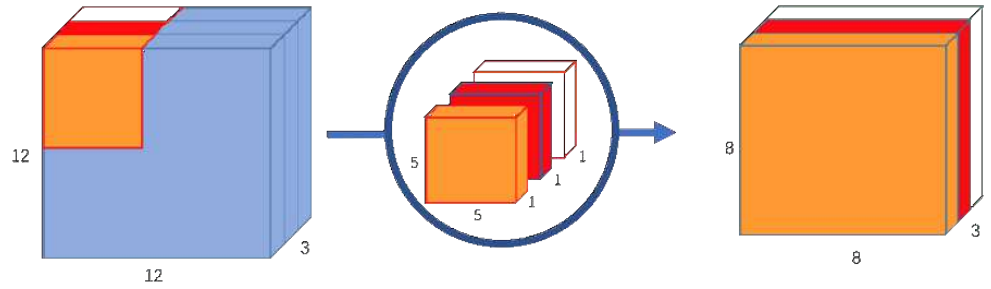
Separable convolution

1) Depthwise: 3 kernels 5x5x1 and 3x5x5x8x8 = 4,800 multiplications

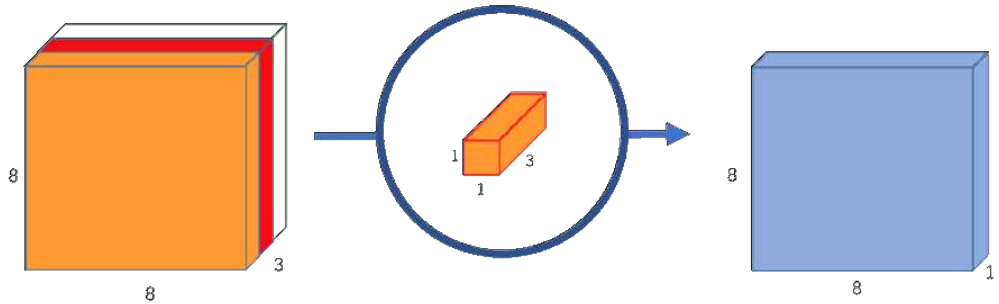
2) Pointwise: 256 kernels 1x1x3, 256x1x1x3x8x8 = 49,152 multiplications

Totally: 53,952 multiplications

Depthwise convolution



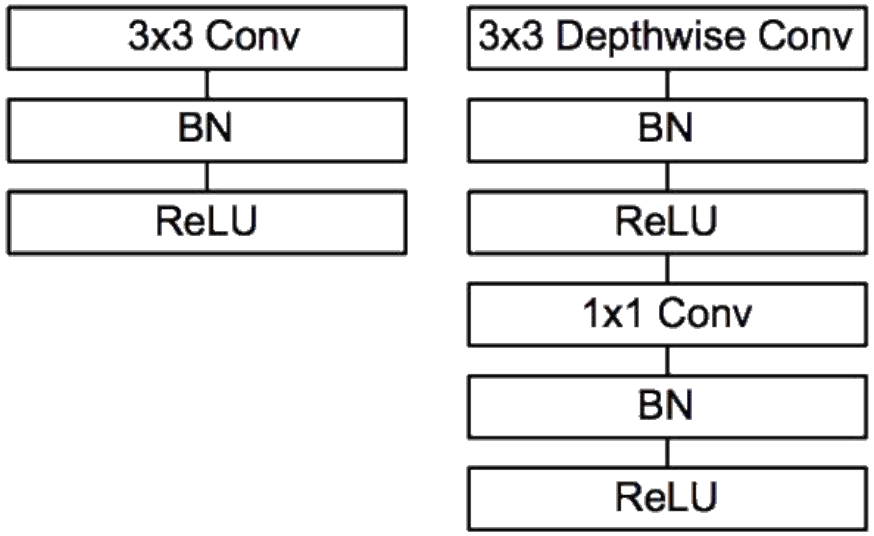
Pointwise convolution



Chi-Feng Wnag/ towardsdatascience.com

MobileNet v1

Depthwise convolution



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

- 28 layers (ConvLayer with stride 2 instead of max-pooling)
- ImageNet Top-5 Error rate: 12.81%
- 4.2M weights

Andrew Howard et al. / arxiv 1704.04861

MobileNet v2

Residual block

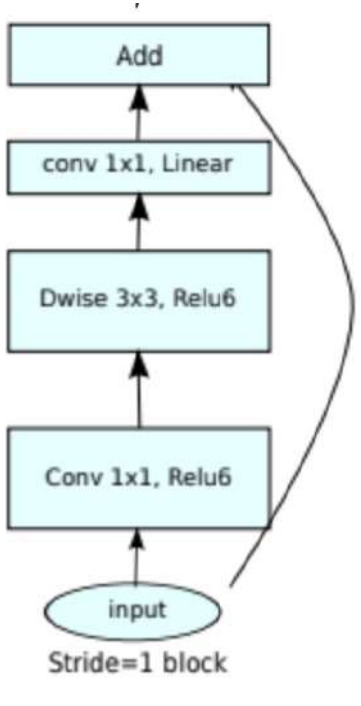
```
def residual_block(x, squeeze=16, expand=64):
  m = Conv2D(squeeze, (1,1), activation='relu')(x)
  m = Conv2D(squeeze, (3,3), activation='relu')(m)
  m = Conv2D(expand, (1,1), activation='relu')(m)
  return Add()([m, x])
```

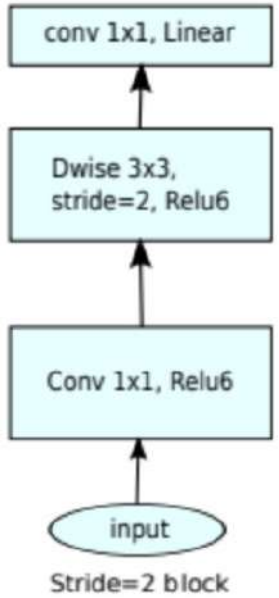
Inverted residual block

```
def inverted_residual_block(x, expand=64, squeeze=16):
  m = Conv2D(expand, (1,1), activation='relu')(x)
  m = DepthwiseConv2D((3,3),    activation='relu')(m)
  m = Conv2D(squeeze, (1,1), activation='relu')(m)
  return Add()([m, x])
```

Input	Operator	Output
h × w × k	1x1 conv2d, ReLU6	h × w × (tk)
h × w × (tk)	Strided Dwise 3x3, ReLU6	(h/s) × (w/s) × k
(h/s) × (w/s) × k	linear 1x1 conv2d	(h/s) × (w/s) × k’

Bottleneck residual block





Bottleneck layer

Depthwise conv layer

Expansion layer (t=6 times)

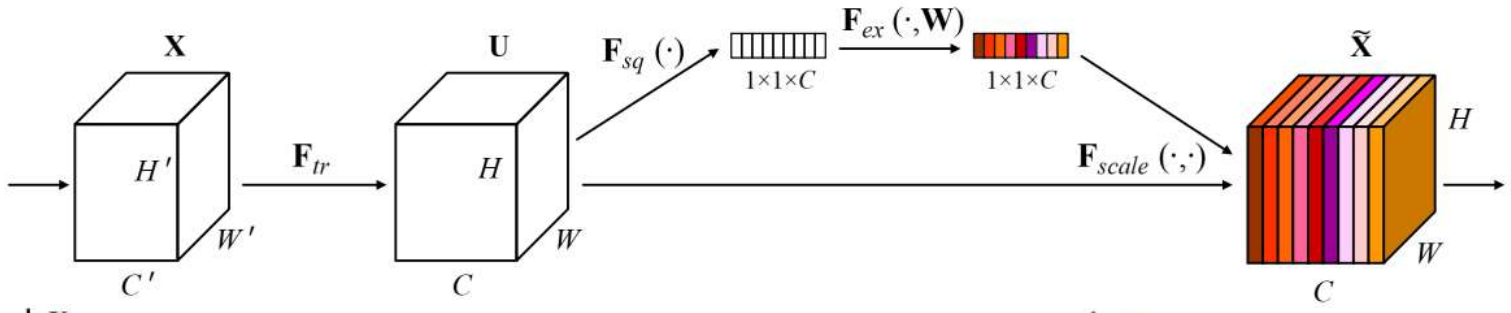
- ImageNet Top-1 Accuracy: 74.7% vs 70.6% for MobileNet v1
- 6…9M weights

