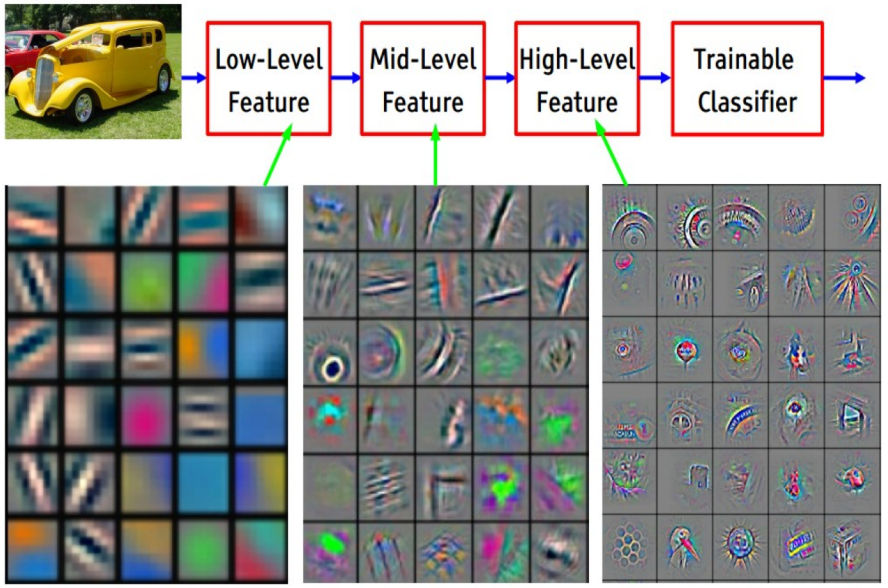
Outline

1. **Introduction**
2. **ML and deep learning review**
3. **What is computer vision?**
    1. Computer vision
    2. Convolutional Neural Nets (CNNs)
4. **Deep NN Architectures for CV Tasks**
    1. Backbone networks
5. **Solving edge-based IoT**
6. **Conclusions and Next steps**

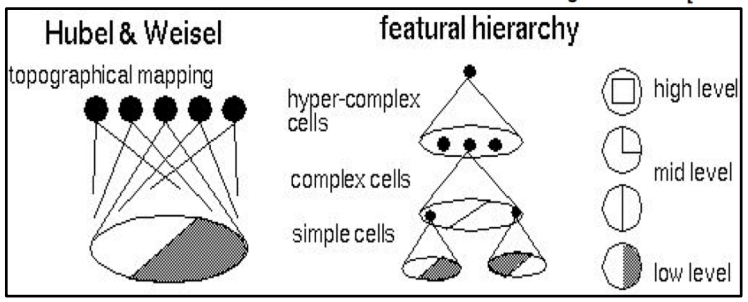
Hubel and Weisel's work inspires ConvNets

Preview



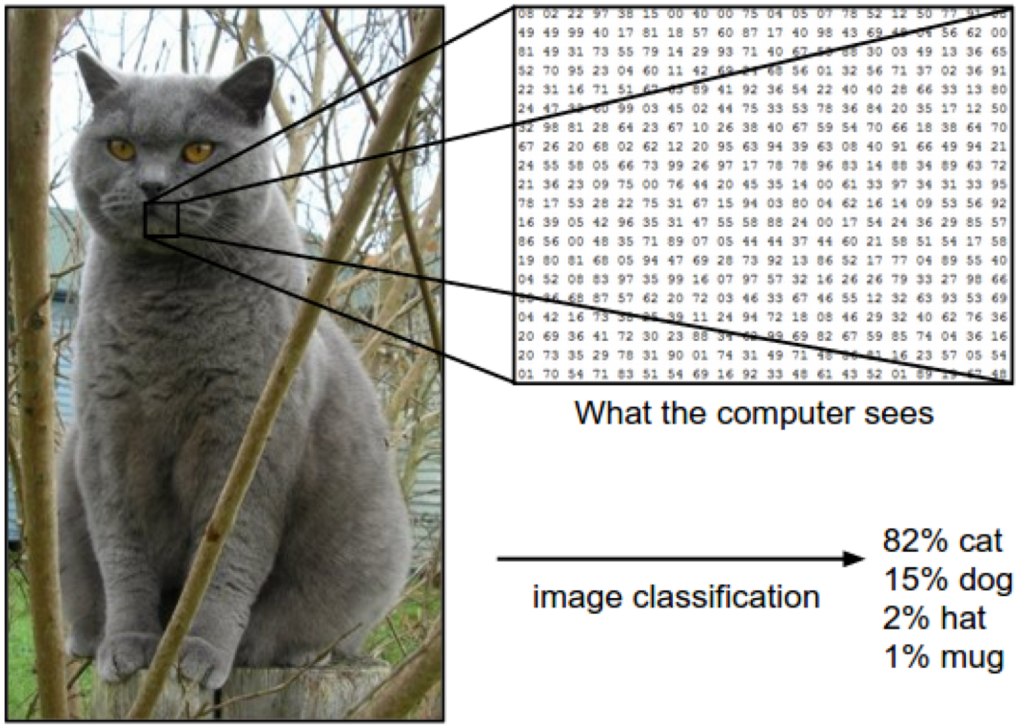
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

http://www.matthewzeiler.com/pubs/arxive2013/arxive2013.pdf



[From recent Yann LeCun slides]

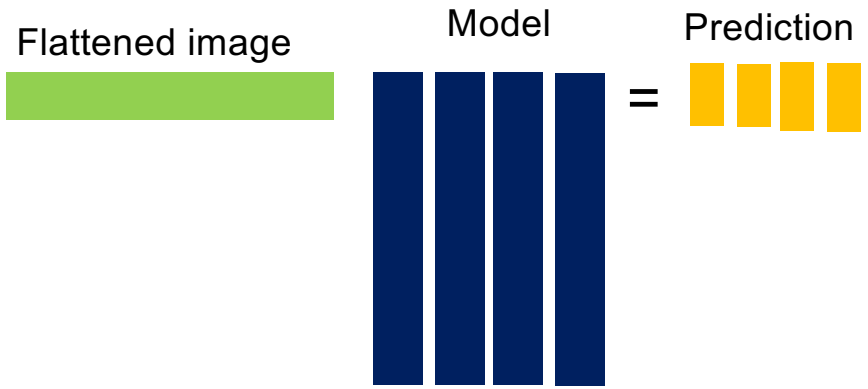
Old fashioned way: logistic regression classifier



What the computer sees

image classification

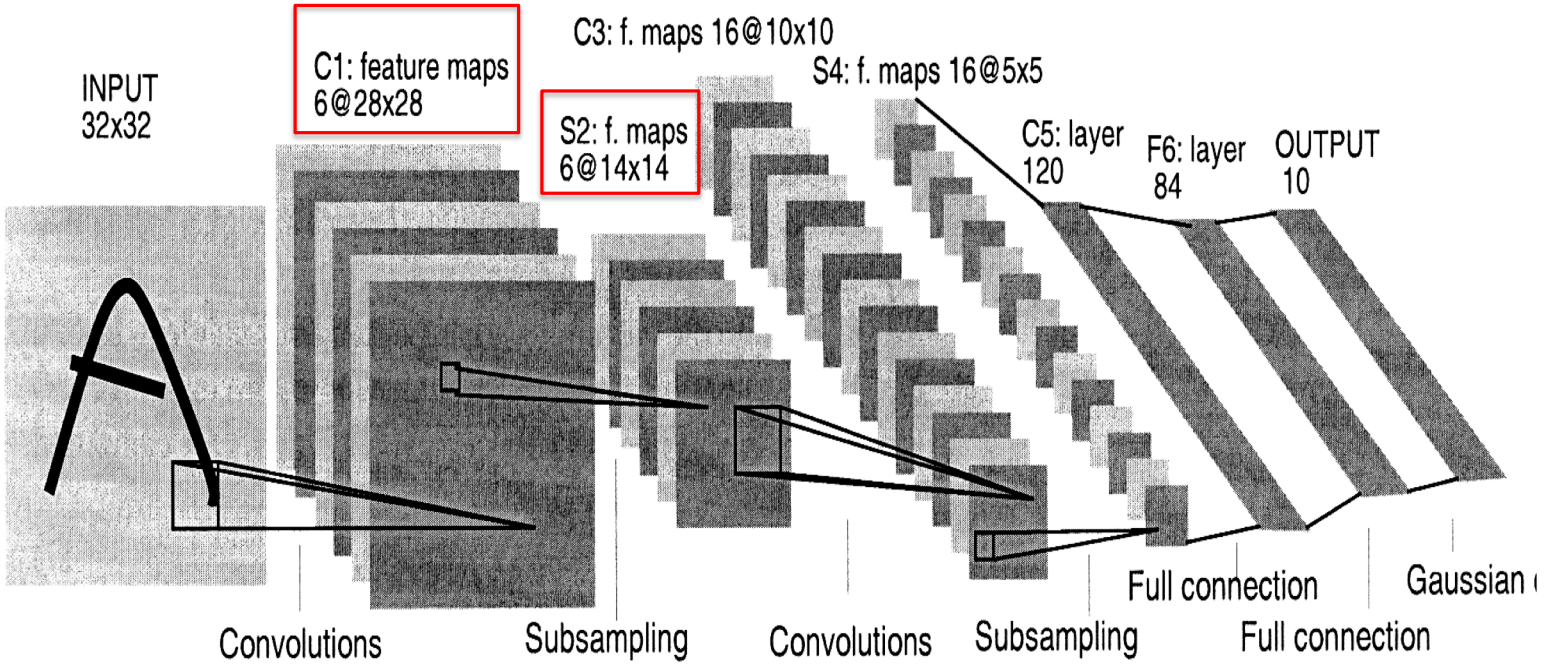
82% cat
15% dog
2% hat
1% mug



Loose spatial data

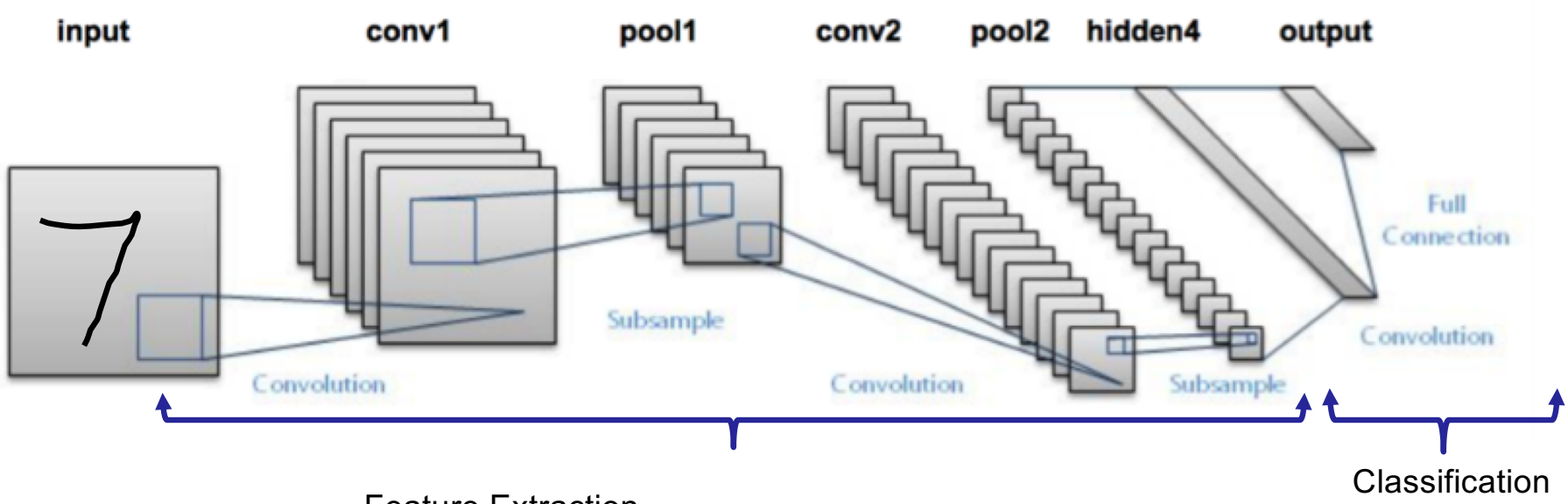
Layered Architecture

The architecture of LeNet5



Yann took Fukushima's architecture and used BackProp to train

LeNet: Took 3 weeks to train in Sun 4 workstation in 1989



- 2 convolutional + 2 fully connected layers
- tanh non-linearity
- 1M parameters
- Training set: MNIST 70K images
- Trained on CPU

LeNet in Keras/Tensorflow

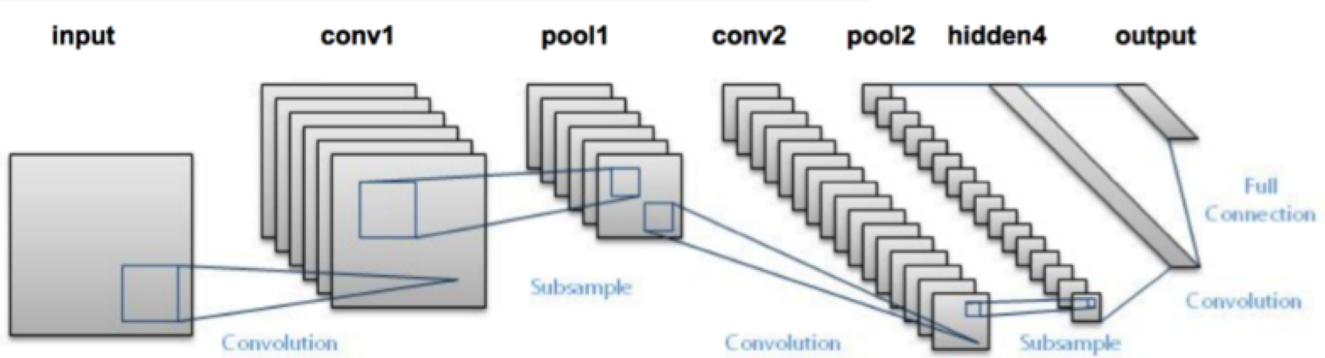
```
model = Sequential()

model.add(Convolution2D(6, 5, 5, border_mode='valid', input_shape = (1, 28, 28)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Activation("sigmoid"))

model.add(Convolution2D(16, 5, 5, border_mode='valid'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Activation("sigmoid"))
model.add(Dropout(0.5))

model.add(Convolution2D(120, 1, 1,

model.add(Flatten())
model.add(Dense(84))
model.add(Activation("sigmoid"))
model.add(Dense(10))
model.add(Activation('softmax'))
```




```
l_rate = 1
sgd = SGD(lr=l_rate, mu=0.8)
model.compile(loss='categorical_crossentropy', optimizer=sgd)
model.fit(X_train, Y_train, batch_size=32, nb_epoch=2,
          verbose=1, show_accuracy=True, validation_data=(X_test, Y_test))
```

jupyter CNN_MNIST (autosaved) ?

File Edit View Insert Cell Kernel Navigate Widgets Help

Markdown

Contents [-]

4 Building CNN model

Now it's time to build the model step-by-step

In [22]:
```
model_cnn = Sequential()
```

Out model is going to be *Sequential* which means that every new added layer will be automatically connected to the pr

Firstly, let's define hyperparameters of the network:

- **filters** — number of filters (or kernels) to use in every layer; in fact this is the same as having multiple channels in th
- **pool_size** — size of the pooling window
- **kernel_size** — size of the convolutional filters

In [23]:
```
filters = 32
kernel_size = (3, 3)
pool_size = (2, 2)
```

Now let's add first layer of the network. It is 2D Convolutional layer. Only unexplained thing here is *padding*. This is the p
pad the data after applying convolutions. padding = 'valid' means that we're not going to pad images and the dimens
layer.

In [24]:
```
model_cnn.add(Convolution2D(filters=filters,
                            kernel_size=kernel_size,
                            padding="valid",
                            input_shape=input_shape))
```

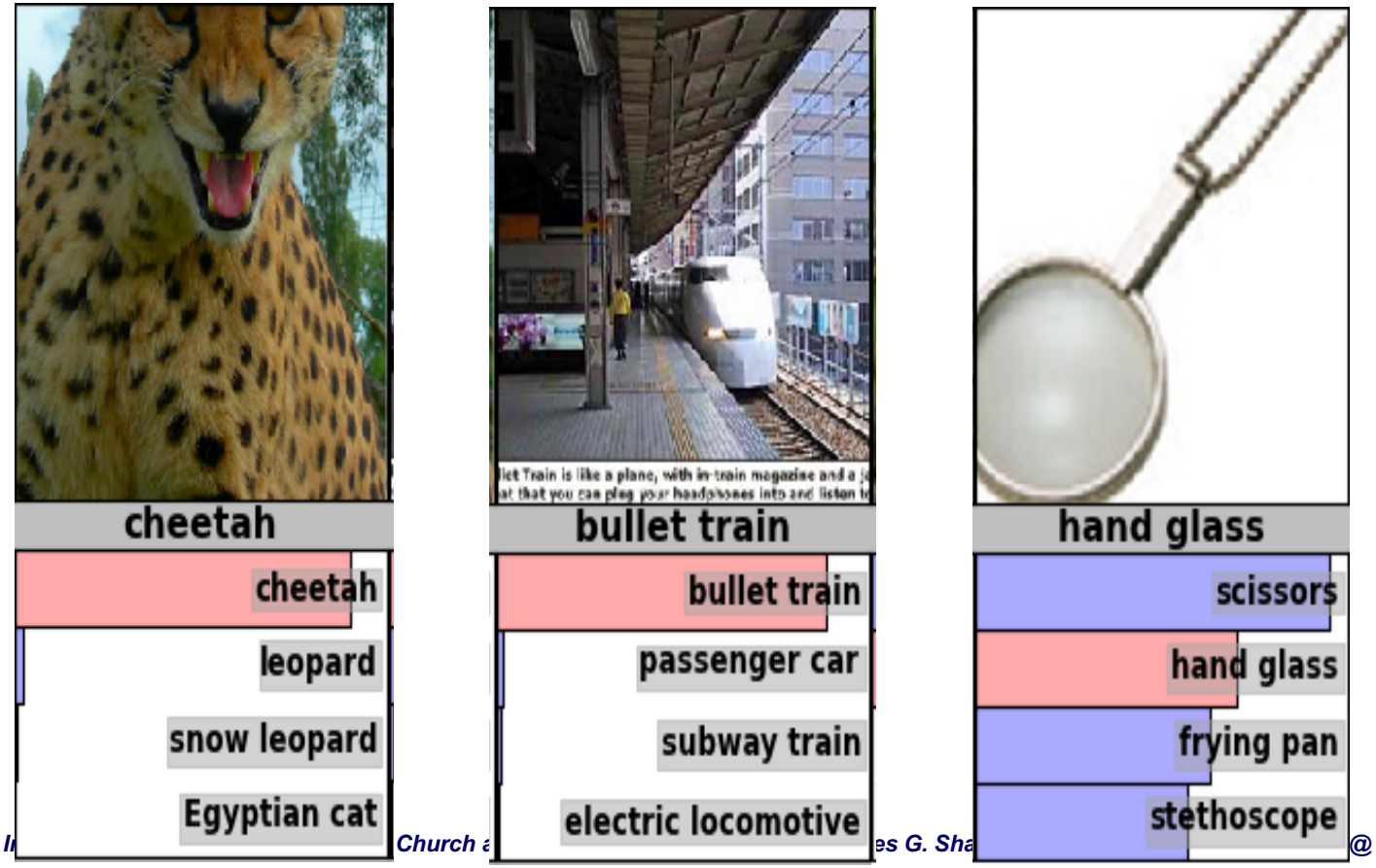
Next step is to add nonlinearity to enable our network to learn complex dependencies. We're going to use ReLU activat
to vanishing gradient problem and faster to train.

