TensorFlow在图像识别中的应用

发表于 2015-12-16 16:11 | 17645次阅读 | 来源 tensorflow.org | 5 条评论 | 作者 TensorFlow

计算机视觉 图形识别 TensorFlow 机器学习

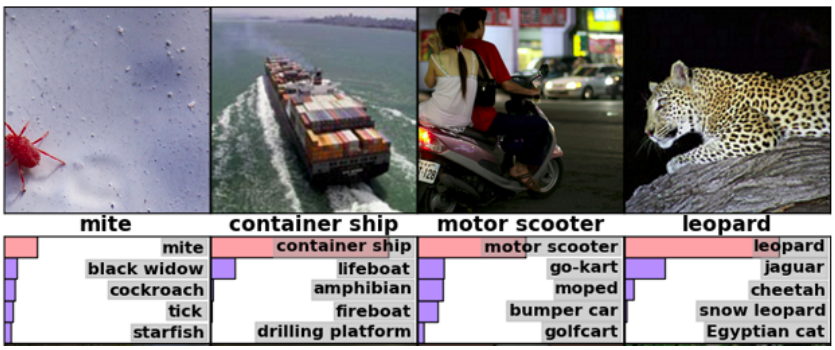
摘要：本教程将会教你如何使用Inception-v3。你将学会如何用Python或者C++把图像分为1000个类别，也会讨论如何从模型中提取高层次的特征，在今后其它视觉任务中可能会用到。本文章重点谈了TensorFlow在图像识别中的应用。

我们大脑的成像过程似乎很容易。人们毫不费力地就能区分出狮子和美洲虎，阅读符号，或是识别面孔。但是这些任务对于计算机而言却是一个大难题：它们之所以看上去简单，是因为我们的大脑有着超乎想象的能力来理解图像。

在过去几年里，机器学习在解决这些难题方面取得了巨大的进步。其中，我们发现一种称为深度卷积神经网络的模型在困难的视觉识别任务中取得了理想的效果 —— 达到人类水平，在某些领域甚至超过。

研究员们通过把他们的成果在ImageNet进行测试，来展示计算机视觉领域的稳定发展进步，ImageNet是计算机视觉领域的一个标准参照集。一系列的模型不断展现了性能的提升，每次都刷新了业界的最好成绩：QuocNet, AlexNet, Inception(GoogLeNet), BN-Inception-v2。谷歌的以及其它的研究员已经发表了论文解释这些模型，但是那些结果仍然很难被重现。我们正在准备发布代码，在最新的模型Inception-v3 上运行图像识别任务。

Inception-v3 是用来训练2012年ImageNet的Large Visual Recognition Challenge数据集。这是计算机视觉领域的一类标准任务，模型要把整个图像集分为1000个类别，例如“斑马”、“达尔玛西亚狗”，和“洗碗机”。如图所示，这里展示了一部分AlexNet的分类结果：



为了比较模型，我们检查模型预测前5个分类结果不包含正确类别的失败率 —— 即“top-5 错误率”。在2012年的验证数据集上，AlexNet取得了15.3%的 top-5 错误率；BN-Inception-v2的错误率是6.66%；Inception-v3的错误率是3.46%。

人类在ImageNet挑战赛上的表现如何呢？Andrej Karpathy写了一篇博文来测试他自己的表现。他的top-5 错误率是5.1%。

这篇教程将会教你如何使用Inception-v3。你将学会如何用Python或者C++把图像分为1000个类别。我们也会讨论如何从模型中提取高层次的特征，在今后其它视觉任务中可能会用到。

Python API的使用方法

第一次运行classify_image.py脚本时，它会从tensorflow.org官网上下载训练好的模型。你需要在磁盘上预留约200M的空间。

接下去的步骤默认你已经通过PIP包安装了TensorFlow，并且已经位于TensorFlow的根目录下。

```
cd tensorflow/models/image/imagenet

python classify_image.py
```

上述命令会对熊猫的图像分类。

如果脚本正确运行， 将会得到如下的输出结果：

```
giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca (score = 0.88493)
indri, indris, Indri indri, Indri brevicaudatus (score = 0.00878)
lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens (score = 0.00317)
custard apple (score = 0.00149)
earthstar (score = 0.00127)
```

如果你还想测试其它JPEG图片，修改 — image_file参数即可。

> 如果你把下载的模型放到了另一个目录下，则需要通过修改 — model_dir 参数指定地址。

C++ API的使用方法

你可以在生产环境中用C++运行同样的Inception-v3模型。按照下面的方式下载定义模型的GraphDef文件（在TensorFlow的根目录下运行）：

```
wget https://storage.googleapis.com/download.tensorflow.org/models
/inception_dec_2015.zip -O tensorflow/examples/label_image/data/inception_dec_2015.zip

unzip tensorflow/examples/label_image/data/inception_dec_2015.zip -d tensorflow/examples
/label_image/data/
```

