武汉大学计算机学院
本科生课程设计报告

基于 ResNet 的细粒度猴子分类

专 业 名 称　：软件工程

课 程 名 称　：商务智能

指 导 教 师　：朱卫平　　职称：副教授

学 生 学 号　：2018302091011

学 生 姓 名　：陈永灿

二〇二〇年十二月

郑 重 声 明

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

本人签名：陈永灿　　日期：2020 年 12 月 1 日

摘要

本次商务智能大作业选择采用人工智能算法解决细粒度猴子分类的问题。本方案以 PaddlePaddle 作为基准框架，以 ResNet[1]构建卷积神经网络，尝试各项能提升模型性能的技术并进行迭代，搭建出能有效提升模型性能和准确率的框架。该框架对模型的训练数据进行基于 imgaug 的随机图像增强，以提升训练集的数量，在训练时使用 Adam 优化器，该方案具有良好的性能与准确性，在验证集上获得最高 94.1%的准确率。

关键词：ResNet；随机图像增强；PaddlePaddle

目录

1 实验背景和需求分析

1.1 实验背景

要说起在动物界中哪种动物智商最高，那么人们通常会想到猴子，的确，猴子从古至今在我国的地位都比较高，就连神话剧西游记里面神通广大的孙悟空都是以猴子为造型而打造出来的，并且按照古老的传统说法，认为人类最早是由猿类进化过来的，由此可见，猴子在中国已经传承了多少年了，而发展到今天，猴子的种类也是多种多样，估计有很多种类人们都叫不上名来。而且很多猴子由于长相过于相似，难以区分，本次实验将对 10 种不同种类的猴子图片进行训练，并进行预测。

1.2 需求分析

正如背景所说的，这是个图像分类问题。典型的图像分类问题往往需要对包含着不同形状、颜色、纹理等特征的实体的图片进行分类，而本次实验需要对图像中记录的猴子进行细粒度分类。对于同类别物种的不同子类往往仅在耳朵形状、毛色等细微处存在差异(见图 1.1)，可谓“差之毫厘，谬以千里”。而且每张图片受视角、背景、遮挡等因素影响较大，算法需要从图像局部性做出对猴子类别的判断，所以需要模型对图像中的局部特征做出正确的预测。

图 1.1

在深度学习时代，训练一个准确度高的图像分类模型通常需要大量的数据以及算力。所以本次实验选择在 aistudio 上进行，项目地址：https://aistudio.baidu.com/aistudio/projectdetail/1250975。数据集包括 1400 张左右照片。数据集下载地址：https://www.datafountain.cn/datasets/102。

2 实验设计

2.1 数据探索

2.1.1 标签分布

本实验共有十种猴子标签，分别是 mantled_howler （鬃毛吼猴） 、patas_monkey （赤猴)、bald_uakari(白秃猴)、japanese_macaque(日本猕猴)、pygmy_marmoset(倭狨)、white_headed_capuchin(巴拿马白面卷尾猴)、silvery_marmoset(银色的狨猴)、 common_squirrel_monkey(松鼠猴)、black_headed_night_monkey(黑头夜猴)、nilgiri_langur(印度乌叶猴) 。图 2.1 为各种类猴子的示例图。