In [25]:
```
model_cnn.add(Activation('relu'))
```

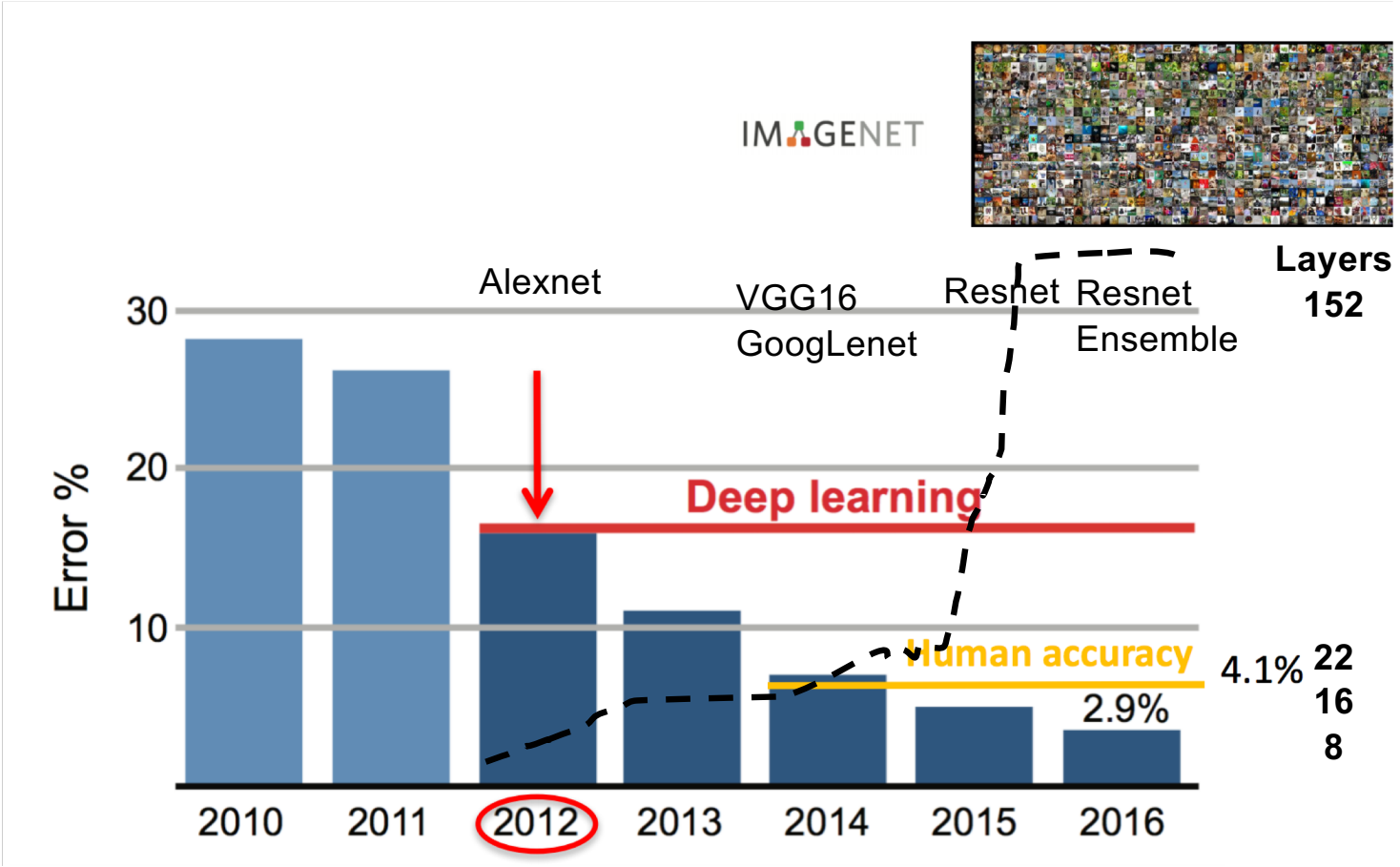
Let's stack one more Convolution layer on top of that:

In [26]:
```
model_cnn.add(Convolution2D(filters=filters,
                            kernel_size=kernel_size,
                            padding="valid"))
model_cnn.add(Activation('relu'))
```

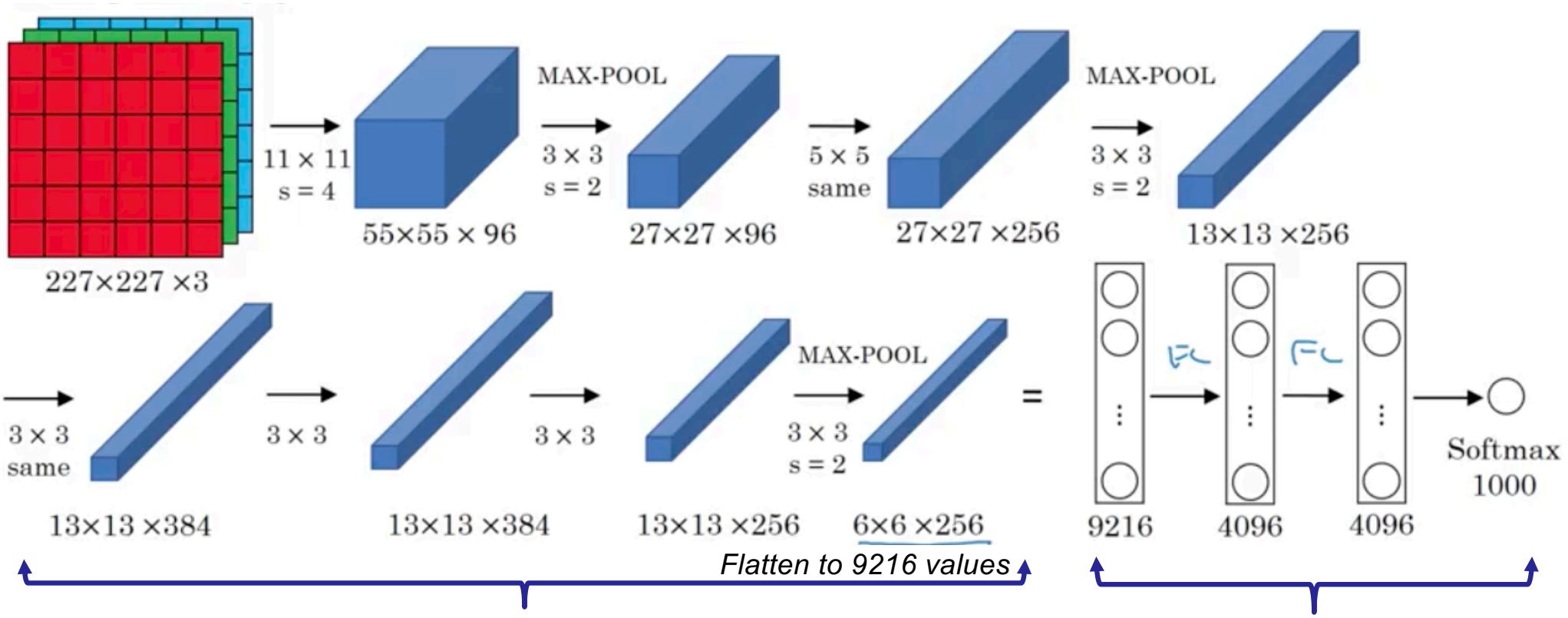
ImageNet Challenge
Examples from the test set
(with the network's predictions)



Computer Vision CNN breakthru [2012]



AlexNet:224x224x3 → 6x6x256

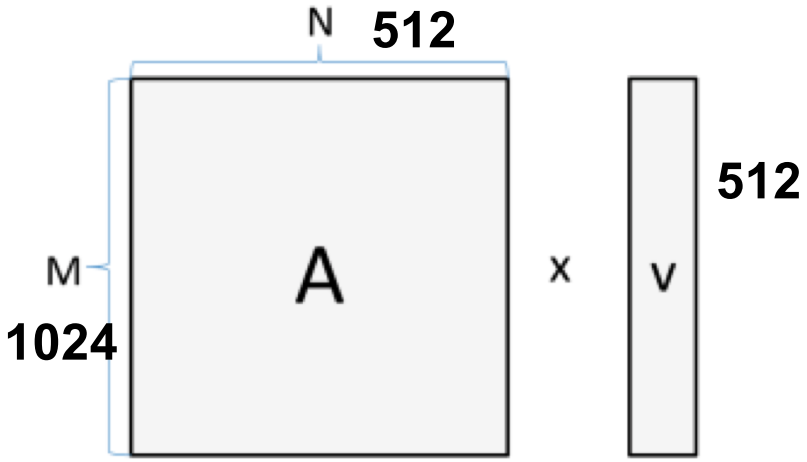


Images shrink from 224x224. → 6x6 x 256

Computational: MxNx2 FLOPs; 10^6 FLOPs

~2Meg of storage

10^6 FLOPs



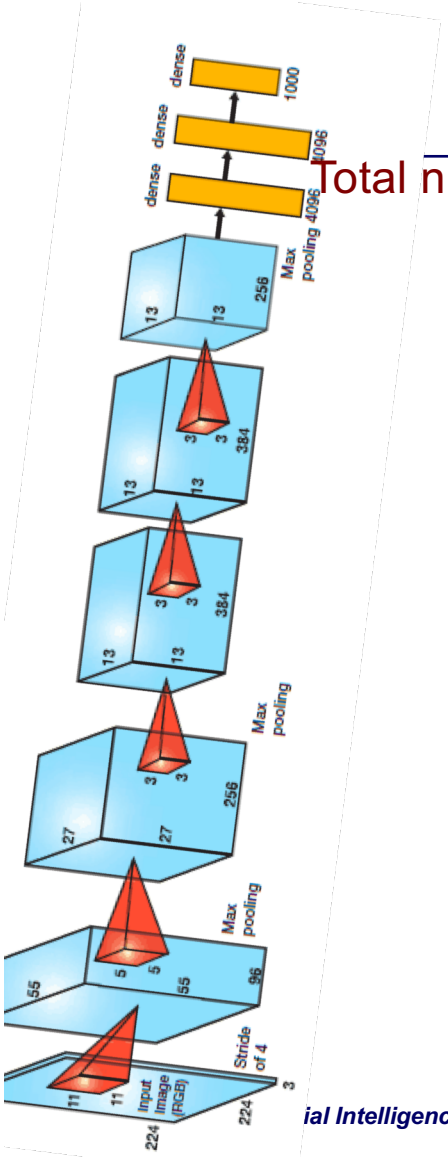
If $M = 1024$ and $N = 512$, then the number of bytes we need to read and store is:

$$4 \text{ bytes} \times (1024 \times 512 + 512 + 1024) = 2.1e6 \text{ bytes}$$

And the number of calculations we need to do is:

$$2 \times 1024 \times 512 = 1e6 \text{ FLOPs}$$

Alexnet: Architecture for Classification



Total nr. params: 60M

Category prediction

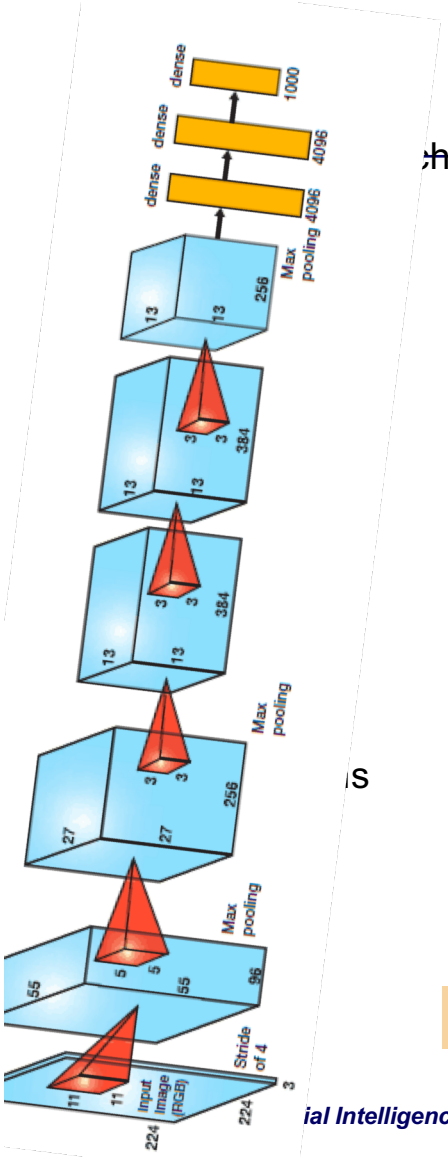
Total nr. flops: 832M

4M	LINEAR	4M
16M	FULLY CONNECTED	16M
37M	FULLY CONNECTED	37M
	MAX POOLING	
442K	CONV	74M
1.3M	CONV	224M
884K	CONV	149M
	MAX POOLING	
	LOCAL CONTRAST NORM	
307K	CONV	223M
	MAX POOLING	
	LOCAL CONTRAST NORM	
35K	CONV	105M

input

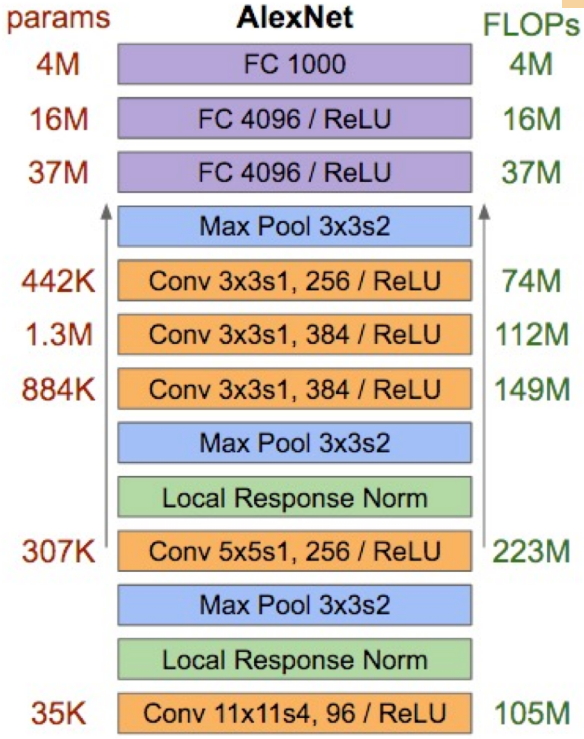
#FLOPs for one image: forward prop only

AlexNet #FLOPs for one image: forward prop only



~~chematic of params and forward FLOPs~~

(on one 3x227x227 image)



10^9 FLOPs per image: GPU has FLOPs and VRAM costs

Layer	WeightOps	BiasOps	% of FLOPS
fc8	4,096,000	1,000	1%
fc7	16,777,216	4,096	2%
fc6	37,748,736	4,096	5%
conv5	74,760,192	43,264	10%
conv4	112,140,288	64,896	15%
conv3	149,520,384	64,896	21%
conv2	**223,948,800**	**186,624**	**31%**
conv1	105,415,200	290,400	15%
			725,066,088
LRN	20,000	0.00%	
pooling	27,000	0.00%	
ReLU	659,272	0.09%	
Conv and forward Prop	725,066,088	99.90%	
	725,772,360	Total FLOPS	

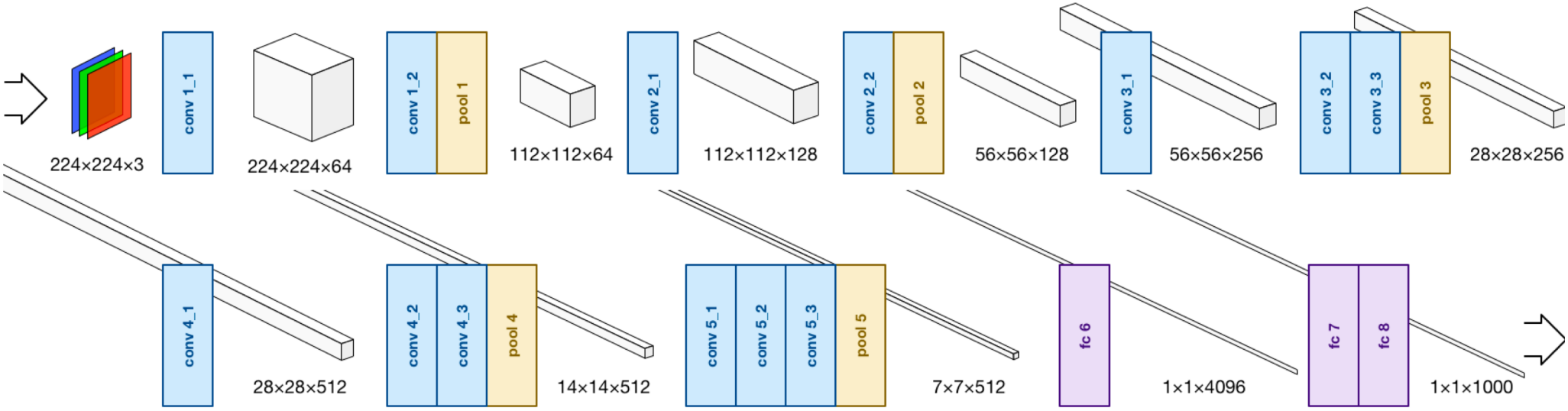
For example, **conv2** has 256 * (96 / 2) * 5^2 = 307,200 params
(on one 3x227x227 image)

