# 黑白图像上色技术说明书

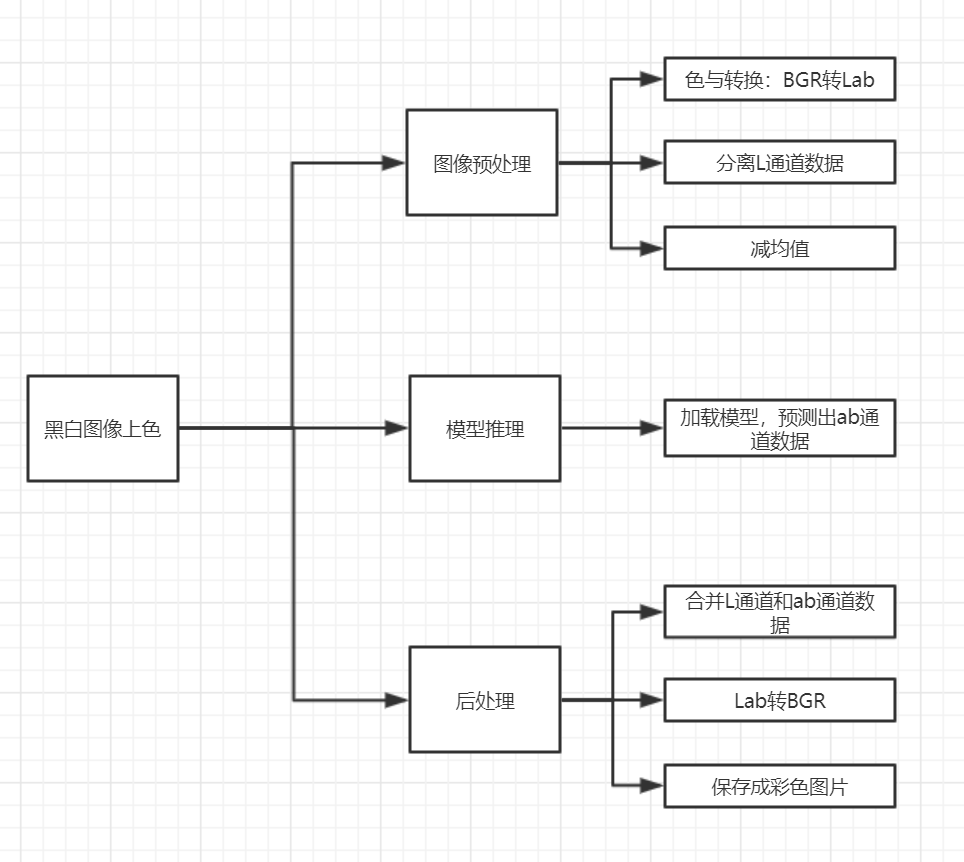
## 1 项目开发目的和意义

在智能手机越来越普及的今天，拍摄一张色彩鲜艳、清晰的照片轻而易举。但是老照片没有如此“幸运”，大多为黑白。借助人工智能，可以一定程度上帮助老照片还原原来色彩。

本项目是黑白图像上色应用，旨在华为Atlas200DK上实现输入黑白图像，自动对黑白图像进行上色，还原彩色图像。本应用能够输入多张黑白图片，上传至Atlas200DK，程序读取图片并加载训练好的自动上色模型进行上色，保存上色后的图片。