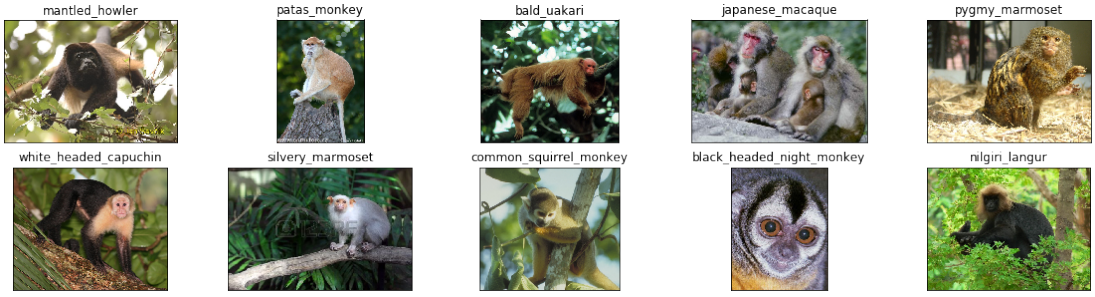

图 2.1

图 2.2 为训练集标签数量分布

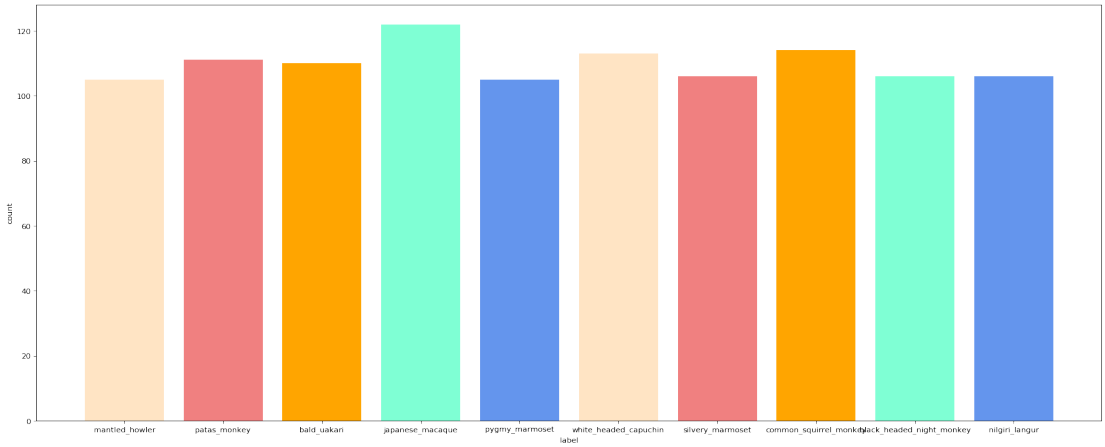

图 2.2

可以看到，10 类标签数量的分布比较均匀。

2.1.2 图像尺寸分布

图像尺寸分布如下，其中横轴与纵轴分别是图片的宽和高，图 2.3 为训练集上图片的尺寸分布，图 2.4 为验证集上图片的尺寸分布。

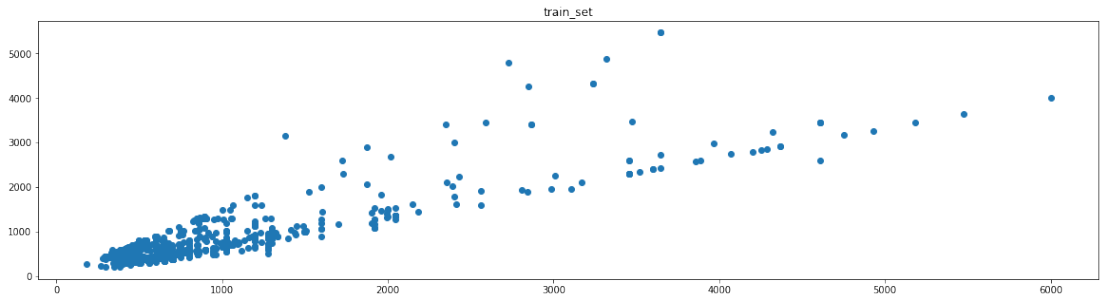

图 2.3

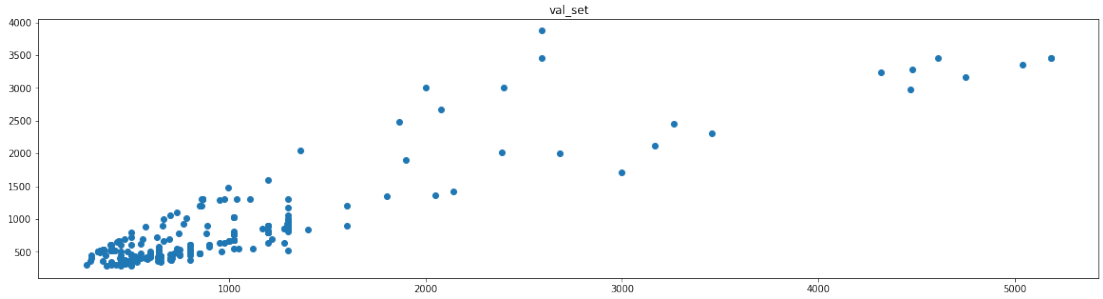

图 2.4

可以发现，训练数据集部分的训练集的尺寸分布与验证数据集尺寸分布十分相似，可以合理推测出验证集是从训练集中抽取出来的。训练集大量的数据的尺寸分布都集中在1000×1000以下，也有少部分图片尺寸偏离较远，但考虑到偏离较远的图片数量不大，而且通常深度学习模型用于训练的数据越多越好，所以在接下来的训练中可以使用全部的数据。

