PLAYING AROUND

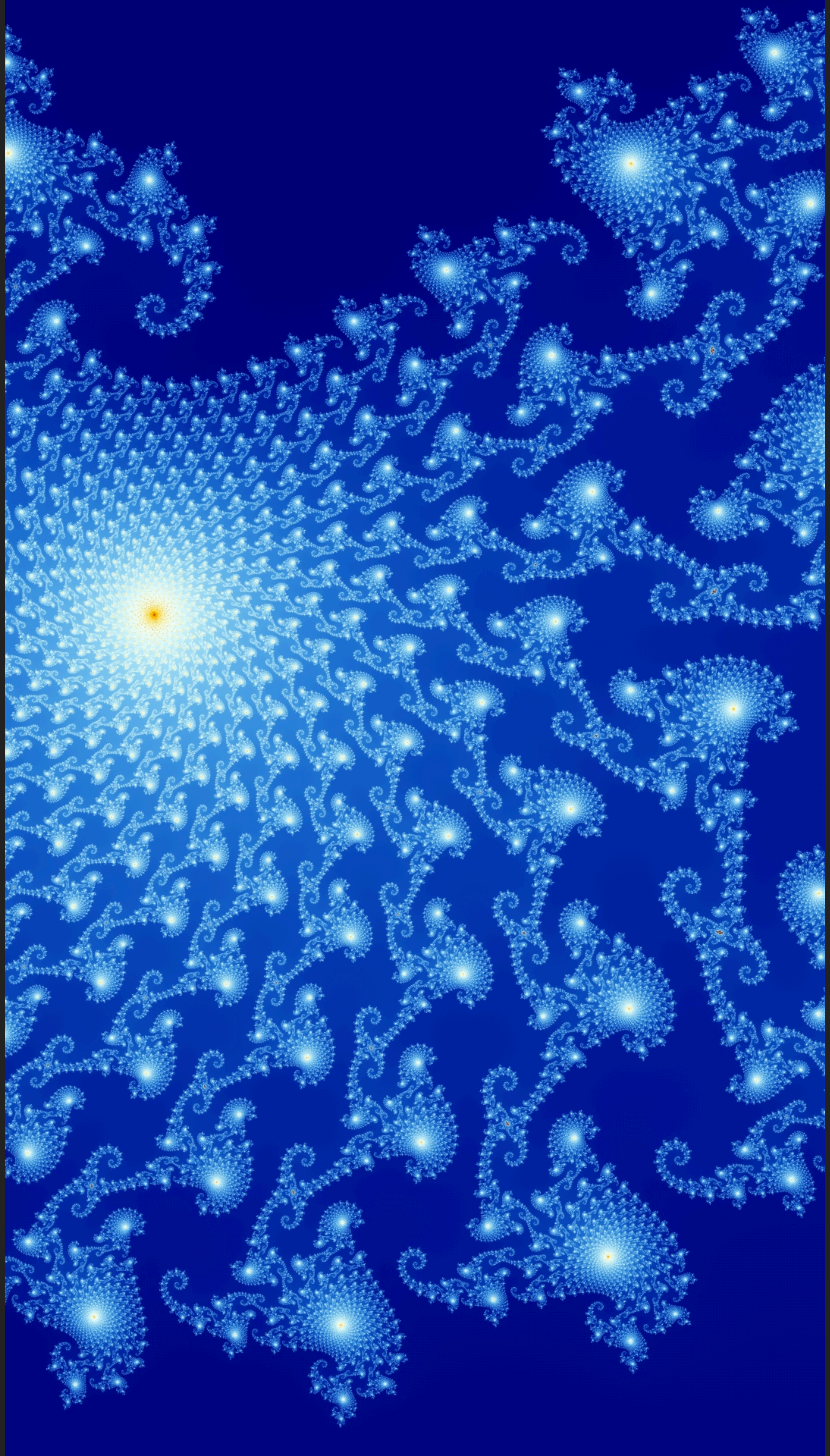
METAL & DEEP LEARNING

WHAT IS METAL? EXACTLY...

RuOyU

METAL IS

- Lowest-overhead access to the GPU
- Maximal of graphics and compute potential
- The better 'OpenGL' in iOS/MacOS
- And more…