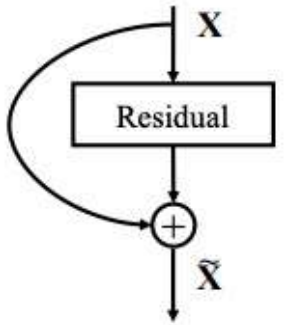
Mark Sandler et al. / arxiv 1801.04381

Squeeze-and-Excitation Networks (SENet)

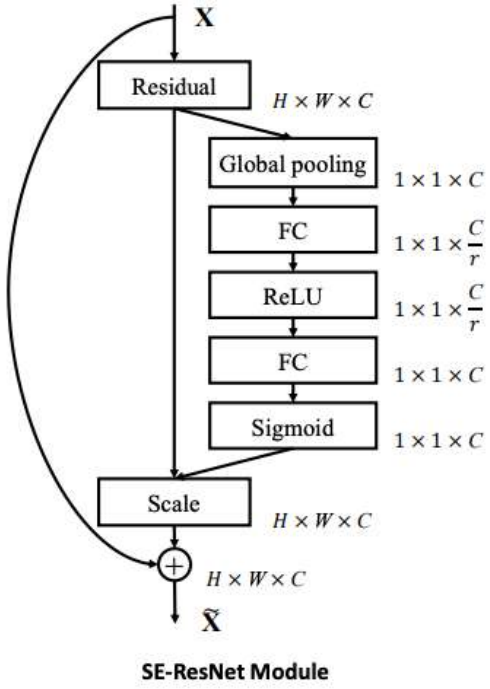
Squeeze-and-Excitation block

```
def se_block(in_block, ch, ratio=16):
 x = GlobalAveragePooling2D()(in_block)
 x = Dense(ch//ratio, activation='relu')(x)
 x = Dense(ch, activation='sigmoid')(x)
 return multiply()([in_block, x])
```

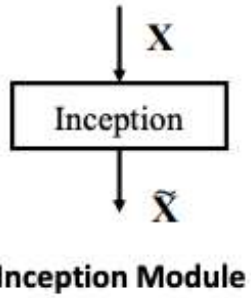




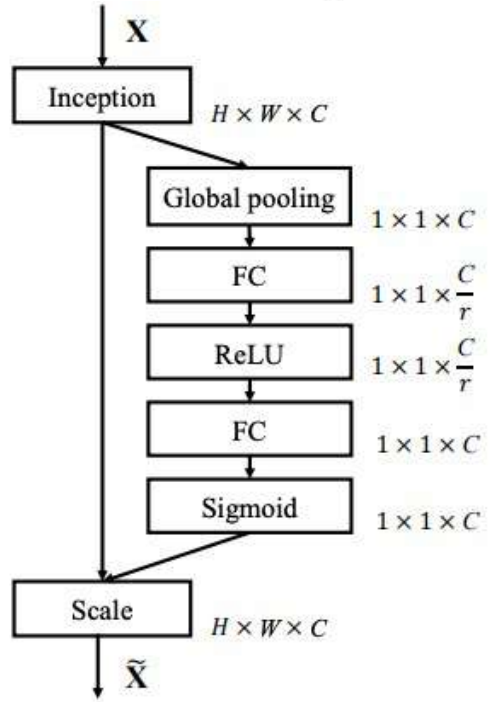
ResNet Module



SE-ResNet Module



Inception Module



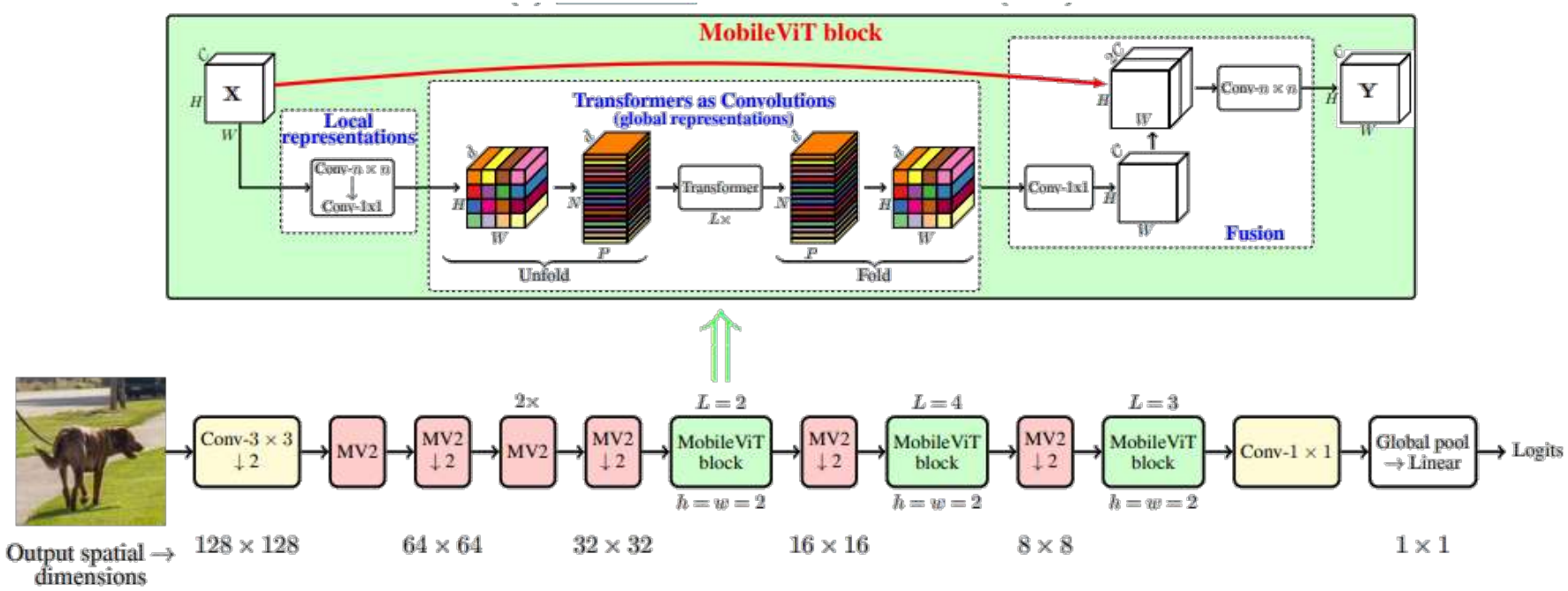
Hu J. et al / CVPR 2018

Mobile Visual Transformer (MobileViT)

Convolution:
(1) unfolding,
(2) matrix multiplication (to learn local representations)
(3) folding

MobileViT replaces the local matrix multiplication) with deeper global processing (a stack of transformer layers)

