ML Summit '19

Google Developers

convolutional neural networks with swift (and 🐍)

brett koonce
cto, quarkworks
GDG Columbia, MO

#MLSummit

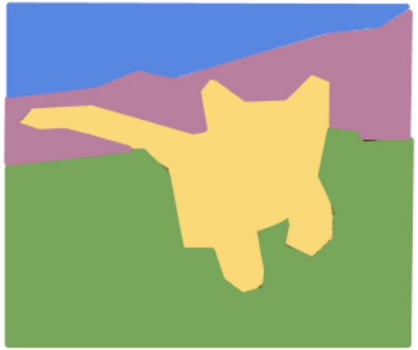
outline

- goal: convolutional neural networks for image recognition
- neural networks, 1d mnist demo
- convolutions, 2d mnist demo
- color, [2d] cifar demo
- vgg, resnet, imagenet + tpu demo
- efficientnet, edge tpu demo
- recap

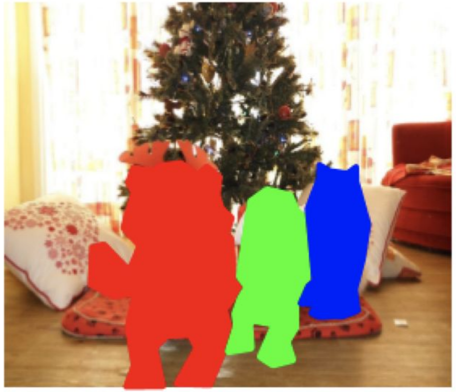
computer vision problems

- image recognition
- object detection
- image segmentation
- instance segmentation

