Fine-grained image classification

Digital Image Processing

Yang Guofeng
Master Graduate Student

Cao Liang
Master Graduate Student

Project Topic and Problem Statement:

图像细粒度分类：识别同一大类中的子类

与一般的图像分类不同，图像细分类旨在对粗粒度的大类（如鸟）进行更加细粒度的子类划分（如海鸥、鹦鹉等）其挑战在于外形、颜色等的相似导致不同类别的差异小，姿态、视角等的不同导致相同类别的差异大。

Dataset, Features and Software:

数据集：CUB_200_2011

图像数量：11788 张（鸟），分为训练集：5994 张，测试集：5794 张

类别：共 200 个类别，均为鸟的子类

每张图片：15 Part Locations, 312 Binary Attributes, 1 Bounding Box

编程语言及环境：Python；Tensorflow，Sublime Text 3

Methodology:

BCNN（Bilinear-CNN），即双线性卷积神经网络，是一种端到端的基于神经网络深度学习的细分类方法。在论文的介绍中，对输入的每张图片，先通过两个卷积神经网络，然后将两个通道中得到的输出按通道进行相乘，然后进行池化操作（每个通道中的元素相加求平均），并通过一个全连接层后得到对应的输出。

迁移学习（Transfer Learning），即把已训练好的模型参数迁移到新的模型来帮助新模型训练。考虑到大部分数据或任务是存在相关性的，所以通过迁移学习可以将已经学到的模型参数（也可理解为模型学到的知识）通过某种方式来分享给新模型从而加快并优化模型的学习效率不用像大多数网络那样从零学习。

Steps of Model

BCNN: 使用 BCNN 网络在较短时间内实现有限数据情况下细粒度任务的高识别分类率。不需要使用位置标记和轮廓标记，需要使用预训练网络，由于使用内积计算的关系，对物体的位置信息不敏感，但也让全连接层的计算结果不需要依赖于定位框的选取，从而能够进行端到端的训练。

加载数据：由 CUB_200_2011 文件夹下的 train_test_split.txt 可以获得 train, test 数据集，并将其存放进 new_train.h5 文件和 new_val.h5 文件，以便数据存储和模型读取数据。

构建模型：基于 VGG16 卷积神经网络并且载入预训练好的网络参数，同时去掉网络最后的全连接层，只留下卷积层。然后对每一组输入的图片，先将图片缩放为 224x224x3 大小，然后通过 vgg16 后变成 14x14x512 大小，一共 512 通道，每个通道 7x7 大小，然后将输出复制一份，两份输出的通道两两进行内积，然后将内积的结果取平均后开方，得到 512x512 维的向量，将向量归一化后通过一层的全连接层，全连接层输出为 200 维向量，对应结果的两百个类别，取其中数值最高的一维作为最后的分类。

训练模型：采用 tf.train.MomentumOptimizer(momentum=0.9)，训练分为两步，第一步锁定卷积层参数，只训练全连接层，学习率为 0.9，第二步载入第一步时的全连接层数据，同时训练卷积层和全连接层参数，学习率为 0.01。为了减少训练过程中的过拟合，采用了三个策略：①图片随机翻转，图片输入网络前，会随机对图片进行上下或左

右翻转。②图片随机变形，图片输入网络前，对图片进行小幅度拉伸变换并裁剪成一样大小。③随机 dropout，在训练过程中随机屏蔽一部分全连接层的参数。

评估模型：在图片输入格式为 224x224 情况下，我们最后训练的结果只有 73%，相比论文中 84%的准确率还是差了不少。我们试着将输入图片放大成 448x448x3 大小，准确率升高，但是受限于时间限制，训练的不够充分，最后准确率为 79.9%。

```
correct_test_count, total_test_count 4795.0 5996
Test Data Accuracy --> 79.96997998665778
```

Transfer Learning：使用已训练好的 ResNet50，VGG19，InceptionV3，Xception 模型，将模型及参数迁移到新的模型（Stacking Model）来帮助新模型训练。

加载数据：调用 CUB200.class 将 CUB_200_2011 数据集图像加载到指定大小的图像，并返回到 X 储存。

构建模型：①使用 Resnet50 模型进行特征提取。使用大小为 244X244 像素的图像，生成整个数据集的向量表示。首次调用时，ResNet50 构造函数将下载预先训练的参数文件。然后将这些特征向量用于具有简单线性 SVM 分类器的交叉验证过程中，构建基于 Resnet 的 LinearSVC 模型。②组合来自 4 种预先训练模型（VGG19，ResNet50，Inception 和 Xception）。将每个预训练模型最初的特征集合（X_vgg, X_resnet, X_incept, X_xcept）叠加到一个矩阵中，同时保留索引。对每个基本分类器中，构建一个 pipeline，并且添加 LogisticRegression。构建基于 4 个预训练的集成学习元分类器 StackingClassifier 模型。

训练模型：由于是基于 Stacking 方法使用预训练好的模型，因此可以直接加载 VGG19，ResNet50，Inception 和 Xception 模型及参数，集成 Stacking Model。