2.2 技术架构

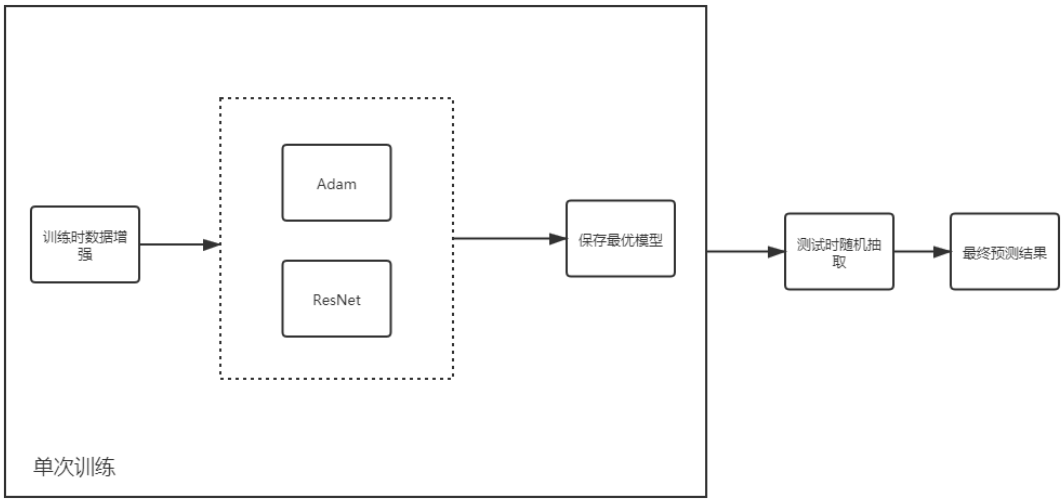

图 2.5

该架构分为训练模块和测试模块两个部分。对于训练模块，在数据输入时首先对训练数据进行随机的数据增强(Augmentation)，以在原有图像的基础上产生更多的模型没见过的新数据；对单次的训练过程，使用最常用的自适应 Adam[2]优化器，采用 ResNet 卷积神经网络模型；同时在每个周期结束时，在验证集中进行验证，取验证集准确率最大的模型并保存。在整个训练过程中，我想使用混合精度技术进行加速并降低显存的使用，但查阅官方文档后发现混合精度还只支持静态图，暂不支持动态图。训练结束后，我们会得到最优模型，从数据集中随机抽取 200 张图片进行测试，将模型参数导入并以准确率作为测试指标进行测试。

2.3 解决方案

2.3.1 数据预处理

CNN 在处理不同尺度的图片时多采用滑动窗口机制，基于网络模型的输入大小(本实验中为 224×224)作为滑动窗口。在猴子分类领域，猴子具体特征如毛色、脸型并不会因为比例缩放而消失，所以此次实验将图片直接 resize 到网络的输入大小 224×224。根据 2.1.2 图像尺寸分布可知，大部分图片的长宽比都在 1：1 到 2：1 之间，很少有图片超过 2：1 的长宽比，所以直接缩放能在保留图片大部分的具体特征的同时减少数据量，并且提升时间性能。

2.3.1 模型选择

AlexNet[3]、VGGNet[4]和 GoogleNet[5]已被证明在图像分类任务上可以获得良好的效果. 本次实验采用的是 ResNet, ResNet 是何凯明于 2015 年提出的 CNN 结构模型, 该方法以 152 层的网络模型在 ILSVRC2015 上获得第一名, 将错误率降低到了 3.75%. ResNet 的主要优点是可以利用更深层次的网络解决训练误差随网络层数的增加而增大的问题. 为了解决该问题, ResNet 对传统的平原网络结构进行了调整, 其关键结构是将基本的网络单元增加了一个恒等的快捷连接(如图 2. 5 所示). 图中 H(x)为理想映射, F(x)为残差映射, H(x)=F(x)+x. 通过将拟合目标函数 H(x)转变为拟合残差函数 F(x), 把输出变为拟合和输入的叠加, 使得网络对输出 H(x)与输入 x 之间的微小波动更加敏感.