CAT



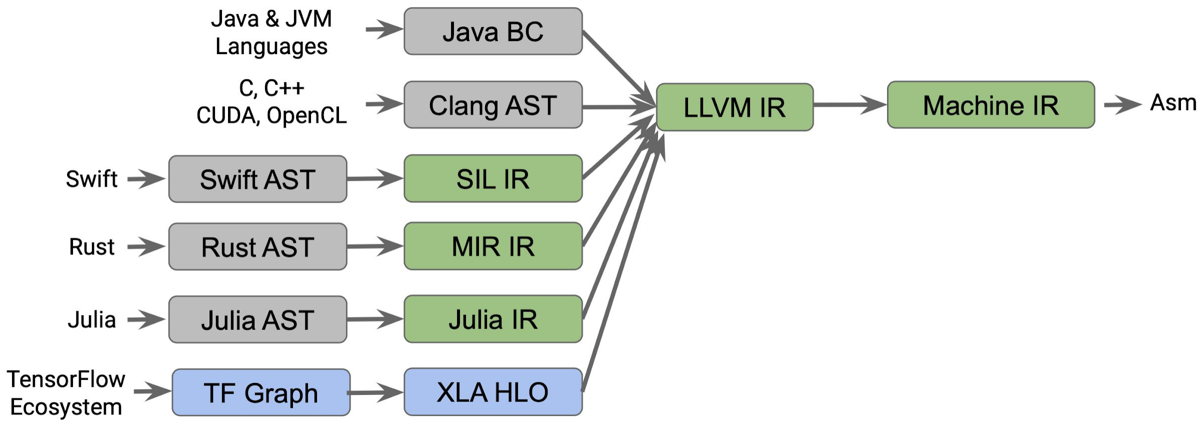
GRASS, CAT, TREE, SKY

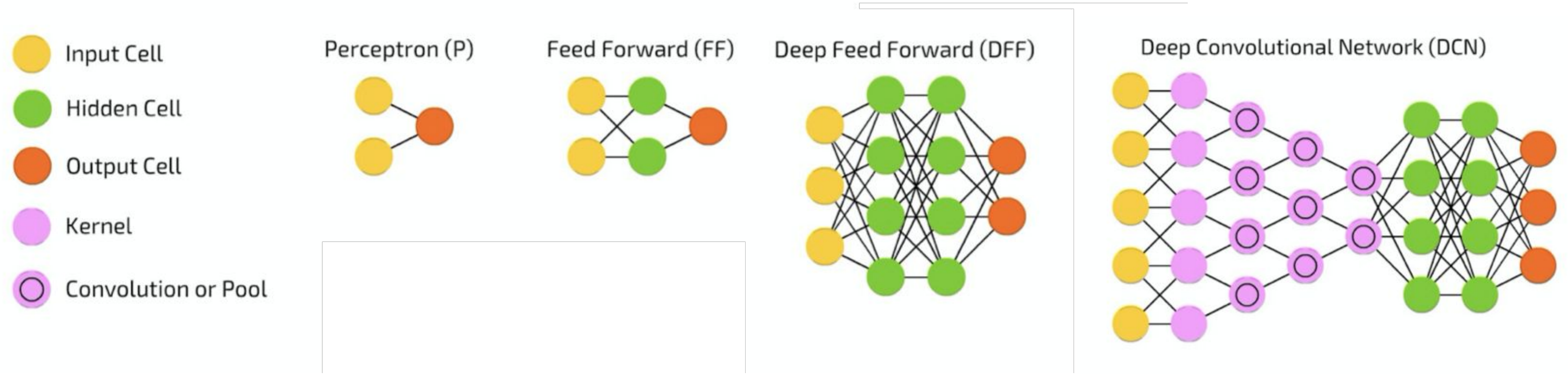


DOG, DOG, CAT

neural networks

- activation function
- seperate high dimensional data
- images are 5d: [r, g, b, h, w] → [c]
- a(b(c(d(e(...))))) →
  a. backpropagation + chain rule
- auto differentiation, swift, mlir

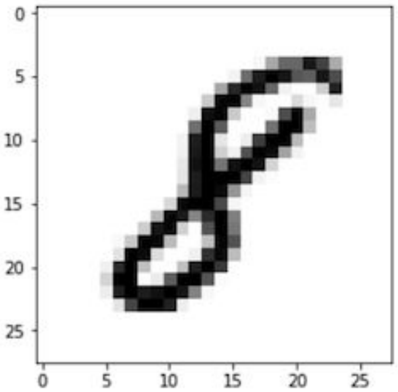




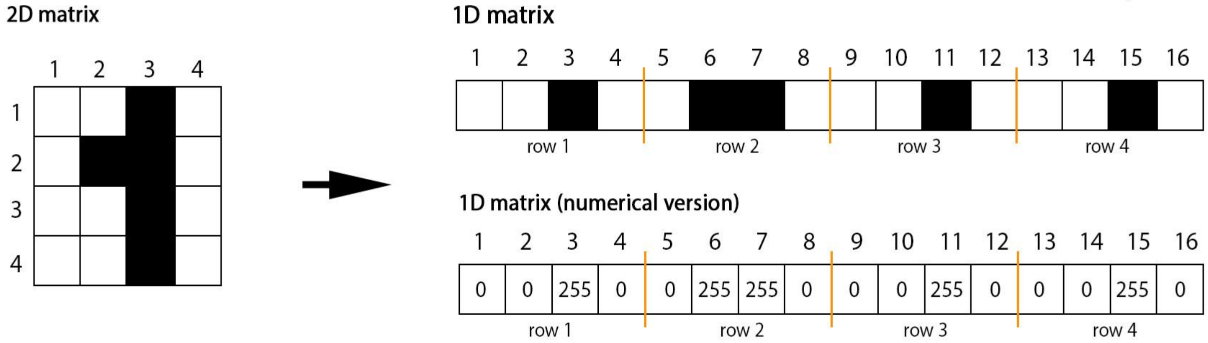
1d mnist

- mnist: 28x28x1 (h, w, 0..255 greyscale)
- Convert to 1d vector: [row1, row2, row3...row28] == (784 x 1)
- 2 * 512 fully connected layers
- github.com/huan/swift-MNIST