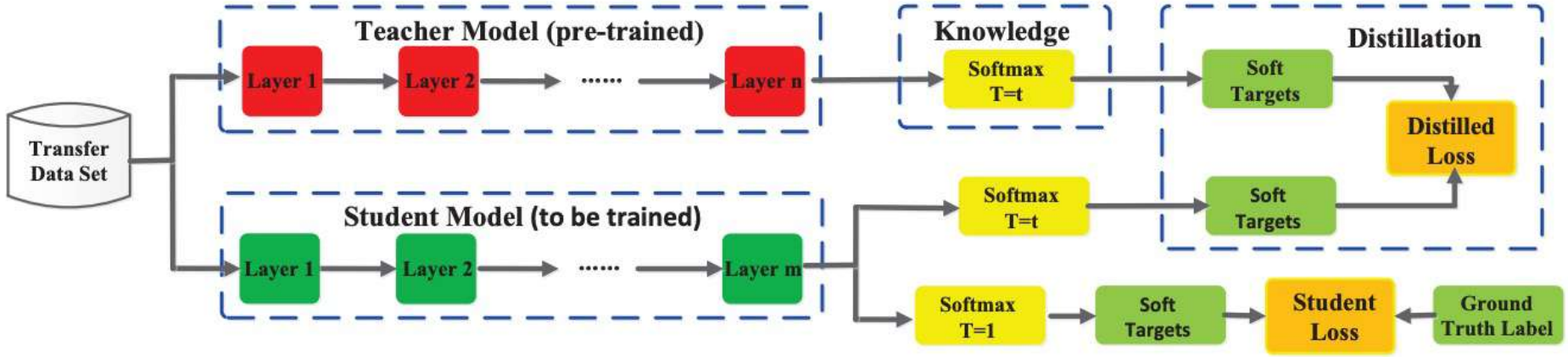
Sequence of convolution, MV2 (MobileNetV2) and MobileViT blocks



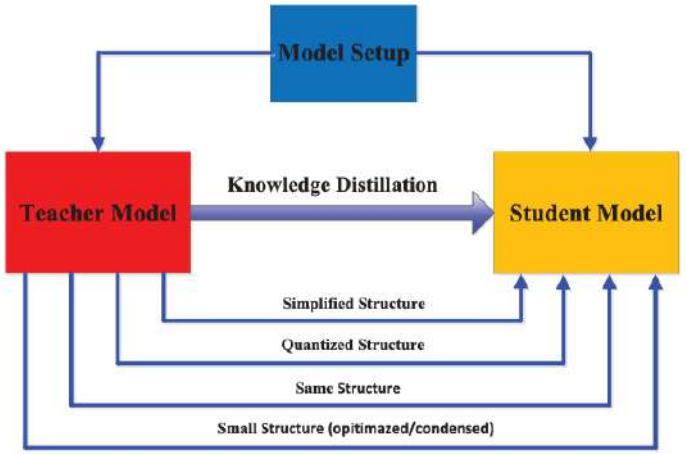
S. Mehta and M. Rastegari (Apple) / arxiv 2110.02178

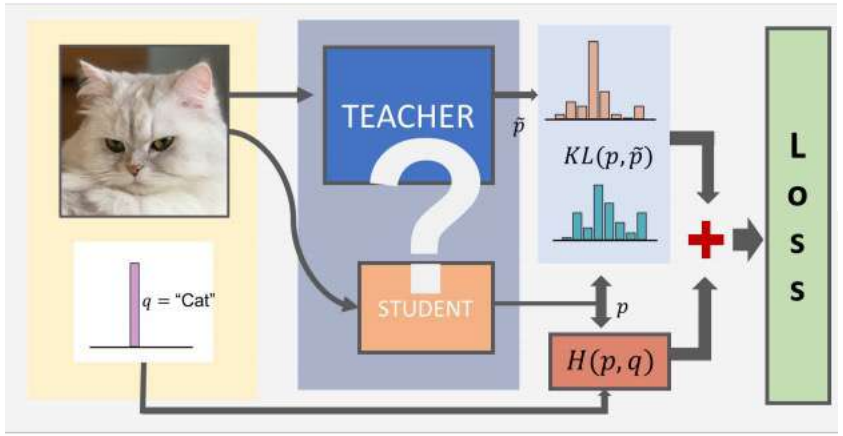
Optimization of neural network architectures

Knowledge distillation



Types of distillation





Loss function

$$\mathcal{L} = \alpha \mathcal{L}_{cls} + (1-\alpha)\mathcal{L}_{KD}$$

$$\mathcal{L}_{KD} = -\tau^2 \sum_k \tilde{p}_k^t(x) \log \tilde{p}_k^s(x)$$

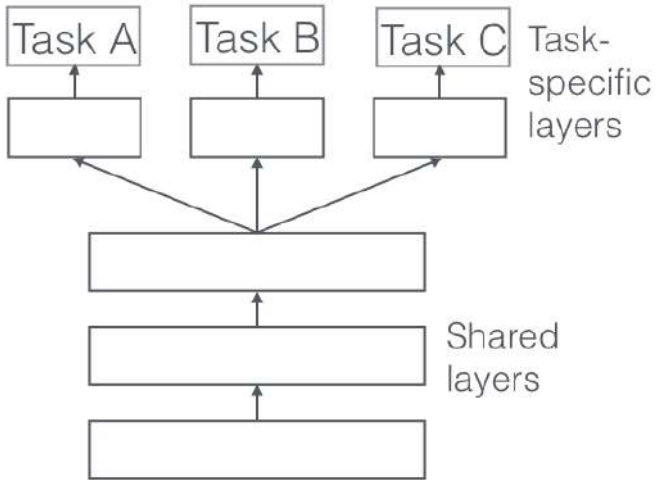
$$\tilde{p}_k^t(x) = \frac{e^{s_k^t(x)/\tau}}{\sum_j e^{s_j^t(x)/\tau}}$$