~60Million parameters X 4 bytes X 2 (ForwardProp+BackProp) for one image

https://groups.google.com/forum/#!topic/caffe-users/cUD3IF5NMOk

VGGNet: 16 weight layers + 5 maxpools

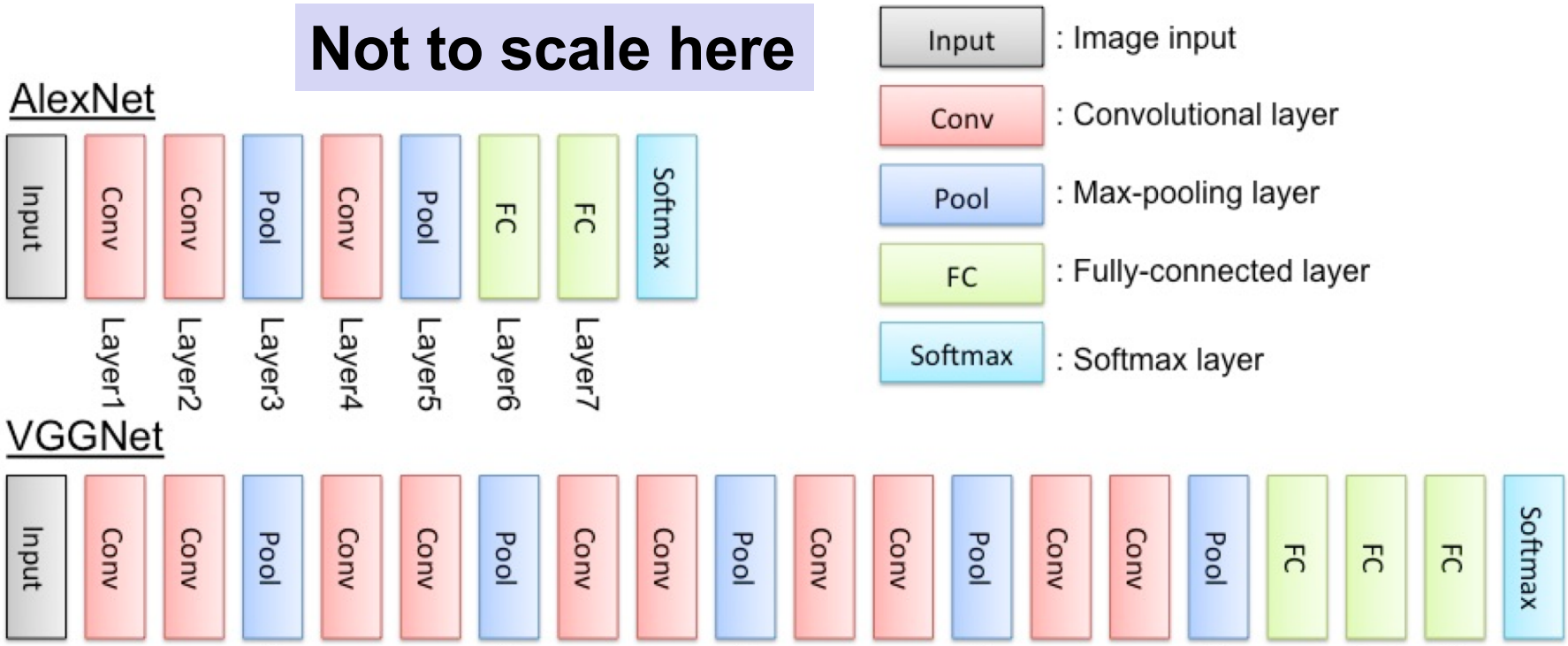
- For example, the first layer **conv1_1** receives a 224×224×3 input volume. It has 64 convolution kernels, so its output volume is 224×224×64.



Images shrink from 224x224. → 7x7 x 512

- The very last pooling layer, **pool5**, outputs a 7×7×512 volume. So we've gone from a 224×224 RGB image to something that is only 7×7 "pixels" in size, but in depth we've grown from 3 to 512 planes of data.

AlexNet vs. VGGNet: 15% vs 7%



VGGNet BP Memory, Model Memory: Forward-Backward

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note: 3.2M parameters:

Most memory is in early CONV

Most params are in late FC

120M out of 138M parms come in the FC
Modern architectures replace these layers

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)

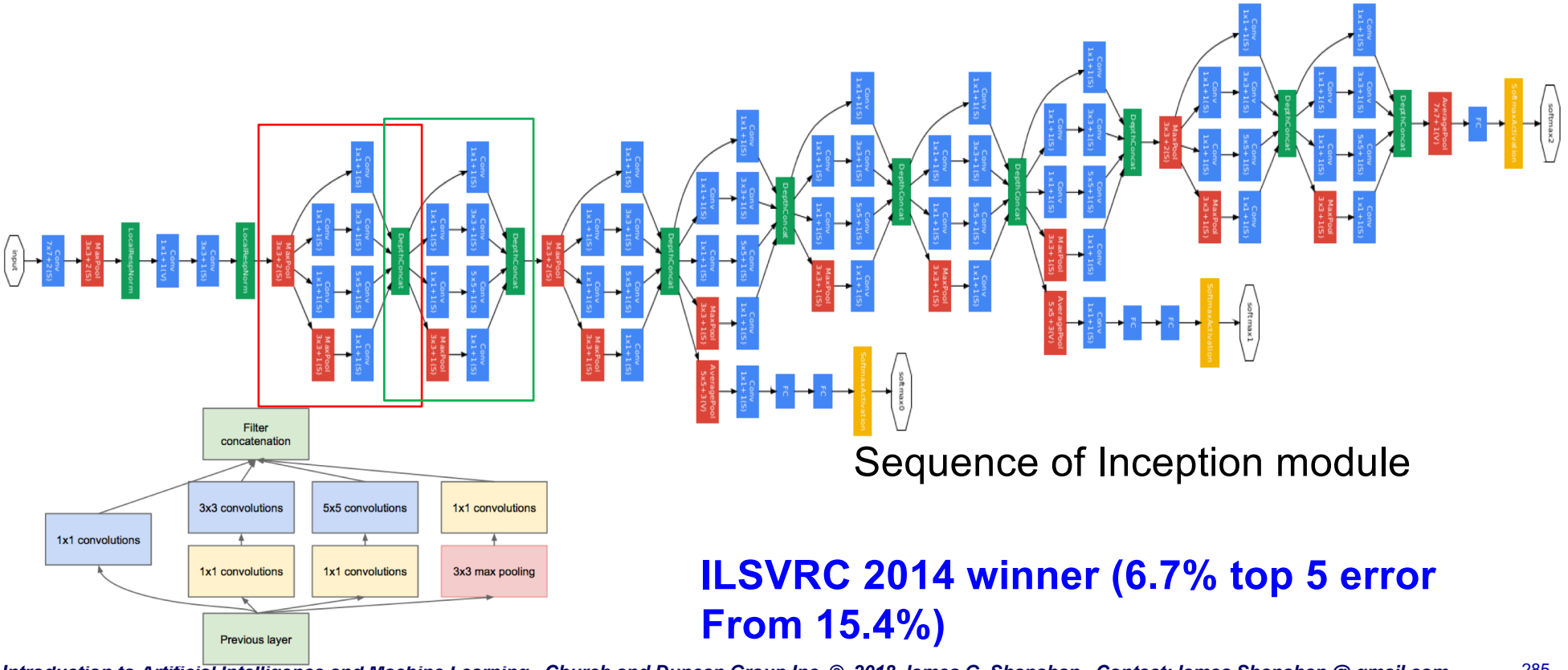
TOTAL params: 138M parameters

~200M for forward+Back

As the space descreases the depth increases (*512 activation image)

GoogLeNet: 2014 winner at 6.7%; 22 weight layers

Case Study: GoogLeNet *[Szegedy et al., 2014]*

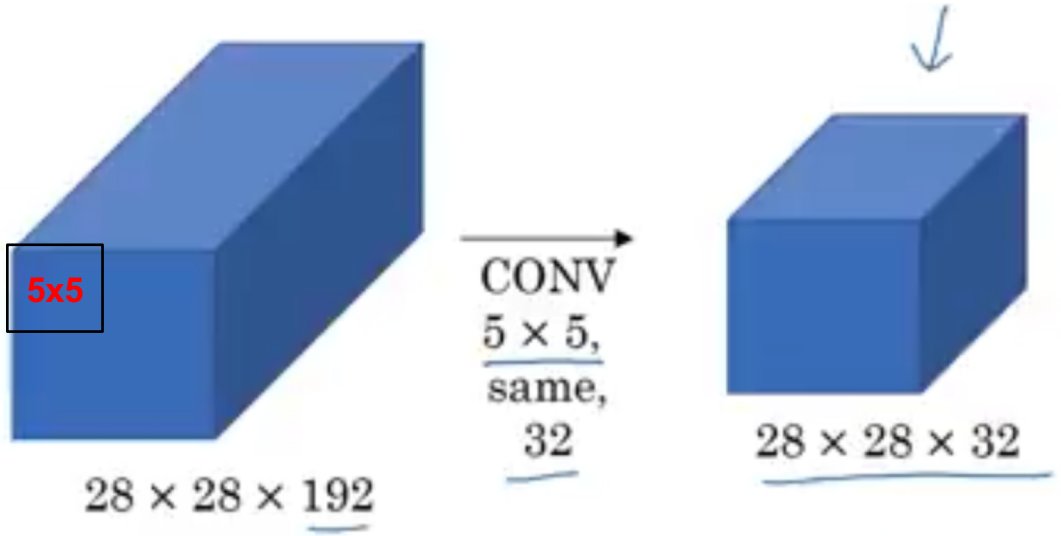


Sequence of Inception module

ILSVRC 2014 winner (6.7% top 5 error From 15.4%)

FLOPS: 120M multiplications + 120M additions

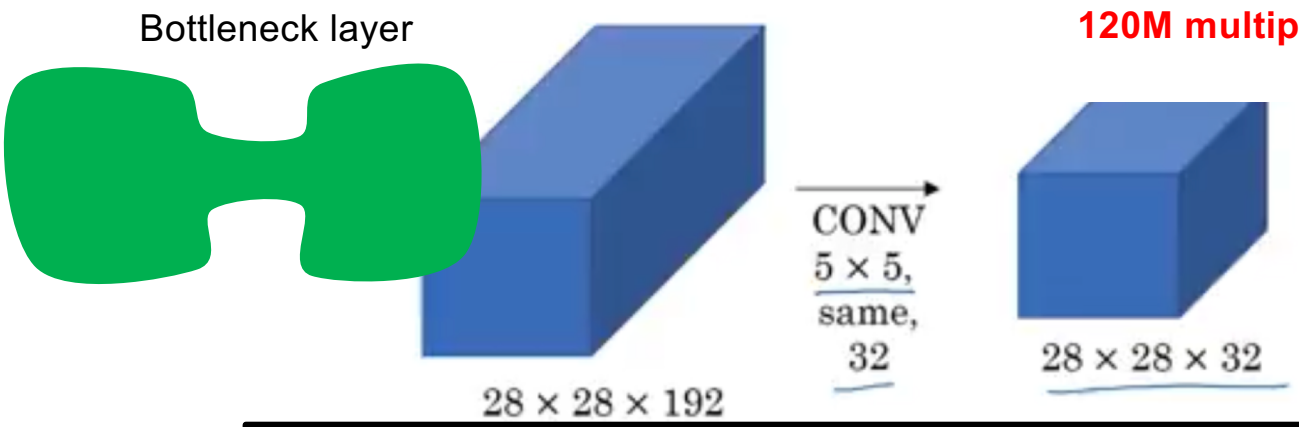
[1, 2,3] [3, 4, 5]



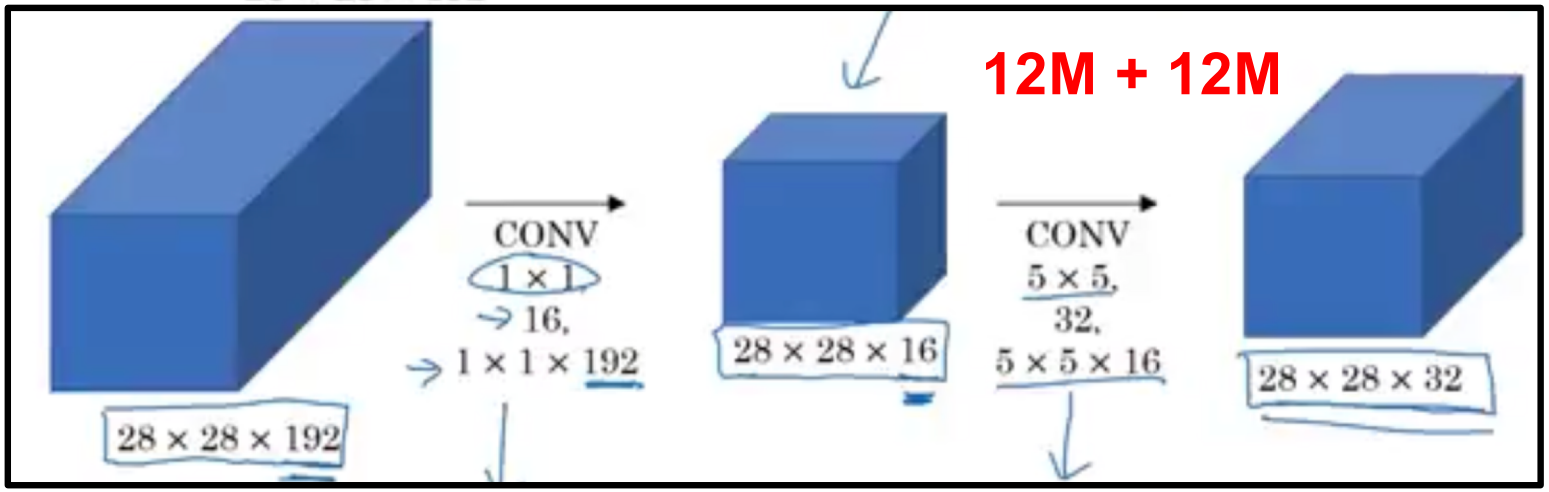
32 filters. filter one 5x5x192.

28x28x32 * 5x5x192 = 120M.

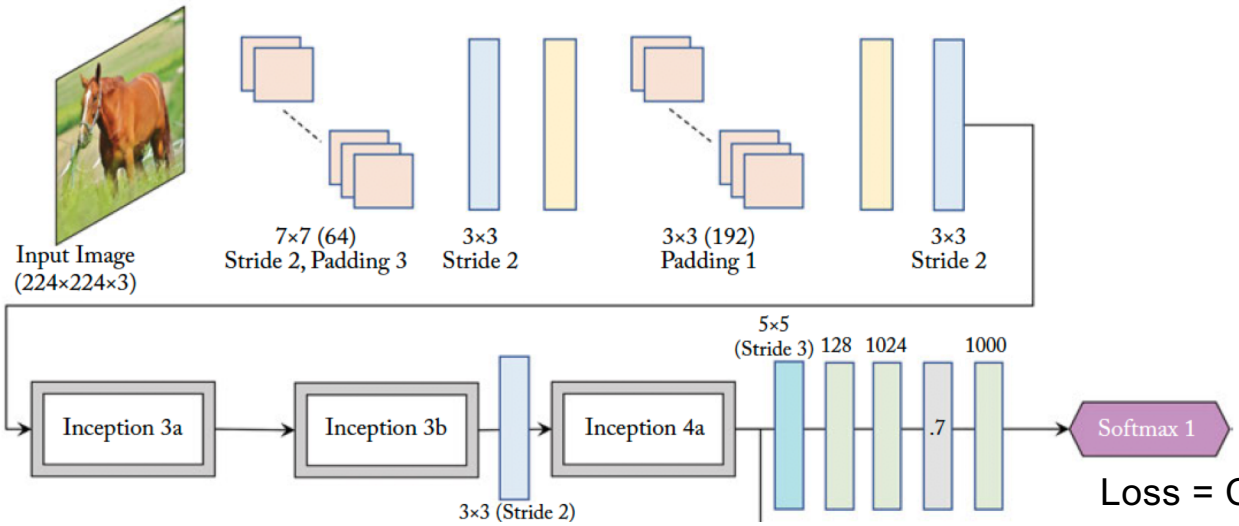
120M multiplications + 120M additions → 12M + 12M
Conv layer to a conv module with layers and a bottleneck



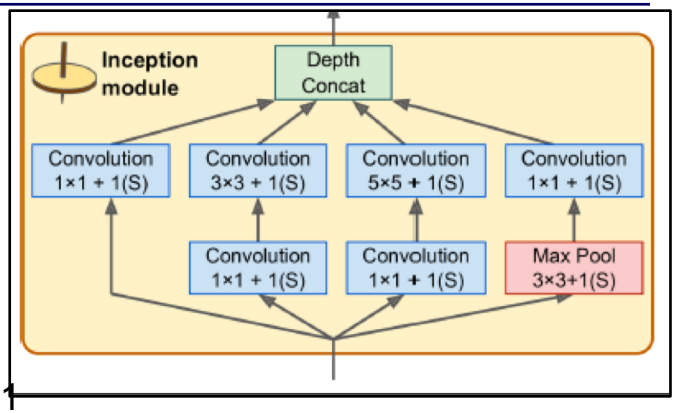
Use a 1x1 convolution bottleneck layer:
10X reduction in FLOPs
120M → 12.4 M multiplication ops