## 2 总体设计

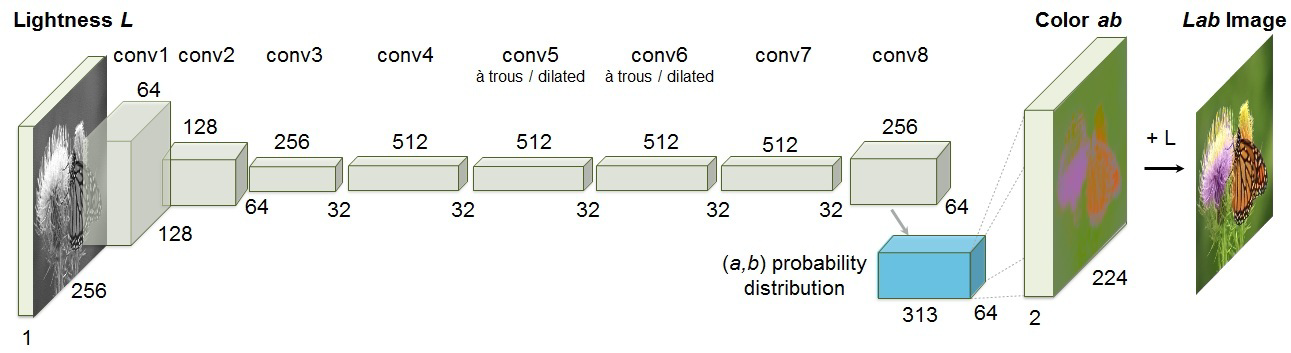
本应用可以划分为图像预处理、模型推理、后处理三个engine。前处理engine读取图片，归一化，把图片转换为模型推理需要的Lab色彩空间，resize到224\*224模型需要的尺寸，提取L通道，减均值等操作，为模型推理做准备。模型推理engine加载模型，对前处理engine处理好的L通道数据进行推理，得到预测的ab通道数据，并把推理结果发送给下游后处理engine。后处理engine接收推理结果，并把ab通道数据和L通道数据合并，resize回原图像大小，生成并保存上色后的图像，至此，自动上色完成。



## 3 算法设计

给定一张灰度图，该模型可以使得图像的颜色显得十分逼真。过去的黑白图像上色要么需要用户交互参与，要么会生成去饱和的图像。以前的作品使用均方误差（Mean Squared Error ,MSE）作为训练模型的损失函数，MSE将尝试“平均”出颜色，以获得最小的平均误差，这将导致平淡的外观。于是该模型将图片着色变为分类问题。该模型使用了LAB色彩空间模式（最常见的使RGB）。在LAB方案中，L通道记录了光强值，另外两个通道则记录了一组对抗色，绿-红和蓝-黄。使用LAB颜色空间的一个很有利的原因是它能使光强值保持分离。好处是黑白图片可以被当作L通道，这样模型在预测时就不必学习如何保持光强（使用RGB则必须）。这样，模型只需学习怎样将图像彩色化，从而专注于关键步骤。模型输出的AB通道值，可以被直接应用到黑白图像是上得到彩色图片。该模型是一个相当标准的卷积神经网络，模型并没有用池化层，取而代之的是使用上/下采样层。该模型使用的损失函数是标准的交叉熵：





模型网络结构图

颜色值在分类中会出现不均衡，使用交叉熵作为损失函数在分类不均衡上不能很好地发挥作用，通常具有较少示例的类具有更高的权重。而不饱和色灰色和浅蓝色相比其他会更加丰富广泛，因为它们常常出现在背景色中。所以该模型中采用了权重方案：



要获得分布，从整个ImageNet训练集上估计量化后的ab空间中色彩的经验概率，并用一个高斯kernel做平滑处理。Q是类型数量（313），λ取经验值0.5有不错的效果。考虑权重后，损失函数最终如下：