https://arxiv.org/pdf/2006.05525.pdf

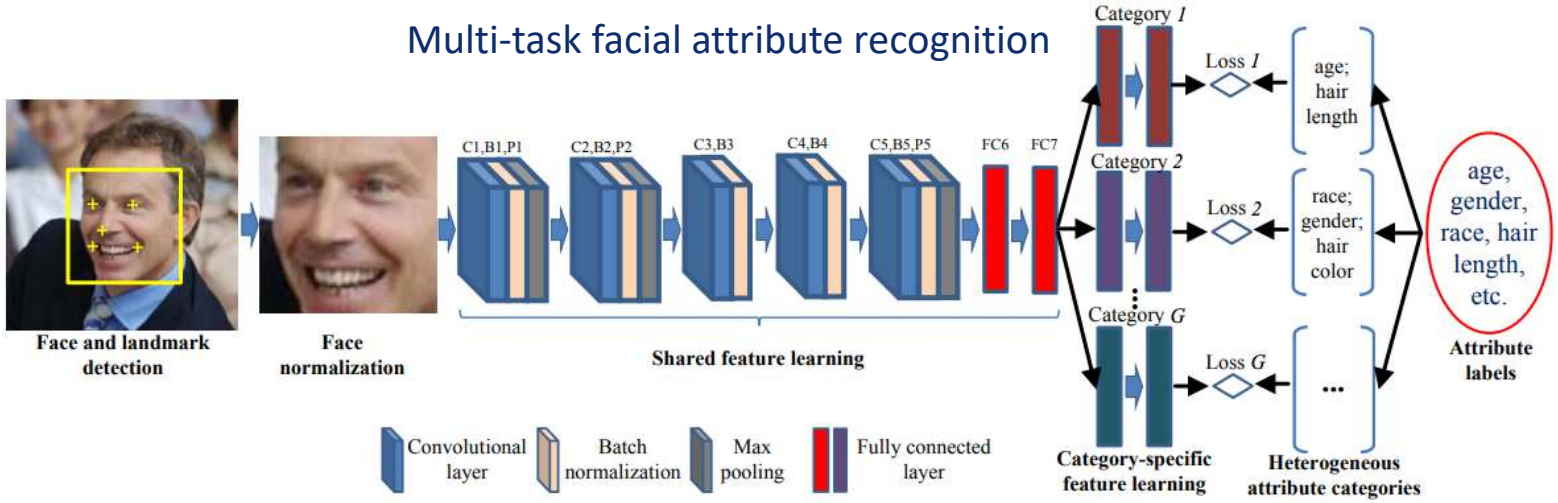
Jang Hyun Cho & B. Hariharan, ICCV 2019

Multi-task learning

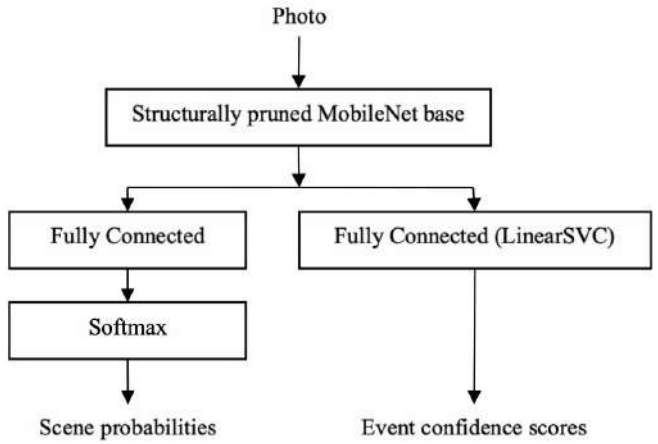
Hard parameter sharing



Multi-task facial attribute recognition



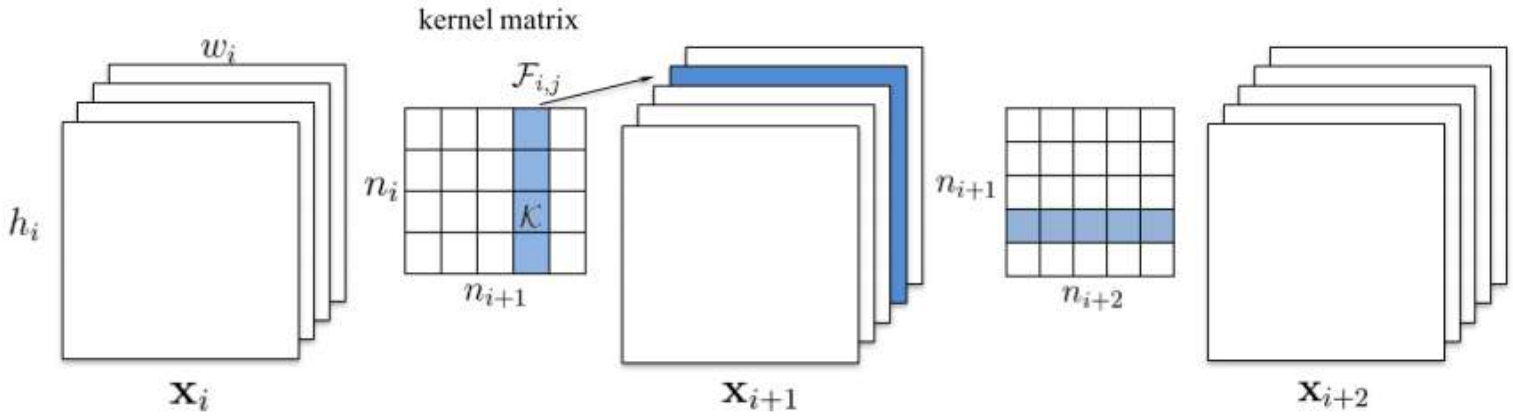
Multi-task scene/event recognition



Han et al / PAMI 2017

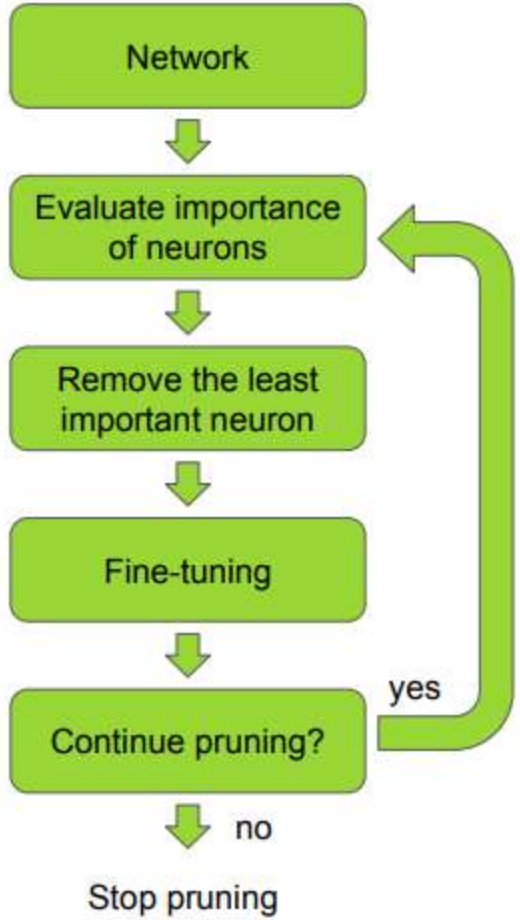
Structural pruning (1)

Remove filters in convolution layer



How to choose channels to remove?

$$\min_{\mathcal{W}'} \left| \mathcal{C}(\mathcal{D}|\mathcal{W}') - \mathcal{C}(\mathcal{D}|\mathcal{W}) \right| \quad \text{s.t.} \quad ||\mathcal{W}'||_0 \leq B,$$



Average Percentage of Zeros (APoZ) (https://arxiv.org/pdf/1607.03250.pdf)

$$APoZ_c^{(i)} = APoZ(O_c^{(i)}) = \frac{\sum_k^N \sum_j^M f(O_{c,j}^{(i)}(k) = 0)}{N \times M}$$

Taylor expansion

$$\Theta_{TE}(z_l^{(k)}) = \left| \frac{1}{M} \sum_m \frac{\delta C}{\delta z_{l,m}^{(k)}} z_{l,m}^{(k)} \right|,$$

https://arxiv.org/pdf/1608.08710.pdf

https://arxiv.org/pdf/1611.06440.pdf

Structural pruning (2)

Example for scene recognition model

		MobileNet v2 (α=1.0)			MobileNet v2 (α=1.4)	
		Original	Structural pruning (25%)	Structural pruning (40%)	Original	Structural pruning (40%)
Size, Mb		11.1	8.3	6.7	20.3	12.2
Inference time, ms	MacBook Pro 2015	18	14	12	31	29
	Galaxy Tab S4	85-105	70-90	60-80	150-180	130-150
	Galaxy S9+	70-90	55-80	50-70	110-160	100-120
All 388 labels	Top-1 accuracy, %	50.7	49.8	48.7	51.3	49.5
	Top-5 accuracy, %	80.4	79.8	79.0	80.7	79.3
	Precision, %	57.5	56.2	54.9	58.0	56.1
	Recall, %	46.7	46.2	45.4	47.1	45.6
Only 323 reliable labels	Top-1 accuracy, %	57.5	56.4	55.3	58.2	56.3
	Top-5 accuracy, %	87.3	86.7	86.2	88.1	86.9

Quantization

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, Microcontrollers
Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

1. If limit to Floating Points (e.g., GPU)? -> convert to FP16
2. Else if no representative dataset? -> Parameters are quantized to int, model executes mixture of FP32 and int
3. Else -> parameters and activations are quantized to int. if limit to int8? -> model executes with int computations only
4. Else -> model executes with int if possible, else FP32