评估模型：采用 ResNet50，在图片输入格式为 224x224 情况下，在 CUB_200_2011 数据集上获得 64.7%的准确率。利用集成学习模型进行分类识别可以在数据集上获得 75.6%准确率。

```
[0.75158268 0.75412332 0.76346845]
Overall accuracy: 75.6
```

Core Code:

BCNN

读入 vgg16 网络中全连接层之外的参数，用于初始化网络。	`def load_weights(self, sess):`
	`  #saver=tf.train.Saver(self.parameters)`
	`  #save_path="../save"`
	`  #saver.restore(sess,save_path)`
	`  weights = np.load(self.weight_file)`
	`  #return`
	`  keys = sorted(weights.keys())`
	`  for i, k in enumerate(keys):`
	`  removed_layer_variables = ['fc6_W','fc6_b','fc7_W','fc7_b','fc8_W','fc8_b']`
	`  if not k in removed_layer_variables:`
	`  print(k)`
	`  print("",i, k, np.shape(weights[k]))`
	`  sess.run(self.parameters[i].assign(weights[k]))`

在卷积层最后的池化部分，将得到的不同通道间的输出进行相乘求平均，然后求平方根正则化和归一化。	`self.InnerPro = tf.einsum('ijkm,ijkn->imn',self.conv5_3,self.conv5_3)`
	`self.InnerPro = tf.reshape(self.InnerPro,[-1,512*512])`
	`self.InnerPro = tf.divide(self.InnerPro,14.0*14.0)`
	`self.ySsqrt = tf.multiply(tf.sign(self.InnerPro),tf.sqrt(tf.abs(self.InnerPro)+1e-12))`
	`self.zL2 = tf.nn.l2_normalize(self.ySsqrt, dim=1)`

在将通道相乘后并池化后，通过最后的全连接层计算。	`def fc_layers(self):`
	`  with tf.name_scope('fc') as scope:`
	`  fcw = tf.get_variable('weights', [512*512, 200],`
	`  initializer=tf.contrib.layers.xavier_initializer(), trainable=True)`
	`  fcb = tf.Variable(tf.constant(1.0, shape=[200], dtype=tf.float32),`
	`  name='biases', trainable=True)`
	`  self.fcl = tf.nn.bias_add(tf.matmul(self.zL2,`
	`  tf.contrib.layers.dropout(fcw,self.drop_prob)),`
	`  tf.contrib.layers.dropout(fcb,self.drop_prob))`
	`  self.last_layer_parameters += [fcw, fcb]`
	`  self.parameters += [fcw, fcb]`

图片训练前将图片随机翻转以增加数据量，减少过拟合。	`def random_flip_right_to_left(image_batch):`
	`  result = []`
	`  for n in range(image_batch.shape[0]):`
	`  if bool(random.getrandbits(1)):`
	`  result.append(image_batch[n][:,::-1,:])`
	`  elif bool(random.getrandbits(1)):`
	`  result.append(image_batch[n][::-1,:,:])`
	`  else:`
	`  result.append(image_batch[n])`
	`  return result`

将图片适度拉伸后切取 224*224 大小，也是为了增加数据量。	`def random_crop(image_batch):`
	`  result = []`
	`  for n in range(image_batch.shape[0]):`
	`  start_x = random.randint(0,19)`
	`  start_y = random.randint(0,19)`
	`  result.append(scipy.misc.imresize(image_batch[n][start_y:start_y+224,start_x:start_x+224,:],(224,224,3)))`
	`  return np.array(result)`

为全连接层添加 dropout，减少数据的过拟合。	`self.fc1 = tf.nn.bias_add(tf.matmul(self.zL2,`
	`  tf.contrib.layers.dropout(fcw,self.drop_prob)),`
	`  tf.contrib.layers.dropout(fcb,self.drop_prob))`

在运行时保存验证集（测试集的前一小部分）得到更优结果时的参数，在第二次以后运行时，取消 load_weights 中的注释，以读入上次运行的结果。	`acc = 100.0*correct_val_count/(1.0*total_val_count)`
	`if acc>best_accuracy:`
	`  best_accuracy = acc`
	`  saver=tf.train.Saver(vgg.parameters)`
	`  save_path="../save"`
	`  saver.save(sess,save_path)`

当全连接层未训练时，isFineTune=False,训练全连接层，同时冻结卷积层参数；
当全连接层训练好后，isFineTune=True，进行学习率更细的训练，同时训练卷积层参数。
	`isFineTune=True`
	
	`if isFineTune:`
	`  optimizer = tf.train.MomentumOptimizer(learning_rate=0.001, momentum=0.9).minimize(loss)`
	`else: optimizer = tf.train.MomentumOptimizer(learning_rate=0.9, momentum=0.9).minimize(loss)`
	
	`with tf.name_scope('conv1_1') as scope:`
	`  kernel = tf.Variable(tf.truncated_normal([3, 3, 3, 64], dtype=tf.float32,`
	`  stddev=1e-1), trainable=trainable, name='weights')`

Transfer_Learning

读取加载数据集。使用程序将数据集图像加载为指定大小的图像。	`X, y = CUB200(CUB_DIR, image_size=im_size).load_dataset()`