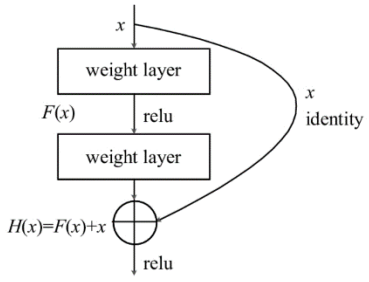

图 2. 6

本次实验构建的是 50 层 ResNet 结构模型, 本次实验利用深度学习框架 PaddlaPaddle 构建网络模型, 构建的模型共有约 590 万个参数, 网络中的神经元连接总数约为 10.68 亿. 相比于 AlexNet 的约 6000 万个参数, 该模型在网络层

数、需训练的参数个数上优势明显. 由于框架分类输出为 10 类, 故将最后全连接层的神经元个数设置为 10, 模型的输入为 224×224 大小的图片, 第一个卷积层 conv_1 的参数是 64 个 7×7 的卷积核, 卷积核的步长为 2. 每一个卷积层之后都设置 BatchNorm 层, 以增加模型的容纳能力. 激活函数使用 Relu, 并通过最大池化层 maxpool 进行下采样. 最后一层是 softmax 回归层, 用于输出图片被分为某一类的概率. softmax 回归是 logistic 回归的一般形式, 其数学公式为公式（2.1）所示, 其中 k 为类别数, 当 k=2 时 softmax 退化为 logistic. 本文实验中训练的模型有 10 个类别, 故 k=10.

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)}=1|x^{(i)};\theta) \\ p(y^{(i)}=2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)}=k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x_i}} \begin{bmatrix} e^{\theta_1^T x_i} \\ e^{\theta_2^T x_i} \\ \vdots \\ e^{\theta_k^T x_i} \end{bmatrix} \tag{2.1}$$

2.3.2 优化器选择

在深度学习种有很多优化器，包括 BGD、SGD、MBGD、Momentum、NAG、Adagrad、Adadelta、RMSprop、Adam。由于本次实验的猴子图像数据是稀疏的，所以选择使用自适用方法，即从 Adagrad, Adadelta, RMSprop, Adam 中选出最好的。RMSprop, Adadelta, Adam 在很多情况下的效果是相似的。Adam 就是在 RMSprop 的基础上加了 bias-correction 和 momentum，随着梯度变的稀疏，在测试后发现 Adam 比 RMSprop 效果会好。整体来讲，Adam 是最好的选择。很多论文里都会用 SGD，SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。如果需要更快的收敛，或者是训练更深更复杂的神经网络，还是需要用一种自适应的算法。图 2.7 为五种优化器在鸢尾花数据集中的效果对比。

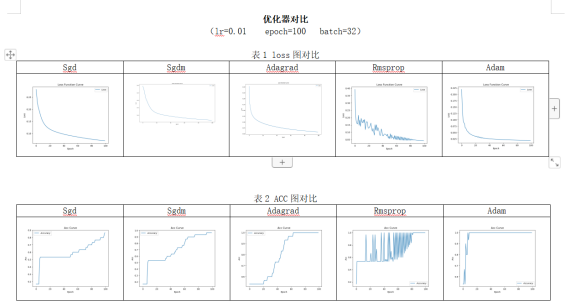

图 2.7

2.3.3 数据增强

数据增强是一个常用且非常有效的，扩充数据减少模型过拟合现象的技术。在计算机视觉领域，数据增强可以通过在训练时对输入图像进行随机且不改变其标签性质的拉伸、旋转、裁剪等操作，在不增加本地存储数据集大小的情况下，大大增加模型接受到的学习样本的多样性。本次实验只在训练时对数据进行随机数据增强操作。

图 2.7 是进行数据增强后的图像实例：

图 2.8

下表是本方案在训练时和测试时分别使用的数据增强操作：

表 2.1

图像增强操作	是否在训练时使用
Flip(翻转)	是
Crop(裁切)	是
Affine(仿射)	是
Blur(模糊)	是
Sharpen(锐化)	是
EdgeDetect(边缘检测)	是
GaussianNoise(高斯噪声)	是