直接用313个色彩分类会使图像很粗糙，只有少数能与真实色彩匹配。为了从预测模型中得到更加多样化的色彩范围，又加了一步后续处理：



Z是模型的输出，T是一个超参数，经试验不通取值有不通效果：



这是很巧妙地一步，原因是模型的输出是一组概率值，相比只取最大概率所对应的颜色值，上述函数试图利用模型输出一组概率值的信息来计算颜色，因此效果更好。

训练这样一个神经网络分为两部分，首先训练数据正向传播，计算输出得到预测结果。然后计算H的倒数来计算损失（Loss）。

### 3.2 数据集制作

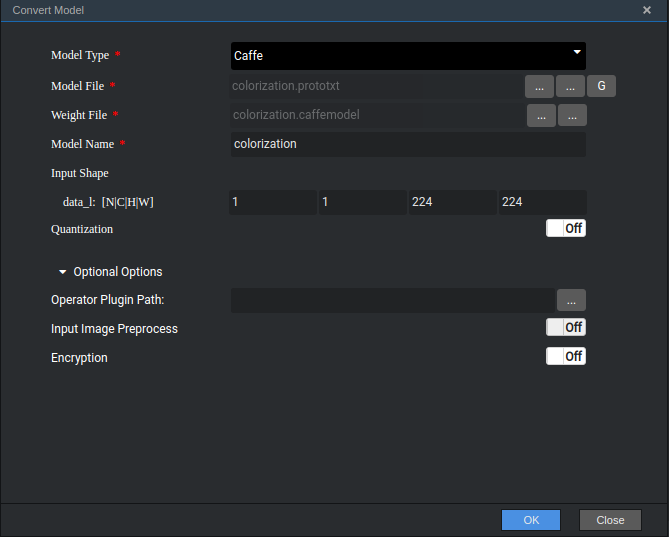
该模型训练的数据集为ImageNet数据集。

### 3.3 模型训练

该模型来自GitHub：<https://github.com/richzhang/colorization> 请从GitHub中获取模型网络结构，预训练模型等。

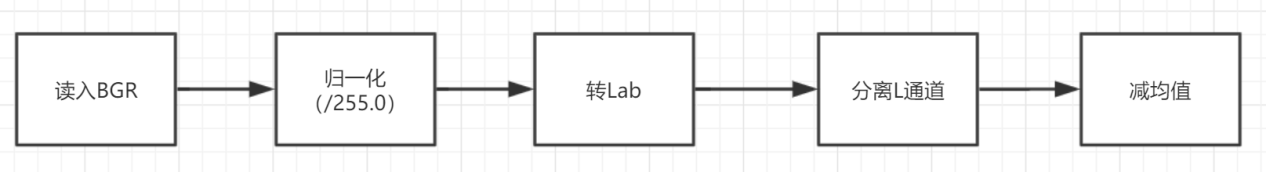
### 3.4 模型转换

要将训练好的caffe模型应用到Atlas200DK上，首先要将其转换为Ascend310芯片支持的离线模型，模型转换的擦书如下图所示。Model File为记录网络机构的prototxt文件，Weight File为记录参数值的caffemodel文件。推理过程的batch size需要设为1，图像的宽、高、通道数不变，因此输入的维度N、C、H、W分别对应为1、3、224、224。



### 3.5 图像预处理

本应用图像预处理部分，使用opencv的imread接口读取图片，读取出来的是BGR格式。模型输入为224×225，因此需要把读取到的图像resize到224×224。Resize后对数据做归一化处理。如算法设计部分所言，该模型采用Lab色彩空间，因此需要把BGR格式转为Lab格式数据。该模型用L通道数据预测出可能的ab空间数据，所以要从Lab数据中分离出L通道数据。再对所得的数据减均值，即可得到模型需要的输入数据。



### 3.6 模型推理

解析预处理engine发送来的数据，调用模型管家的Process接口进行推理，并把推理数据发送给下游后处理engine。

// 2. process

  HIAI\_ENGINE\_LOG("aiModelManager->Process start!");

  ret = ai\_model\_manager\_->Process(ai\_context, input\_data\_vec, output\_data\_vec,

                                   kAiModelProcessTimeout);