使用 Resnet50 模型进行特征提取。使用大小为 244X244 像素的图像，生成整个数据集的向量表示。	`X = model(include_top=False, weights="imagenet", pooling='avg').predict(preprocess(X))`
	`pd.DataFrame(X, index=y).to_csv(save_path, compression='gzip', header=True, index=True)`
	`X_resnet, y = load_features_compute_once(ResNet50, (244, 244), preprocess_type1,`
	`  os.path.join(FEATURES_DIR, "CUB200_resnet"))`

首次调用时，ResNet50 构造函数将下载预先训练的参数文件。然后将这些特征向量用于具有简单线性 SVM 分类器的交叉验证过程中。	`results = cross_val_score(clf, X_resnet, y, cv=3, n_jobs=-1)`
	`print(results)`
	`print("Overall accuracy: {:.3}".format(np.mean(results) * 100.))`

采用 ResNet50，在 CUB_200_2011 数据集上获得 64.7%的准确率。	[0.64978476 0.63486425 0.65623397] Overall accuracy: 64.7
组合来自 4 种预先训练模型（VGG19，ResNet50，Inception 和 Xception）。每个基本分类器将是一个简单的逻辑回归。然后对这些概率输出进行平均，并输入线性 SVM，然后提供最终决策。	base_classifier = LogisticRegression meta_classifier = LinearSVC
将每个预训练模型最初的特征集合（X_vgg, X_resnet, X_incept, X_xcept）叠加到一个矩阵中，同时保留索引。	X_all = np.hstack([X_vgg, X_resnet, X_incept, X_xcept]) inx = np.cumsum([0] + [X_vgg.shape[1], X_resnet.shape[1], X_incept.shape[1], X_xcept.shape[1]])
对每个基本分类器中，构建一个 pipeline，并且添加 LogisticRegression。	pipes = [make_pipeline(ColumnSelector(cols=list(range(inx[i], inx[i+1]))), base_classifier()) for i in range(4)]
一种用于堆叠的集成学习元分类器 StackingClassifier，使用每个基本分类器提供的平均概率作为聚合函数。	stacking_classifier = StackingClassifier(classifiers=pipes, meta_classifier=meta_classifier(), use_probas=True, average_probas=True, verbose=1)
对数据集进行指定次数的交叉验证。	results = cross_val_score(stacking_classifier, X_all, y, cv=3, n_jobs=-1) print(results) print("Overall accuracy: {:.3}".format(np.mean(results) * 100.))

Results:

基于 BCNN 模型，在图片输入格式为 224x224 情况下，最后训练的结果只有 73%，相比论文中 84%的准确率还是差了不少。通过将输入图片放大成 448x448x3，准确率升高，但是受限于时间限制，训练的不够充分，最后准确率为 79.9%。

基于 ResNet50 模型，在 CUB_200_2011 数据集上可以获得 64.7%的准确率。利用 stacking 方法，构建基于 4 个预训练的模型分类器对 CUB_200_2011 数据集 200 类鸟进行分类，可以获得 74.5%的准确性。

BCNN and Transfer_Learning identify CUB_200_2011

Model	Test Accuracy
BCNN	79.9%
Transfer_Learning	74.5%

Bonus 1

BCNN 算法在我们的实现中，使用的都是卷积层的输出作为 bcnn 的混合层的两个输入，我们曾尝试过使用卷积层的中间层输出作为输入或是相邻两层的输出做两个输入，但无论从准确率还是损失率的收敛速度都不如卷积层的最后一层高。我们之后打算尝试不同的卷积算子得到混合层输入，并且尝试对输入图像用物体框定位后作为输入。

Bonus 2

我们使用基于预训练好的 ImageNet 中的模型，将模型及参数迁移到新的模型（Stacking Model）来帮助新模型训练，从而得到不错的分类识别准确率，既减少了模型对相关信息（如：Part Locations, Bounding Box）的需求也能让模型构建者快速的构建完整的集成模型。当然，在特定情况下迁移学习甚至会产生相反的效果。但是，迁移学习可以帮助我们处理全新的场景，它是机器学习在没有大量标签数据的任务和领域中规模化应用所必须的。到目前为止，我们已经把我们的模型应用在能够容易获取数据的任务和领域中。我们将继续研究在强化学习中的任务迁移，利用迁移学习来实现更多更有趣的项目。

Team Contribution:

Cao Liang: BCNN Model and Complete the report,slides
Yang Guofeng: Collection of papers,material and Transfer_Learning and Method discussion and Complete the report,slides

Sources:

Lin T Y, RoyChowdhury A, Maji S. Bilinear CNN models for fine-grained visual recognition.
https://arxiv.org/abs/1504.07889v5
Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. NIPS(2014), How transfer-able are features in deep neural networks?
https://arxiv.org/abs/1411.1792
J. Donahue, Y. Jia, O. Vinyals.., ICML(2014): A deep convolutional activation feature for generic visual recognition.
https://arxiv.org/abs/1310.1531?context=cs

Supplementary Materials:

1. Python code File
2. Slides