Inception



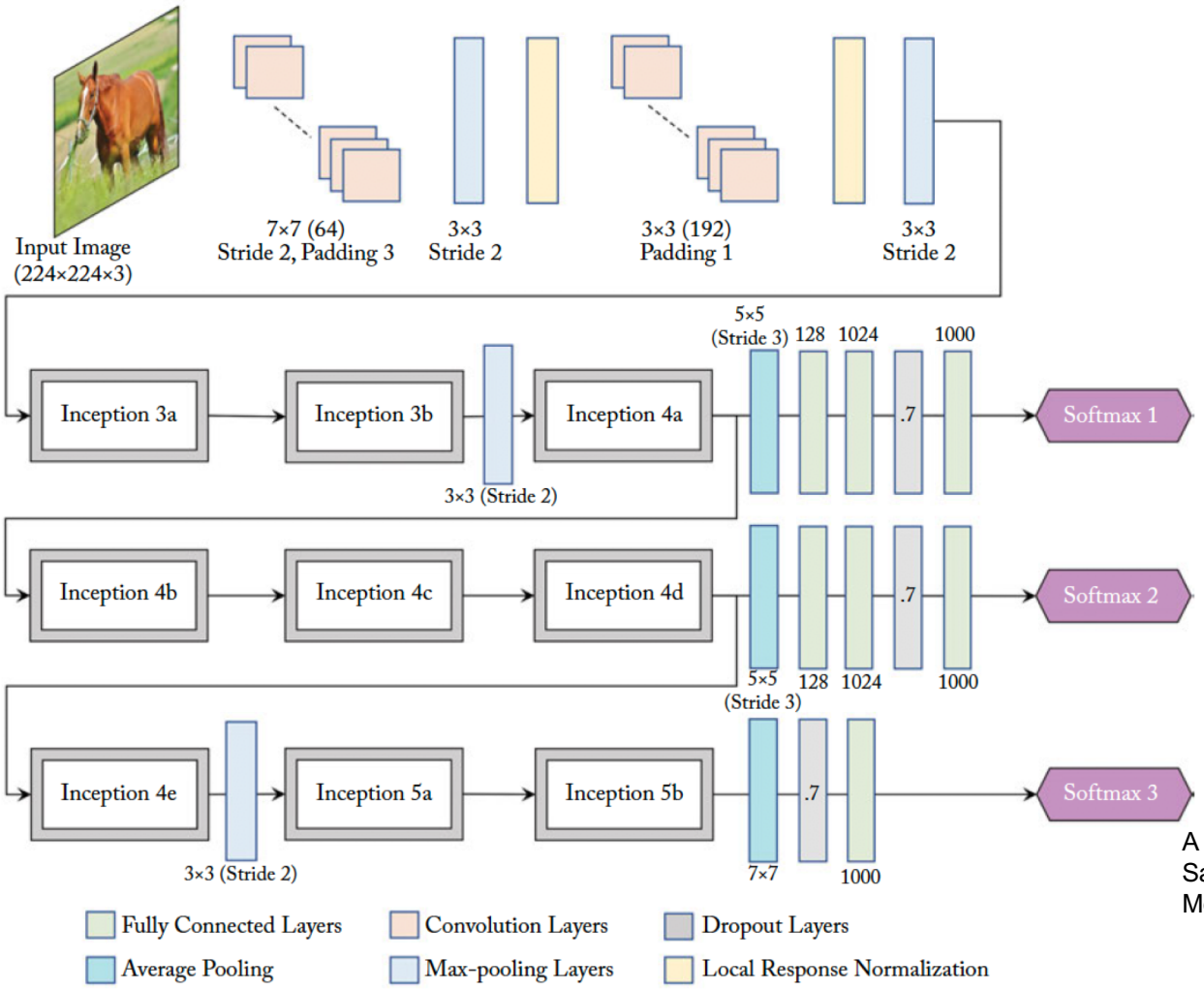
Loss = CX1

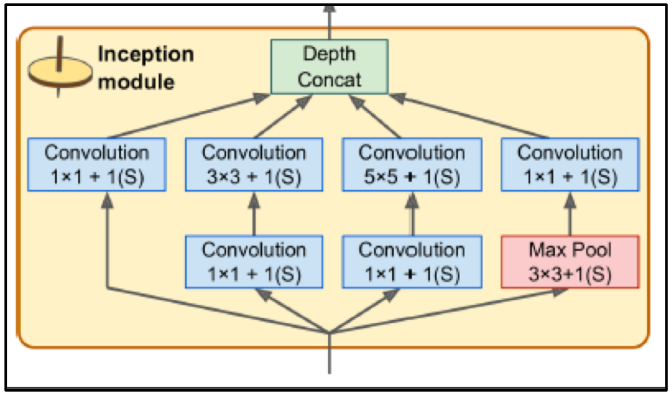


Guide to Convolutional Neural Networks for Computer Vision
…alman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and
…ohammed Bennamoun. 2018

Fully Connected Layers
Convolution Layers
Dropout Layers
Average Pooling
Max-pooling Layers
Local Response Normalization

Inception: inception module + pyramid loss



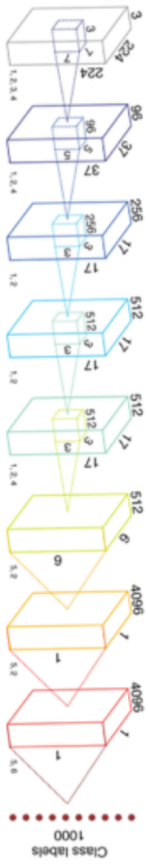


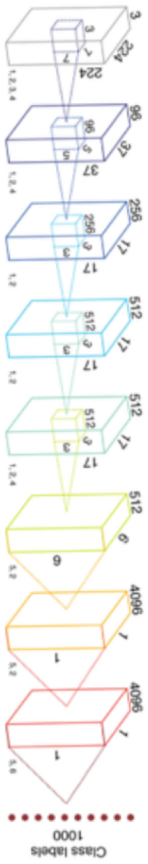
$$\text{Loss} = CX_1 + CX_2 + CX_3$$

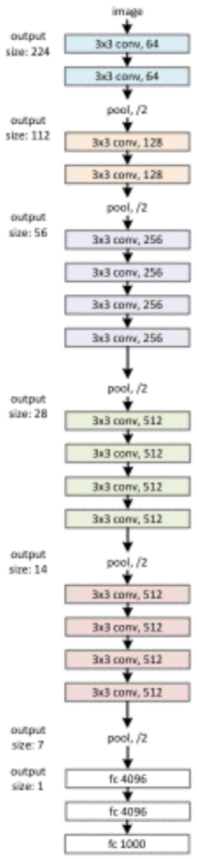
A Guide to Convolutional Neural Networks for Computer Vision
Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun. 2018

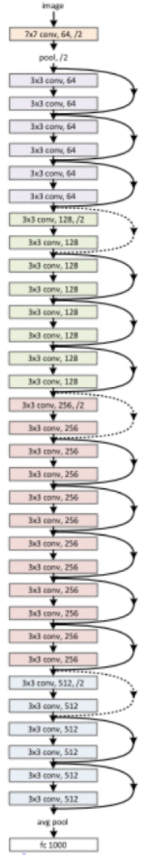
ResNet (skip connections)

2012 | 2013 | 2014 | 2015









8 layers
15.31% error

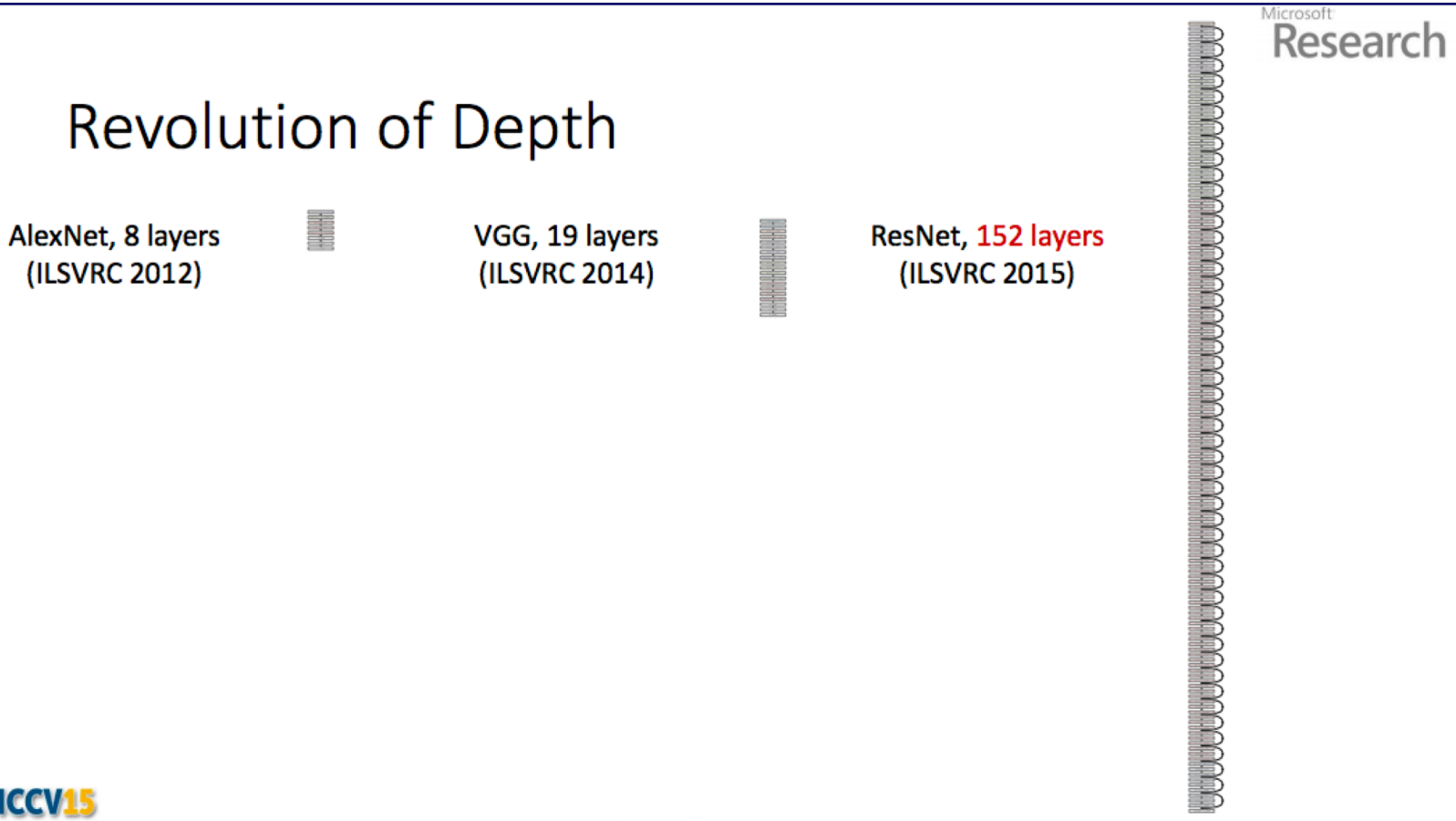
9 layers, 2x params
11.74% error

19 layers
7.41% error

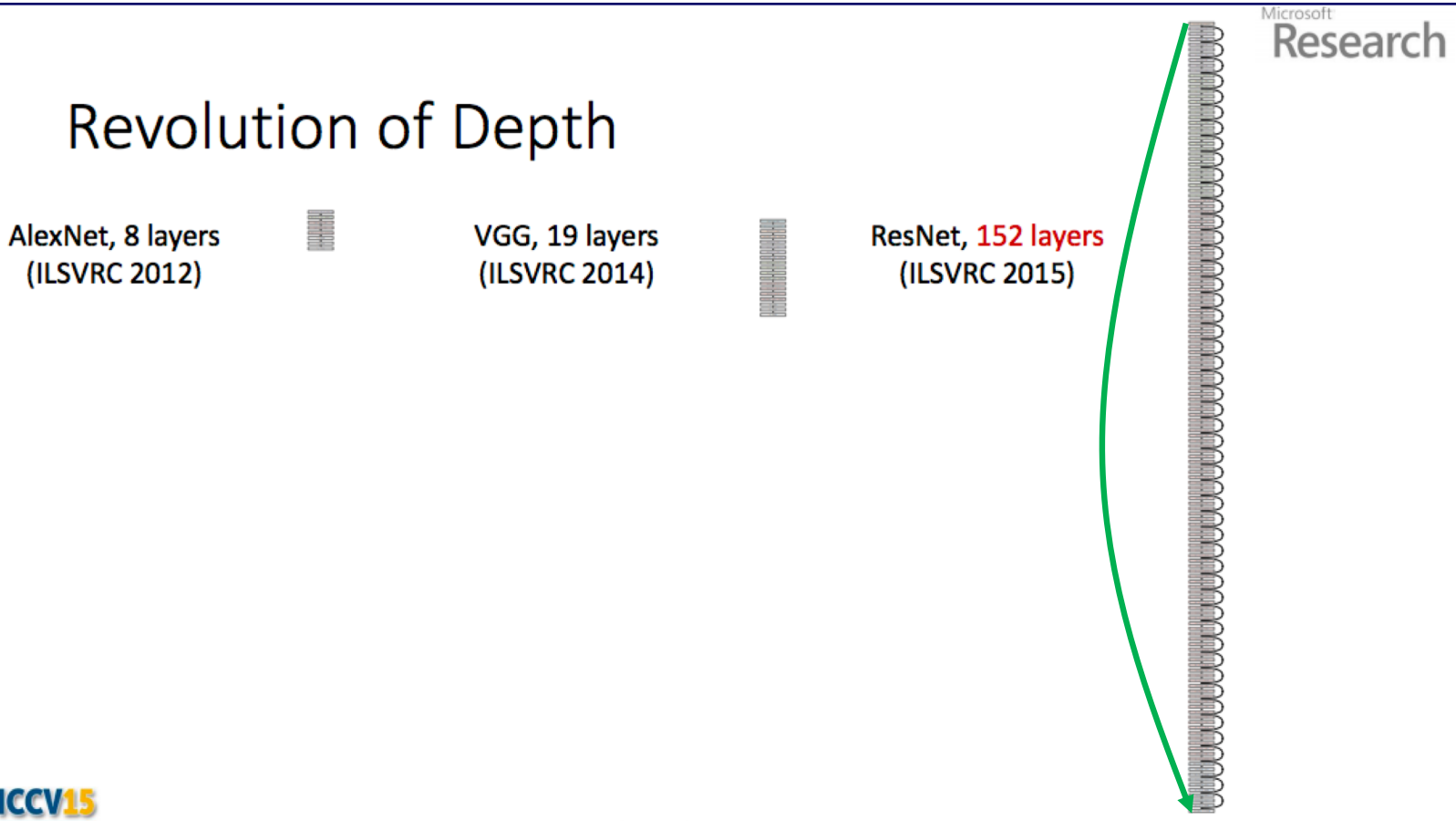
152 layers
3.57% error

http://www.slideshare.net/iljakuzovkin/paper-overview-deep-residual-learning-for-image-recognition?from_action=save