https://www.tensorflow.org/lite/performance/post_training_quantization

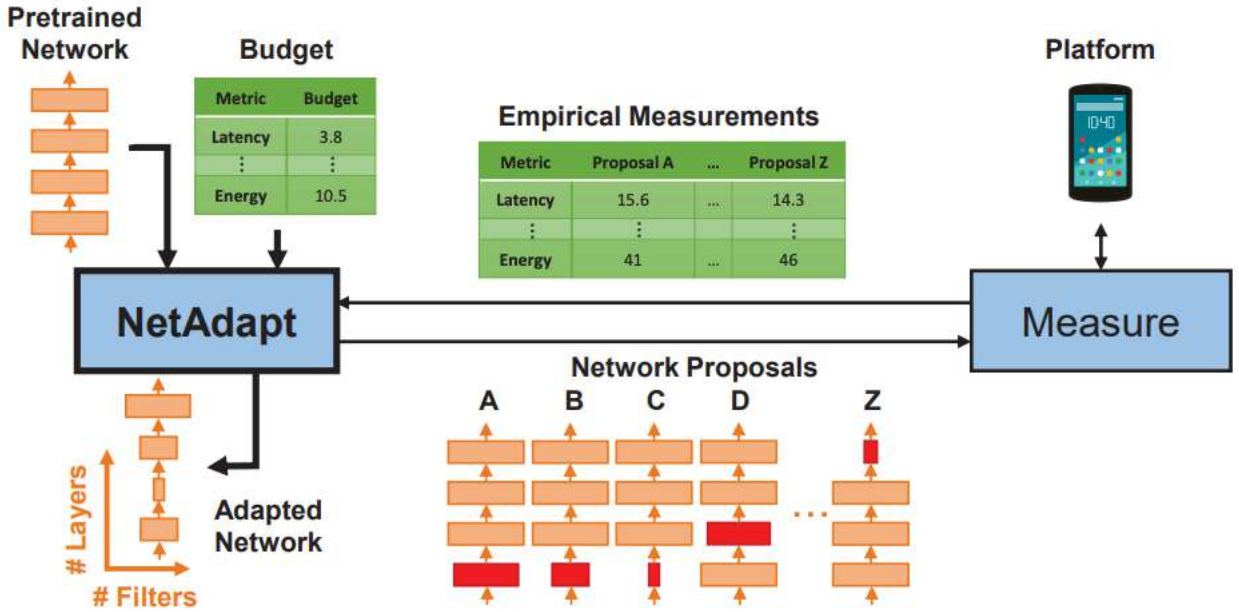
Typical quantization results (Flowers dataset):

Technique	Accuracy change	Technique
Original MobileNet v2	98.30%	Fine-tuning
TfLite (Fp32)	+0.07 98.37	TfLite (Fp32)
Fp16	+0.07 98.37	Fp16
int8	-12.74 85.56	int8
Post training: Weight quantization (int8)	-0.48 97.82	Post training: Weight quantization (int8)

https://gist.github.com/NobuoTsukamoto/b42128104531a7612e5c85e246cb2dac

https://colab.research.google.com/gist/sayakpaul/468f70d075e5608bfebeb9ba639265b6/qat-bad-accuracy.ipynb

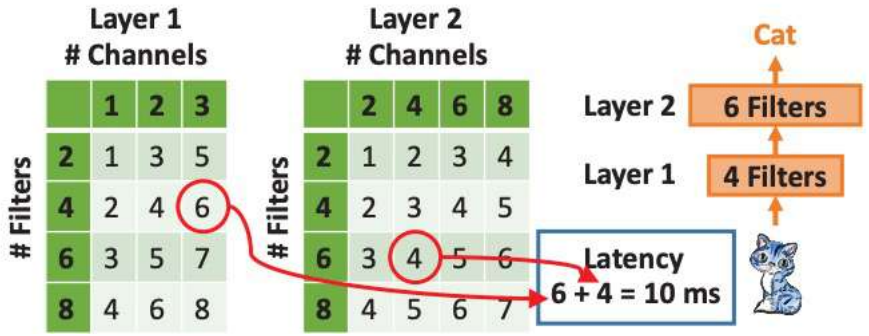
Neural Architecture Search (NAS). NetAdapt

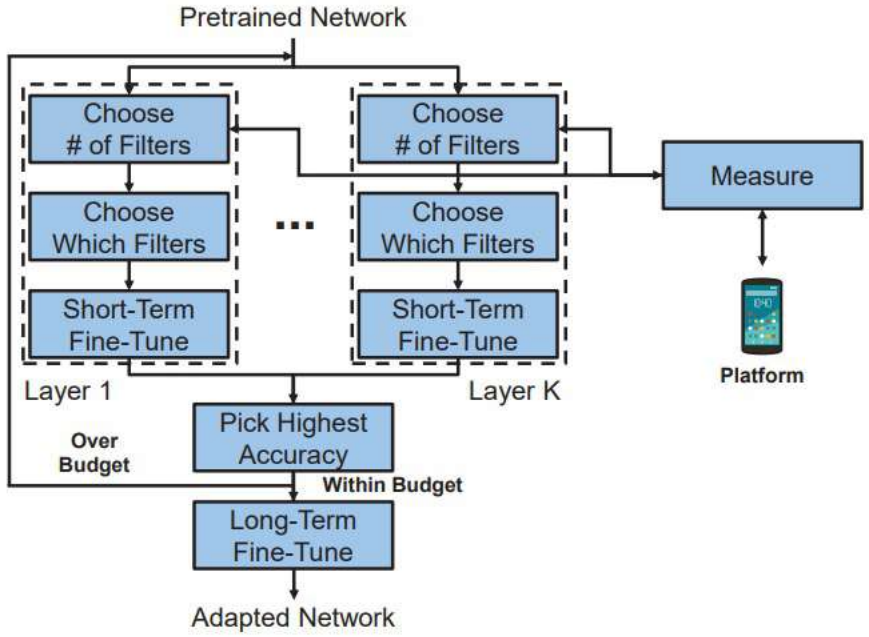


$$\underset{Net}{\text{maximize}} \quad Acc(Net)$$
$$\text{subject to} \quad Res_j(Net) \leq Bud_j,\ j = 1, \ldots, m,$$

$$\underset{Net_i}{\text{maximize}} \quad Acc(Net_i)$$
$$\text{subject to} \quad Res_j(Net_i) \leq Res_j(Net_{i-1}) - \Delta R_{i,j},\ j = 1, \ldots, m,$$