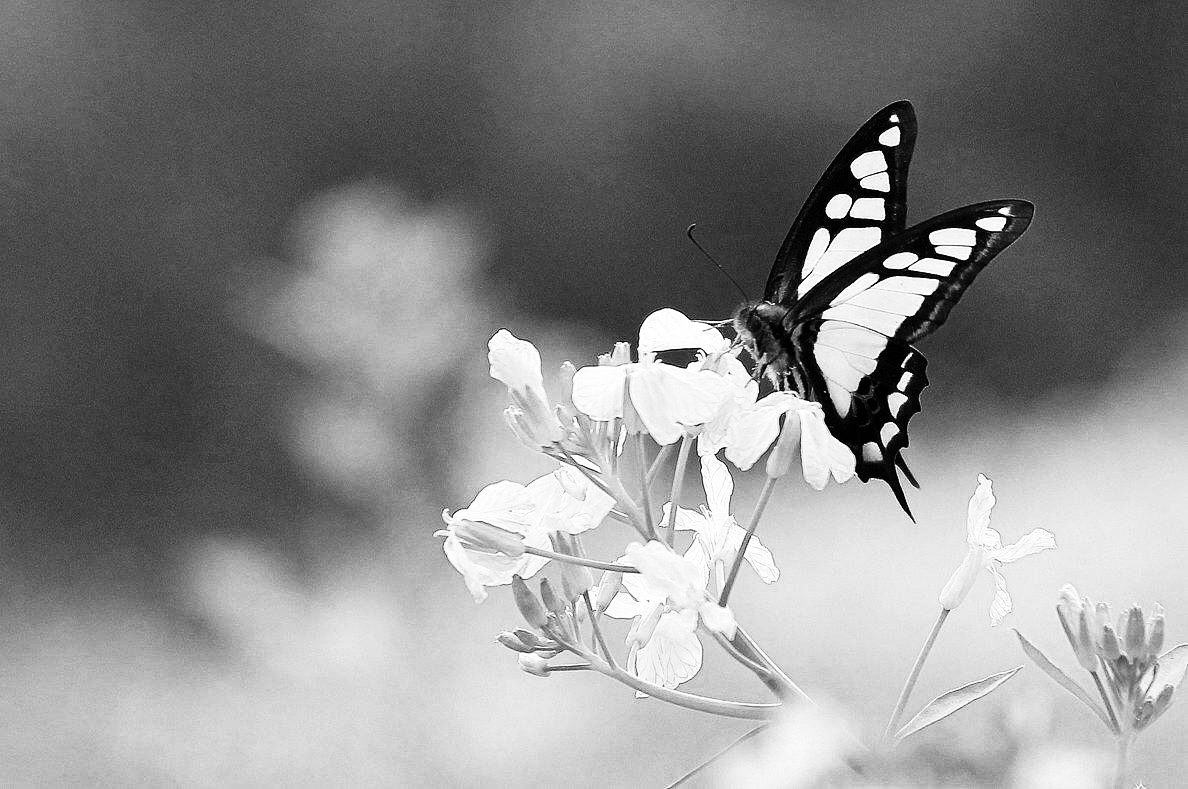
### 3.6 模型后处理

模型推理得到预测出的ab空间数据。首先把得到的数据resize回原图像大小，然后和原图像L通道数据合并，即得到完整Lab图像。把Lab图像转回BGR格式即可保存为jpeg图片，得到上色后的图像。



## 4 运行示例

黑白图像1：



自动上色后的图片1：



黑白图像2：



自动上色后的图片2：



黑白图像3：



自动上色后的图像3：



黑白图像4：



自动上色后的图像4：



## 5 关键代码说明

5.1 图像预处理关键代码

bool GeneralImage::ArrangeImageInfo(shared\_ptr<EngineTrans> &image\_handle,

const string &image\_path) {

// read image using OPENCV

cv::Mat mat = cv::imread(image\_path, CV\_LOAD\_IMAGE\_COLOR);

//resize

cv::Mat mat\_rs;

cv::resize(mat, mat\_rs, cv::Size(224,224));