DATA-PARALLEL COMPUTE PROCESSING

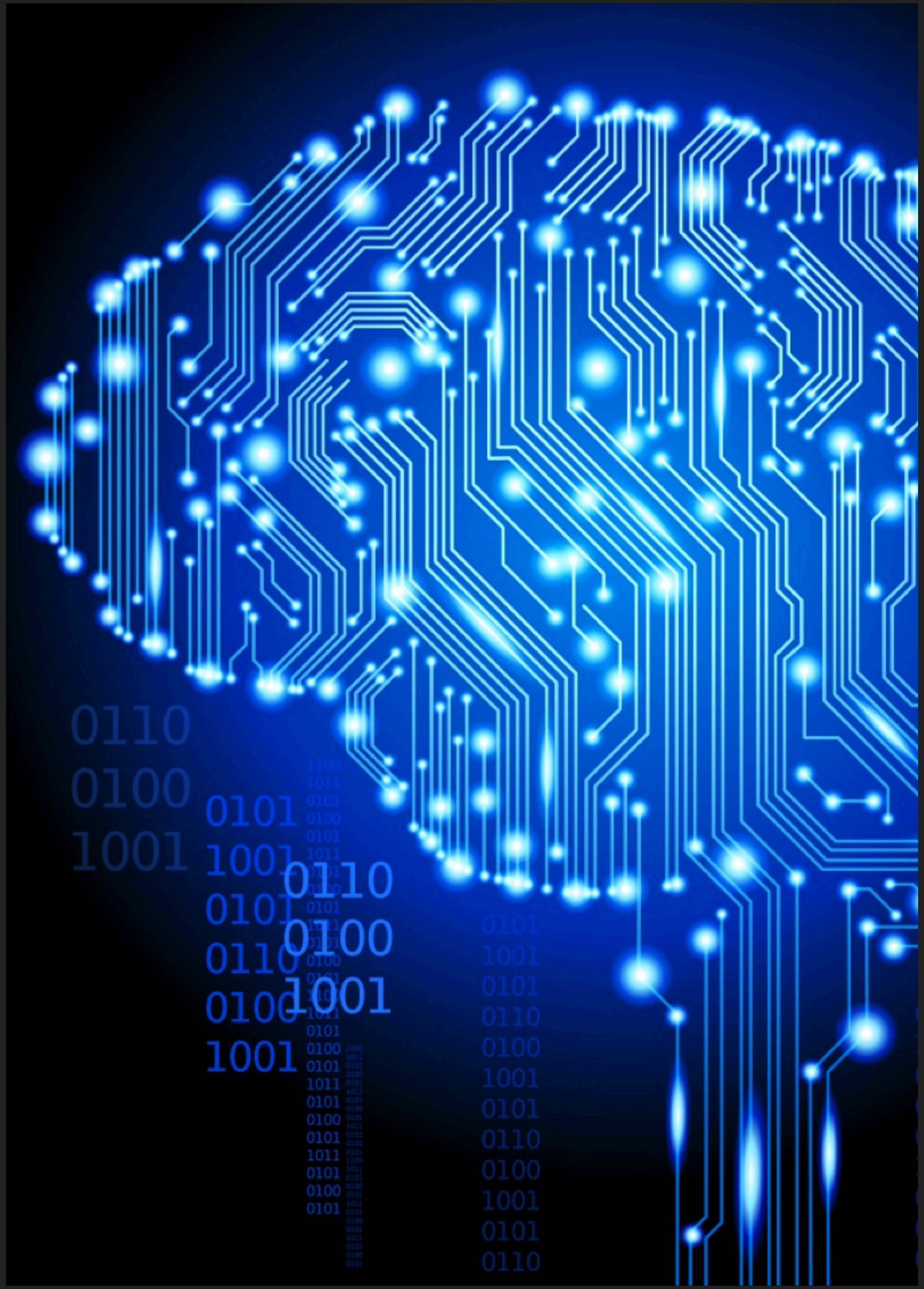
THE “MORE”

AND OF CAUSE DEEP LEARNING

RuOyU

DEEP LEARNING WITH METAL

- Fundamental Ideas About ML
- Metal Performance Shaders
- Metal Shading Language
- TensorFlow (或者其他类似工具)
- And a curious mind