fast resource consumption estimation

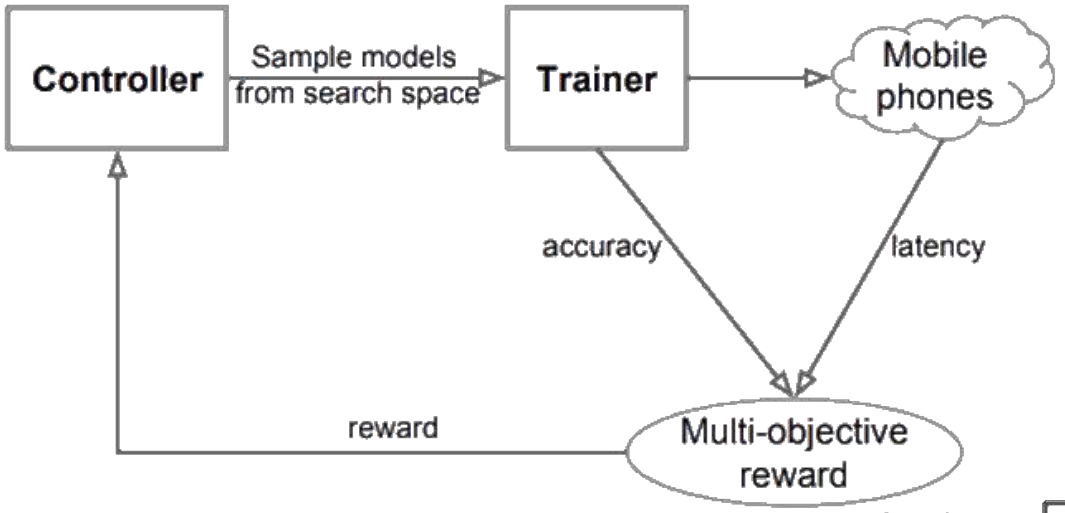




Tien-Ju Yang et al, ECCV 2018

MnasNet

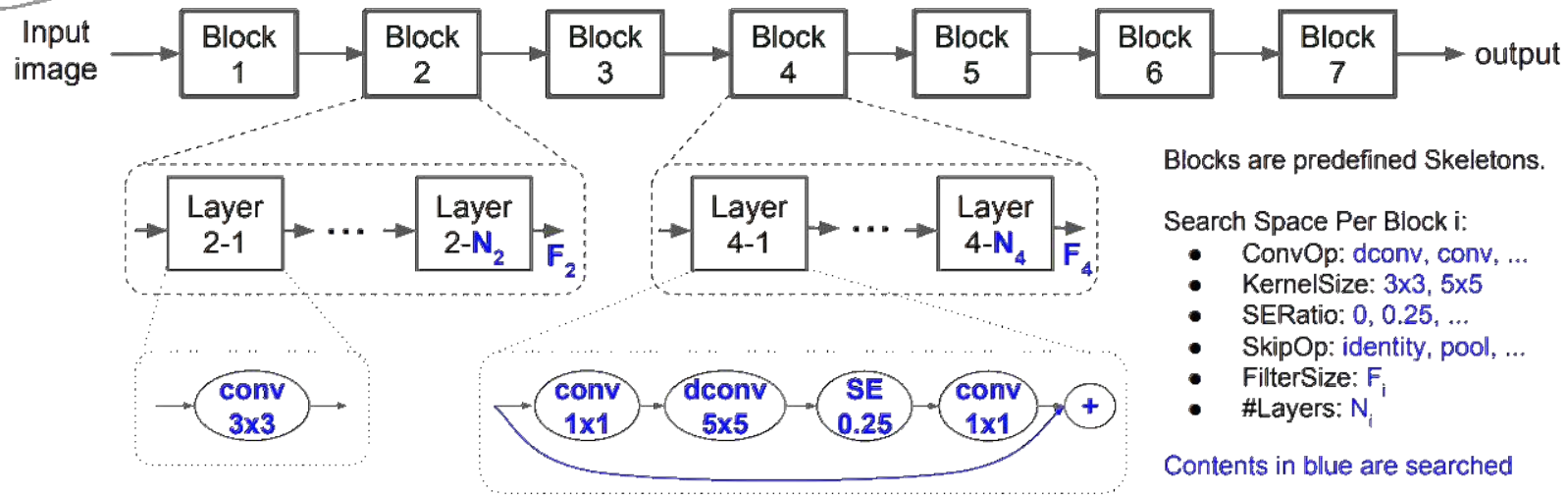
Platform-aware NAS



Max Acc(Net)
subject to Lat(Net)<T

Reward:
Max Acc(Net) x (Lat(Net)/T)^{w}

Reinforcement learning to find Pareto optimal solutions



Blocks are predefined Skeletons.

Search Space Per Block i:

- ConvOp: dconv, conv, ...
- KernelSize: 3x3, 5x5
- SERatio: 0, 0.25, ...
- SkipOp: identity, pool, ...
- FilterSize: F_i
- #Layers: N_i

Contents in blue are searched

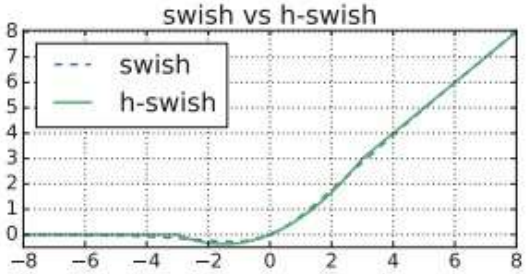
Mingxing Tan et al. /CVPR 2019

MobileNet v3

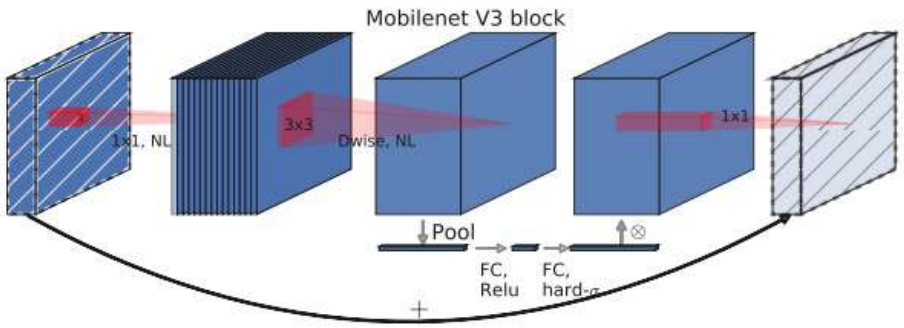
- Platform-aware NAS + NetAdapt for layer-wise search
- MobileNetV2 + Squeeze-and-Excite (apply the squeeze and excite in the residual layer)
- New activation functions

$$\text{swish}\, x = x \cdot \sigma(x)$$

$$\text{h-swish}[x] = x \frac{\text{ReLU6}(x+3)}{6}$$



MobileNetV2 + Squeeze-and-Excite



Andrew Howard et al. / arxiv 1905.02244

MobileNetV3-Small

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

EfficientNet

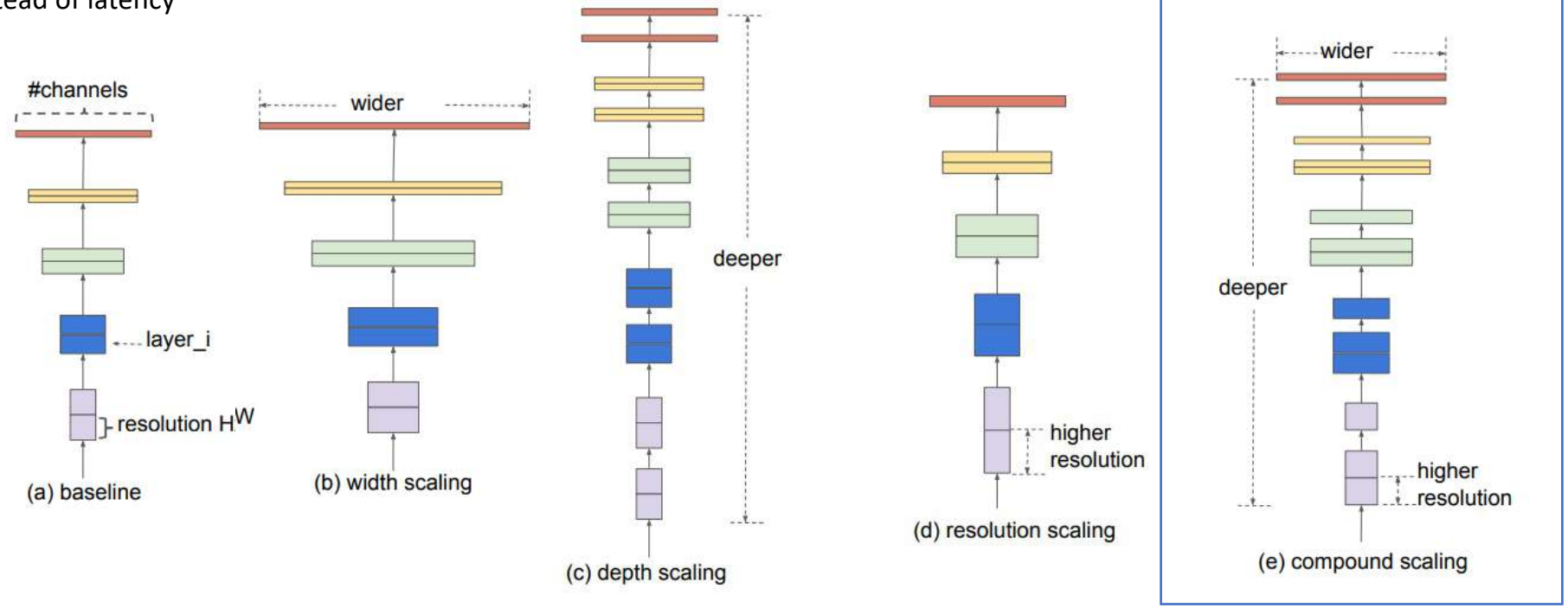
•Platform-aware NAS + NetAdapt for layer-wise search
AutoML (MnasNet), but optimize FLOPS instead of latency