INPUT	784x1
Dense	512
Dense	512
OUTPUT	10



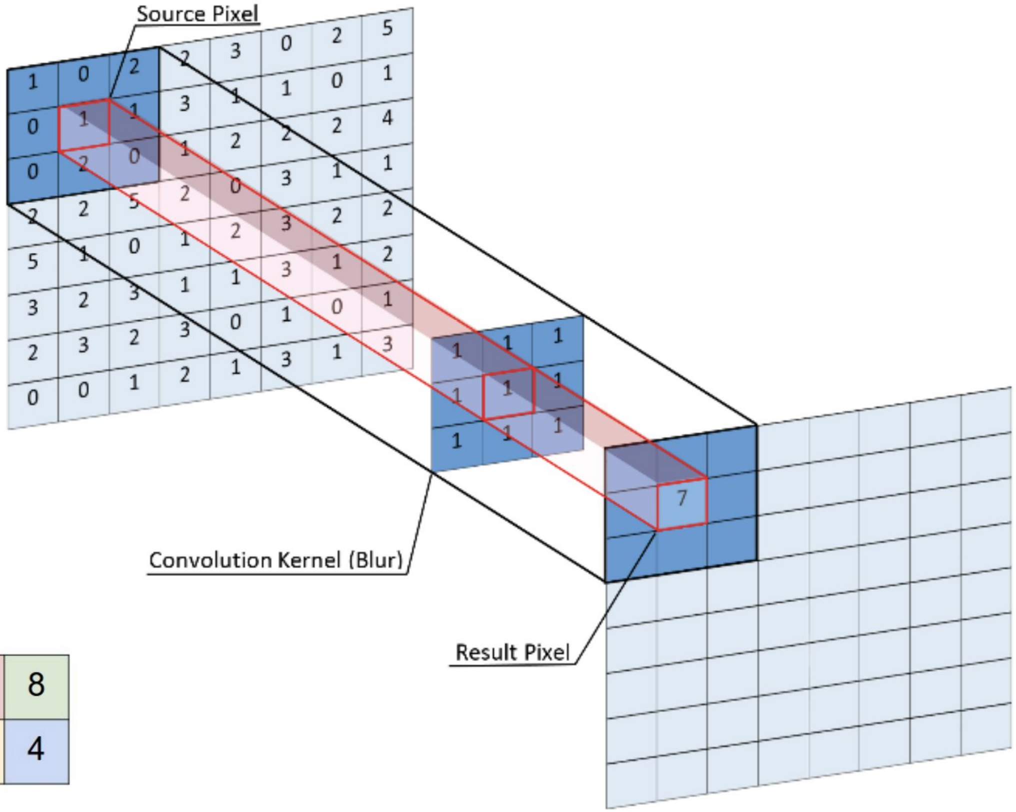
2D matrix to 1D matrix conversion

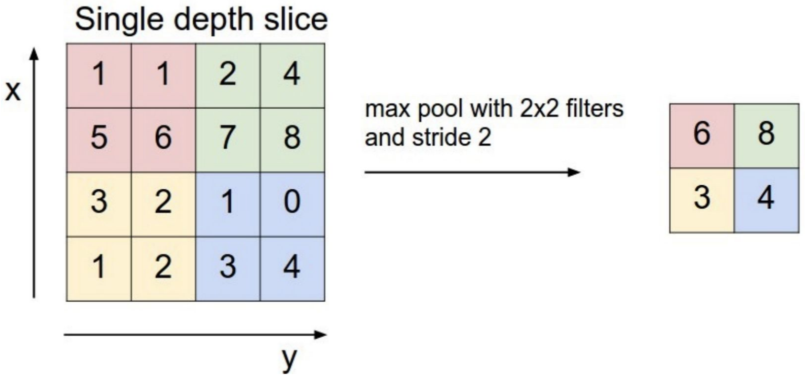


```swift
struct MLP: Layer {
  typealias Input = Tensor<Float>
  typealias Output = Tensor<Float>

  var flatten = Flatten<Float>()
  var dense = Dense<Float>(inputSize: 784, outputSize: 512, activation: relu)
  var innerLayer = Dense<Float>(inputSize: 512, outputSize: 512, activation: relu)
  var output = Dense<Float>(inputSize: 512, outputSize: 10, activation: softmax)

  @differentiable
  public func callAsFunction(_ input: Input) -> Output {
    return input.sequenced(through: flatten, dense, innerLayer, output)
  }
}
```

convolutions

- 3x3 blur example
- 3x3 striding
- 2x2 maxpool





2d mnist

- 3x3 convolution
- 3x3 convolution
- 2x2 maxpool
- 2 * 512 fully connected layers
- demo (modify 1d demo)