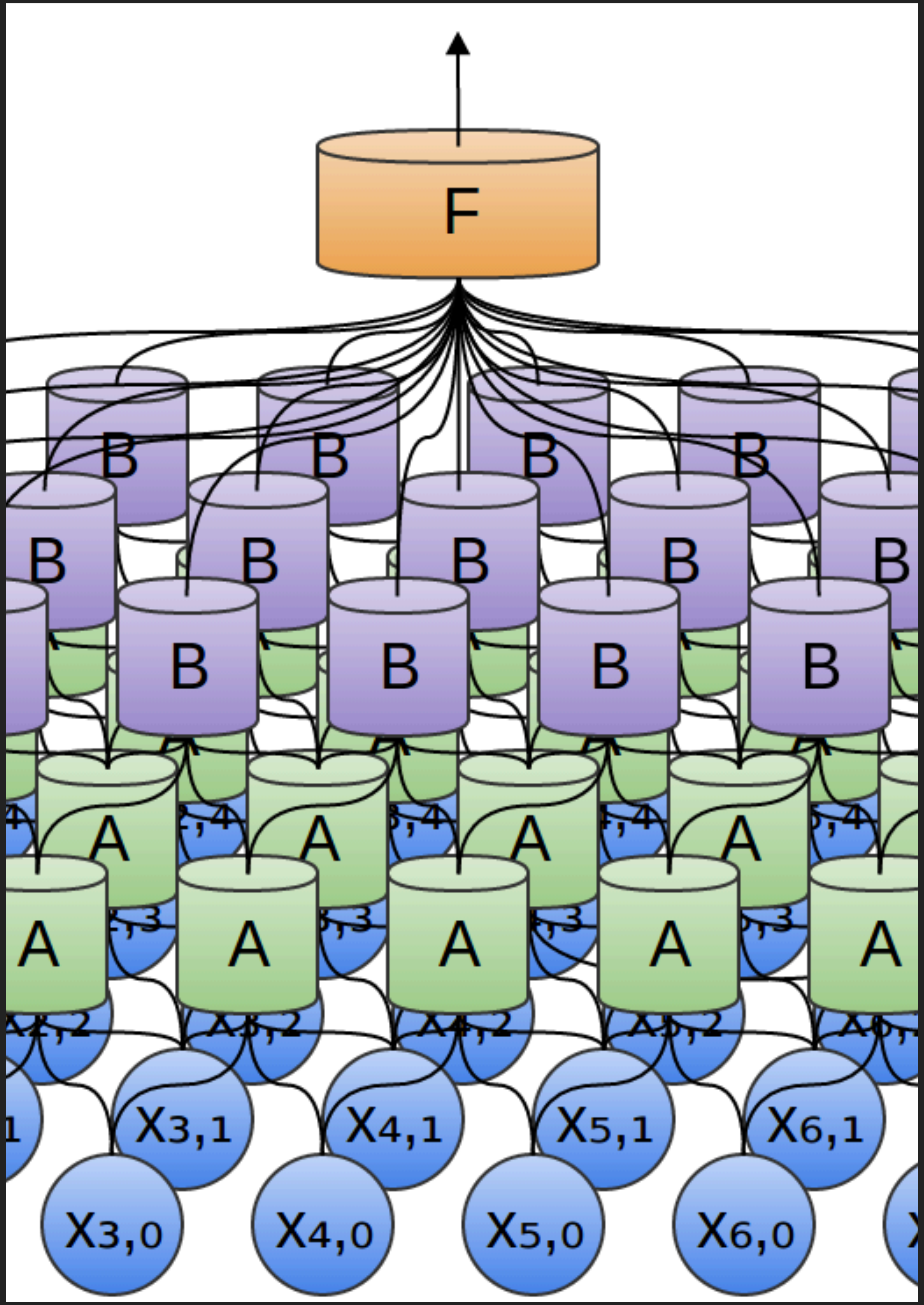


MACHINE LEARNING 101

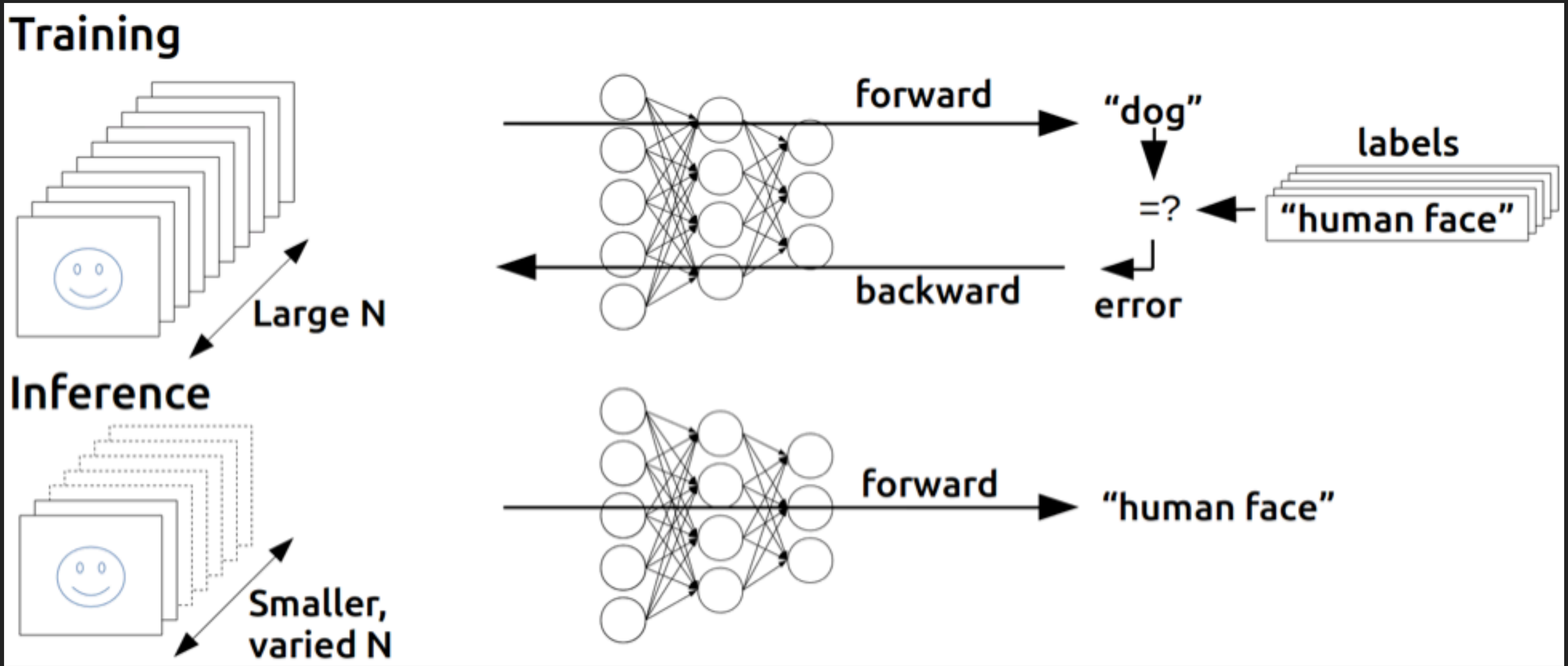
- Supervised Learning(监督学习)
  - Classification(分类)
    - 各种识别
  - Regression(回归)
    - 各种生成
- Unsupervised Learning (非监督学习)
  - Out of scope today…

ARCHITECTURES OF SUPERVISED ML

- Deep Neural Networks
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Deep Belief Networks
- Support Vector Machine