Dropout(降低像素)	是
Invert(反转通道)	否
Contrast(对比度)	是
Grayscale(灰度覆盖)	否
ElasticTransformation(局部移动)	是
Distort(扭曲)	是

需要注意的是，颜色能很好的区分猴子的种类，因此需要谨慎并减少使用颜色变换类的增强操作。并且为了防止随机裁切导致图片的主要部分被裁切掉，一般以物体为中心随机切割。而且对图片进行随机模糊（Blur）可以降低图像的高频细节，让模型专注于对整体图像感知的学习而不是物体的局部细节。

2.2.4 训练及参数调试

损失函数选择交叉熵损失函数，优化方式选择 Adam 优化。初始学习率设置为 0.001，batch size 设置为 128，迭代周期为 300，采用准确率作为评价指标。

图 2.8 为最终的训练结果，图 2.9 为最终的验证结果

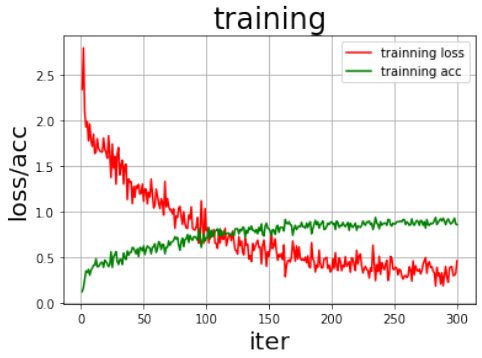

图 2.9

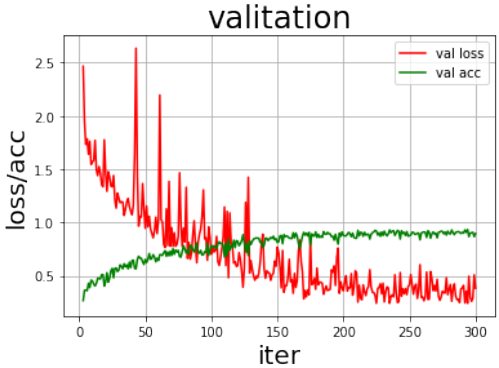

图 2.10

ResNet 模型在验证集准确率上最高达到了 94.1%

2.4 关键代码

2.4.1 图像数据归一化

将输入的所有图像缩放到 224×224，并调用 image_augment_imgauglib()函数对图像进行随机增强，最后将数据范围调整到[-1.0,1.0]之间

```
# 数据读取
def normalize_img(img, augment=''): # 图片数据归一化
    # 将图片尺寸缩放到 224x224
    img = cv2.resize(img, (224, 224))

    if augment == 'imgaugLib':  # 使用 imgaug 库进行数据增广
        img = image_augment_imgauglib(img)
    #plt.imshow(img)
    # 读入的图像数据格式是[H, W, C]。使用转置操作将其变成[C, H, W], 以适应 Paddle 的卷积操作对数据格式的要求
    img = np.transpose(img, (2,0,1))
    img = img.astype('float32')
    # 将数据范围调整到[-1.0, 1.0]之间
    img = img / 255.
    img = img * 2.0 - 1.0
    #img = np.mean(img, axis = 0).reshape((1, 28, 28))
    return img
```