// deal image

mat\_rs.convertTo(mat\_rs,CV\_32FC3);

mat\_rs = 1.0\*mat\_rs/255;

cv::cvtColor(mat\_rs, mat\_rs, CV\_BGR2Lab);

// pull out L channel and subtract 50 for mean-centering

std::vector<cv::Mat> channels;

cv::split(mat\_rs, channels);

cv::Mat mat\_rs\_l = channels[0] - 50;

if (mat.empty()) {

ERROR\_LOG("Failed to deal file=%s. Reason: read image failed.",

image\_path.c\_str());

return false;

}

// set property

image\_handle->image\_info.path = image\_path;

image\_handle->image\_info.width = mat\_rs\_l.cols;

image\_handle->image\_info.height = mat\_rs\_l.rows;

// set image data

uint32\_t size = mat\_rs\_l.total() \* mat\_rs\_l.channels()\*4;

u\_int8\_t \*image\_buf\_ptr = new (nothrow) u\_int8\_t[size];

if (image\_buf\_ptr == nullptr) {

HIAI\_ENGINE\_LOG("new image buffer failed, size=%d!", size);

ERROR\_LOG("Failed to deal file=%s. Reason: new image buffer failed.",

image\_path.c\_str());

return false;

}

error\_t mem\_ret = memcpy\_s(image\_buf\_ptr, size, mat\_rs\_l.ptr<u\_int8\_t>(),

mat\_rs\_l.total() \* mat\_rs\_l.channels()\*4);

if (mem\_ret != EOK) {

delete[] image\_buf\_ptr;

ERROR\_LOG("Failed to deal file=%s. Reason: memcpy\_s failed.",

image\_path.c\_str());

image\_buf\_ptr = nullptr;

return false;

}

image\_handle->image\_info.size = size;

image\_handle->image\_info.data.reset(image\_buf\_ptr,

default\_delete<u\_int8\_t[]>());

return true;

}

bool GeneralImage::SendToEngine(const shared\_ptr<EngineTrans> &image\_handle) {

// can not discard when queue full

HIAI\_StatusT hiai\_ret;

do {

hiai\_ret = SendData(kSendDataPort, "EngineTrans",

static\_pointer\_cast<void>(image\_handle));

// when queue full, sleep

if (hiai\_ret == HIAI\_QUEUE\_FULL) {

HIAI\_ENGINE\_LOG("queue full, sleep 200ms");

usleep(kSleepInterval);

}

} while (hiai\_ret == HIAI\_QUEUE\_FULL);

// send failed

if (hiai\_ret != HIAI\_OK) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"call SendData failed, err\_code=%d", hiai\_ret);

return false;

}

return true;

}

HIAI\_IMPL\_ENGINE\_PROCESS("general\_image",

GeneralImage, INPUT\_SIZE) {

HIAI\_StatusT ret = HIAI\_OK;

// Step1: check arg0

if (arg0 == nullptr) {

ERROR\_LOG("Failed to deal file=nothing. Reason: arg0 is empty.");

return HIAI\_ERROR;

}

// Step2: get all files

shared\_ptr<ConsoleParams> console\_param = static\_pointer\_cast<ConsoleParams>(

arg0);

string input\_path = string(console\_param->input\_path);

vector<string> file\_vec;

GetAllFiles(input\_path, file\_vec);

if (file\_vec.empty()) {

ERROR\_LOG("Failed to deal all empty path=%s.", input\_path.c\_str());

return HIAI\_ERROR;

}

// Step3: send every image to inference engine

for (string path : file\_vec) {

shared\_ptr<EngineTrans> image\_handle = nullptr;

MAKE\_SHARED\_NO\_THROW(image\_handle, EngineTrans);

if (image\_handle == nullptr) {

ERROR\_LOG("Failed to deal file=%s. Reason: new EngineTrans failed.",

path.c\_str());

continue;

}

// arrange image information, if failed, skip this image

if (!ArrangeImageInfo(image\_handle, path)) {

continue;

