dlib\_dnn库研究

目 录

[1 dlib库人脸特征提取算法 3](#_Toc525042000)

[1.1 ResNet简介 3](#_Toc525042001)

[1.2 网络结构 4](#_Toc525042002)

[参考文献 1](#_Toc525042003)

# core.h

namespace:

impl::

struct layer\_helper; // 多个

struct layer\_helper\_match; // 多个

dimpl::

get\_input\_details(); // 多个

函数：

tuple\_tail()； re

tuple\_head()； re

double get\_learning\_rate\_multiplier(const T& obj);

double get\_weight\_decay\_multiplier(const T& obj);

impl::call\_layer\_forward

template <unsigned int i, typename T>

typename impl::layer\_helper<i,T>::type& layer (T& n); //

类：

sstack：re，堆栈相关

template <typename LAYER\_DETAILS, typename SUBNET>

class add\_layer: 在SUBNET上增加LAYER\_DETAILS网络

private:

tensor& private\_get\_output() const;

tensor& private\_get\_gradient\_input();

void disable\_output\_and\_gradient\_getters ();

bool this\_layer\_operates\_inplace();

bool this\_layer\_requires\_forward\_output();

void swap(add\_layer& item);

LAYER\_DETAILS details;

std::unique\_ptr<subnet\_type> subnetwork;

bool this\_layer\_setup\_called;

bool gradient\_input\_is\_stale;

bool get\_output\_and\_gradient\_input\_disabled;

resizable\_tensor x\_grad;

resizable\_tensor cached\_output;

resizable\_tensor params\_grad;

resizable\_tensor temp\_tensor;

public:

template <typename forward\_iterator>

void to\_tensor (forward\_iterator ibegin, forward\_iterator iend,

resizable\_tensor& data) const;

将[ibegin, iend)的迭代范围转化为tensor，并存进data里，调用输入网络里的to\_tensor()函数。

template <typename forward\_iterator>

const tensor& operator() (forward\_iterator ibegin, forward\_iterator iend)

重载()运算符，运行前向网络的操作

const tensor& get\_output() const;

tensor& get\_gradient\_input();

const tensor& get\_final\_data\_gradient();

void back\_propagate\_error(const tensor& x);

void update\_parameters(sstack<solver\_type> solvers, double learning\_rate);

const tensor& get\_parameter\_gradient();

tensor& get\_parameter\_gradient();

const subnet\_type& subnet()const ;

subnet\_type& subnet();

const layer\_details\_type& layer\_details() const;

layer\_details\_type& layer\_details();

unsigned int sample\_expansion\_factor() const;

void clean();

friend void serialize(const add\_layer& item, std::ostream& out);

friend void deserialize(add\_layer& item, std::istream& in);

friend std::ostream& operator<< (std::ostream& out, const add\_layer& item);

void print (std::ostream& out, unsigned long idx, int& min\_length) const;

class add\_layer; for input layer object only

template <unsigned long ID, typename SUBNET>

class add\_tag\_layer

template <size\_t num, template<typename> class REPEATED\_LAYER, typename SUBNET>

class repeat

template <typename LOSS\_DETAILS, typename SUBNET>

class add\_loss\_layer

template <template<typename> class TAG\_TYPE, typename SUBNET>

class add\_skip\_layer

# layers.h

类：

template <long \_num\_filters, long \_nr, long \_nc, int \_stride\_y, int \_stride\_x,

int \_padding\_y = \_stride\_y!=1? 0 : \_nr/2,

int \_padding\_x = \_stride\_x!=1? 0 : \_nc/2>

class con\_

{

private:

resizable\_tensor params;

alias\_tensor filters, biases;

tt::tensor\_conv conv;

double learning\_rate\_multiplier;

double weight\_decay\_multiplier;

double bias\_learning\_rate\_multiplier;

double bias\_weight\_decay\_multiplier;

long num\_filters\_;

int padding\_y\_;

int padding\_x\_;

public:

con\_& operator= (const con\_& item);

template <typename SUBNET>

void setup (const SUBNET& sub)

template <typename SUBNET>

void forward(const SUBNET& sub, resizable\_tensor& output)

template <typename SUBNET>

void backward(const tensor& gradient\_input, SUBNET& sub, tensor& params\_grad)

const tensor& get\_layer\_params() const

tensor& get\_layer\_params()

friend void serialize(const con\_& item, std::ostream& out)

friend void deserialize(con\_& item, std::istream& in)

friend std::ostream& operator<<(std::ostream& out, const con\_& item)

friend void to\_xml(const con\_& item, std::ostream& out)