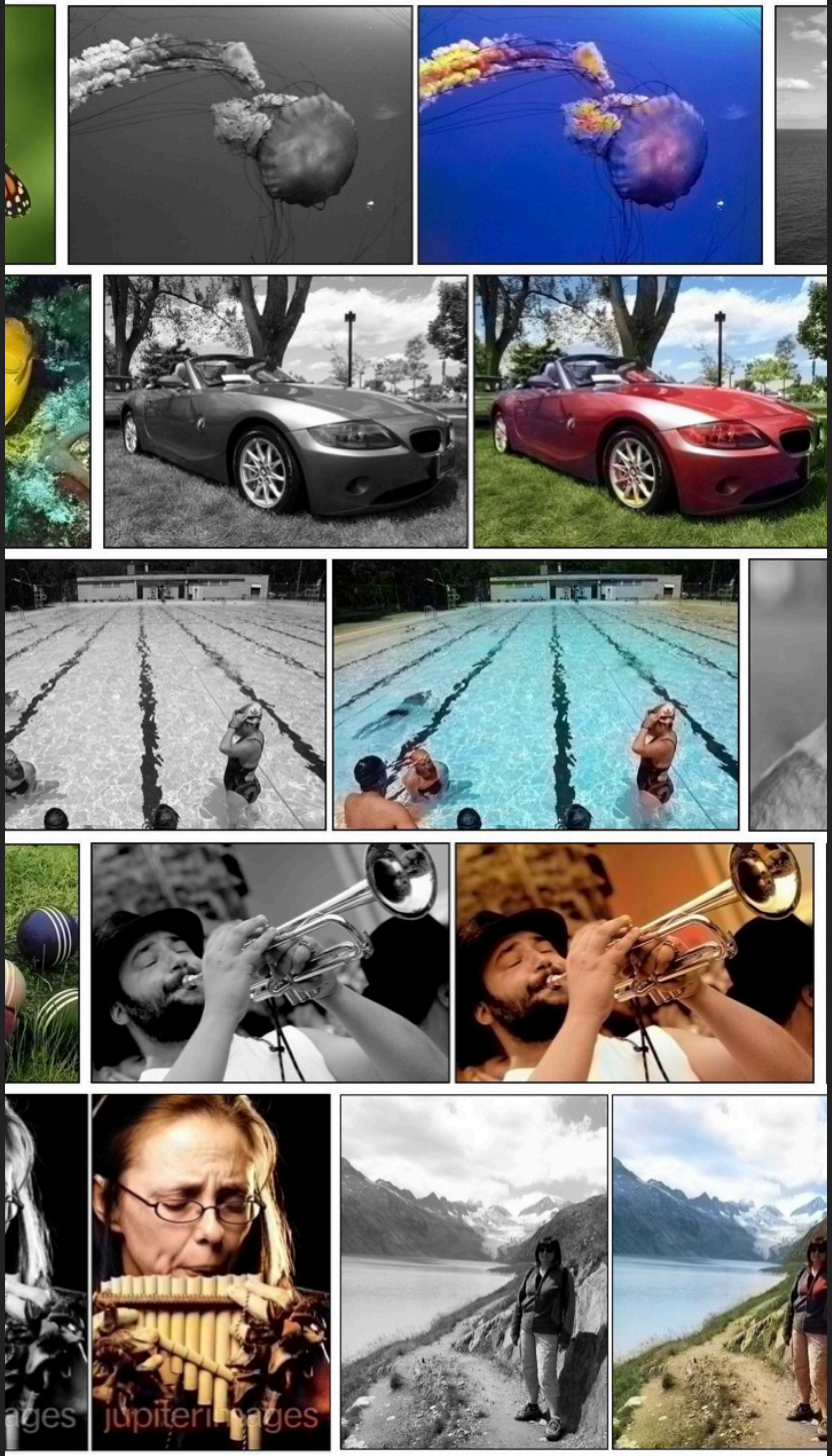
TRAINING & INFERENCE



- 目前，Training 阶段就不要想在iOS或者Mac上做了，暂时还不太靠谱
- 但是，Inference 这个事儿，挺适合在客户端去做的：
  - 1. 可以实时Inference
  - 2. 可以实时Inference
  - 3. 可以实时Inference