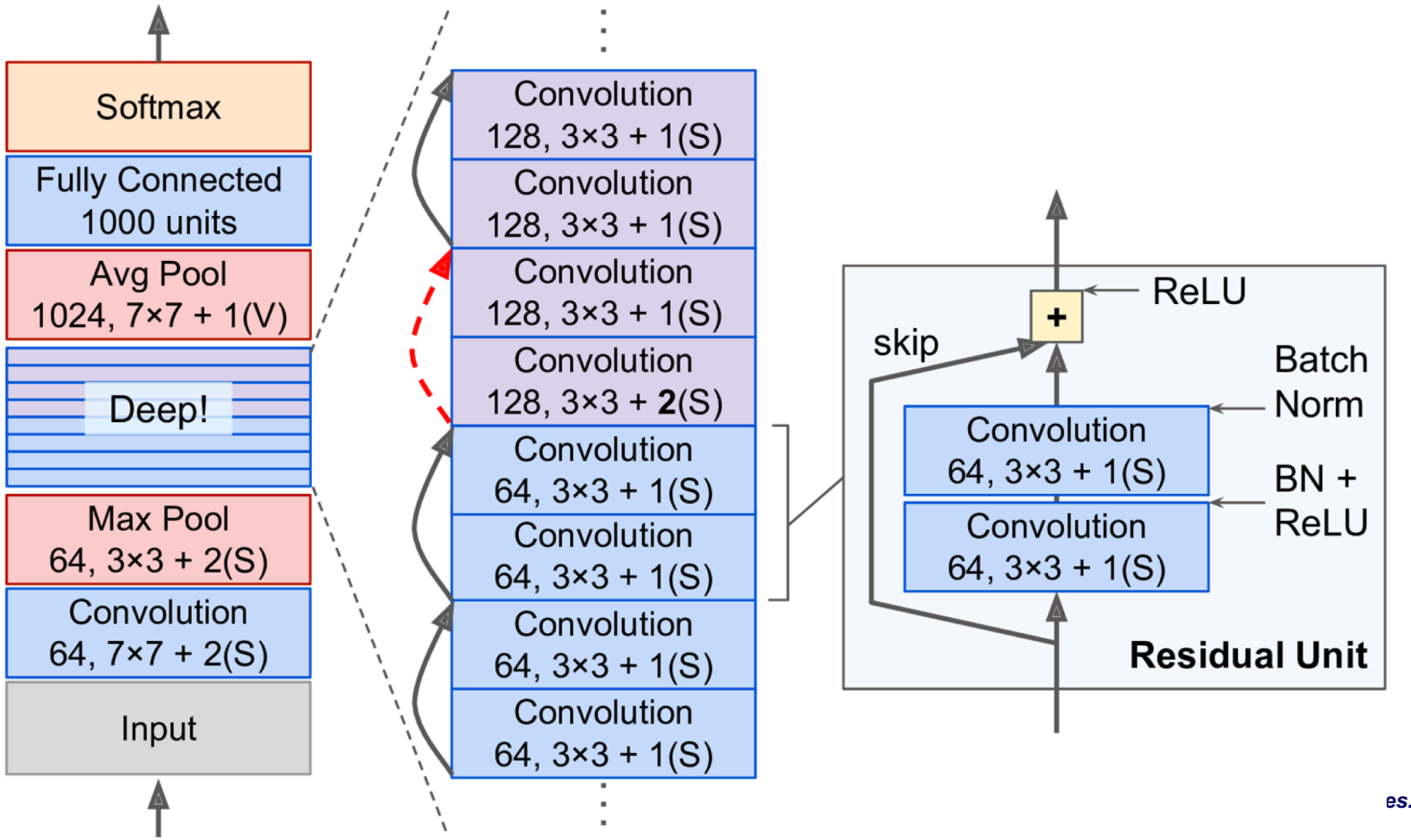
8 → 20 -> 152 layers



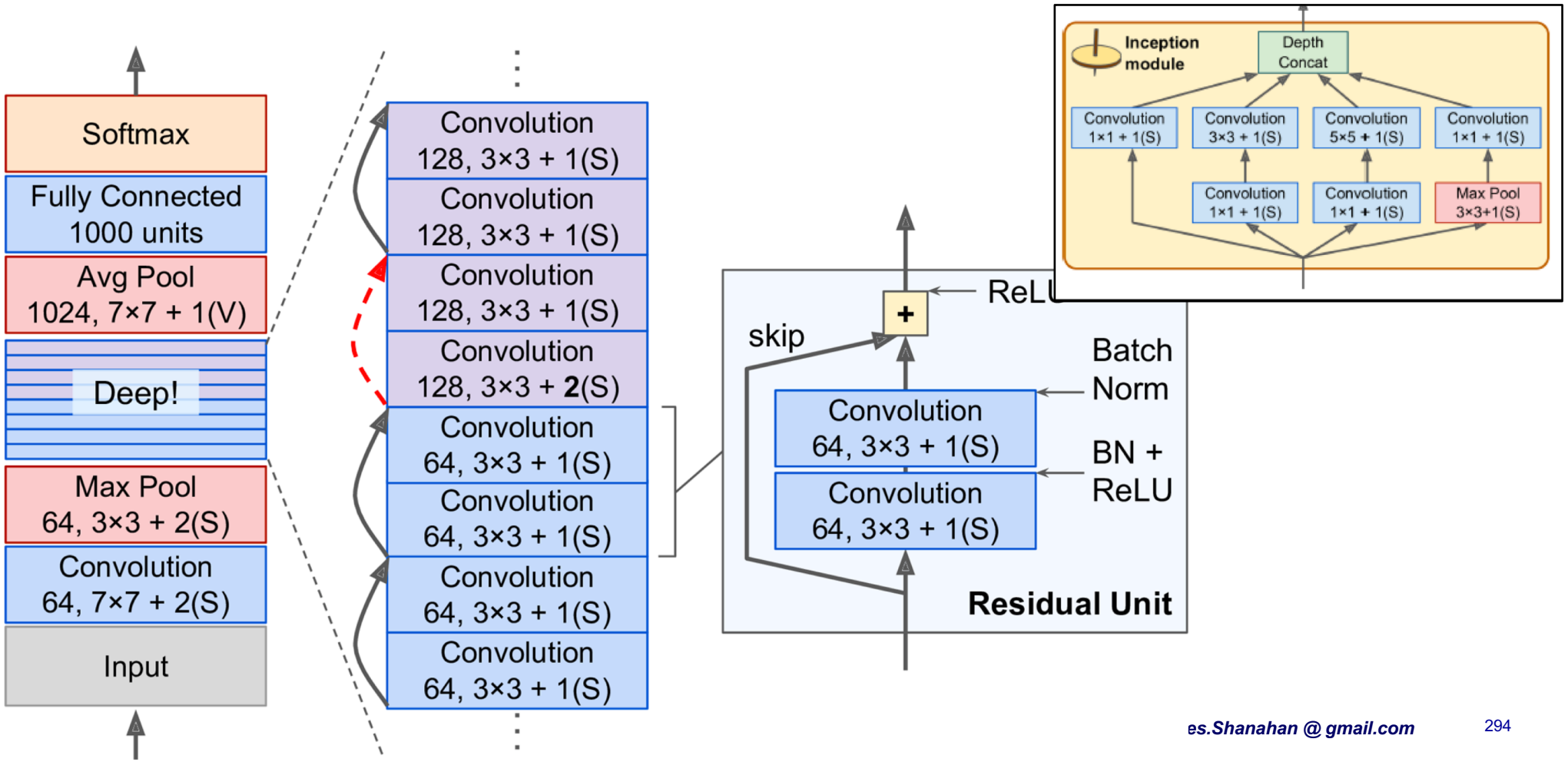
8 → 20 -> 152 layers



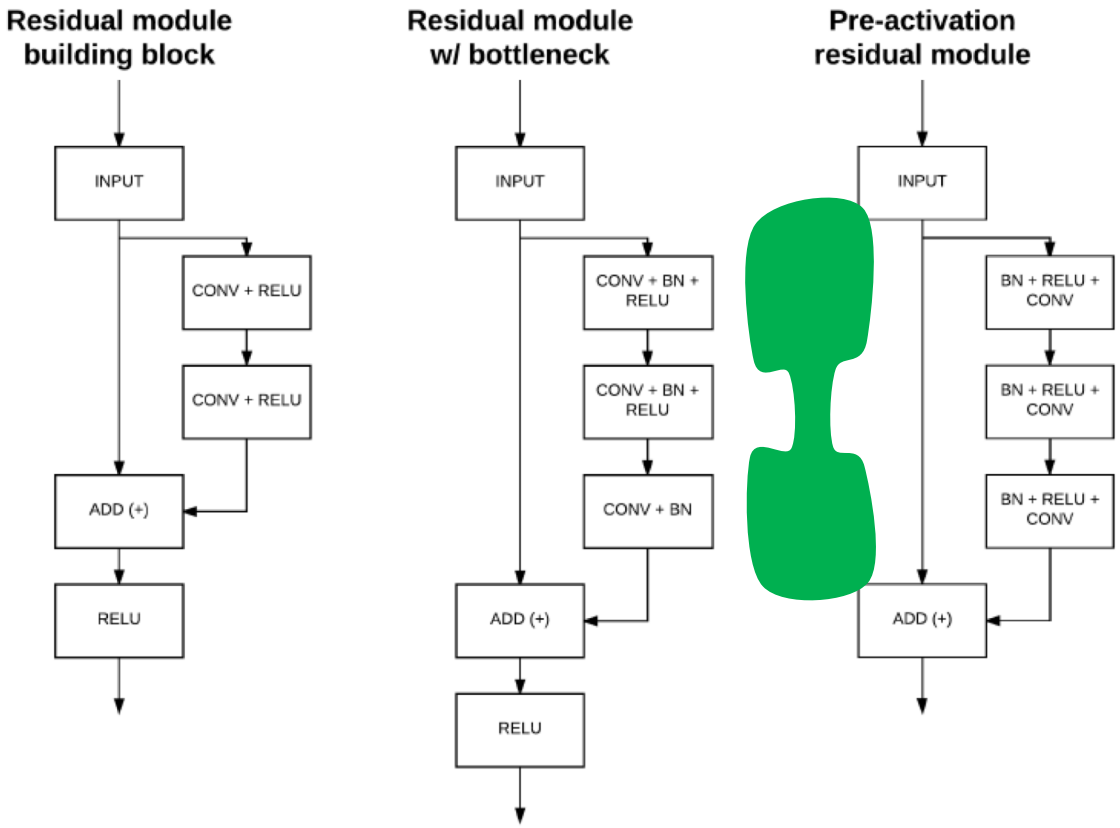
ResNet Unit


Softmax
Fully Connected
1000 units
Avg Pool
1024, 7×7 + 1(V)
Deep!
Max Pool
64, 3×3 + 2(S)
Convolution
64, 7×7 + 2(S)
Input
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 2(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
+
ReLU
skip
Batch
Norm
Convolution
64, 3×3 + 1(S)
BN +
ReLU
Convolution
64, 3×3 + 1(S)
Residual Unit

Inception module vs. ResNet Unit



Different flavors:ResNet with bottleneck layers



Bottleneck layer

Figure 8.1: **Left:** The original residual module building block. **Center:** The residual module with a bottleneck. The bottleneck adds in an extra CONV layer. This module also helps reduce the dimensionality of spatial volumes. **Right:** The updated pre-activation module that changes the order of RELU, CONV, and BN.

Complexity for classification tasks

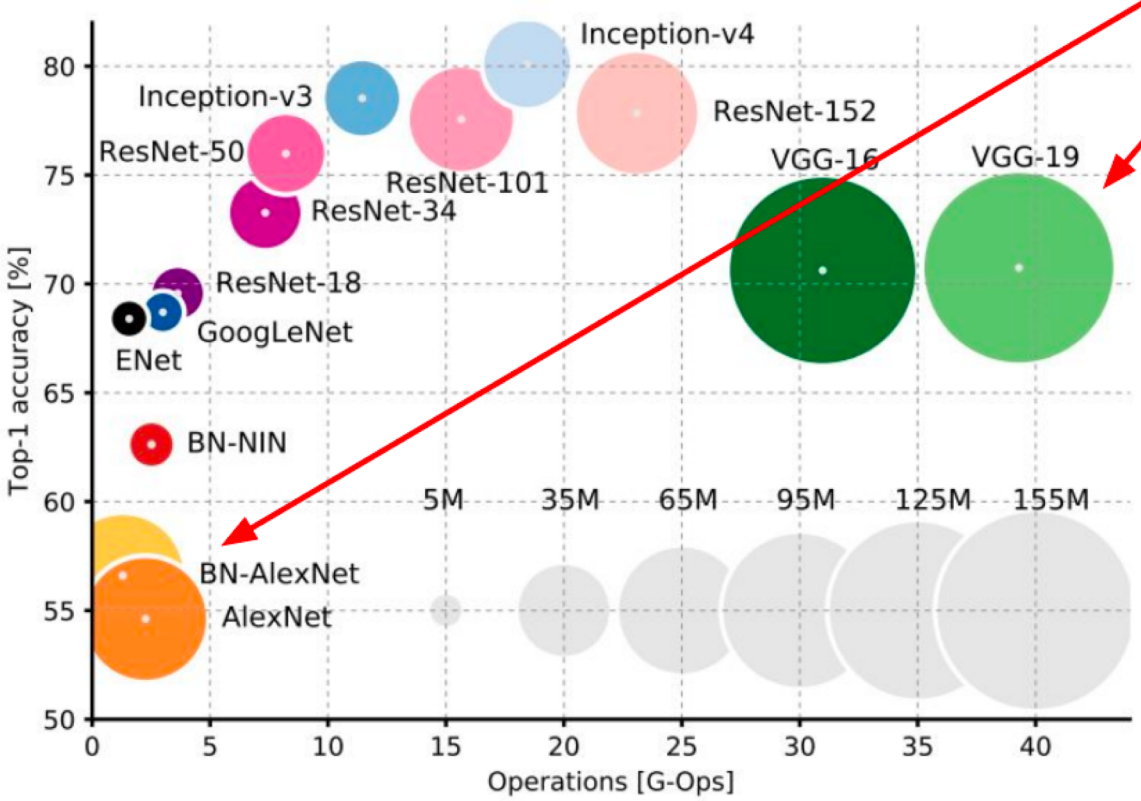
CONVOLUTIONAL NEURAL NETWORKS ARCHITECTURES

Architecture	#Params	#Multiply-Adds	Top-1 Accuracy	Top-5 Accuracy	Year
Alexnet	61M	724M	57.1	80.2	2012
VGG	138M	15.5B	70.5	91.2	2013
Inception-V1	7M	1.43B	69.8	89.3	2013
Resnet-50	25.5M	3.9B	75.2	93	2015

CV-Tricks.com

Top1 Accuracy vs. Compute vs. Size

Alexnet 10^9 FLOPs (1 G-Ops) per image forward prop only



AlexNet and VGG have tons of parameters in the fully connected layers

AlexNet: ~62M parameters

FC6: 256x6x6 -> 4096: 38M params
FC7: 4096 -> 4096: 17M params
FC8: 4096 -> 1000: 4M params
~59M params in FC layers!

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	99 MB	0.749	0.921	25,636,712	168
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation da

- Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.
- Weights are downloaded automatically when instantiating a model.
- They are stored at ~/.keras/models/.

- **Models for image classification with weights trained on ImageNet:**
- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- DenseNet
- NASNet
- MobileNetV2

Training an Imagenet classifier - 50-layer ResNet-50 from 3 weeks to 1 (one) hour with 256 GPUs

Accurate, Large Minibatch SGD:
Training ImageNet in 1 Hour

Priya Goyal Piotr Dollár Ross Girshick Pieter Noordhuis
Lukasz Wesolowski Aapo Kyrola Andrew Tulloch Yangqing Jia Kaiming He

Facebook

Abstract

Deep learning thrives with large neural networks and large datasets. However, larger networks and larger datasets result in longer training times that impede research and development progress. Distributed synchronous SGD offers a potential solution to this problem by dividing SGD minibatches over a pool of parallel workers. Yet to make this scheme efficient, the per-worker workload must be large, which implies nontrivial growth in the SGD minibatch size. In this paper, we empirically show that on the ImageNet dataset large minibatches cause optimization difficulties, but when these are addressed the trained networks exhibit good generalization. Specifically, we show no loss of accuracy when training with large minibatch sizes up to 8192 images. To achieve this result, we adopt a linear scaling rule for adjusting learning rates as a function of minibatch size and develop a new warmup scheme that overcomes optimization challenges early in training. With these simple techniques, our Caffe2-based system trains ResNet-50 with a minibatch size of 8192 on 256 GPUs in one hour, while matching small minibatch accuracy. Using commodity hardware, our implementation achieves ~90% scaling efficiency when moving from 8 to 256 GPUs. This system enables us to train visual recognition models on internet-scale data with high efficiency.

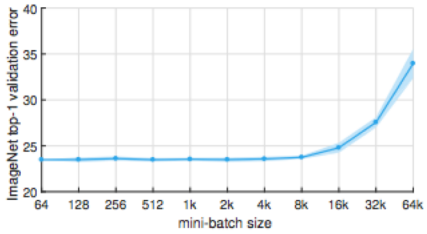
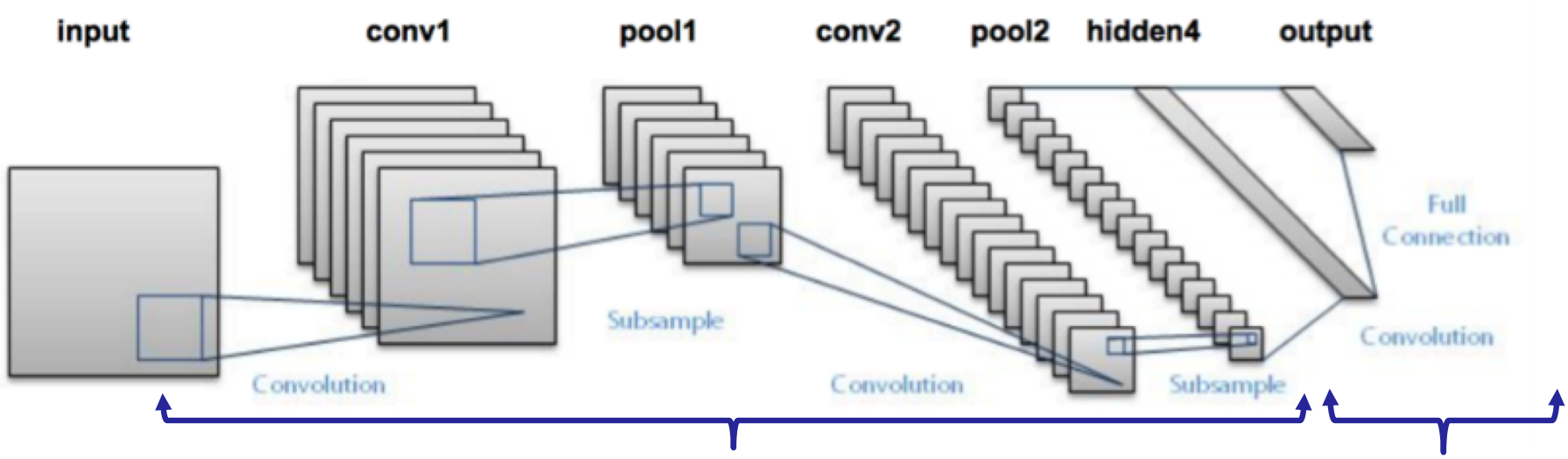


Figure 1. **ImageNet top-1 validation error *vs.* minibatch size.** Error range of plus/minus *two* standard deviations is shown. We present a simple and general technique for scaling distributed synchronous SGD to minibatches of up to 8k images *while maintaining the top-1 error of small minibatch training*. For all minibatch sizes we set the learning rate as a *linear* function of the minibatch size and apply a simple warmup phase for the first few epochs of training. All other hyper-parameters are kept fixed. Using this simple approach, accuracy of our models is invariant to minibatch size (up to an 8k minibatch size). Our techniques enable a linear reduction in training time with ~90% efficiency as we scale to large minibatch sizes, allowing us to train an accurate 8k minibatch ResNet-50 model in 1 hour on 256 GPUs.

mentation [8, 10, 27]. Moreover, this pattern generalizes: larger datasets and network architectures consistently yield improved accuracy across all tasks that benefit from pre-training [22, 40, 33, 34, 35, 16]. But as model and data

https://research.fb.com/wp-content/uploads/2017/06/imagenet1kin1h5.pdf

Transfer learning



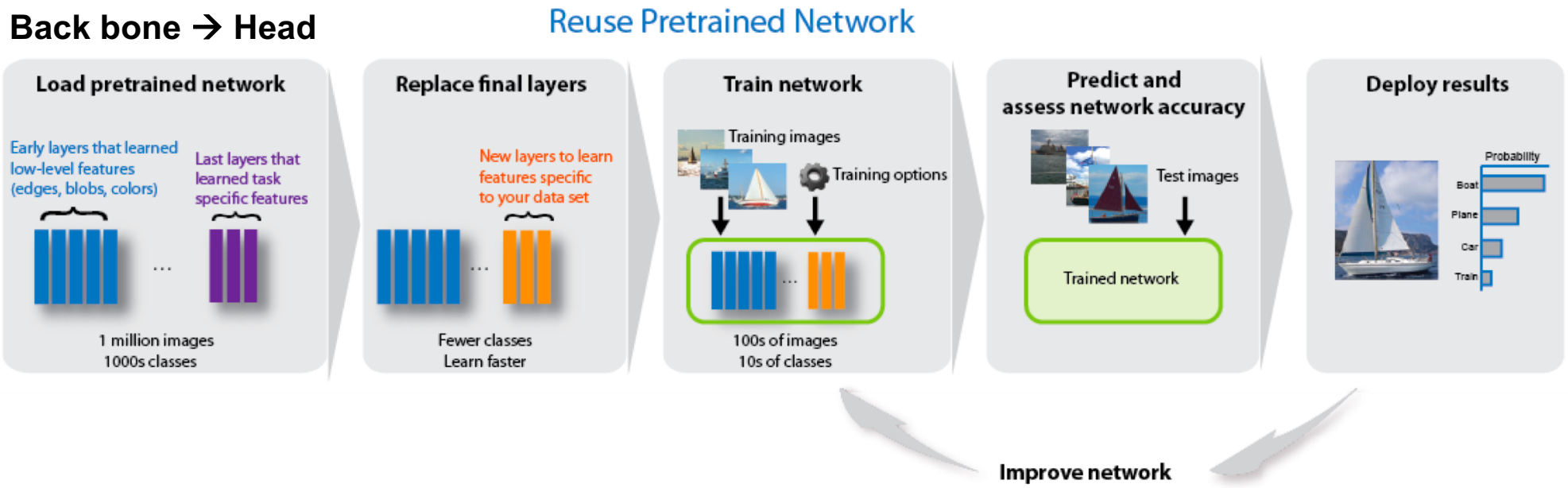
Feature Extraction

Back bone

→. **Head**

Transfer learning

Back bone → Head

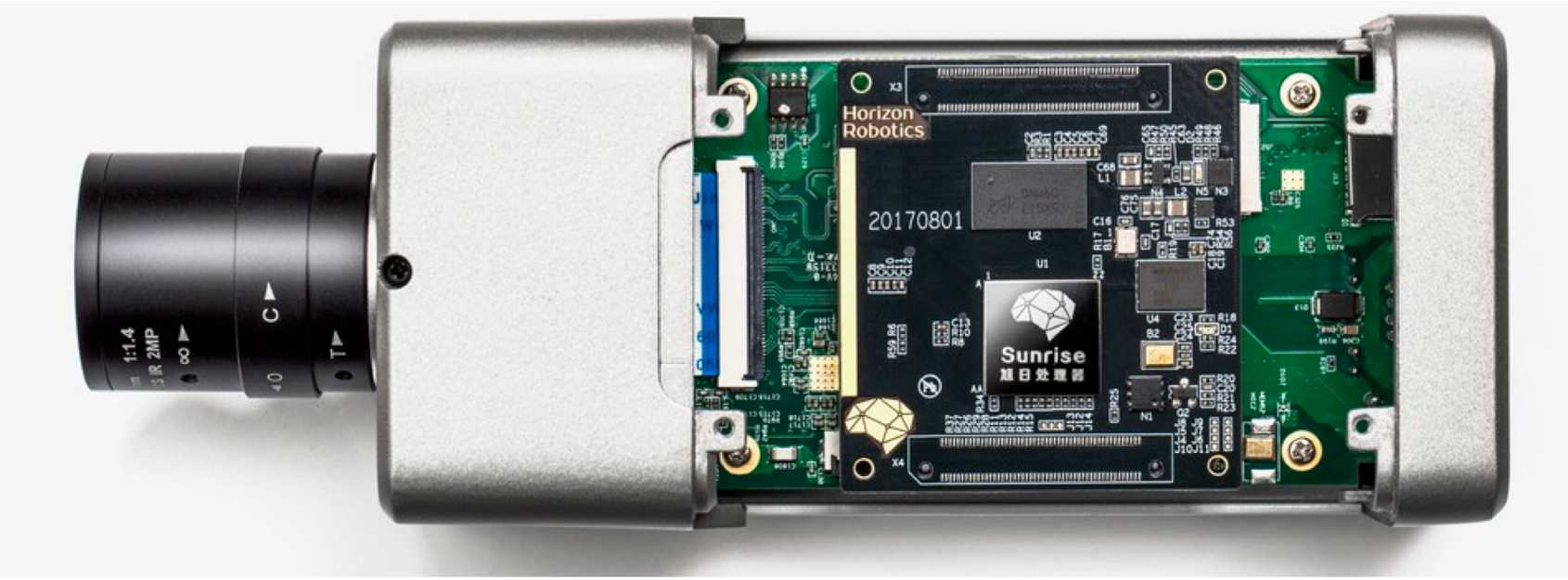


Outline

1. **Introduction**
2. **ML and deep learning review**
3. **What is computer vision?**
    1. Computer vision
    2. Convolutional Neural Nets (CNNs)
4. **Deep NN Architectures for CV Tasks**
    1. Backbone networks
5. **Solving edge-based IoT**
6. **Conclusions and Next steps**

IoT: Devices will know what they are seeing

- **A new breed of chips tuned for artificial intelligence is arriving to help cameras/devices around stores, sidewalks, and homes make sense of what they see.**
- **Cloud based → Edge-based processing**



IoT: smart edge devices

- Even relatively cheap devices will be able to know your name, what you're holding, or that you've been loitering for exactly 17.5 minutes.
- It's the latest development in the tech industry's campaign to build out the internet of things, a slogan for linking everyday devices to the internet so they are interactive and gather data.