}

// send data to inference engine

image\_handle->console\_params.input\_path = console\_param->input\_path;

image\_handle->console\_params.model\_height = console\_param->model\_height;

image\_handle->console\_params.model\_width = console\_param->model\_width;

image\_handle->console\_params.output\_nums = console\_param->output\_nums;

image\_handle->console\_params.output\_path = console\_param->output\_path;

if (!SendToEngine(image\_handle)) {

ERROR\_LOG("Failed to deal file=%s. Reason: send data failed.",

path.c\_str());

continue;

}

// sleep

usleep(kSleepInterval);

}

// Step4: send finished data

shared\_ptr<EngineTrans> image\_handle = nullptr;

MAKE\_SHARED\_NO\_THROW(image\_handle, EngineTrans);

if (image\_handle == nullptr) {

ERROR\_LOG("Failed to send finish data. Reason: new EngineTrans failed.");

ERROR\_LOG("Please stop this process manually.");

return HIAI\_ERROR;

}

image\_handle->is\_finished = true;

if (SendToEngine(image\_handle)) {

return HIAI\_OK;

}

ERROR\_LOG("Failed to send finish data. Reason: SendData failed.");

ERROR\_LOG("Please stop this process manually.");

return HIAI\_ERROR;

}

5.2 模型推理部分关键代码

bool GeneralInference::Inference(

shared\_ptr<EngineTrans> &image\_handle,

vector<shared\_ptr<hiai::IAITensor>> &output\_data\_vec) {

// neural buffer

shared\_ptr<hiai::AINeuralNetworkBuffer> neural\_buf = nullptr;

MAKE\_SHARED\_NO\_THROW(neural\_buf, hiai::AINeuralNetworkBuffer);

if (neural\_buf == nullptr) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"new AINeuralNetworkBuffer failed");

return false;

}

neural\_buf->SetBuffer((void\*) image\_handle->image\_info.data.get(), image\_handle->image\_info.size);

// input data

shared\_ptr<hiai::IAITensor> input\_data = static\_pointer\_cast<hiai::IAITensor>(

neural\_buf);

vector<shared\_ptr<hiai::IAITensor>> input\_data\_vec;

input\_data\_vec.push\_back(input\_data);

// Call Process

// 1. create output tensor

hiai::AIContext ai\_context;

hiai::AIStatus ret = ai\_model\_manager\_->CreateOutputTensor(input\_data\_vec,

output\_data\_vec);

// create failed

if (ret != hiai::SUCCESS) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"call CreateOutputTensor failed");

return false;

}

// 2. process

HIAI\_ENGINE\_LOG("aiModelManager->Process start!");

ret = ai\_model\_manager\_->Process(ai\_context, input\_data\_vec, output\_data\_vec,

kAiModelProcessTimeout);

// process failed, also need to send data to post process

if (ret != hiai::SUCCESS) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT, "call Process failed");

return false;

}

HIAI\_ENGINE\_LOG("aiModelManager->Process end!");

return true;

}

bool GeneralInference::SendToEngine(

const shared\_ptr<EngineTrans> &image\_handle) {

// can not discard when queue full

HIAI\_StatusT hiai\_ret;

do {

hiai\_ret = SendData(kSendDataPort, "EngineTrans",

static\_pointer\_cast<void>(image\_handle));

// when queue full, sleep

if (hiai\_ret == HIAI\_QUEUE\_FULL) {

HIAI\_ENGINE\_LOG("queue full, sleep 200ms");

usleep(kSleepInterval);

}

} while (hiai\_ret == HIAI\_QUEUE\_FULL);

// send failed

if (hiai\_ret != HIAI\_OK) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"call SendData failed, err\_code=%d", hiai\_ret);

return false;

}

return true;