}

template <long num\_filters, long nr, long nc,

int stride\_y, int stride\_x, typename SUBNET>

using con = add\_layer<con\_<num\_filters,nr,nc,stride\_y,stride\_x>, SUBNET>;

template <long \_num\_filters, long \_nr, long \_nc, int \_stride\_y, int \_stride\_x,

int \_padding\_y = \_stride\_y!=1? 0 : \_nr/2,

int \_padding\_x = \_stride\_x!=1? 0 : \_nc/2>

class con\_{}

template <long num\_filters, long nr, long nc,

int stride\_y, int stride\_x, typename SUBNET>

using cont = add\_layer<cont\_<num\_filters,nr,nc,stride\_y,stride\_x>, SUBNET>;

template <int scale\_y, int scale\_x >

class upsample\_{}

template <int scale, typename SUBNET>

using upsample = add\_layer<upsample\_<scale,scale>, SUBNET>;

template <long NR\_, long NC\_>

class resize\_to\_{}

template <long NR, long NC, typename SUBNET>

using resize\_to = add\_layer<resize\_to\_<NR,NC>, SUBNET>;

template <long \_nr, long \_nc, int \_stride\_y, int \_stride\_x,

int \_padding\_y = \_stride\_y!=1? 0 : \_nr/2,

int \_padding\_x = \_stride\_x!=1? 0 : \_nc/2>

class max\_pool\_{}

template <long nr, long nc, int stride\_y, int stride\_x, typename SUBNET>

using max\_pool = add\_layer<max\_pool\_<nr,nc,stride\_y,stride\_x>, SUBNET>;

template <typename SUBNET>

using max\_pool\_everything = add\_layer<max\_pool\_<0,0,1,1>, SUBNET>;

template <long \_nr, long \_nc, int \_stride\_y, int \_stride\_x,

int \_padding\_y = \_stride\_y!=1? 0 : \_nr/2,

int \_padding\_x = \_stride\_x!=1? 0 : \_nc/2>

class avg\_pool\_{}

template <long nr, long nc, int stride\_y, int stride\_x, typename SUBNET>

using avg\_pool = add\_layer<avg\_pool\_<nr,nc,stride\_y,stride\_x>, SUBNET>;

template <typename SUBNET>

using avg\_pool\_everything = add\_layer<avg\_pool\_<0,0,1,1>, SUBNET>;

enum layer\_mode{CONV\_MODE = 0,FC\_MODE = 1};

template <layer\_mode mode>

class bn\_{}

template <typename SUBNET>

using bn\_con = add\_layer<bn\_<CONV\_MODE>, SUBNET>;

template <typename SUBNET>

using bn\_fc = add\_layer<bn\_<FC\_MODE>, SUBNET>;

enum fc\_bias\_mode{FC\_HAS\_BIAS = 0, FC\_NO\_BIAS = 1};

template <unsigned long num\_outputs\_,fc\_bias\_mode bias\_mode>

class fc\_

{

private:

unsigned long num\_outputs;

unsigned long num\_inputs;

resizable\_tensor params;

alias\_tensor weights, biases;

double learning\_rate\_multiplier;

double weight\_decay\_multiplier;

double bias\_learning\_rate\_multiplier;

double bias\_weight\_decay\_multiplier;

public:

friend void to\_xml(const fc\_& item, std::ostream& out);

friend std::ostream& operator<<(std::ostream& out, const fc\_& item);

const tensor& get\_layer\_params() const

tensor& get\_layer\_params()

alias\_tensor\_const\_instance get\_biases() const

alias\_tensor\_instance get\_biases()

alias\_tensor\_const\_instance get\_weights() const

alias\_tensor\_instance get\_weights()