- Currently
  - Cloud based solutions
  - Relies on being connected; big pipe
  - Privacy is a concern

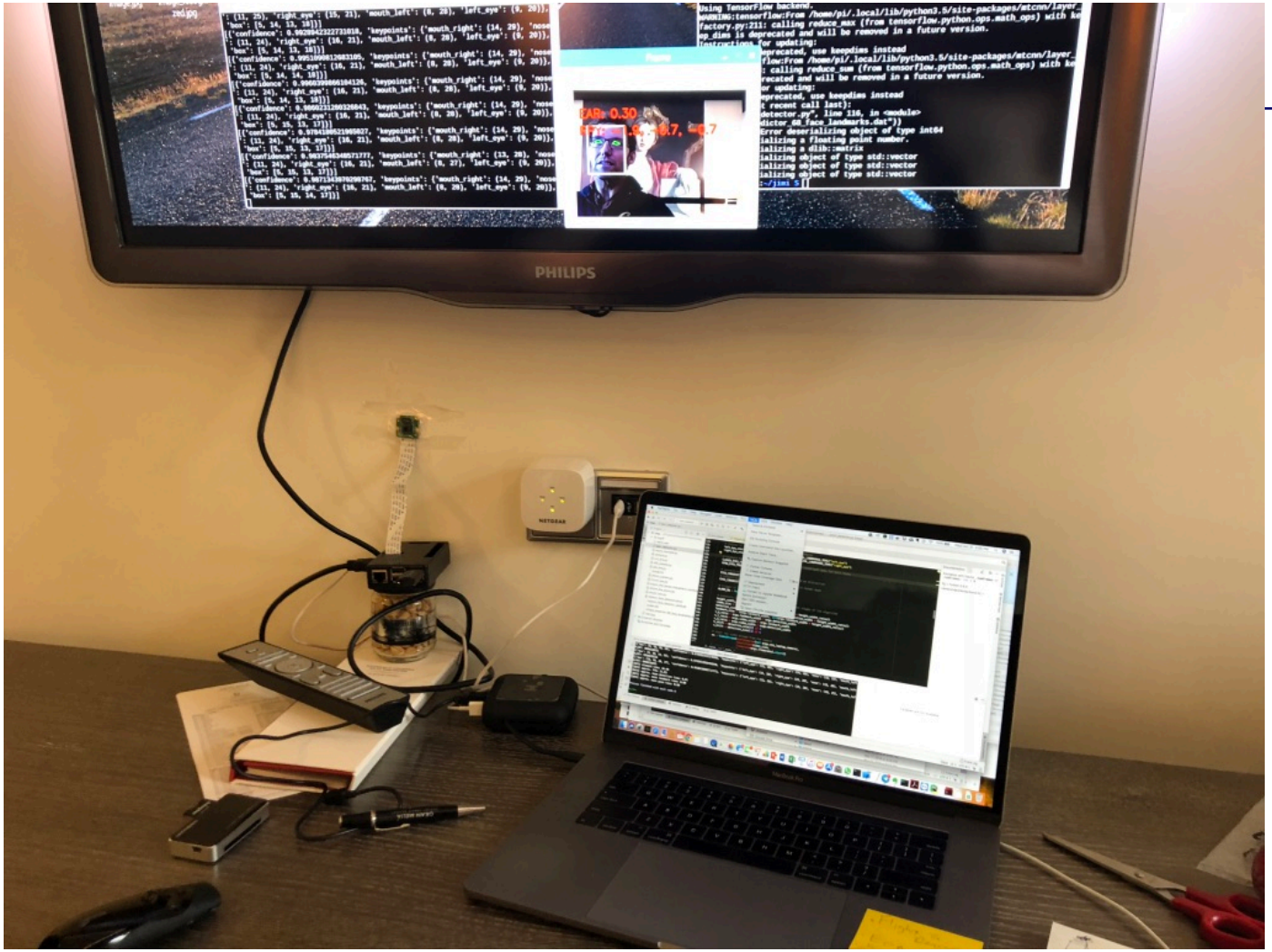
Smart cameras: Sensors + AI

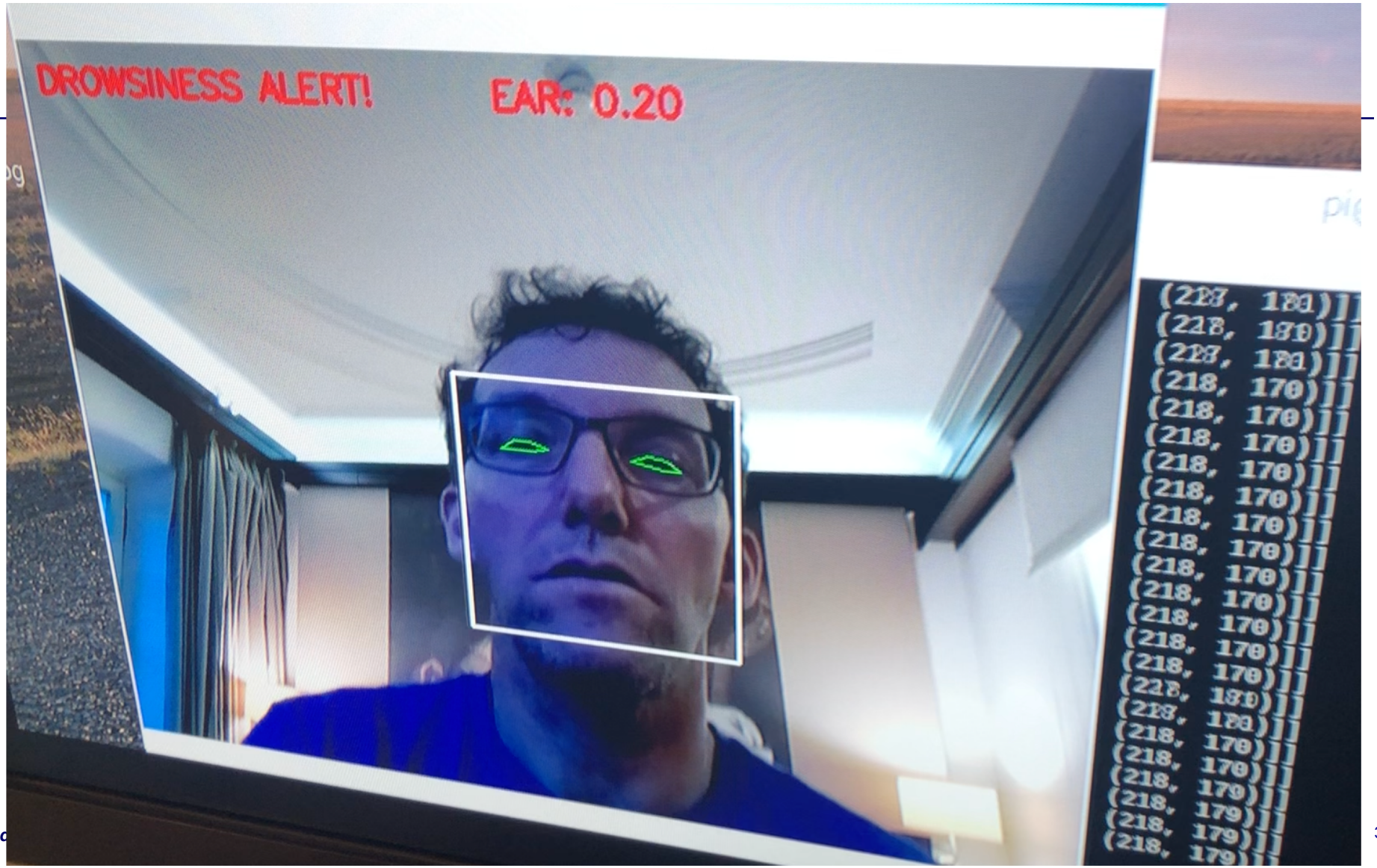
- Smart cameras with built-in AI capabilities can offer new conveniences, like a phone notification that a child just arrived home safely, or that the dog walker really did walk the dog.
- They will also bring new risks to privacy in public and private spaces.
- Companies are exploring how smart cameras can be used to gather marketing data, or assist law enforcement, for example.

Driver Monitoring: 5* safety standard in 2020

- The vehicle is equipped with a camera built into the dashboard and aimed at the driver's face. It allows functionalities such as :
- Identification of the driver in order to allow the vehicle to automatically restore its preferences and settings
- Monitor driver fatigue and alert him when potential drowsiness situation is detected
- Monitor driver attentiveness by ensuring he's keeping his eyes on the road and that he is aware of any dangerous situation
- Pilot an user interface thanks to the eyes by automatically selecting HMI areas, highlight features

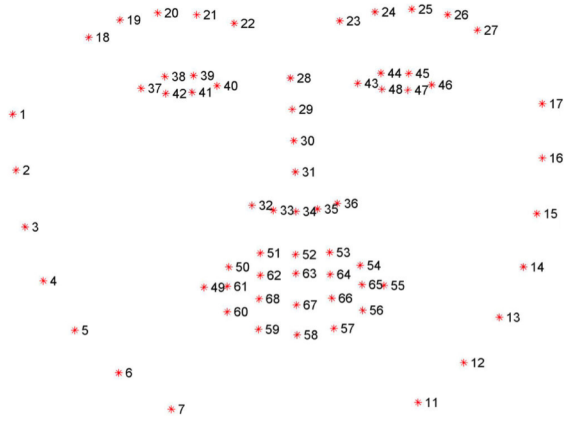
MVP on Raspberry Pi




DROWSINESS ALERT!
EAR: 0.20

Bbox→68 face landmarks

- **Detecting facial landmarks is therefore a two step process:**
- **Step #1: Localize the face in the image.**
- **Step #2: Detect the key facial structures on the face ROI.**



Rasperry Pi https://www.pyimagesearch.com/2018/06/25/raspberry-pi-face-recognition/.

Bbox + 68 face landmarks https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/#

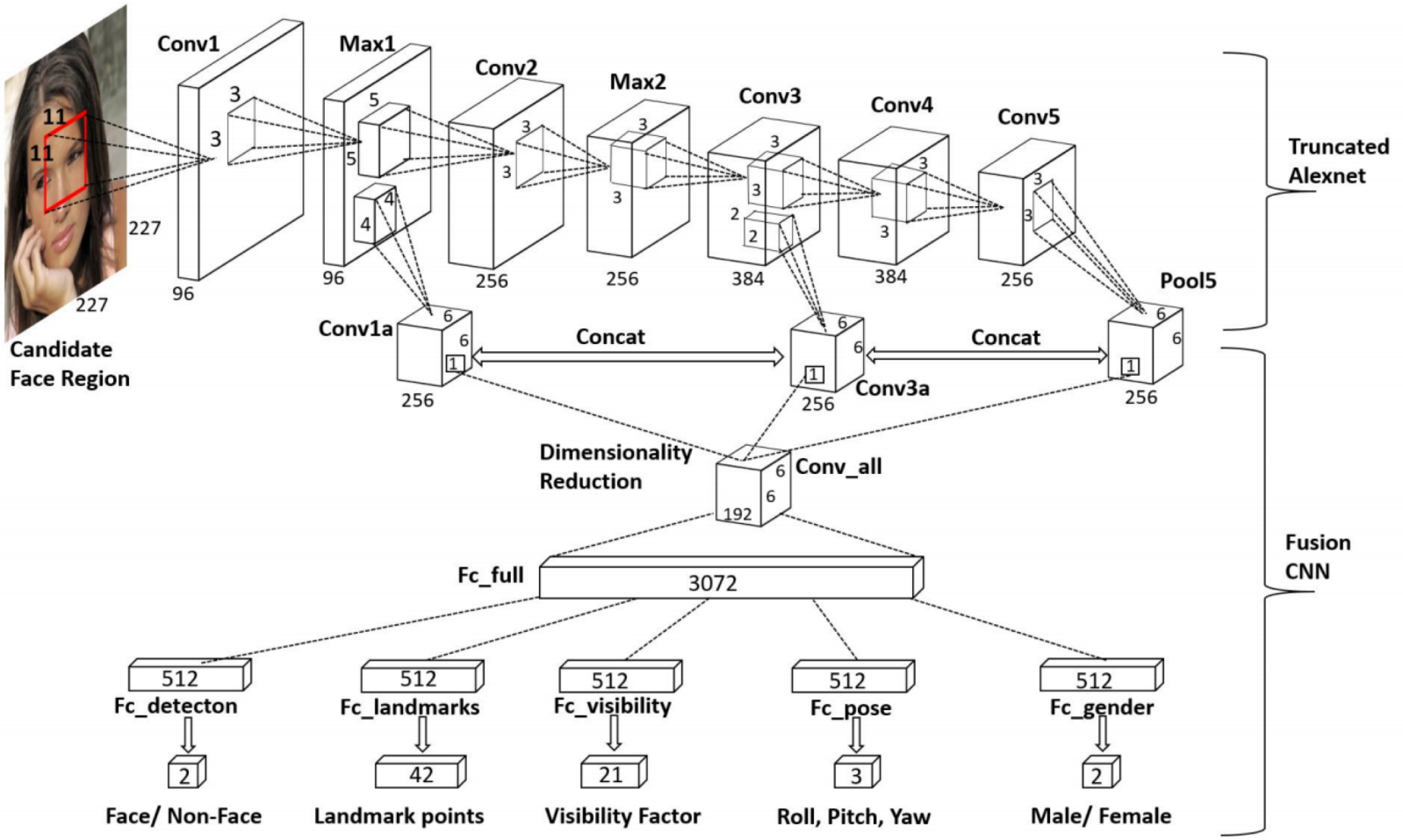


Fig. 2. The architecture of the proposed HyperFace. The network is able to classify a given image region as face or non-face, estimate the head pose, locate face landmarks and recognize gender.

HyperFace with Resnet Backbone

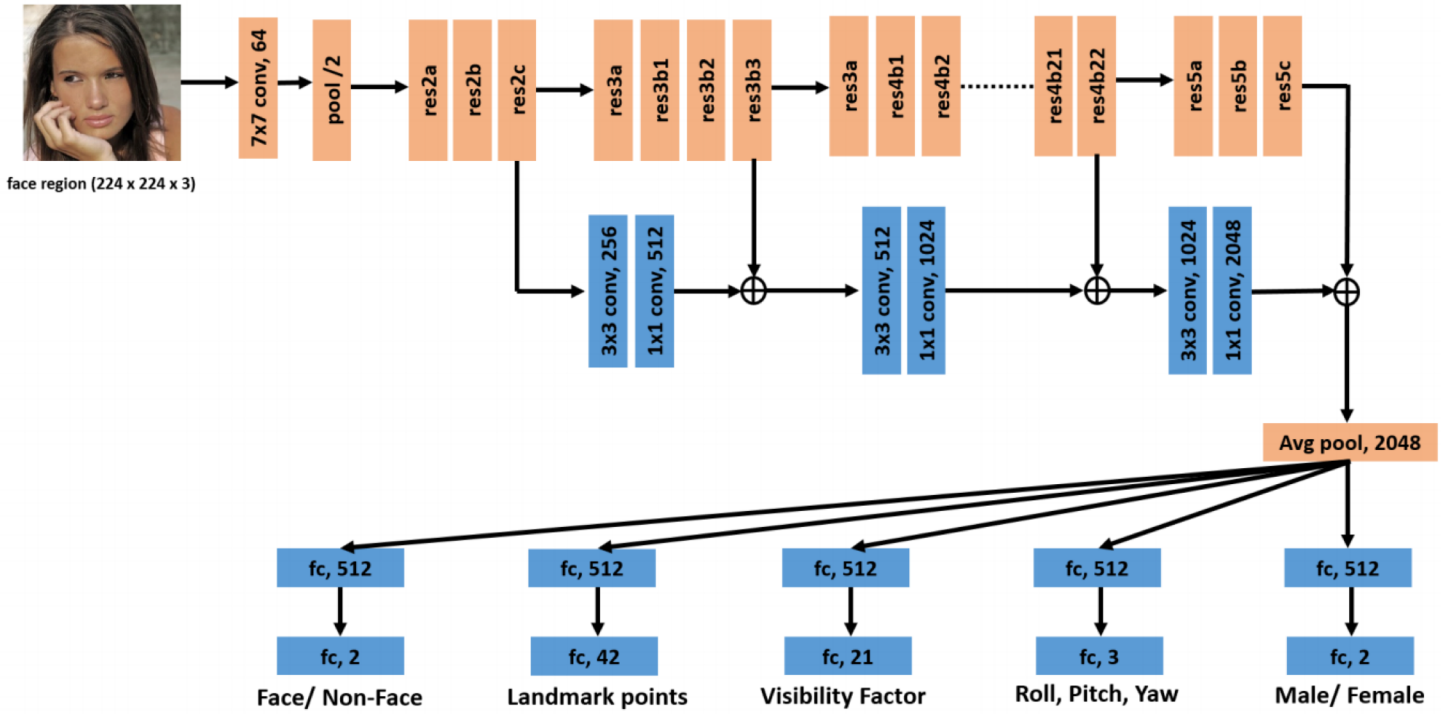


Fig. 6. The architecture of the proposed HyperFace-Resnet (HF-ResNet). ResNet-101 model is used as the backbone network, represented in color orange. The new layers added are represented in color blue. The network is able to classify a given image region as face or non-face, estimate the head pose, locate face landmarks and recognize gender.