•MBConv - mobile inverted bottleneck

•Compound scaling

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Model scaling

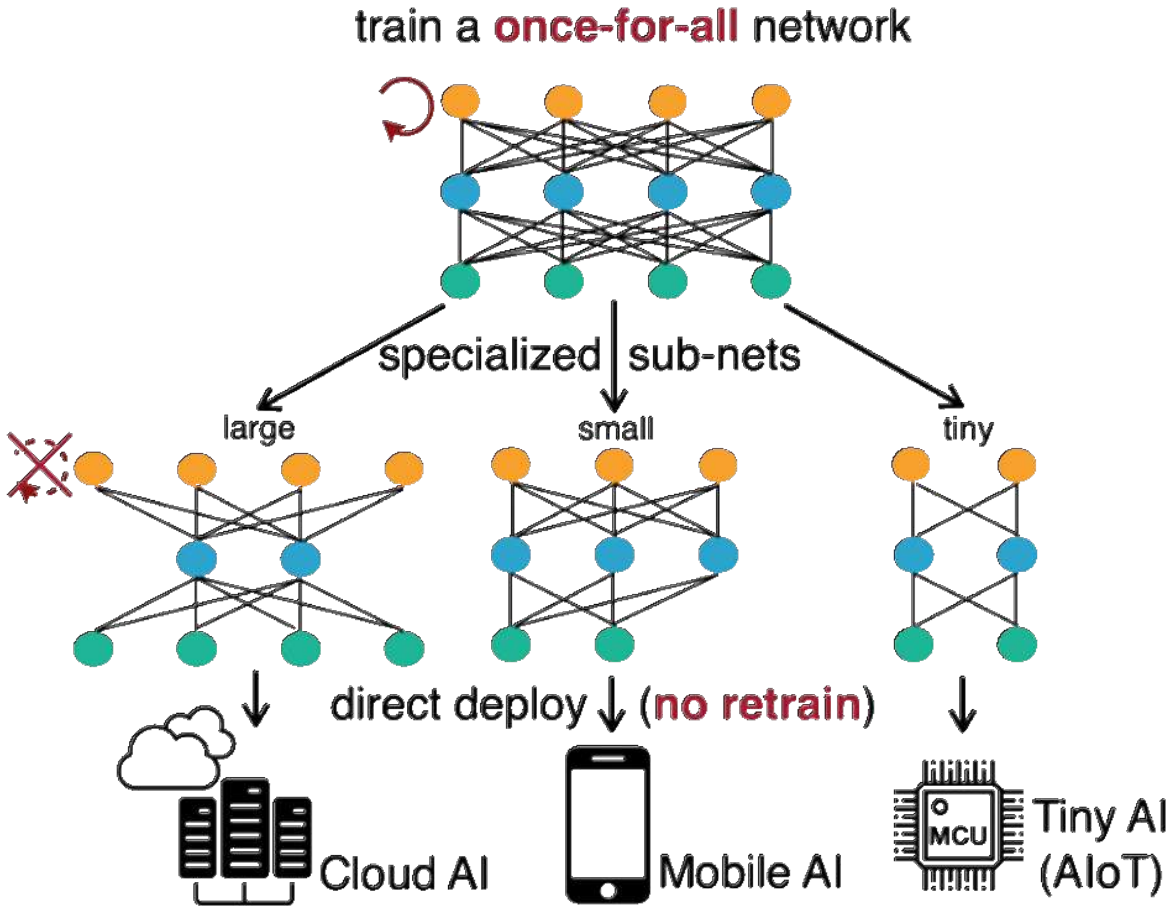


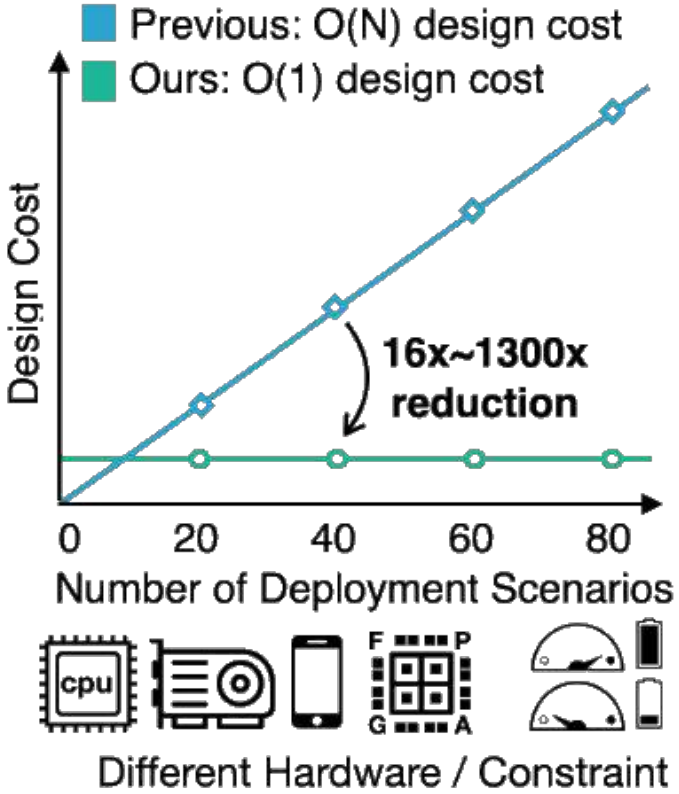
•EfficientNet-B7: top-1 accuracy on ImageNet 84.3%, 8.4x smaller and 6.1x faster on inference than the best existing ConvNet