有一天，你女朋友想看彩色版的<罗马假日>

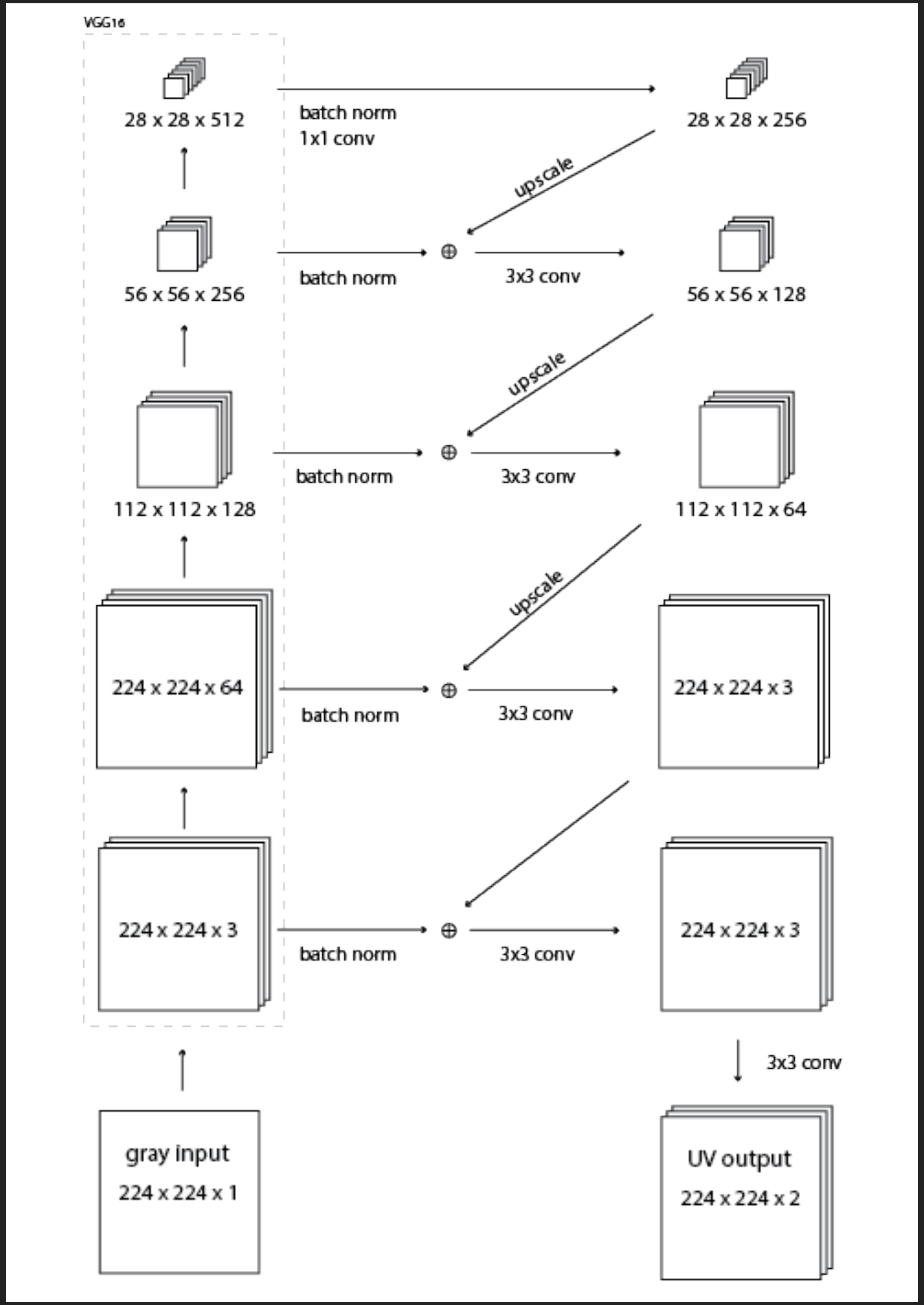
RuOyU



事实上！
这个真不难！

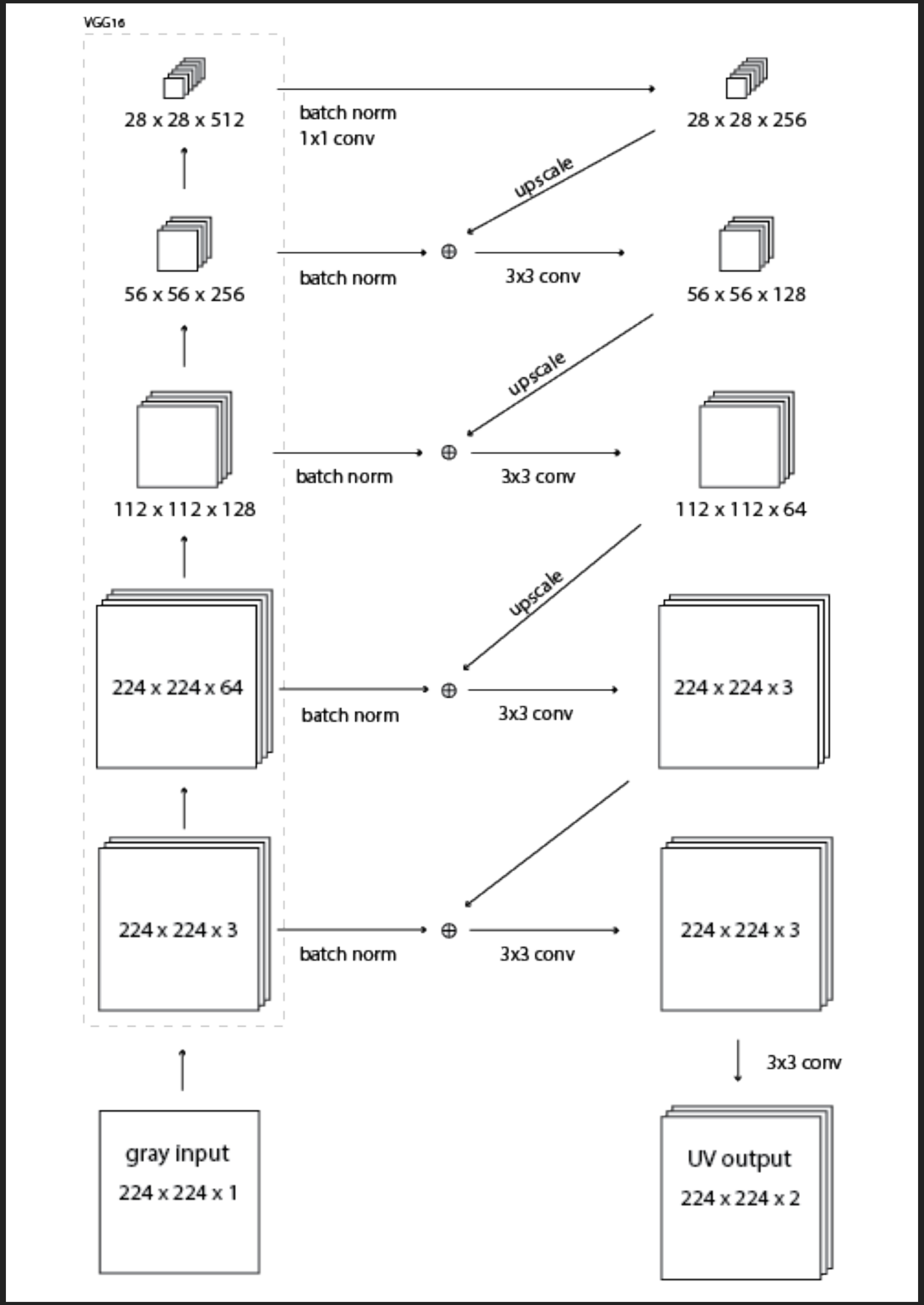
就三步搞定……

- 第一步：寻找合适的网络
  - 目前已经有很多现成的算法和网络结构了
  - 比如右图这个来自Ryan Dahl 的 Automatic Colorization
  - http://tinyclouds.org/colorize/
- 第二步：拿TensorFlow训练
  - 实在自己不行，还能找现成训练好的Model
- 第三步：用TensorFlow训练出来的参数，在iOS上面跑Inference！