INPUT	28x28x1
Conv 1-1	3x3
Conv 1-2	3x3
Pooling	2x2
Dense	512
Dense	512
OUTPUT	10

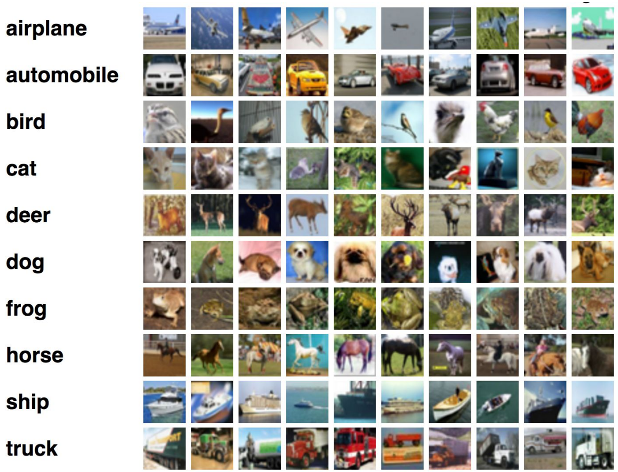
```swift
struct CNN: Layer {
  typealias Input = Tensor<Float>
  typealias Output = Tensor<Float>

  var conv1a = Conv2D<Float>(filterShape: (3, 3, 1, 32), padding: .same, activation: relu)
  var conv1b = Conv2D<Float>(filterShape: (3, 3, 32, 32), activation: relu)
  var pool1 = MaxPool2D<Float>(poolSize: (2, 2), strides: (2, 2))

  var flatten = Flatten<Float>()
  var inputLayer = Dense<Float>(inputSize: 13 * 13 * 32, outputSize: 512, activation: relu)
  var hiddenLayer = Dense<Float>(inputSize: 512, outputSize: 512, activation: relu)
  var outputLayer = Dense<Float>(inputSize: 512, outputSize: 10, activation: softmax)

  @differentiable
  public func callAsFunction(_ input: Input) -> Output {
    let cnn_input = input.sequenced(through: conv1a, conv1b, pool1)
    return cnn_input.sequenced(through: flatten, inputLayer, hiddenLayer, outputLayer)
  }
}
```

2d stacking, color, cifar

- [3x3 striding, 3x3 striding, 2x2 maxpool]

  → [block1] + [block2]

- 2 * 512 fully connected layers
- Cifar: (32x32x3, 10 categories)