•EfficientNets transfer well to other dataset

Tan and Le / ICML 2019

Once-for-All (OFA) SuperNet

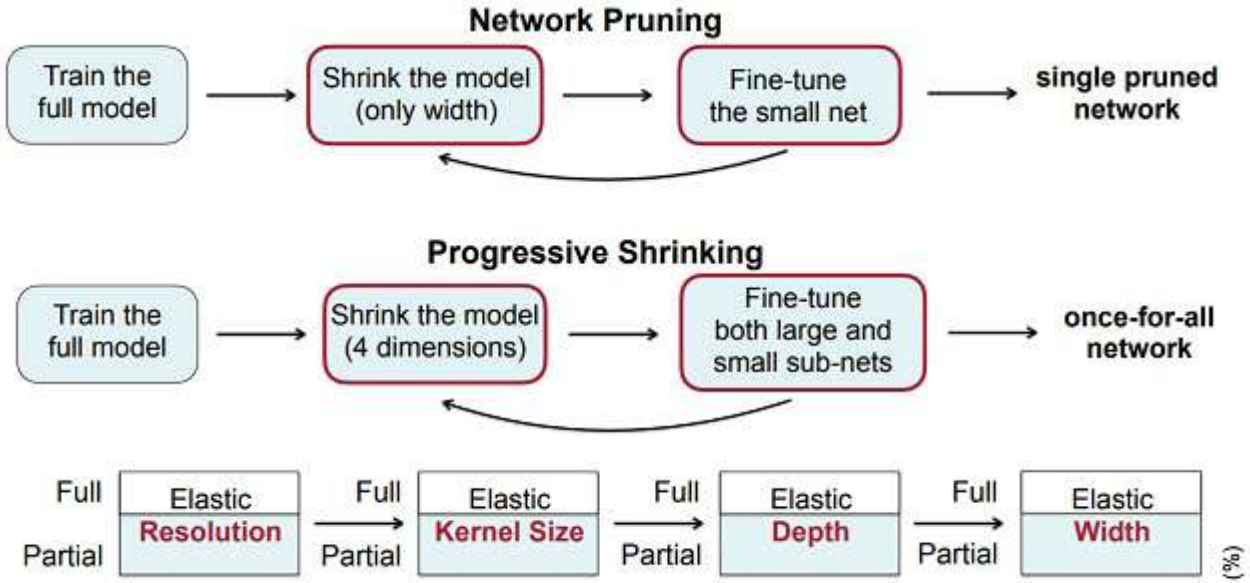


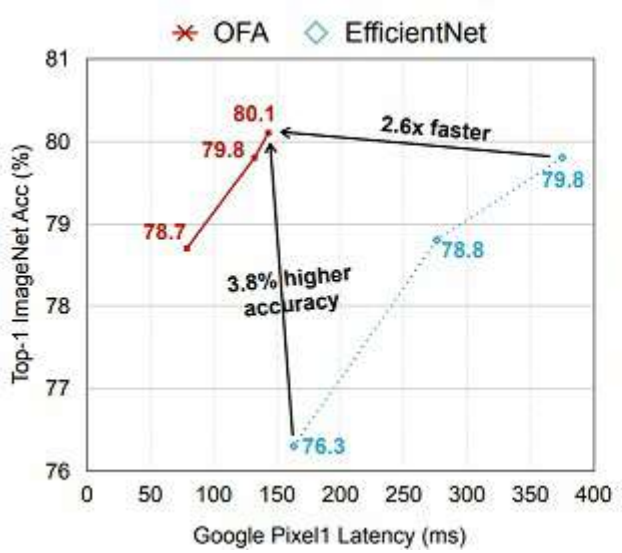


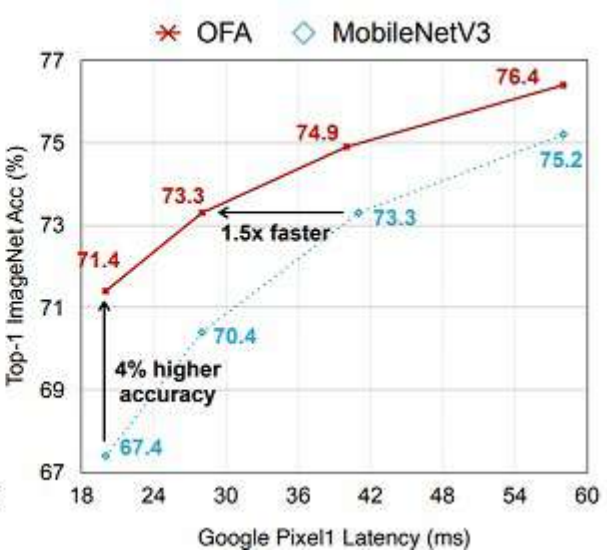
Han Cai et al. / https://github.com/mit-han-lab/once-for-all

Once-for-All (OFA) SuperNet

Progressive shrinking







•Up to 4.0% ImageNet top1 accuracy improvement over MobileNetV3, or same accuracy but 1.5x faster than MobileNetV3

•2.6x faster than EfficientNet w.r.t measured latency

Han Cai et al. / https://github.com/mit-han-lab/once-for-all

Example: OFA for face recognition

Rank-1 accuracy (%) for LFW of fine-tuned models

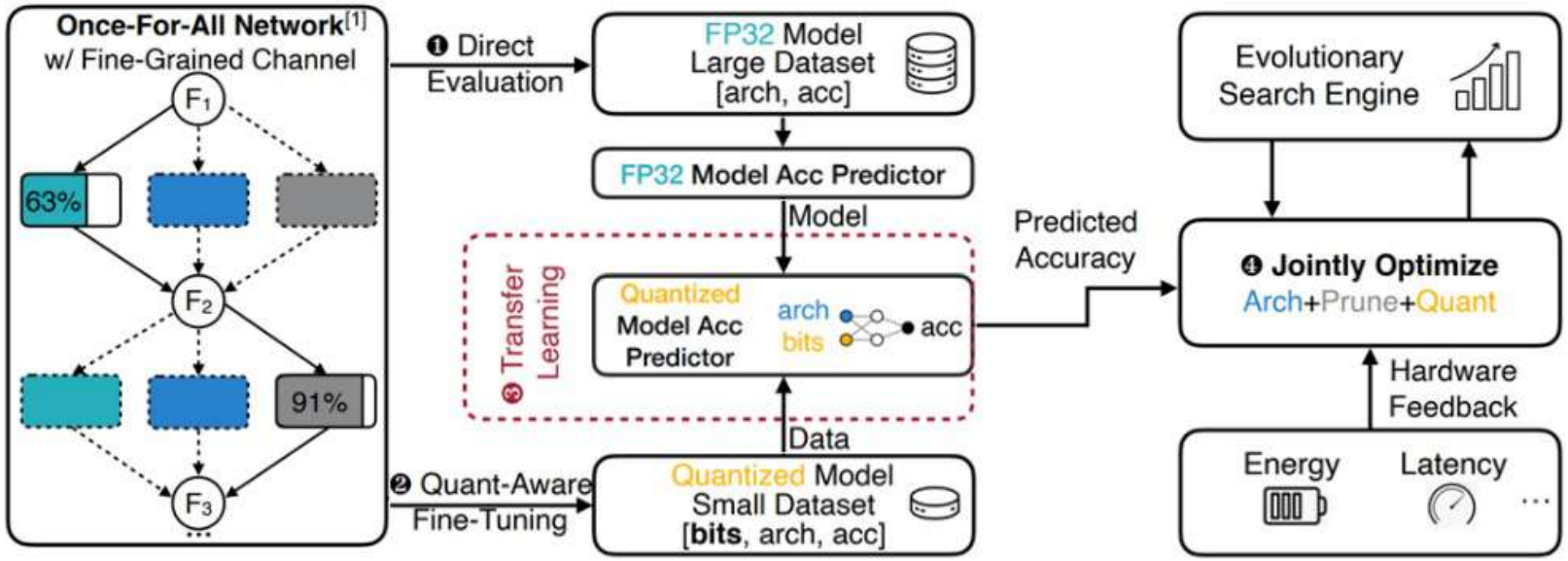
CNN	Center crop	Cropped by MTCNN w/o margins
SENet-50 [10]	97.21± 4.19	96.61± 2.02
MobileNet-v1	90.80±3.96	92.60±4.01
EfficientNet-B0	94.07±4.18	94.70±4.67
EfficientNet-B2	94.58±3.83	95.00±3.81
RexNet-150	94.76±4.45	96.59±3.95

Rank-1 accuracy (%), OFA and subnets for Snapdragon 865

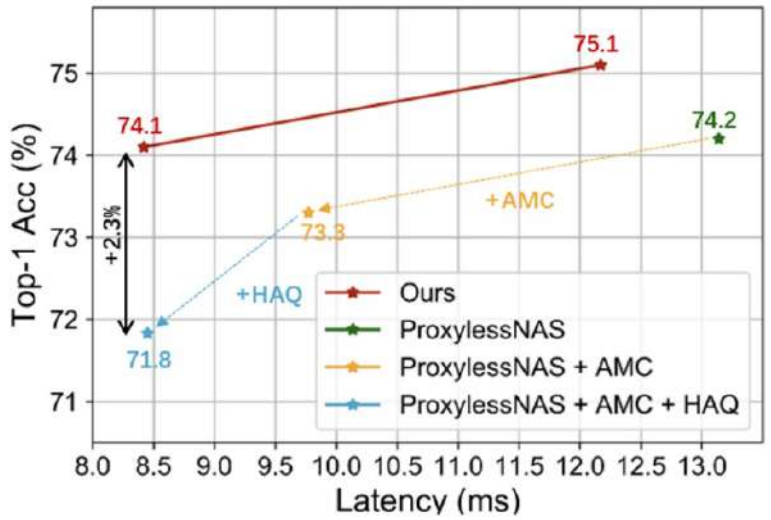
CNN	Center crop	Cropped by MTCNN w/o margins
OFAMobileNet v3	98.97±1.0	99.12±0.83
subnet_device_865_acc_98.3_lut_12.1ms_w12_d234_nac_gbdt	98.13±2.55	98.71±1.05
subnet_device_865_acc_97.8_lut_9.15ms_w12_d234_nac_gbdt	96.33±3.34	97.34±2.18

APQ: Joint Search for Architecture, Pruning and Quantization

APQ



Better performance than sequential design



Han Cai et al. / https://github.com/mit-han-lab/apq

CV for mobile devices is still under active research:

- Novel architectures
- Novel transformations and optimizations of existing models
- Fast inference

AI packages are constantly upgraded:

- Adding support of novel layers
- Adding new accelerators (GPU, DSP, NN-API)

Do not forget about client-server mode!

Thank you!

This work is supported by RSF grant 20-71-10010