2.4.2 图像增广函数

image_augment_imgauglib()函数是基于 imgaug 库所构建的，函数中随机选择 Flip、Crop、Blur 等操作对图像进行随机增强，使得每次训练时，模型看到的图像都不一样。

```
#图像增广
def image_augment_imgauglib(img):
    img = np.array([img])
    ia.seed(random.randint(0, 10000))
    sometimes = lambda aug: iaa.Sometimes(0.5, aug)
    seq = iaa.Sequential(
        [
            iaa.Fliplr(0.5),
            iaa.Flipud(0.2),
```

```
            sometimes(iaa.Crop(percent=(0, 0.1))),
            sometimes(iaa.Affine(scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},rotate=(-45, 45),shear=(-16, 16),order=[0, 1],cval=(0, 255),mode=ia.ALL)),
            iaa.SomeOf((0, 5),
                [
                    sometimes(iaa.Superpixels(p_replace=(0, 1.0),n_segments=(20, 200))),
                    iaa.OneOf([iaa.GaussianBlur((0, 3.0)),iaa.AverageBlur(k=(2, 7)),iaa.MedianBlur(k=(3, 11)),]),
                    iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)),
                    iaa.Emboss(alpha=(0, 1.0), strength=(0, 2.0)),
                    sometimes(iaa.OneOf([iaa.EdgeDetect(alpha=(0, 0.7)),iaa.DirectedEdgeDetect(alpha=(0, 0.7), direction=(0.0, 1.0)),])),
                    iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5),
                    iaa.OneOf([iaa.Dropout((0.01, 0.1), per_channel=0.5),iaa.CoarseDropout((0.03, 0.15), size_percent=(0.02, 0.05),per_channel=0.2),]),
                    # iaa.Invert(0.05, per_channel=True),
                    iaa.Add((-10, 10), per_channel=0.5),
                    iaa.Multiply((0.5, 1.5), per_channel=0.5),
                    iaa.LinearContrast((0.5, 2.0), per_channel=0.5),
                    # iaa.Grayscale(alpha=(0.0, 1.0)),
                    sometimes(iaa.ElasticTransformation(alpha=(0.5, 3.5), sigma=0.25)),
                    sometimes(iaa.PiecewiseAffine(scale=(0.01, 0.05)))
                ],
                random_order=True
            )
        ],
        random_order=True
    )
    img = seq(images=img)
    img = img[0]
    return img
```

2.4.3ResNet 模型

本次实验采用的是 50 层 ResNet 模型，第一个卷积层 conv_1 的参数是 64 个 7×7 的卷积核，卷积核的步长为 2。每一个卷积层之后都设置 BatchNorm 层，以增加模型的容纳能力. 激活函数使用 Relu，并通过最大池化层 maxpool 进行下采样。最后一层是 softmax 回归层，用于输出图片被分为某一类的概率。每个残

差块会对输入图片做三次卷积，然后跟输入图片进行短接，如果残差块中第三次卷积输出特征图的形状与输入不一致，则对输入图片做 1x1 卷积，将其输出形状调整成一致。在卷积层的后面加上 BatchNorm 是为了提升数值稳定性。BatchNorm 层和残差块的定义没有在此贴出。

```
# 定义 ResNet 模型
class ResNet(fluid.dygraph.Layer):
    def __init__(self, layers=50, class_dim=10):
        super(ResNet, self).__init__()
        self.layers = layers
        supported_layers = [50, 101, 152]
        assert layers in supported_layers, \
            "supported layers are {} but input layer is {}".format(supported_layers, layers)
        depth=[3,4,6,3]
        # 残差块中使用到的卷积的输出通道数
        num_filters = [64, 128, 256, 512]
        # ResNet 的第一个模块，包含 1 个 7x7 卷积，后面跟着 1 个最大池化层
        self.conv = ConvBNLayer(
            num_channels=3,
            num_filters=64,
            filter_size=7,
            stride=2,
            act='relu')
        self.pool2d_max = Pool2D(
            pool_size=3,
            pool_stride=2,
            pool_padding=1,
            pool_type='max')
        # ResNet 的第二到第五个模块 c2、c3、c4、c5
        self.bottleneck_block_list = []
        num_channels = 64
        for block in range(len(depth)):
            shortcut = False
            for i in range(depth[block]):
                bottleneck_block = self.add_sublayer(
                    'bb_%d_%d' % (block, i),
                    BottleneckBlock(
                        num_channels=num_channels,
                        num_filters=num_filters[block],
                        stride=2 if i == 0 and block != 0 else 1, # c3、c4、c5 将会在第一个残差块使用 stride=2；其余所有残差块 stride=1
```

```
                            shortcut=shortcut))
                num_channels = bottleneck_block._num_channels_out
                self.bottleneck_block_list.append(bottleneck_block)
                shortcut = True
        # 在 c5 的输出特征图上使用全局池化
        self.pool2d_avg = Pool2D(pool_size=7, pool_type='avg', global_poolin
g=True)
        # stdv 用来作为全连接层随机初始化参数的方差
        stdv = 1.0 / math.sqrt(2048 * 1.0)
        # 创建全连接层，输出大小为类别数目
        self.out = Linear(input_dim=2048, output_dim=class_dim,
                          param_attr=fluid.param_attr.ParamAttr(
                              initializer=fluid.initializer.Uniform(-
stdv, stdv)))

    def forward(self, inputs):
        y = self.conv(inputs)
        y = self.pool2d_max(y)
        for bottleneck_block in self.bottleneck_block_list:
            y = bottleneck_block(y)
        y = self.pool2d_avg(y)
        y = fluid.layers.reshape(y, [y.shape[0], -1])
        y = self.out(y)
        return y
```