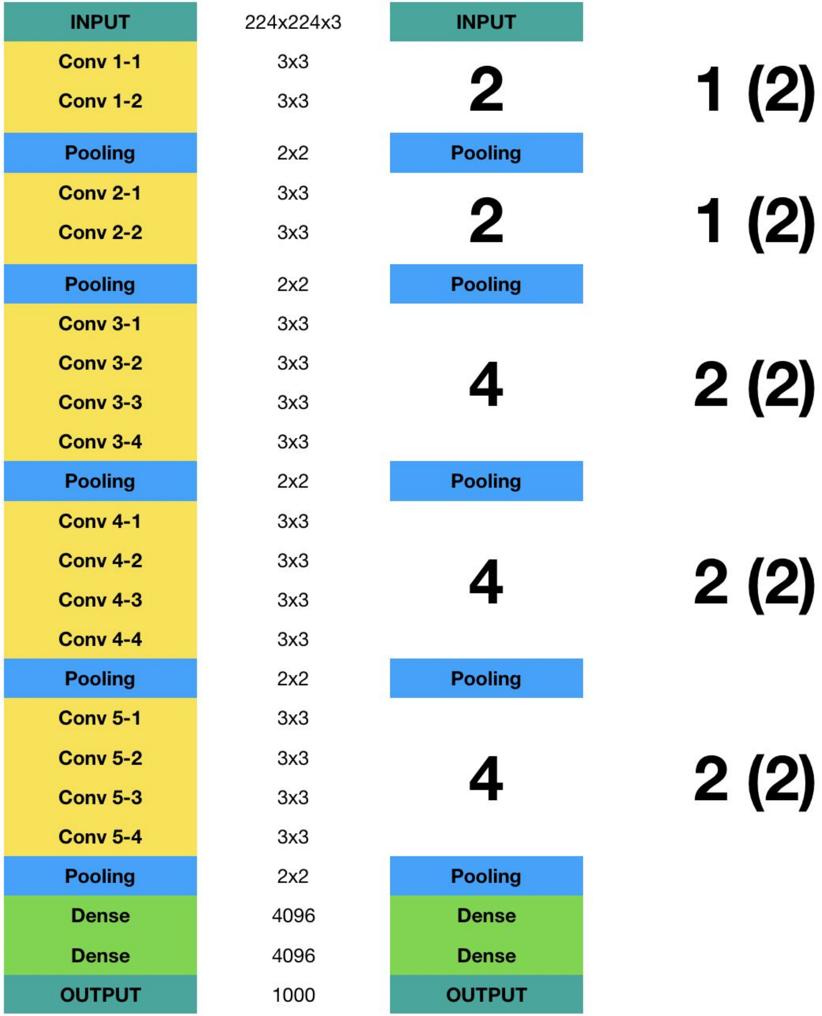


INPUT	32x32x3	INPUT
Conv 1-1	3x3	2
Conv 1-2	3x3	
Pooling	2x2	Pooling
Conv 2-1	3x3	2
Conv 2-2	3x3	
Pooling	2x2	Pooling
Dense	512	Dense
Dense	512	Dense
OUTPUT	10	OUTPUT

```swift
struct CIFARModel: Layer {
    typealias Input = Tensor<Float>
    typealias Output = Tensor<Float>

    var conv1a = Conv2D<Float>(filterShape: (3, 3, 3, 32), padding: .same, activation: relu)
    var conv1b = Conv2D<Float>(filterShape: (3, 3, 32, 32), activation: relu)
    var pool1 = MaxPool2D<Float>(poolSize: (2, 2), strides: (2, 2))

    var conv2a = Conv2D<Float>(filterShape: (3, 3, 32, 64), padding: .same, activation: relu)
    var conv2b = Conv2D<Float>(filterShape: (3, 3, 64, 64), activation: relu)
    var pool2 = MaxPool2D<Float>(poolSize: (2, 2), strides: (2, 2))

    var flatten = Flatten<Float>()
    var dense1 = Dense<Float>(inputSize: 6 * 6 * 64, outputSize: 512, activation: relu)
    var dense2 = Dense<Float>(inputSize: 512, outputSize: 512, activation: relu)
    var output = Dense<Float>(inputSize: 512, outputSize: 10, activation: identity)

    @differentiable
    func callAsFunction(_ input: Input) -> Output {
        let conv1 = input.sequenced(through: conv1a, conv1b, pool1)
        let conv2 = conv1.sequenced(through: conv2a, conv2b, pool2)
        return conv2.sequenced(through: flatten, dense1, dense2, output)
    }
}
```

vgg (2014)

- arxiv:1409.1556
- vgg16: [2, 2, 3, 3, 3]
- vgg19: [2, 2, 4, 4, 4]
- 2 * 4096 fully connected layers
- imagenet: (224x224x3, 1000 categories)



residual networks (2015)

- arxiv: 1512.03385
- cnn backbone + skip connections → resnet 34
- resnet 34 + bottleneck blocks (1x3x1) → resnet 50
- imagenet + tpu demo