拿这个网络举例

- 左边是一个VGG-NET 的一部分
  - VGG-NET本来是一个图片识别网络
  - 这里用来提取特征信息
  - 输入数据从左下往上，每上升一层，都能提供能多的形状和意义信息
- 把VGG输出的Tensor接入右边通过另一个网络计算色彩信息
  - 注意，所有用到的网络层，就是普通的卷积层
  - 这非常适合用Metal Performance Shader来快速搭建



```swift
poolMax.encode(commandBuffer: commandBuff

conv2_1.encode(commandBuffer: commandBuff
conv2_2.encode(commandBuffer: commandBuff
poolMax.encode(commandBuffer: commandBuff

conv3_1.encode(commandBuffer: commandBuff
conv3_2.encode(commandBuffer: commandBuff
conv3_3.encode(commandBuffer: commandBuff
poolMax.encode(commandBuffer: commandBuff

conv4_1.encode(commandBuffer: commandBuff
conv4_2.encode(commandBuffer: commandBuff
conv4_3.encode(commandBuffer: commandBuff

let cc1_0Image = MPSTemporaryImage(comman

let bn4_3Image = MPSTemporaryImage(comman
bn1.batch_normal(commandBuffer: commandBu
convc1.encode(commandBuffer: commandBuffe
c4_3Image.readCount -= 1

let cc1_1Image = MPSTemporaryImage(comman
let bn3_3Image = MPSTemporaryImage(comman
bn2.batch_normal(commandBuffer: commandBu

c3_3Image.readCount -= 1
scaleAdd(commandBuffer: commandBuffer, so
cc1_0Image.readCount -= 1
bn3_3Image.readCount -= 1

let cc2_0Image = MPSTemporaryImage(comman
convc2.encode(commandBuffer: commandBuffe
let cc2_1Image = MPSTemporaryImage(comman
```

```swift
conv2_1       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv2_1")
conv2_2       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv2_2")
conv3_1       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv3_1")
conv3_2       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv3_2")
conv3_3       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv3_3")
conv4_1       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv4_1")
conv4_2       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv4_2")
conv4_3       = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3, inputFeatureC
                      kernelParamsBinaryName: "vgg16_conv4_3")

convc1        = SlimMPSCNNConvolution(kernelWidth: 1, kernelHeight: 1,
                      inputFeatureChannels: 512, outputFeatureChanne
                      kernelParamsBinaryName: "color4", withBias: tr
convc2        = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3,
                      inputFeatureChannels: 256, outputFeatureChanne
                      kernelParamsBinaryName: "color3", withBias: tr
convc3        = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3,
                      inputFeatureChannels: 128, outputFeatureChanne
                      kernelParamsBinaryName: "color2", withBias: tr
convc4        = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3,
                      inputFeatureChannels: 64, outputFeatureChannel
                      kernelParamsBinaryName: "color1", withBias: tr
convc5        = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3,
                      inputFeatureChannels: 3, outputFeatureChannels
                      kernelParamsBinaryName: "color0", withBias: tr
convc6        = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3,
                      inputFeatureChannels: 3, outputFeatureChannels
                      kernelParamsBinaryName: "uv", withBias: true)
```

```swift
func metalInit() {
    device = MTLCreateSystemDefaultDevice()
    guard MPSSupportsMTLDevice(device) else { return }
    commandQueue = device!.makeCommandQueue()
    textureLoader = MTKTextureLoader(device: device!)
    net = Net(withCommandQueue: commandQueue)
    CVMetalTextureCacheCreate(kCFAllocatorDefault, nil,
}


func videoInit() {
    output.setSampleBufferDelegate(self, queue: Dispatc
    output.videoSettings = [kCVPixelBufferPixelFormatTy
    guard let videoDevice = AVCaptureDevice.defaultDevi
    try! session.addInput(AVCaptureDeviceInput(device:
    session.addOutput(output)
    output.connection(withMediaType: AVMediaTypeVideo).
    session.sessionPreset = AVCaptureSessionPresetPhoto
    session.startRunning()
}
var sampleBuffer: CMSampleBuffer! = nil
```

构造你的网络

- 首先，定义出网络中的每一个神经元：

```
conv1_1        = SlimMPSCNNConvolution(kernelWidth: 3, kernelHeight: 3,
                                       inputFeatureChannels: 3, outputFeatureChannels: 64,
                                       neuronFilter: relu, device: device,
                                       kernelParamsBinaryName: "vgg16_conv1_1")
```

- 接着，定义出网络中数据的流动过程，
  即各个神经元的输入输出方式：

```
conv1_1.encode(commandBuffer: commandBuffer, sourceImage: srcImage, destinationImage: c1_1Image)
conv1_2.encode(commandBuffer: commandBuffer, sourceImage: c1_1Image, destinationImage: c1_2Image)
poolMax.encode(commandBuffer: commandBuffer, sourceImage: c1_2Image, destinationImage: c1Image)
```

构造你的网络

- 遇到Metal原生没有的神经元，你需要去写一些Shader

```
kernel void batch_normal(
                         texture2d_array<half, access::read> inTexture [[texture(0)]],
                         texture2d_array<half, access::write> outTexture [[texture(1)]],
                         constant float*                    beta      [[ buffer(0) ]],
                         constant float*                    gamma      [[ buffer(1) ]],
                         constant float*                    mean      [[ buffer(2) ]],
                         constant float*                    varian      [[ buffer(3) ]],
                         uint3 gid [[thread_position_in_grid]])
{
    if (gid.x >= outTexture.get_width() ||
        gid.y >= outTexture.get_height()) {
        return;
    }
    half4 inColor = inTexture.read(gid.xy, gid.z);
    half r = gamma[gid.z*4+0] * (inColor.r - mean[gid.z*4+0]) / sqrt(varian[gid.z*4+0] + 0.001) + beta[gid.z*4+0];
    half g = gamma[gid.z*4+1] * (inColor.r - mean[gid.z*4+1]) / sqrt(varian[gid.z*4+1] + 0.001) + beta[gid.z*4+1];
    half b = gamma[gid.z*4+2] * (inColor.r - mean[gid.z*4+2]) / sqrt(varian[gid.z*4+2] + 0.001) + beta[gid.z*4+2];
    half a = gamma[gid.z*4+3] * (inColor.r - mean[gid.z*4+3]) / sqrt(varian[gid.z*4+3] + 0.001) + beta[gid.z*4+3];
    outTexture.write(half4(r,g,b,a), gid.xy,gid.z);

}
```

加载和处理每一个神经元预训练好的参数

- 推荐用TensorFlow来进行参数训练……

```
init(device: MTLDevice, length: Int, name: String) {

    let sizeL:Int = length * Int(MemoryLayout<Float>.size)
    self.length = length
    self.device = device
    let bPath = Bundle.main.path( forResource: name + "_bn_beta", ofType: "dat")
    let fd_b = open( bPath!, O_RDONLY, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)
    let hdrB = mmap(nil, Int(sizeL), PROT_READ, MAP_FILE | MAP_SHARED, fd_b, 0)
    let beta = UnsafePointer(hdrB!.bindMemory(to: Float.self, capacity: Int(sizeL)))
    bBuffer = device.makeBuffer(bytes: beta, length: sizeL, options: [])
```

处理数据的输入与输出

- 遇到Metal原生没有的神经元，你需要去写一些Compute Shader

```
kernel void batch_normal(
                         texture2d_array<half, access::read> inTexture [[texture(0)]],
                         texture2d_array<half, access::write> outTexture [[texture(1)]],
                         constant float*                     beta       [[ buffer(0) ]],
                         constant float*                     gamma       [[ buffer(1) ]],
                         constant float*                     mean       [[ buffer(2) ]],
                         constant float*                     varian       [[ buffer(3) ]],
                         uint3 gid [[thread_position_in_grid]])
{
    if (gid.x >= outTexture.get_width() ||
        gid.y >= outTexture.get_height()) {
        return;
    }
    half4 inColor = inTexture.read(gid.xy, gid.z);
    half r = gamma[gid.z*4+0] * (inColor.r - mean[gid.z*4+0]) / sqrt(varian[gid.z*4+0] + 0.001) + beta[gid.z*4+0];
    half g = gamma[gid.z*4+1] * (inColor.r - mean[gid.z*4+1]) / sqrt(varian[gid.z*4+1] + 0.001) + beta[gid.z*4+1];
    half b = gamma[gid.z*4+2] * (inColor.r - mean[gid.z*4+2]) / sqrt(varian[gid.z*4+2] + 0.001) + beta[gid.z*4+2];
    half a = gamma[gid.z*4+3] * (inColor.r - mean[gid.z*4+3]) / sqrt(varian[gid.z*4+3] + 0.001) + beta[gid.z*4+3];
    outTexture.write(half4(r,g,b,a), gid.xy,gid.z);

}
```