接着，我们需要编译加载和运行模型的C++代码。如果你已经根据自己的平台环境，按照教程下载并安装了TensorFlow，那么在shell终端执行这条命令就能编译例子了：

```
bazel build tensorflow/examples/label_image/...
```

这一步生成了二进制可执行程序，然后这样运行：

```
bazel-bin/tensorflow/examples/label_image/label_image
```

它使用了框架自带的示例图片，输出的结果大致是这样：

```
I tensorflow/examples/label_image/main.cc:200] military uniform (866): 0.647296
I tensorflow/examples/label_image/main.cc:200] suit (794): 0.0477196
I tensorflow/examples/label_image/main.cc:200] academic gown (896): 0.0232411
```

```
I tensorflow/examples/label_image/main.cc:200] bow tie (817): 0.0157356
I tensorflow/examples/label_image/main.cc:200] bolo tie (940): 0.0145024
```

这里，我们使用的默认图像是 Admiral Grace Hopper，网络模型正确地识别出她穿着一套军服，分数高达0.6。

接着，通过修改 —image=argument参数来试一试你自己的图像。

```
bazel-bin/tensorflow/examples/label_image/label_image --image=my_image.png
```

如果你进入 tensorflow/examples/label_image/main.cc 文件仔细阅读，就能明白其中的原理。我们希望这段代码能帮助你把TensorFlow融入到你自己的产品中，因此我们一步步来解读主函数：

命令行指定了文件的加载路径，以及输入图像的属性。模型期望输入 299x299 RGB 图片，因此有 input_width 和 input_height两个标志。我们还需要把像素值从0~255的整数值转换为浮点数值。我们通过 input_mean 和 input_std 来控制归一化：首先给每个像素值减去 input_mean，然后除以 input_std。

这些数字可能看起来有些神奇，但它们是模型的原作者根据自己当时的想法定义的数值。如果你有一张自己训练的图片，你只需调整数值以匹配训练过程所使用的值。

你阅读ReadTensorFromImageFile() 函数就能够明白它们是如何被应用到一张图片上的。

```
// Given an image file name, read in the data, try to decode it as an image,
// resize it to the requested size, and then scale the values as desired.
Status ReadTensorFromImageFile(string file_name, const int input_height,
                               const int input_width, const float input_mean,
                               const float input_std,
                               std::vector<Tensor>* out_tensors) {
  tensorflow::GraphDefBuilder b;
```

首先创建一个GraphDefBuilder 对象，我们可以用它来指定运行或加载的模型。

```
  string input_name = "file_reader";
  string output_name = "normalized";
  tensorflow::Node* file_reader =
      tensorflow::ops::ReadFile(tensorflow::ops::Const(file_name, b.opts()),
                                b.opts().WithName(input_name));
```

接着，我们来为希望运行的模型创建节点，用于加载图像、调整大小和归一化像素值，使得其符合模

型的输入条件。我们创建的第一个节点只是一个Const操作，一个用来存放我们希望加载图像的文件名的tensor。然后它作为第一个输入传给ReadFile操作。你也许注意到了我们把 b.opts() 作为最后一个参数传给所有的op 创建函数。这个参数确保了节点被添加到GraphDefBuilder定义的模型下。我们也通过 b.opts() 调用 WithName() 函数来给ReadFile操作命名。给节点赋名字并不是严格要求的，因为即使我们不做，节点也会自动被分配一个名字，但这会让debug变得容易些。

```
// Now try to figure out what kind of file it is and decode it.
const int wanted_channels = 3;
tensorflow::Node* image_reader;
if (tensorflow::StringPiece(file_name).ends_with(".png")) {
  image_reader = tensorflow::ops::DecodePng(
      file_reader,
      b.opts().WithAttr("channels", wanted_channels).WithName("png_reader"));
} else {
  // Assume if it's not a PNG then it must be a JPEG.
  image_reader = tensorflow::ops::DecodeJpeg(
      file_reader,
      b.opts().WithAttr("channels", wanted_channels).WithName("jpeg_reader"));
}
// Now cast the image data to float so we can do normal math on it.
tensorflow::Node* float_caster = tensorflow::ops::Cast(
    image_reader, tensorflow::DT_FLOAT, b.opts().WithName("float_caster"));
// The convention for image ops in TensorFlow is that all images are expected
// to be in batches, so that they're four-dimensional arrays with indices of
// [batch, height, width, channel]. Because we only have a single image, we
// have to add a batch dimension of 1 to the start with ExpandDims().
tensorflow::Node* dims_expander = tensorflow::ops::ExpandDims(
    float_caster, tensorflow::ops::Const(0, b.opts()), b.opts());
// Bilinearly resize the image to fit the required dimensions.
tensorflow::Node* resized = tensorflow::ops::ResizeBilinear(
    dims_expander, tensorflow::ops::Const({input_height, input_width},
                                          b.opts().WithName("size")),
    b.opts());
// Subtract the mean and divide by the scale.
tensorflow::ops::Div(
    tensorflow::ops::Sub(
        resized, tensorflow::ops::Const({input_mean}, b.opts()), b.opts()),
    tensorflow::ops::Const({input_std}, b.opts()),
    b.opts().WithName(output_name));
```