}

void GeneralInference::SendError(const std::string &err\_msg,

std::shared\_ptr<EngineTrans> &image\_handle) {

image\_handle->err\_msg.error = true;

image\_handle->err\_msg.err\_msg = err\_msg;

if (!SendToEngine(image\_handle)) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT, "SendData err\_msg failed");

}

}

bool GeneralInference::SendResult(

shared\_ptr<EngineTrans> &image\_handle,

vector<shared\_ptr<hiai::IAITensor>> &output\_data\_vec) {

for (uint32\_t i = 0; i < output\_data\_vec.size(); i++) {

shared\_ptr<hiai::AISimpleTensor> result\_tensor = static\_pointer\_cast<

hiai::AISimpleTensor>(output\_data\_vec[i]);

Output out;

out.size = result\_tensor->GetSize();

out.data = std::shared\_ptr<uint8\_t>(new (nothrow) uint8\_t[out.size],

std::default\_delete<uint8\_t[]>());

if (out.data == nullptr) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"dealing results: new array failed");

continue;

}

errno\_t mem\_ret = memcpy\_s(out.data.get(), out.size,

result\_tensor->GetBuffer(),

result\_tensor->GetSize());

// memory copy failed, skip this result

if (mem\_ret != EOK) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT,

"dealing results: memcpy\_s() error=%d", mem\_ret);

continue;

}

image\_handle->inference\_res.emplace\_back(out);

}

return SendToEngine(image\_handle);

}

HIAI\_IMPL\_ENGINE\_PROCESS("general\_inference",

GeneralInference, INPUT\_SIZE) {

HIAI\_StatusT ret = HIAI\_OK;

// arg0 is empty

if (arg0 == nullptr) {

HIAI\_ENGINE\_LOG(HIAI\_ENGINE\_RUN\_ARGS\_NOT\_RIGHT, "arg0 is empty.");

return HIAI\_ERROR;

}

// just send data when finished

shared\_ptr<EngineTrans> image\_handle = static\_pointer\_cast<EngineTrans>(arg0);

if (image\_handle->is\_finished) {

if (SendToEngine(image\_handle)) {

return HIAI\_OK;

}

SendError("Failed to send finish data. Reason: Inference SendData failed.",

image\_handle);

return HIAI\_ERROR;

}

// inference

vector<shared\_ptr<hiai::IAITensor>> output\_data;

if (!Inference(image\_handle, output\_data)) {

string err\_msg = "Failed to deal file=" + image\_handle->image\_info.path

+ ". Reason: inference failed.";

SendError(err\_msg, image\_handle);

return HIAI\_ERROR;

}

// send result

if (!SendResult(image\_handle, output\_data)) {

string err\_msg = "Failed to deal file=" + image\_handle->image\_info.path

+ ". Reason: Inference SendData failed.";

SendError(err\_msg, image\_handle);

return HIAI\_ERROR;

}

return HIAI\_OK;

}

5.3 图像后理关键代码

HIAI\_StatusT GeneralPost::ColorizationPostProcess(

const std::shared\_ptr<EngineTrans> &result) {

string file\_path = result->image\_info.path;

cout << "file\_path = " <<file\_path << endl;

// check vector

if (result->inference\_res.empty()) {

ERROR\_LOG("Failed to deal file=%s. Reason: inference result empty.",

file\_path.c\_str());

return HIAI\_ERROR;

}

// only need to get first one

Output out = result->inference\_res[0];

int32\_t size = out.size / sizeof(float);

// get a channel and b channel result data

cv::Mat mat\_a(56, 56, CV\_32FC1, const\_cast<float \*>((float \*)out.data.get()));

cv::Mat mat\_b(56, 56, CV\_32FC1, const\_cast<float \*>((float \*)out.data.get()+size/2));

// pull out L channel in original image

cv::Mat mat = cv::imread(file\_path, CV\_LOAD\_IMAGE\_COLOR);

mat.convertTo(mat,CV\_32FC3);

mat = 1.0\*mat/255;

cv::cvtColor(mat, mat, CV\_BGR2Lab);

std::vector<cv::Mat> channels;

cv::split(mat, channels);