}

template <unsigned long num\_outputs, typename SUBNET>

using fc = add\_layer<fc\_<num\_outputs,FC\_HAS\_BIAS>, SUBNET>;

template <unsigned long num\_outputs, typename SUBNET>

using fc\_no\_bias = add\_layer<fc\_<num\_outputs,FC\_NO\_BIAS>, SUBNET>;

class dropout\_{}

template <typename SUBNET>

using dropout = add\_layer<dropout\_, SUBNET>;

class multiply\_{}

template <typename SUBNET>

using multiply = add\_layer<multiply\_, SUBNET>;

class affine\_

{

private:

resizable\_tensor params, empty\_params;

alias\_tensor gamma, beta;

layer\_mode mode;

计算elementwise的A\*INPUT+B的运算

2个模式，FC\_MODE和CONV\_MODE，对于FC\_MODE，A和B与输入张量维度(n, k, nr, nc)一致，只是n维度为1；对于CONV\_MODE，A和B只有k维度与输入一致，其他维度为1。

}

template <typename SUBNET>

using affine = add\_layer<affine\_, SUBNET>;

template <template<typename> class tag>

class add\_prev\_{}

template <template<typename> class tag, typename SUBNET>

using add\_prev = add\_layer<add\_prev\_<tag>, SUBNET>;

template <typename SUBNET> using add\_prev1 = add\_prev<tag1, SUBNET>;

template <template<typename> class tag>

class mult\_prev\_{}

template <template<typename> class tag, typename SUBNET>

using mult\_prev = add\_layer<mult\_prev\_<tag>, SUBNET>;

template <typename SUBNET> using mult\_prev1 = mult\_prev<tag1, SUBNET>;

template <template<typename> class tag>

class scale\_{}

template <template<typename> class tag, typename SUBNET>

using scale = add\_layer<scale\_<tag>, SUBNET>;

template <typename SUBNET> using scale1 = scale<tag1, SUBNET>;

class relu\_{}

template <typename SUBNET>

using relu = add\_layer<relu\_, SUBNET>;

class prelu\_{}

template <typename SUBNET>

using prelu = add\_layer<prelu\_, SUBNET>;

class sig\_{}

template <typename SUBNET>

using sig = add\_layer<sig\_, SUBNET>;

class htan\_{}

template <typename SUBNET>

using htan = add\_layer<htan\_, SUBNET>;

class softmax\_{}

template <typename SUBNET>

using softmax = add\_layer<softmax\_, SUBNET>;

class softmax\_all\_{}

template <typename SUBNET>

using softmax\_all = add\_layer<softmax\_all\_, SUBNET>;

template<template<typename> class... TAG\_TYPES>

class concat\_{}

…

class l2normalize\_{}

template <typename SUBNET>

using l2normalize = add\_layer<l2normalize\_, SUBNET>;

template <long \_offset, long \_k, long \_nr, long \_nc>

class extract\_{}

template <long offset, long k, long nr, long nc, typename SUBNET>

using extract = add\_layer<extract\_<offset,k,nr,nc>, SUBNET>;

# input.h

类：

template <typename T, long NR, long NC, typename MM, typename L>

class input<matrix<T,NR,NC,MM,L>>

template <typename T, long NR, long NC, typename MM, typename L, size\_t K>

class input<std::array<matrix<T,NR,NC,MM,L>,K>>

template <typename T, typename MM>

class input<array2d<T,MM>>

class input\_rgb\_image

{

private:

float avg\_red;

float avg\_green;

float avg\_blue;

public:

typedef matrix<rgb\_pixel> input\_type;

}

template <size\_t NR, size\_t NC>

class input\_rgb\_image\_sized{}

template <typename PYRAMID\_TYPE>

class input\_rgb\_image\_pyramid{}

# loss.h

class loss\_binary\_hinge\_

class loss\_binary\_log\_

class loss\_multiclass\_log\_

class loss\_multimulticlass\_log\_

class loss\_mmod\_

class loss\_metric\_

class loss\_ranking\_

class loss\_mean\_squared\_

class loss\_epsilon\_insensitive\_

class loss\_mean\_squared\_multioutput\_

class loss\_multiclass\_log\_per\_pixel\_

class loss\_mean\_squared\_per\_pixel\_

class loss\_multiclass\_log\_per\_pixel\_weighted\_

class loss\_dot\_

参考文献

1. Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. Deep Residual Learning for Image Recognition. CVPR 2016.