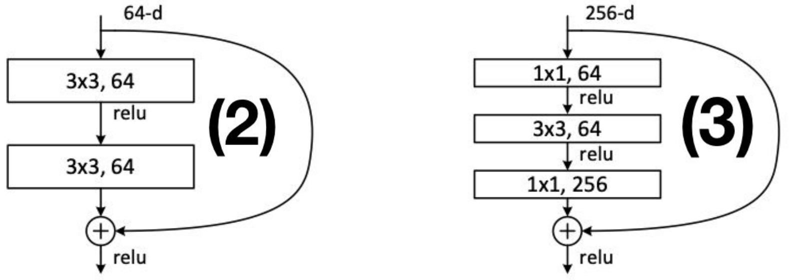
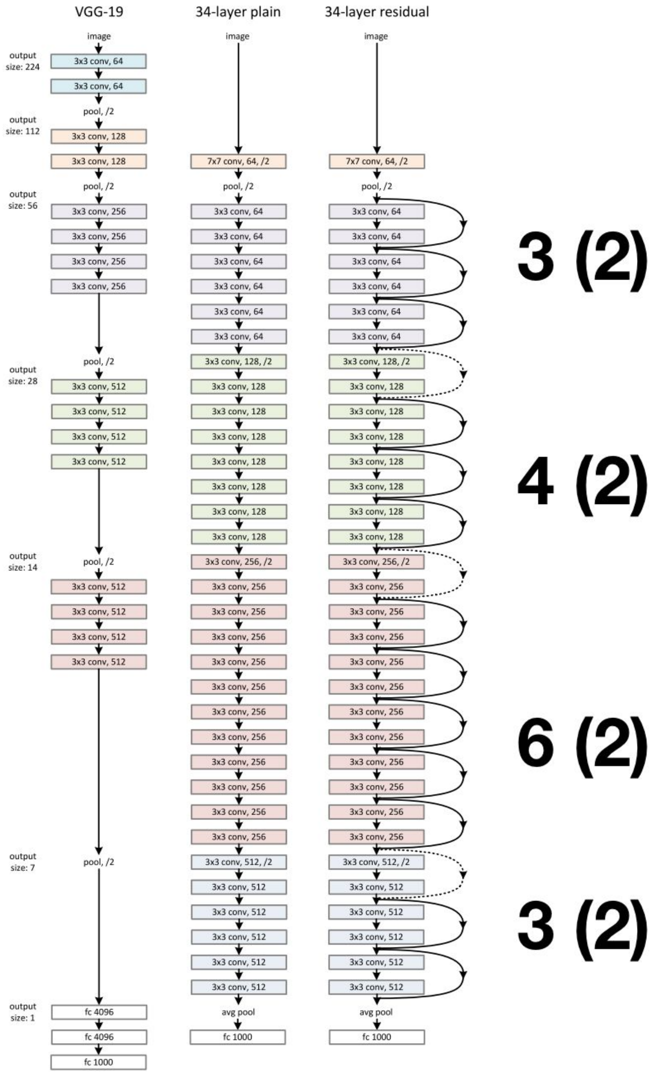


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.



efficientnet (may 2019)