HyperFace: Multi-task loss

Pose Estimation: We use the Euclidean loss to train the head pose estimates of roll (p_1), pitch (p_2) and yaw (p_3). We compute the loss for a candidate region having an overlap more than 0.5 with the ground truth, from (5)

$$loss_P = \frac{(\hat{p_1} - p_1)^2 + (\hat{p_2} - p_2)^2 + (\hat{p_3} - p_3)^2}{3}, \quad (5)$$

where $(\hat{p_1}, \hat{p_2}, \hat{p_3})$ are the estimated pose labels.

Gender Recognition: Predicting gender is a two class problem similar to face detection. For a candidate region with overlap of 0.5 with the ground truth, we compute the softmax loss given in (6)

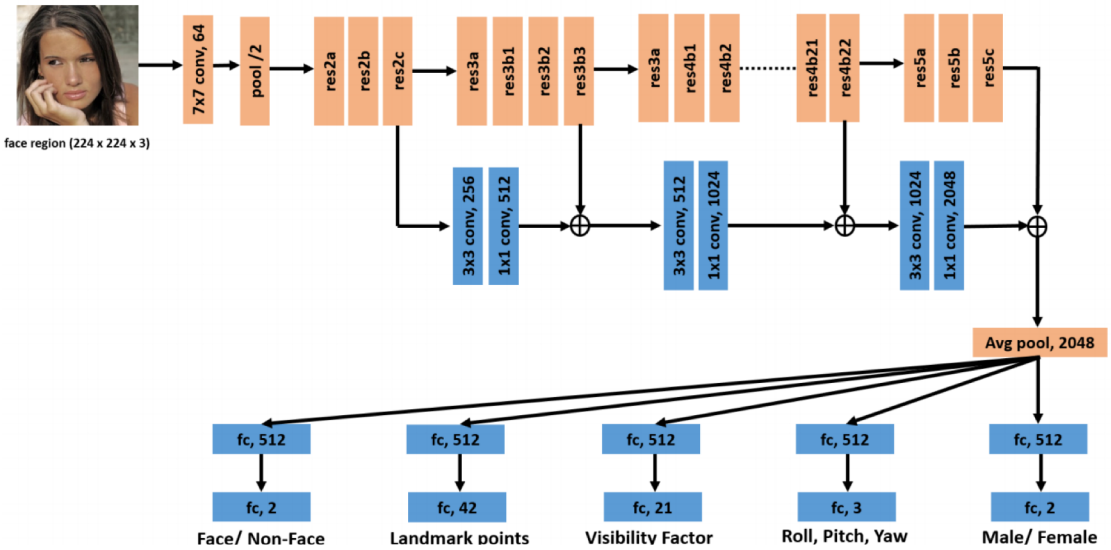
$$loss_G = -(1-g) \cdot \log(1-p_g) - g \cdot \log(p_g), \quad (6)$$

where $g = 0$ if the gender is male, or else $g = 1$. Here, (p_0, p_1) is the two dimensional probability vector computed from the network.

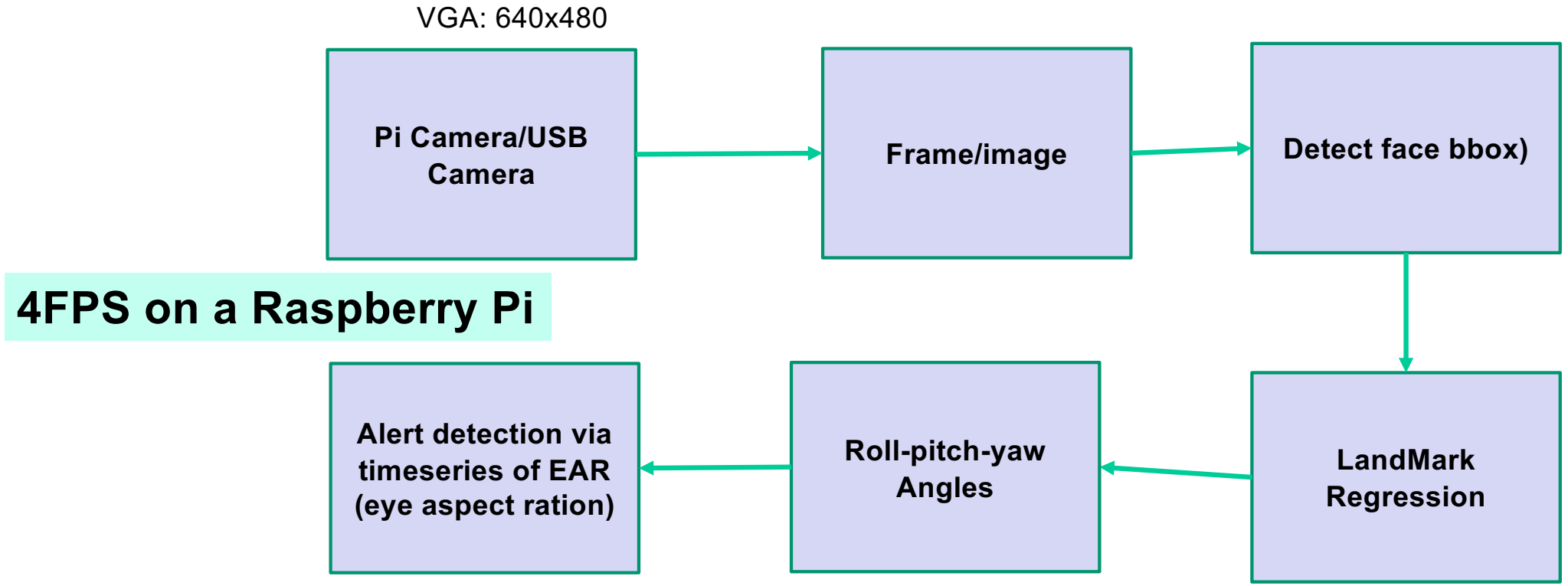
The total loss is computed as the weighted sum of the five individual losses as shown in (7)

$$loss_{full} = \sum_{i=1}^{i=5} \lambda_{t_i} loss_{t_i}, \quad (7)$$

where t_i is the i^{th} element from the set of tasks $T = \{D, L, V, P, G\}$. The weight parameter λ_{t_i} is decided based on the importance of the task in the overall loss. We choose $(\lambda_D = 1, \lambda_L = 5, \lambda_V = 0.5, \lambda_P = 5, \lambda_G = 2)$ for our experiments. Higher weights are assigned to landmark localization and pose estimation tasks as they need spatial accuracy.



Pi Camera → Frame→ bbox→ Landmarks→RowPitchYaw→ Alert Classifier



4FPS on a Raspberry Pi

4 models in this pipeline

Complex architectures

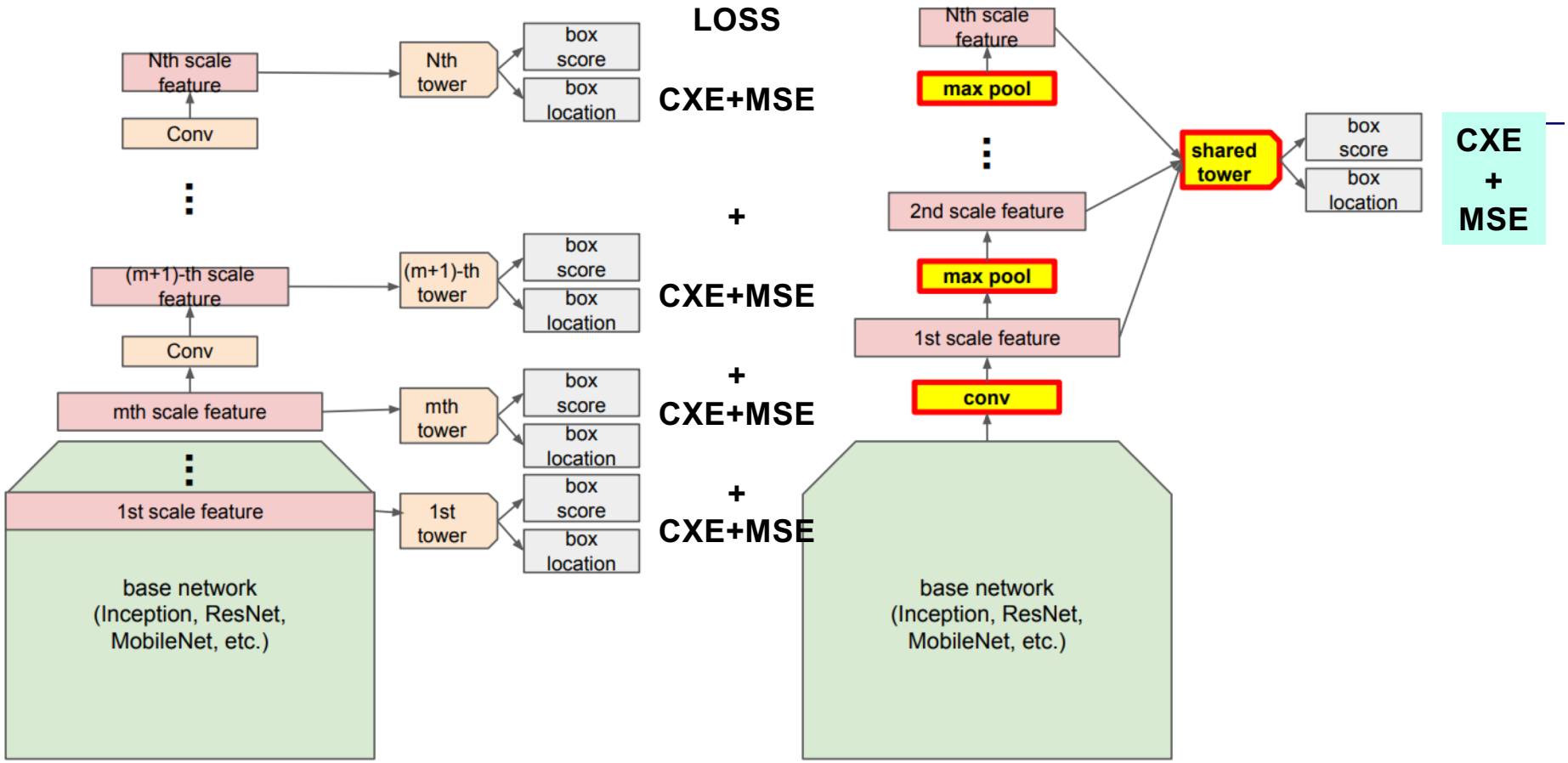


Figure 1. Architecture comparison between the Pooling Pyramid Network (PPN) and vanilla SSD. Left: vanilla SSD, Right: PPN. Note that the changes in PPN are highlighted: (1) using max pool to build the feature pyramid, (2) using shared convolutional predictors for box classification and regression.

4Meg of memory for model and image

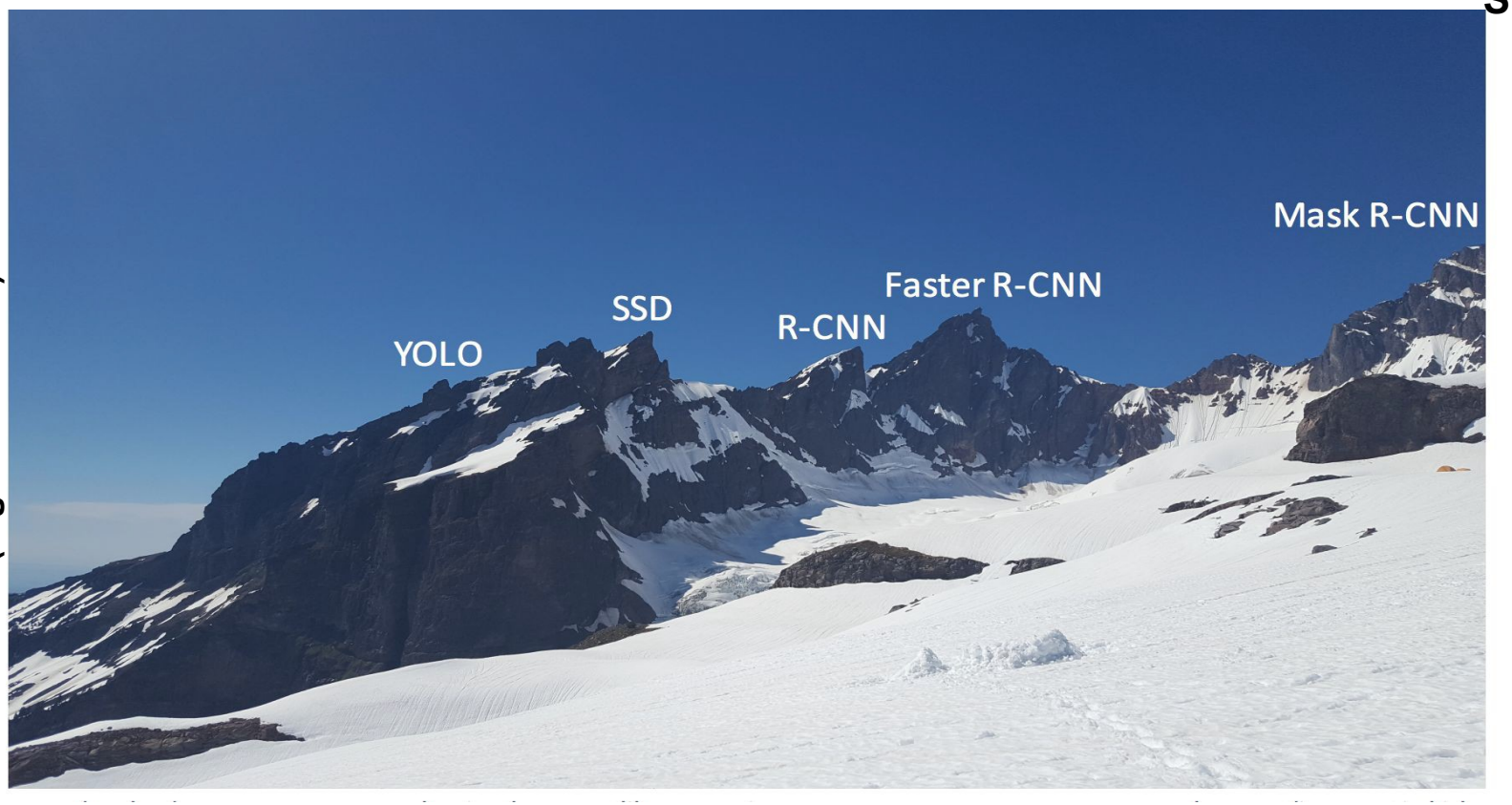
Model	Top-1	Top-5	Ops	Size
AlexNet	57.0	80.3	2.27 Bn	238 MB
Darknet Reference	**61.1**	**83.0**	**0.81 Bn**	28 MB
SqueezeNet	57.5	80.3	2.17 Bn	4.8 MB
Tiny Darknet	58.7	81.7	0.98 Bn	**4.0 MB**

Make the pipeline run on a Chip

4Meg of memory
16-32 tasks
32Gig ops

Modern Convolutional Object Detectors

Size versus ThruPut



mAP. (higher is better)

Complexity

Image from: http://deeplearning.csail.mit.edu/instance_ross.pdf

Classification + Localization: Task

Classification: C classes

Input: Image

Output: Class label

Evaluation metric: Accuracy

CAT

Localization:

Input: Image

Output: Box in the image (x, y, w, h)

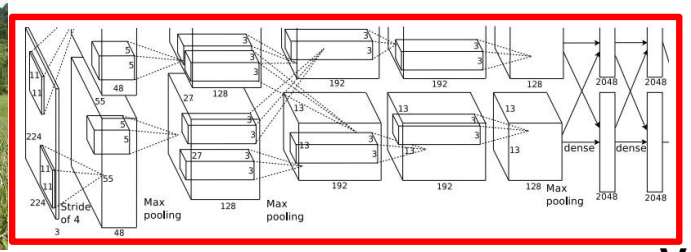
Evaluation metric: Intersection over Union

(x, y, w, h)

Classification + Localization: Do both

Classification + Localization

This image is CC0 public domain



Often pretrained on ImageNet
(Transfer learning)

Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label: Cat

Softmax Loss

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

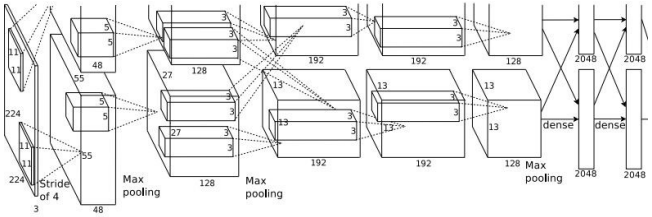
Correct box: (x', y', w', h')

+ → **Loss**

Treat localization as a regression problem!

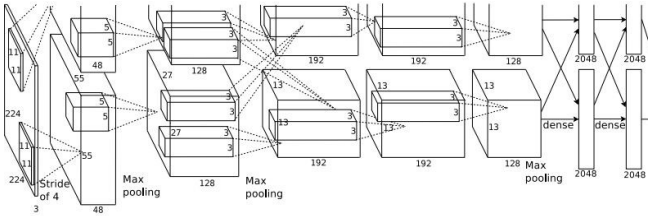
Each image needs a different number of outputs!

Object Detection as Regression?



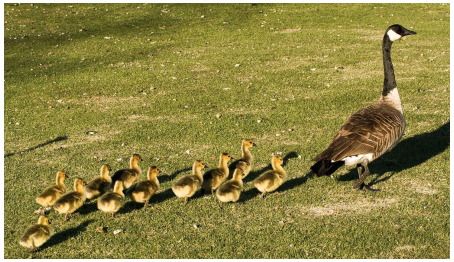
CAT: (x, y, w, h)

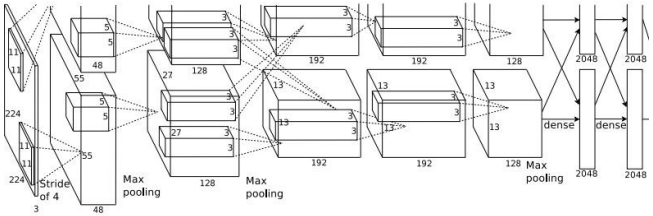
4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

16 numbers





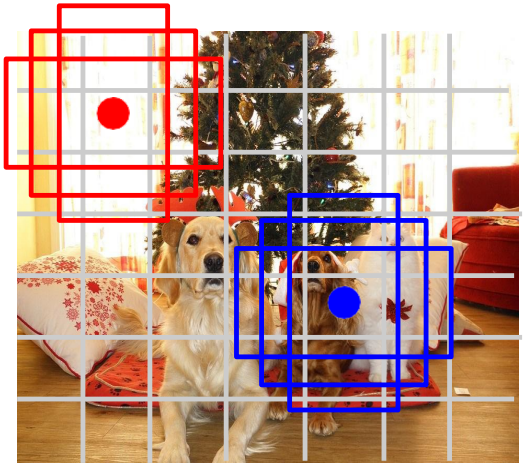
DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Many numbers!