2.4.4 训练代码

训练时通过交叉熵损失函数计算 loss，每一轮训练完进行方向传播、更新权重、清除梯度等操作。Epoch_num 设为 300，训练所花费的时间很长，所以要 Early stopping。如 Geoff Hinton 所说：“Early Stopping 是美好的免费午餐”。所以必须在训练的过程中时常在验证集上监测误差，在验证集上如果损失函数不再显著地降低，那么应该提前结束训练。验证误差的代码与训练的代码类似，这里就不贴出来了。需要注意的是在验证时需要进行 model.eval()操作，进入到验证模式，防止影响权重。

```
for epoch in range(epoch_num):
            for batch_id, data in enumerate(train_loader()):
                x_data, y_data = data
                img = fluid.dygraph.to_variable(x_data)
                label = fluid.dygraph.to_variable(y_data)
                # 运行模型前向计算，得到预测值
```

```
                    logits= model(img)
                    # 进行 loss 计算
                    loss = fluid.layers.softmax_with_cross_entropy(logits, label
    )
                    avg_loss = fluid.layers.mean(loss)
                    #进行 accuracy 计算
                    acc = fluid.layers.accuracy(logits,label)
                    if batch_id % 10 == 0:
                        print("epoch: {}, batch_id: {}, loss is: {}, accuracy is
     {}".format(epoch, batch_id, avg_loss.numpy(),acc.numpy()))
                        train_iter=train_iter+1
                        train_iters.append(train_iter)
                        train_loss.append(avg_loss.numpy())
                        train_accuracy.append(acc.numpy())
                    # 反向传播，更新权重，清除梯度
                    avg_loss.backward()
                    opt.minimize(avg_loss)
                    model.clear_gradients()
```

3.总结

3.1 结论

最后验证集的准确率最高达到了 94.1%，所以此次实验产生的模型能够很高程度上预测猴子的种类，帮助人们更好地识别不同的猴子。图 3.1 为随机抽取的 16 张图片进行的预测。