// resize to match size of original image L

int r = mat.rows;

int c = mat.cols;

cv::Mat mat\_a\_up(r,c,CV\_32FC1);

cv::Mat mat\_b\_up(r,c,CV\_32FC1);

cv::resize(mat\_a, mat\_a\_up, cv::Size(c,r));

cv::resize(mat\_b, mat\_b\_up, cv::Size(c,r));

// result Lab image

cv::Mat newChannels[3] = {channels[0], mat\_a\_up, mat\_b\_up};

cv::Mat resultImage;

cv::merge(newChannels, 3, resultImage);

//convert back to rgb

cv::cvtColor(resultImage, resultImage, CV\_Lab2BGR);

// generate colorized image

int pos = file\_path.find\_last\_of(kFileSperator);

string file\_name(file\_path.substr(pos + 1));

cout << "file\_name = " << file\_name << endl;

stringstream sstream;

sstream.str("");

sstream << result->console\_params.output\_path << kFileSperator

<< kOutputFilePrefix << file\_name;

string outputPath = sstream.str();

resultImage = resultImage \* 255;

cv::imwrite(outputPath, resultImage);

if (size <= 0) {

ERROR\_LOG("Failed to deal file=%s. Reason: inference result size=%d error.",

file\_path.c\_str(), size);

return HIAI\_ERROR;

}

// transform results

float \*res = new (nothrow) float[size];

if (res == nullptr) {

ERROR\_LOG("Failed to deal file=%s. Reason: new float array failed.",

file\_path.c\_str());

return HIAI\_ERROR;

}

errno\_t mem\_ret = memcpy\_s(res, sizeof(float) \* size, out.data.get(),

out.size);

if (mem\_ret != EOK) {

delete[] res;

ERROR\_LOG("Failed to deal file=%s. Reason: call memcpy\_s failed.",

file\_path.c\_str());

return HIAI\_ERROR;

}

// get topN

int32\_t top\_n = result->console\_params.output\_nums;

vector<float> varr(res, res + size);

string top\_n\_str = GenerateTopNStr(top\_n, varr);

delete[] res;

//INFO\_LOG("Success to deal file=%s.", file\_path.c\_str());

//INFO\_LOG("Top index and confidence:%s", top\_n\_str.c\_str())

return HIAI\_OK;

}

HIAI\_IMPL\_ENGINE\_PROCESS("general\_post", GeneralPost, INPUT\_SIZE) {

HIAI\_StatusT ret = HIAI\_OK;

// check arg0

if (arg0 == nullptr) {

ERROR\_LOG("Failed to deal file=nothing. Reason: arg0 is empty.");

return HIAI\_ERROR;

}

// just send to callback function when finished

shared\_ptr<EngineTrans> result = static\_pointer\_cast<EngineTrans>(arg0);

if (result->is\_finished) {

if (SendSentinel()) {

return HIAI\_OK;

}

ERROR\_LOG("Failed to send finish data. Reason: SendData failed.");

ERROR\_LOG("Please stop this process manually.");

return HIAI\_ERROR;

}

// inference failed

if (result->err\_msg.error) {

ERROR\_LOG("%s", result->err\_msg.err\_msg.c\_str());

return HIAI\_ERROR;

}

// arrange result

return ColorizationPostProcess(result);

}

## 6 重要问题及解决

问题1：模型转换时报Silence算子不支持

解答：Silence算子是屏蔽无用打印信息作用，可以从prototxt中删除

问题2：无论换什么图片，模型输出都是0

解答：模型输出层节点需要加载一个cluster centers数据。先用caffe读取权重文件，再把该数据赋值给class8\_ab层即可

问题3：自动上色后的图像总是有一块区域色彩上色错了位置

解答：ab通道数据resize时宽高指定反了