我们接着添加更多的节点，解码数据文件得到图像内容，将整型的像素值转换为浮点型值，调整图像大小，最后对像素值做减法和除法的归一化运算。

```
// This runs the GraphDef network definition that we've just constructed, and
// returns the results in the output tensor.
tensorflow::GraphDef graph;
TF_RETURN_IF_ERROR(b.ToGraphDef(&graph));
```

最终，变量b包含了模型定义的信息，我们用ToGraphDef() 函数将其转换为一个完整的图定义。

```
std::unique_ptr<tensorflow::Session> session(
    tensorflow::NewSession(tensorflow::SessionOptions()));
TF_RETURN_IF_ERROR(session->Create(graph));
TF_RETURN_IF_ERROR(session->Run({}, {output_name}, {}, out_tensors));
return Status::OK();
```

然后，我们再创建一个 Session 对象，它是真正用来运行图的接口，并且运行它，同时指定我们从哪个节点得到输出结果以及输出数据存放在哪儿。

我们会得到一组 Tensor 对象，在这个例子中一组tensor对象仅有一个成员（只有一张输入图片）。这里你可以把 Tensor 当做是一个多维数组，它以浮点数组的形式存放299像素高、299像素宽、3个通道的图像。如果你现有的产品中已经有了自己的图像处理框架，可以继续使用它，只需要保证在输入图像之前进行同样的预处理步骤。

这是用C++动态创建小型 TensorFlow 图的简单例子，但是对于预训练的Inception模型，我们则需要从文件中加载大得多的定义内容。查看 LoadGraph() 函数我们是如何实现的。

```
// Reads a model graph definition from disk, and creates a session object you
// can use to run it.
Status LoadGraph(string graph_file_name,
                 std::unique_ptr<tensorflow::Session>* session) {
  tensorflow::GraphDef graph_def;
  Status load_graph_status =
      ReadBinaryProto(tensorflow::Env::Default(), graph_file_name, &graph_def);
  if (!load_graph_status.ok()) {
    return tensorflow::errors::NotFound("Failed to load compute graph at '",
                                        graph_file_name, "'");
  }
```

如果你仔细阅读图像加载的代码，会发现很多熟悉的术语。不同于用 GraphDefBuilder 来生产一个 GraphDef 对象，我们直接加载包含 GraphDef 的protobuf文件。

```
  session->reset(tensorflow::NewSession(tensorflow::SessionOptions()));
  Status session_create_status = (*session)->Create(graph_def);
  if (!session_create_status.ok()) {
    return session_create_status;
  }
  return Status::OK();
}
```

我们然后从那个 GraphDef 创建一个 Session 对象，将它传回给调用者以便后续调用执行。

GetTopLabels() 函数和图像加载的过程很像，差别在于这里我们想获取运行完main graph的结果，将其按照得分从高到低排序取前几位的标签。如同 image loader，它创建一个 GraphDefBuilder，往里添加一些节点，然后运行short graph得到一对输出的tensor。本例中是输出有序的得分和得分最高结果的索引号。

```
// Analyzes the output of the Inception graph to retrieve the highest scores and
// their positions in the tensor, which correspond to categories.
Status GetTopLabels(const std::vector<Tensor>& outputs, int how_many_labels,
                    Tensor* indices, Tensor* scores) {
  tensorflow::GraphDefBuilder b;
  string output_name = "top_k";
  tensorflow::ops::TopK(tensorflow::ops::Const(outputs[0], b.opts()),
                        how_many_labels, b.opts().WithName(output_name));
  // This runs the GraphDef network definition that we've just constructed, and
  // returns the results in the output tensors.
  tensorflow::GraphDef graph;
  TF_RETURN_IF_ERROR(b.ToGraphDef(&graph));
  std::unique_ptr<tensorflow::Session> session(
      tensorflow::NewSession(tensorflow::SessionOptions()));
  TF_RETURN_IF_ERROR(session->Create(graph));
```

```
  // The TopK node returns two outputs, the scores and their original indices,
  // so we have to append :0 and :1 to specify them both.
  std::vector<Tensor> out_tensors;
  TF_RETURN_IF_ERROR(session->Run({}, {output_name + ":0", output_name + ":1"},
                                  {}, &out_tensors));
  *scores = out_tensors[0];
  *indices = out_tensors[1];
  return Status::OK();
```