- arxiv: 1905.11946

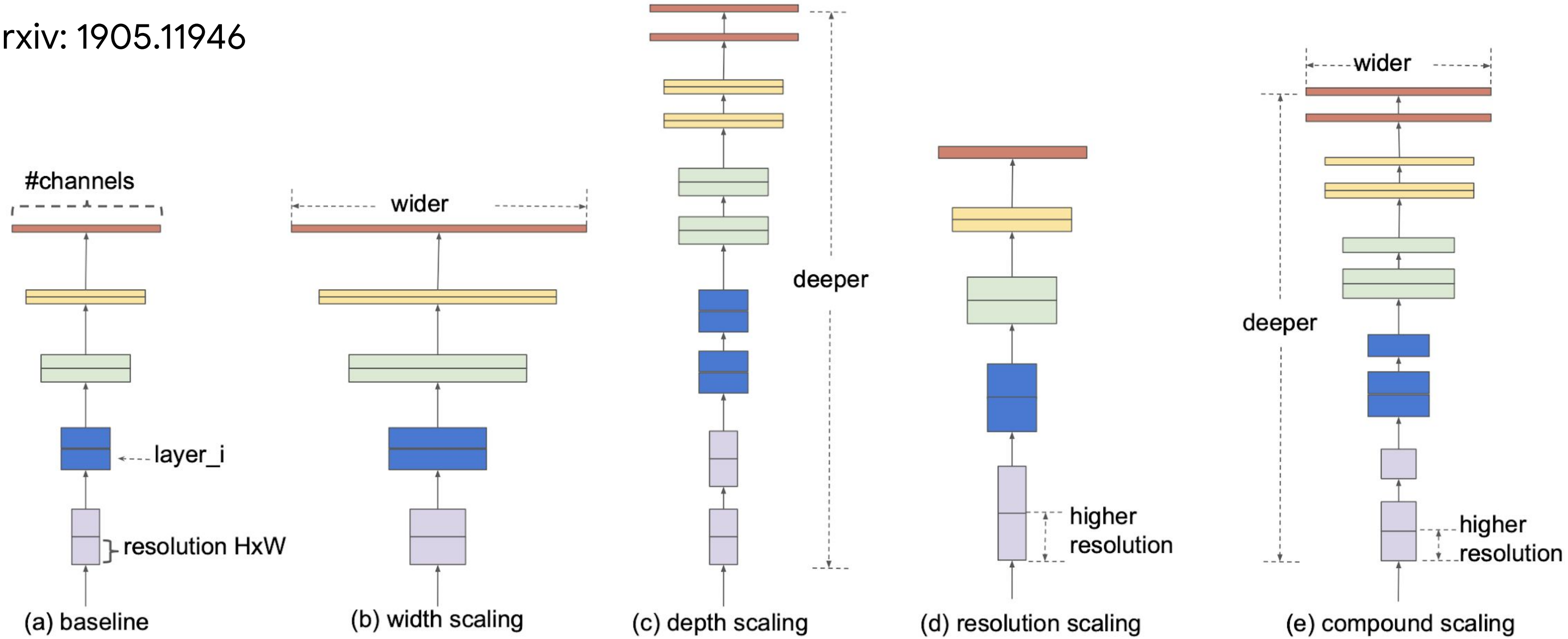
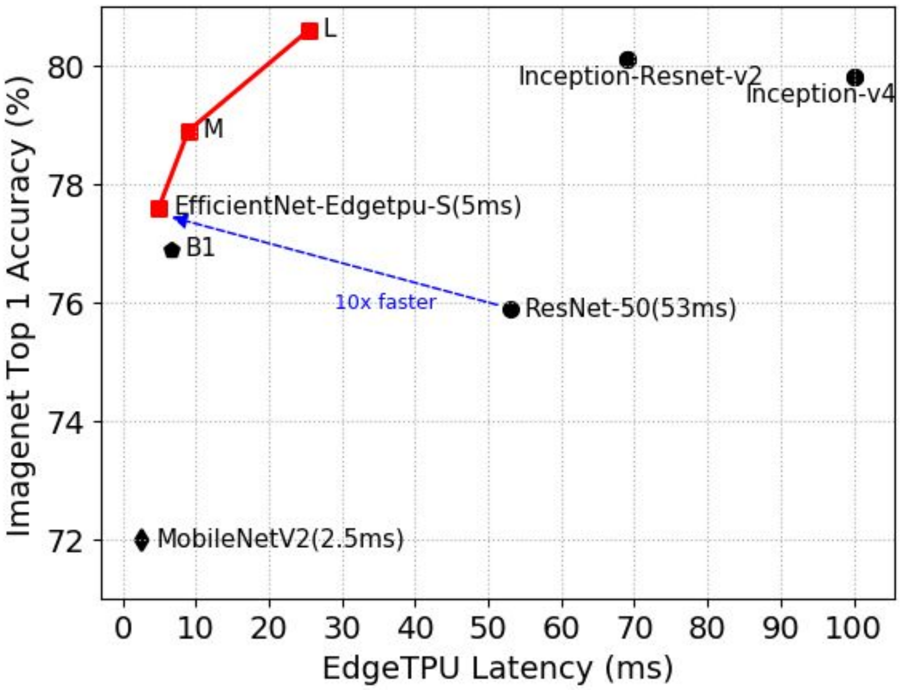
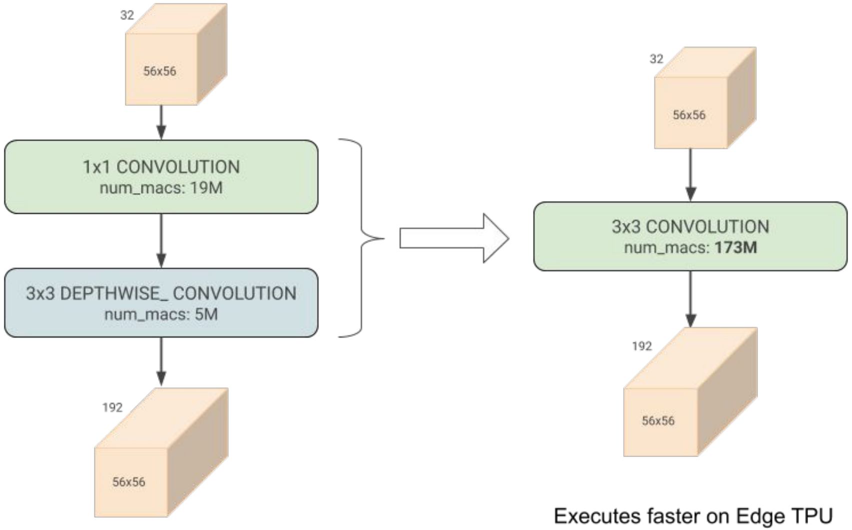