Detection without Proposals: Faster R-CNN

Go from input image to tensor of scores with one big convolutional network!

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

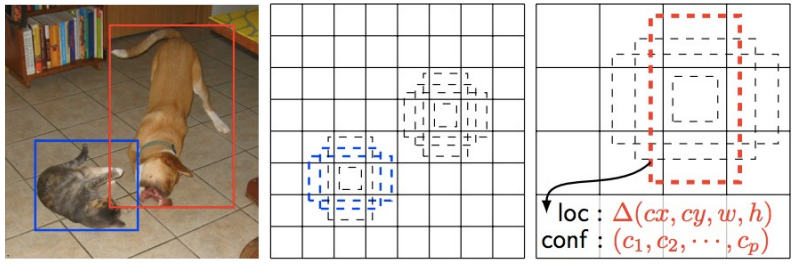
- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

Output:
7 x 7 x (5 * B + C)

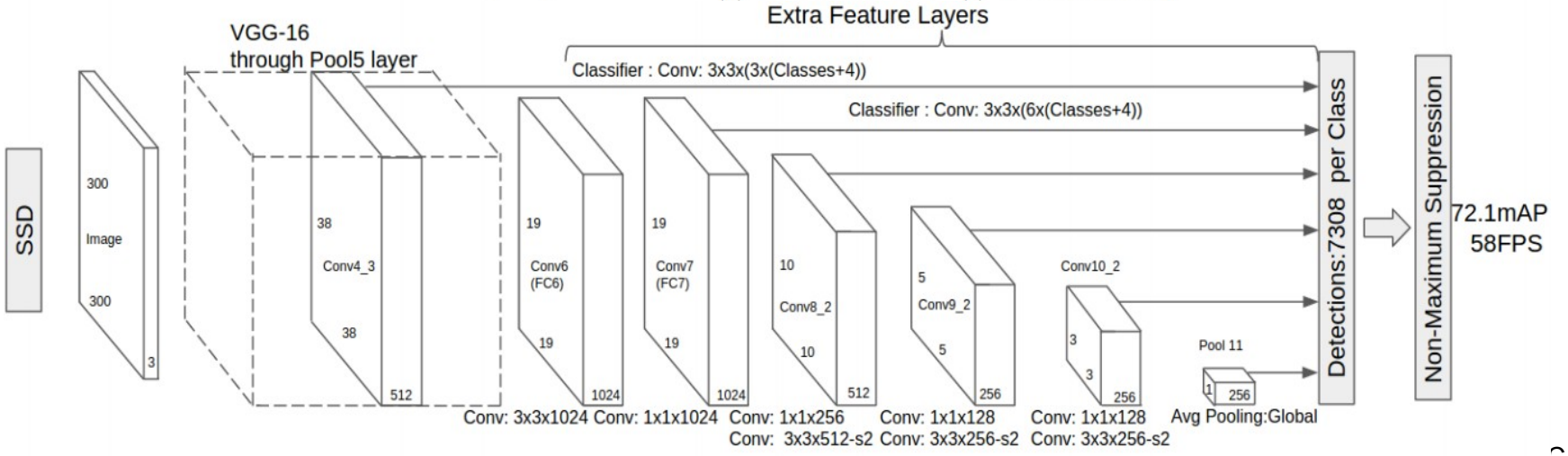
Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

Object detection trends

- Fully convolutional detection networks



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map



W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Ber
SSD: Single Shot MultiBox Detector, arXiv 2016.

Segnet Architecture

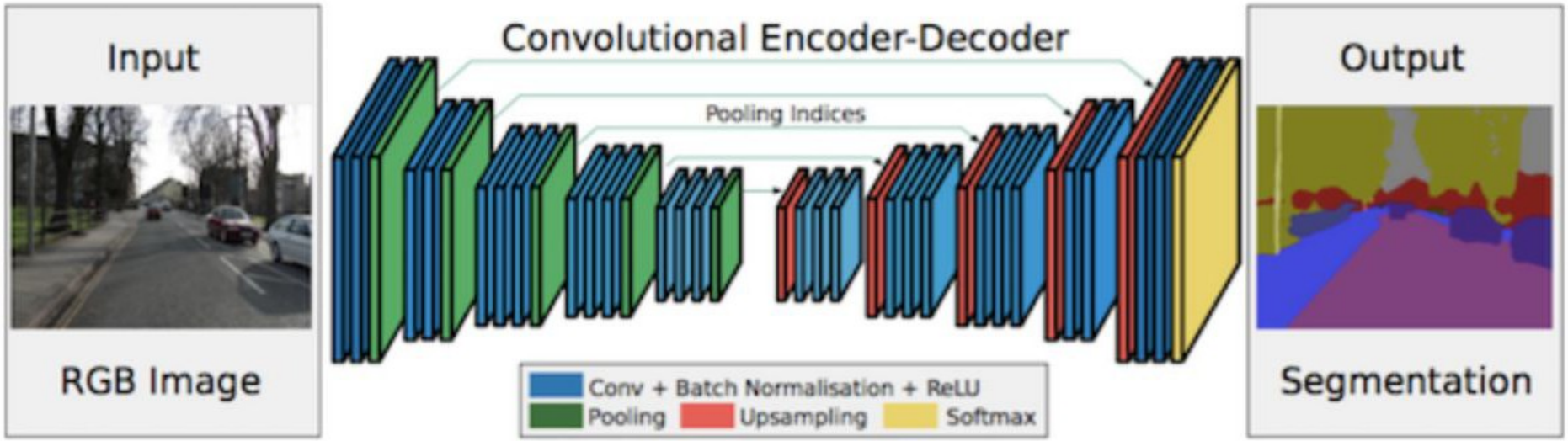
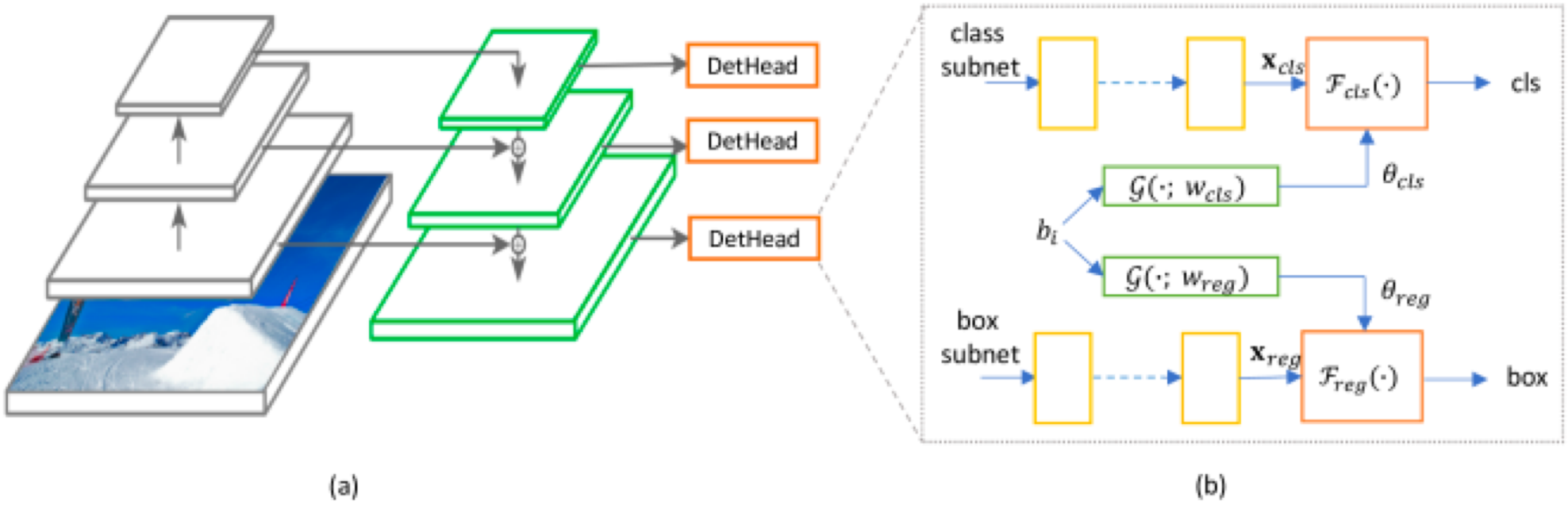


Image from: http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review

RetinaNet Architecture



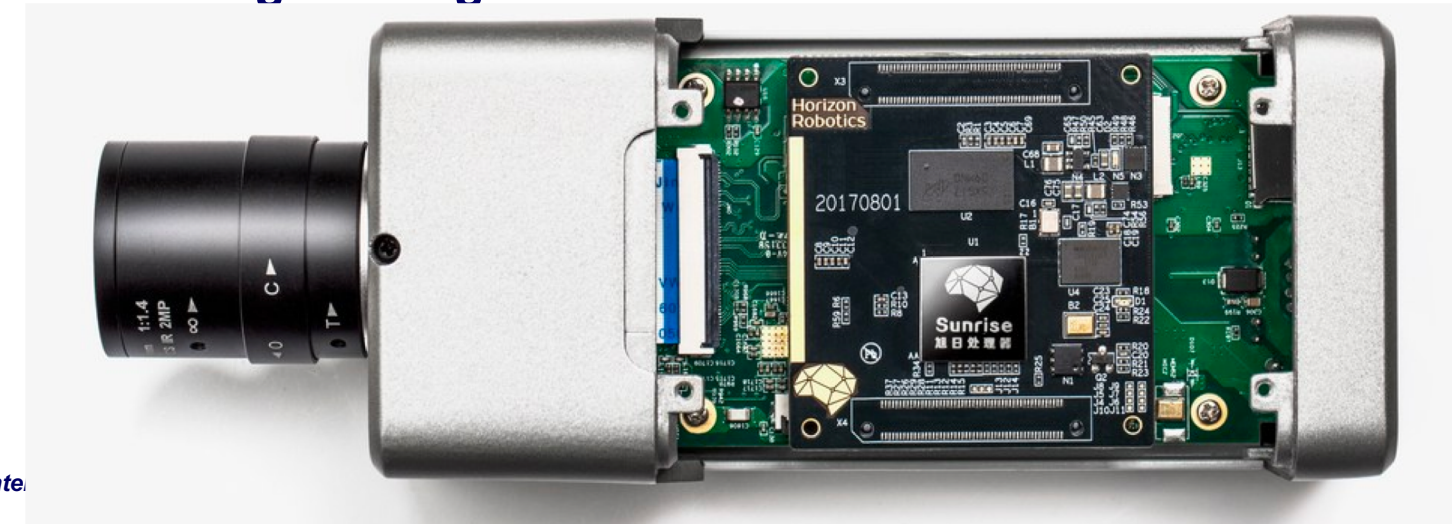

truck 0.959
person 0.968
tie 0.899
car 0.967
car 0.983
car 0.957
truck 0.940
truck 0.779
bicycle 0.761

Outline

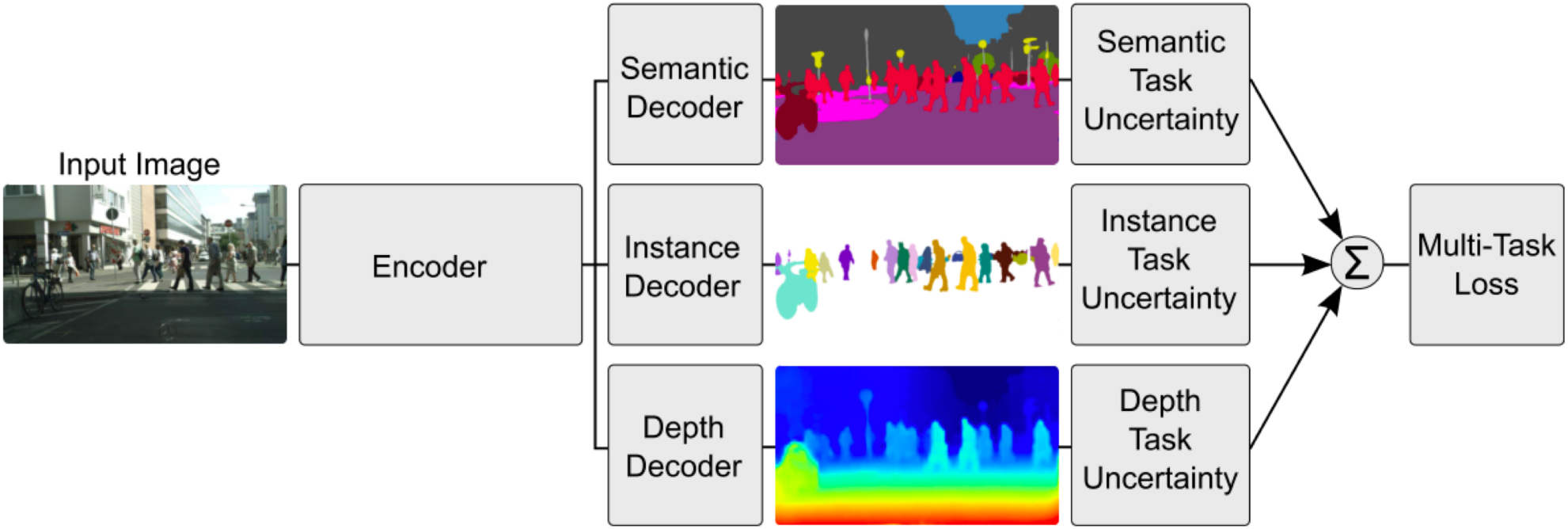
1. **Introduction**
2. **ML and deep learning review**
3. **What is computer vision?**
    1. Computer vision
    2. Convolutional Neural Nets (CNNs)
4. **Deep NN Architectures for CV Tasks**
    1. Backbone networks
5. **Solving edge-based IoT**
6. **Conclusions and Next steps**

Conclusions:
IoT devices will know what they are seeing

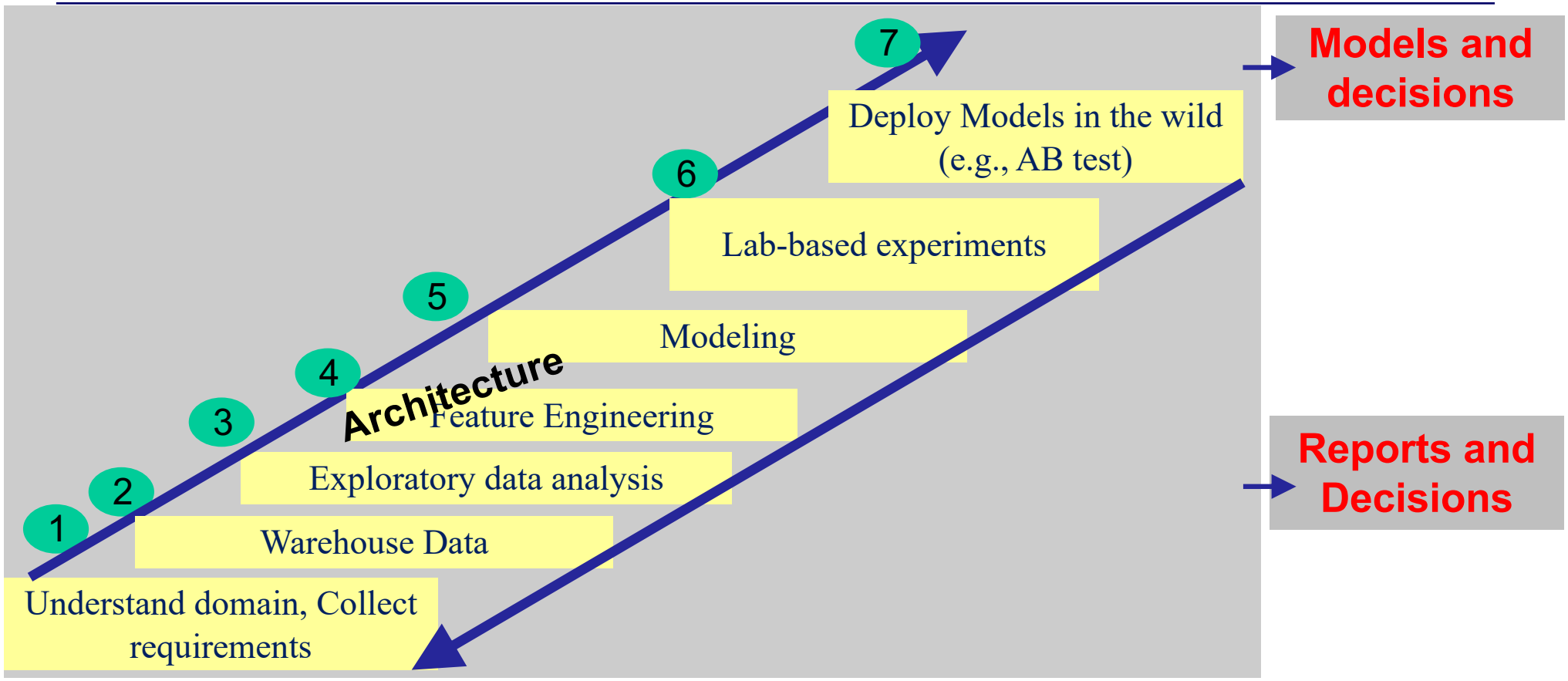
- **A new breed of chips tuned for artificial intelligence is arriving to help cameras/devices around stores, sidewalks, and homes make sense of what they see.**
- **Cloud based → Edge-based processing**
  - small memory, high throughput, low cost
- **Privacy**
- **Architecture engineering**



Architecture and loss function Engineering



Typical Abstract ML Pipeline



Plug and Play Zoo!

Limitations of AI currently

- **Despite the flurry of recent progress in AI and wild prognostications about its near future, there are still many things that machines can't do,**
  - understanding the nuances of language,
  - common-sense reasoning,
  - learning a new skill from just one or two examples.
- **Some models can be big**
  - 50-100 Gig model is massive for workstations never mind for mobile devices
- **Biases**
  - E.g., Amplified societal biases around demographics such as race or gender
  - Metropolitan versus rural

End of slides