PrintTopLabels() 函数接收排序完的结果，然后打印输出到控制台。CheckTopLabel() 函数的功能也非常相似，只是验证顶部的标签符合我们的结果预期，为了调试的时候方便。

最后，main() 函数串联所有的调用方法。

```
int main(int argc, char* argv[]) {
  // We need to call this to set up global state for TensorFlow.
  tensorflow::port::InitMain(argv[0], &argc, &argv);
  Status s = tensorflow::ParseCommandLineFlags(&argc, argv);
  if (!s.ok()) {
    LOG(ERROR) << "Error parsing command line flags: " << s.ToString();
    return -1;
  }

  // First we load and initialize the model.
  std::unique_ptr<tensorflow::Session> session;
  string graph_path = tensorflow::io::JoinPath(FLAGS_root_dir, FLAGS_graph);
  Status load_graph_status = LoadGraph(graph_path, &session);
  if (!load_graph_status.ok()) {
    LOG(ERROR) << load_graph_status;
    return -1;
  }
```

加载main graph。

```
  // Get the image from disk as a float array of numbers, resized and normalized
  // to the specifications the main graph expects.
  std::vector<Tensor> resized_tensors;
  string image_path = tensorflow::io::JoinPath(FLAGS_root_dir, FLAGS_image);
  Status read_tensor_status = ReadTensorFromImageFile(
      image_path, FLAGS_input_height, FLAGS_input_width, FLAGS_input_mean,
      FLAGS_input_std, &resized_tensors);
  if (!read_tensor_status.ok()) {
    LOG(ERROR) << read_tensor_status;
    return -1;
  }
  const Tensor& resized_tensor = resized_tensors[0];
```

加载输入图像，调整大小，完成预处理。

```
  // Actually run the image through the model.
  std::vector<Tensor> outputs;
  Status run_status = session->Run({{FLAGS_input_layer, resized_tensor}},
                                   {FLAGS_output_layer}, {}, &outputs);
  if (!run_status.ok()) {
    LOG(ERROR) << "Running model failed: " << run_status;
    return -1;
```

```
}
```

我们以图片作为输入，运行加载完的graph。

```
// This is for automated testing to make sure we get the expected result with
// the default settings. We know that label 866 (military uniform) should be
// the top label for the Admiral Hopper image.
if (FLAGS_self_test) {
  bool expected_matches;
  Status check_status = CheckTopLabel(outputs, 866, &expected_matches);
  if (!check_status.ok()) {
    LOG(ERROR) << "Running check failed: " << check_status;
    return -1;
  }
  if (!expected_matches) {
    LOG(ERROR) << "Self-test failed!";
    return -1;
  }
}
```

为了完成测试，我们可以检查输出的结果是否符合预期。

```
// Do something interesting with the results we've generated.
Status print_status = PrintTopLabels(outputs, FLAGS_labels);
```

最后，打印输出得到的标签。

```
if (!print_status.ok()) {
  LOG(ERROR) << "Running print failed: " << print_status;
  return -1;
}
```

异常处理使用了TensorFlow的Status对象，非常方便，调用ok() 函数就能知道是否出现了任何错误，还可以将错误信息以易读的方式打印出来。

我们在这个例子中演示了物体识别功能，今后无论在什么领域，你都应该学会将类似的代码用于其它模型或者你自己训练的模型。希望这个小例子能带给你一些启发，将TensorFlow用于自己的产品。

> 练习：迁移学习（transfer learning）的思想是人们若是擅长解决一类任务，那就应该能迁移其中的理解内容，用它来解决另一类相关的问题。实现迁移学习的方法之一就是移除网络的最后一层分类层，并且提取CNN的倒数第二层，在本例中是一个2048维的向量。可以通过C++的API设置 -- output_layer=pool_3 来指定，然后修改输出tensor。尝试在一个图像集里提取这个特征，看看你是否能够预测不属于ImageNet的新类型。

延伸阅读

想要获取更多的神经网络普及资料，Michael Niesen 的免费电子书是个极好的资源。针对卷积神经网络，Chris Olah写过一些很赞的博客，Michael Nielsen的书里也有一个章节详细介绍。

若是要了解更多卷积神经网络的应用，你可以直接前去阅读TensorFlow的深度卷积神经网络章节，或是从ML beginner和ML expert MNIST初学者教程逐渐深入。最后，若果想要追赶此领域的前沿动态，可以阅读本教程所引用的所有文献。

原文链接：Image Recognition （翻译/赵屹华 审校/刘翔宇 责编/周建丁）

译者简介：赵屹华，计算广告工程师@搜狗，前生物医学工程师，关注推荐算法、机器学习领域。

已有5条评论

共1页 首页 上一页 1 下一页 末页