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

efficientnet-edgetpu (august 2019)





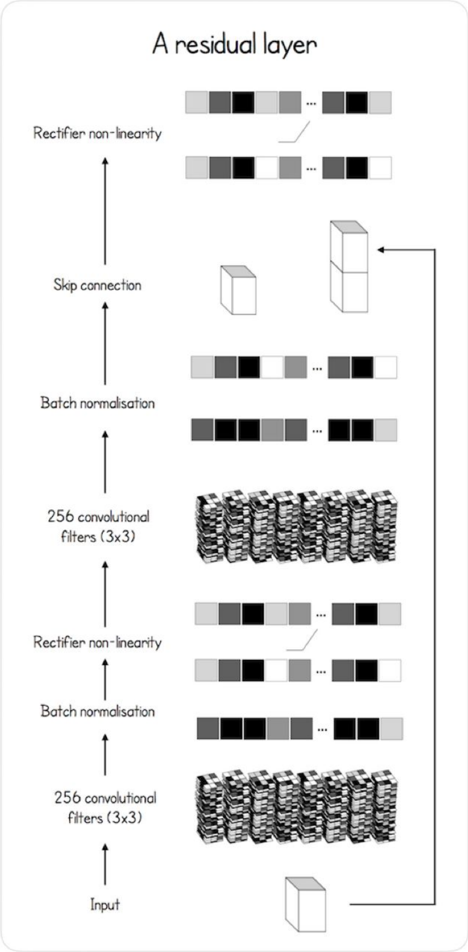
- tensorflow/tpu/blob/master/models/official/efficientnet/edgetpu

recap

- goal: convolutional neural networks for image recognition
- 1d neural network
- add convolutions, 2d neural network
- add more layers, larger problems
- add residual blocks, different block types
- state of the art approaches

other applications of cnn's

- 3D CNN's
- QANet
- AstroNet
- AlphaFold/Multicom
- AlphaGo/AlphaZero



Thanks for coming!

- brettkoonce.com/talks
- quarkworks.co

Thanks to:

- s4tf + fast.ai team
- Huan (李卓桓): swift-mnist, Brad Larson: cifar
- Gaiatri Dasu, Kübra Zengin
- Tensorflow Research Cloud
- Chris Fregly, Tanmay Bakshi

💪 + ❤️:

- seattle: James/Jin Maki
- gdg stl: Neem Serra
- gdg como: Hannah Pratte, Miranda Reese, Cassie Ferrick, Joy Park
- gde program: 🛏️ + ✈️