处理数据的输入与输出

- 通过MPSImage, MTLTexture, MTLBuffer来进行数据交换

```
func batch_normal(commandBuffer: MTLCommandBuffer, source: MTLTexture, target: MTLTexture) {
    let commandEncoder = commandBuffer.makeComputeCommandEncoder()
    commandEncoder.setTexture(source, at: 0)
    commandEncoder.setTexture(target, at: 1)
    commandEncoder.setBuffer(bBuffer, offset: 0, at: 0)
    commandEncoder.setBuffer(gBuffer, offset: 0, at: 1)
    commandEncoder.setBuffer(mBuffer, offset: 0, at: 2)
    commandEncoder.setBuffer(vBuffer, offset: 0, at: 3)
    commandEncoder.dispatch(pipeline: bn, width: target.width, height: target.height, featureChannels: length)
    commandEncoder.endEncoding()
}
```

大体可能遇到的坑……

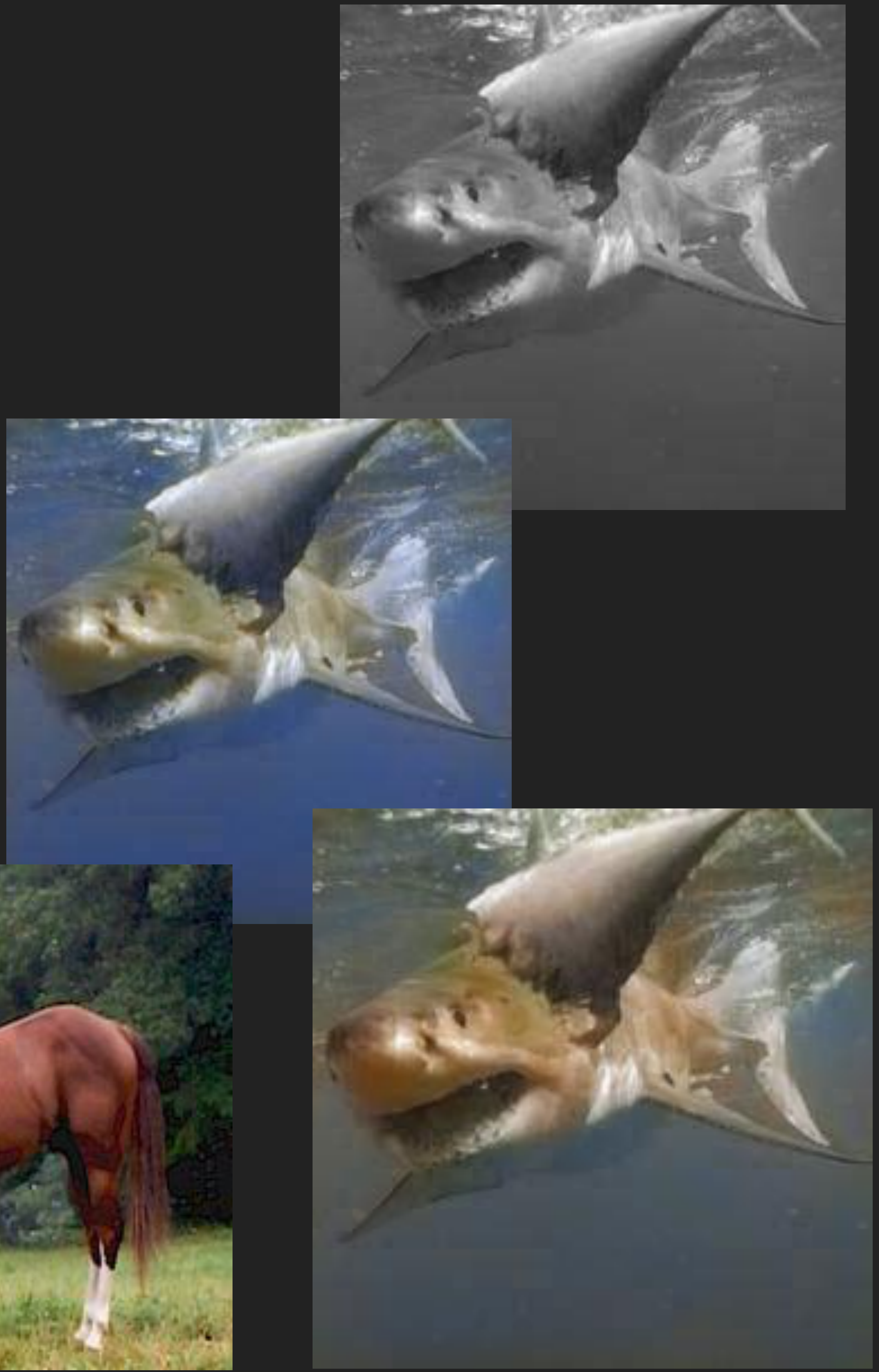
- 1. 你需要一个MTKView, 拿到它'currentRenderable'的Texture
  - 坑：注意它有一个属性叫做'frameBufferOnly'，必须手动设置为false才能将计算结果render出来
- 2. 你需要用Metal Performance Shader提供的工具来搭建你网络中的卷积层，尽量不要用太复杂的网络
  - 坑：经常一般卷积层中间会有concat的操作，注意MPS不直接提供这个方法，你必须通过设置targetFeatureChannelOffset才能做到
  - MPS中不能做Batch Normalization，官方建议的搞法是直接在TensorFlow中加到对应卷积层的Weight和Bias上面去……当然你也可以自己拿shader来写个BN，但毕竟那样消耗额外的性能
  - 小心MPSImage在Feature超过4个的时候Texture的格式会变……这是个神坑……

