|  |
| --- |
| function prediction = VGG(varargin) |
|  | %VGG Returns a VGG-16/19 (VGG-VeryDeep) model for ImageNet |
|  | % M = models.VGG() returns the model VGG-16 proposed in: |
|  | % models.VGG(..., 'option', value, ...) accepts the following options: |
|  | % |
|  | % `variant`:: '16' |
|  | % Model variant: 16 or 19. |
|  | % |
|  | % `pretrained`:: false |
|  | % If true, returns a model pre-trained on ImageNet (using the |
|  | % MatConvNet example code). |
|  | % |
|  | % `input`:: default input |
|  | % Specifies an input (images) layer for the network. If unspecified, a |
|  | % new one is created. |
|  | % |
|  | % `numClasses`:: 5 |
|  | % Number of output classes. |
|  | % |
|  | % `batchNorm`:: true |
|  | % Whether to use batch normalization. |
|  | % |
|  | % `normalization`:: [5 1 0.0001/5 0.75] |
|  | % Parameters for vl\_nnnormalize layer (only used without batch-norm). |
|  | % |
|  | % Any other options will be passed to models.ConvBlock(), and can be used |
|  | % to change the activation function, weight initialization, etc. |
|  | % |
|  | % Suggested SGD training options are also returned in the struct M.meta. |
|  | % parse options. unknown arguments will be passed to ConvBlock (e.g. |
|  | % activation). |
|  | opts.pretrained = false ; % whether to fetch a pre-trained model |
|  | opts.input = Input('name', 'images', 'gpu', true) ; % default input layer |
|  | opts.numClasses = 1000 ; % number of predicted classes |
|  | opts.variant = '16' ; % choose between variants 16/19 (number of layers) |
|  | opts.batchNorm = true ; % whether to use batch normalization |
|  | opts.preActivationBatchNorm = true ; % whether batch-norm comes before or after activations |
|  | opts.normalization = [5 1 0.0001/5 0.75] ; % for LRN layer (vl\_nnnormalize) |
|  | [opts, convBlockArgs] = vl\_argparse(opts, varargin, 'nonrecursive') ; |
|  |  |
|  |  |
|  | % default training options for this network (returned as output.meta) |
|  | if isnumeric(opts.variant) % accept variant 16 instead of '16' |
|  | opts.variant = int2str(opts.variant) ; |
|  | end |
|  | switch lower(opts.variant) |
|  | case '16' |
|  | meta.batchSize = 32 ; |
|  | case '19' |
|  | meta.batchSize = 24 ; |
|  | otherwise |
|  | error('Unknown variant.') ; |
|  | end |
|  | meta.imageSize = [224, 224, 3] ; |
|  | meta.augmentation.crop = 224 / 256; |
|  | meta.augmentation.location = true ; |
|  | meta.augmentation.flip = true ; |
|  | meta.augmentation.brightness = 0.1 ; |
|  | meta.augmentation.aspect = [2/3, 3/2] ; |
|  | meta.weightDecay = 0.0005 ; |
|  |  |
|  | % the default learning rate schedule |
|  | if ~opts.pretrained |
|  | if ~opts.batchNorm |
|  | meta.learningRate = logspace(-2, -4, 60) ; |
|  | else |
|  | meta.learningRate = logspace(-1, -4, 20) ; |
|  | end |
|  | meta.numEpochs = numel(meta.learningRate) ; |
|  | else % fine-tuning has lower LR |
|  | meta.learningRate = 1e-5 ; |
|  | meta.numEpochs = 20 ; |
|  | end |
|  |  |
|  |  |
|  | % return a pre-trained model |
|  | if opts.pretrained |
|  | if opts.batchNorm |
|  | warning('The pre-trained model does not include batch-norm (set batchNorm to false).') ; |
|  | end |
|  | if opts.numClasses ~= 1000 |
|  | warning('Model options are ignored when loading a pre-trained model.') ; |
|  | end |
|  | prediction = models.pretrained(['imagenet-vgg-verydeep-' opts.variant]) ; |
|  |  |
|  | % return prediction layer (not softmax) |
|  | assert(isequal(prediction{1}.func, @vl\_nnsoftmax)) ; |
|  | prediction = prediction{1}.inputs{1} ; |
|  |  |
|  | % replace input layer with the given one |
|  | input = prediction.find('Input', 1) ; |
|  | prediction.replace(input, opts.input) ; |
|  |  |
|  | prediction.meta = meta ; |
|  | return |
|  | end |
|  |  |
|  |  |
|  | % get conv block generator with the given options. default activation is |
|  | % ReLU, with pre-activation batch normalization (can be overriden). |
|  | conv = models.ConvBlock('batchNorm', opts.batchNorm, ... |
|  | 'preActivationBatchNorm', opts.preActivationBatchNorm, convBlockArgs{:}) ; |
|  |  |
|  | % build network |
|  | images = opts.input ; |
|  |  |
|  | x = conv(images, 'size', [3, 3, 3, 64], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 64, 64], 'pad', 1) ; |
|  | x = vl\_nnpool(x, 2, 'stride', 2) ; |
|  |  |
|  | x = conv(x, 'size', [3, 3, 64, 128], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 128, 128], 'pad', 1) ; |
|  | x = vl\_nnpool(x, 2, 'stride', 2) ; |
|  |  |
|  | x = conv(x, 'size', [3, 3, 128, 256], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 256, 256], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 256, 256], 'pad', 1) ; |
|  | if strcmp(opts.variant, '19') |
|  | x = conv(x, 'size', [3, 3, 256, 256], 'pad', 1) ; |
|  | end |
|  | x = vl\_nnpool(x, 2, 'stride', 2) ; |
|  |  |
|  | x = conv(x, 'size', [3, 3, 256, 512], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 512, 512], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 512, 512], 'pad', 1) ; |
|  | if strcmp(opts.variant, '19') |
|  | x = conv(x, 'size', [3, 3, 512, 512], 'pad', 1) ; |
|  | end |
|  | x = vl\_nnpool(x, 2, 'stride', 2) ; |
|  |  |
|  | x = conv(x, 'size', [3, 3, 512, 512], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 512, 512], 'pad', 1) ; |
|  | x = conv(x, 'size', [3, 3, 512, 512], 'pad', 1) ; |
|  | if strcmp(opts.variant, '19') |
|  | x = conv(x, 'size', [3, 3, 512, 512], 'pad', 1) ; |
|  | end |
|  | x = vl\_nnpool(x, 2, 'stride', 2) ; |
|  |  |
|  | x = conv(x, 'size', [7, 7, 512, 4096]) ; |
|  | if ~opts.batchNorm |
|  | x = vl\_nndropout(x) ; |
|  | end |
|  |  |
|  | x = conv(x, 'size', [1, 1, 4096, 4096]) ; |
|  | if ~opts.batchNorm |
|  | x = vl\_nndropout(x) ; |
|  | end |
|  |  |
|  | prediction = conv(x, 'size', [1, 1, 4096, opts.numClasses], ... |
|  | 'batchNorm', false, 'activation', 'none') ; |
|  |  |
|  | prediction.meta = meta ; |
|  |  |
|  | end |