大体步骤和可能遇到的坑

- 3. 你需要写一部分shader来去实现MPS中不支持的各种操作
  - 坑：事实上……绝大部分操作MPS都不支持，你需要写非常，非常，非常多shader……
  - 坑：如前述，MPSImage的Channel多过4个的时候，你必须用texture array来接收数据，也就是说，绝大部分情况下，你的shader都要写两份
- 4. 你需要在运行时导入TensorFlow训练出来的各种参数
  - 坑：MPS CNN的参数张量和TensorFlow的维度不一样，你要在TensorFlow中做个转置
  - 坑：注意，参数的默认数值类型是Float——这是32位浮点数，同时我们在shader中往往使用的是"half"，即16位浮点数。小心有时该做数值转换
  - 再次提一下，不要用太复杂的网络——小心遇到几百上千M的模型把手机给烧了

效果如何呢?

- 图片分辨率有限——224X224
- 不见得啥都能着色
- 但是可以实时输出视频！！！
  - 只在iPhone7及以上……

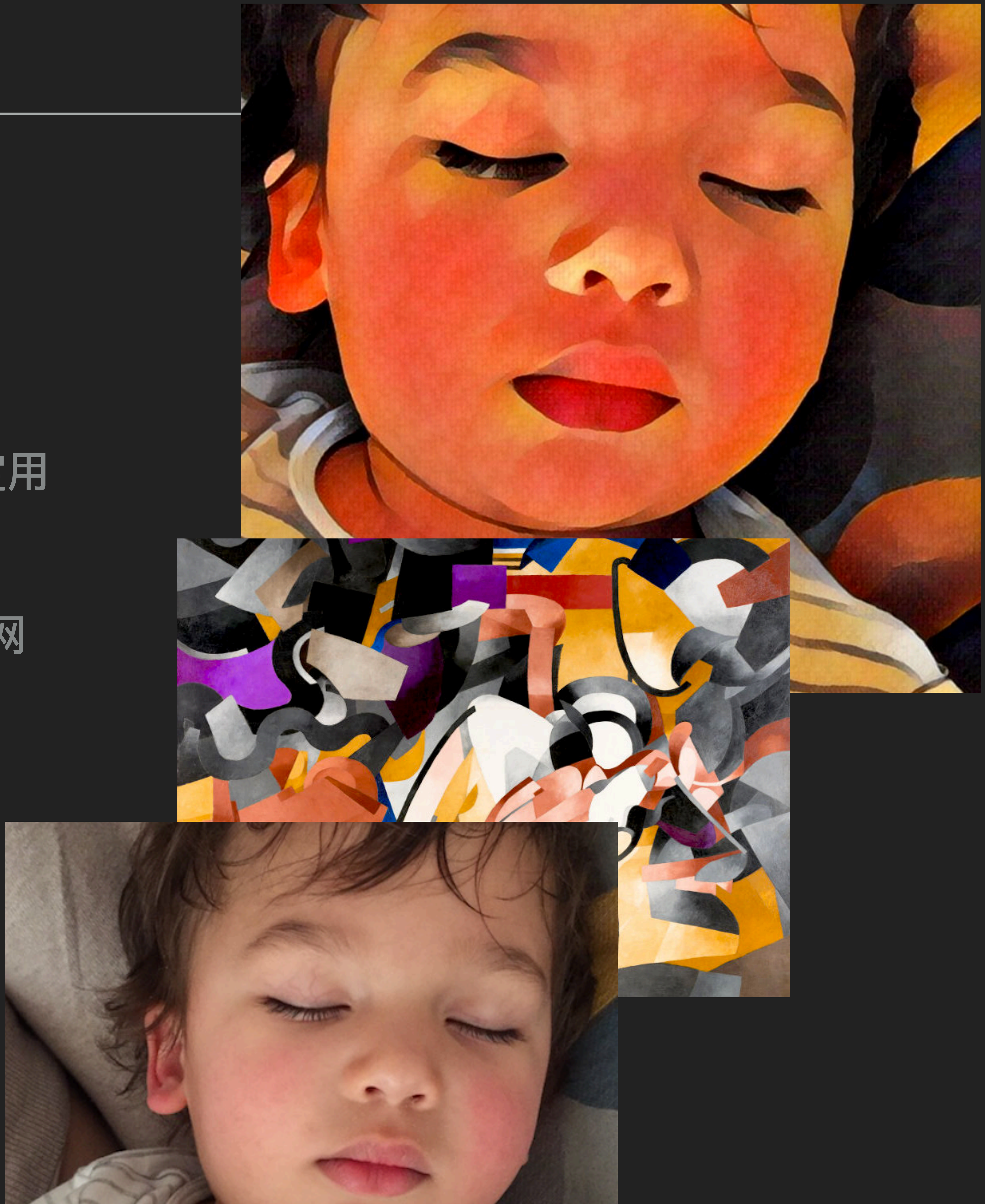


当你这些都搞定以后，想象一下

- 试试把摄像头的输入直接Map成Texture，输入到网络，得到彩色化以后的图片后再实时显示在MTKView中
    - 坑：VideoOutput和MTKView都是用Delegate来控制的，但注意VideoOutput设置Delegate的时候可以指明你想要的dispatch queue，即方便扔后台
- 然后……你发现你需要一部最新iPhone7……
    - 坑：更老的设备很难实现实时的inference，甚至很多Metal的API都没有
    - 坑：即便iPhone7，有时也都还会有点儿卡……
    - 炫酷：鬼~~~~~~你终于可以拿手机屏幕对着电视屏幕看彩色版地雷战和地道战了！！！

我们其实还可以做更多

- 利用摄像头实时的风格化
- 利用摄像头实时物体识别，说不定用来找钥匙
- 嗯，前一阵子Google出了个神经网络，可以用来去除马赛克……所以……



谢谢