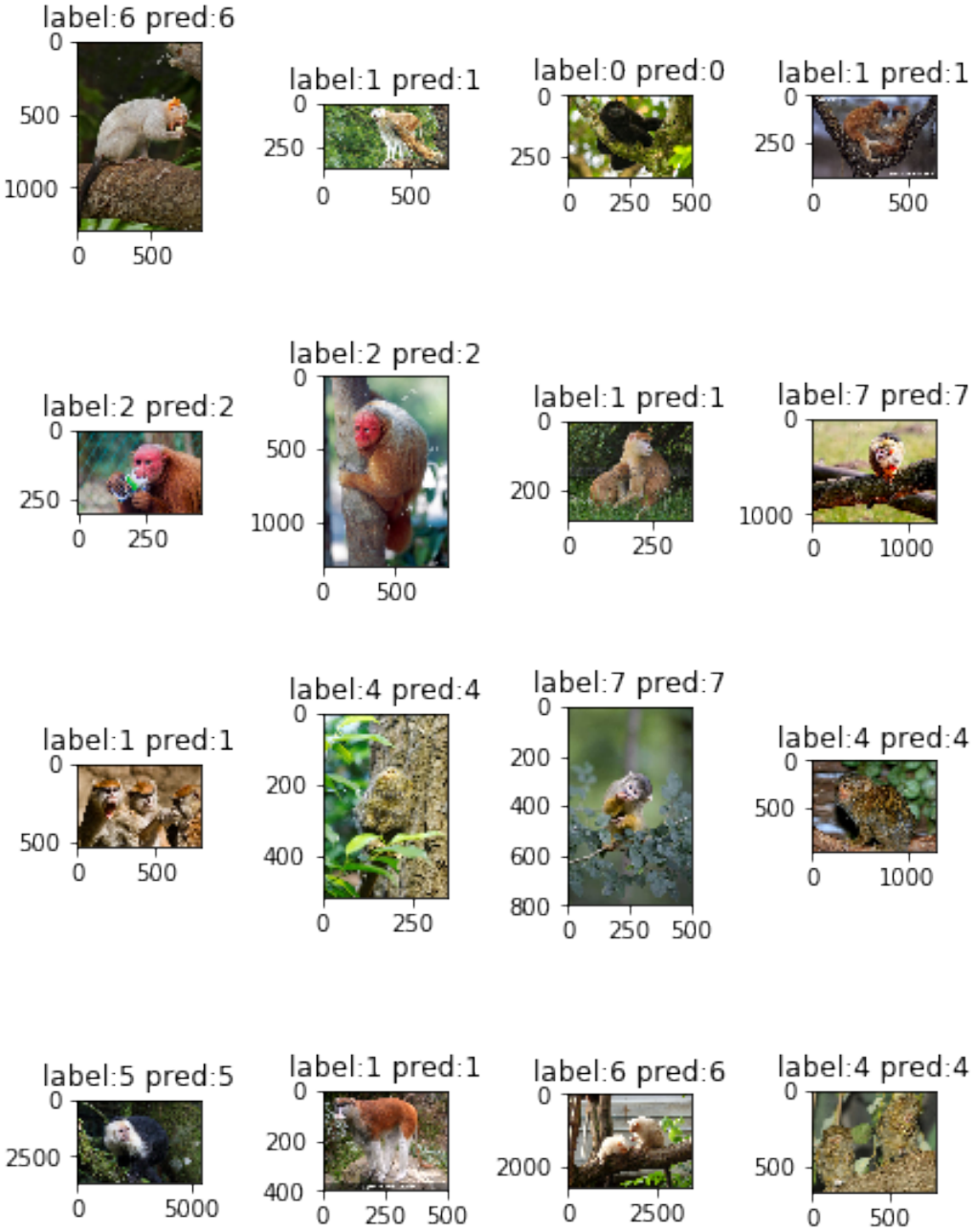

图 3.1

3.2 思考与后续改进

本次实验选用的是 ResNet 模型，虽然 ResNet 有很多优点如：使前馈/反馈传播算法顺利进行，结构更加简单，恒等映射增加基本不会降低网络的性能。但是，我在训练时发现 ResNet 训练的时间很长，平均每一轮需要耗费 45 秒，导致 300 轮的训练要将近 3 个小时，而且有时候设置的 batch_size 过大会导致显存爆炸，只能通过重启实验环境来清理缓存。所以以后可以选择更好的模型，比如谷歌的 EfficientNet[6]模型等，这些模型也许用起来会比较轻便，能够兼顾性能和准确率。

因为训练的时间过长，我在 paddlepaddle 官方文档里查询到可以通过混合精度来加速模型的训练，但根据文档的代码写出后，会产生错误，无法产生加速，后来在 paddlepaddle 的 github 的 issue 里发现 paddlepaddle 还没有支持动态图的混合精度加速，所以等动态图的混合精度支持后，可以尝试使用一下。

准确率还没有达到 95%以上的水平，可能是因为训练的轮数 300 轮还不够多，以后可以尝试 400 或者更多轮的训练。

本次实验中的学习率是固定值 0.001，对于固定的学习率，太低则需要太多时间来让模型收敛，太高则模型难以收敛；而我们往往希望在训练的初期模型能快速收敛到局部最小值附近，而在后期模型能在一个较小的学习率下充分收敛。所以需要进行学习率调整。在网上查询后发现，以后可以使用余弦退火 (Cosine Annealing)[7]学习率衰减策略进行学习率的动态调整，和 WarmUp 策略在模型训练的开始，让学习率通过某种方式（e.g.线性）从 0 逐渐增加至优化器的最大学习率，随后按照既定的学习率调整策略进行学习。

在优化器的选择上，本次实验只选择了最常用的 Adam 优化器，没有选择更加适用于图像分类的优化器，在查阅资料后发现，有对 Adam 进行提升的 AdamP[8]优化器，该优化器从视觉任务到语言任务都表现出了比 Adam 更佳的性能，因此后续可以将 Adam 替换为 AdamP。

最后，感谢老师和两位助教的辛勤付出，让我能够更深入地了解深度学习的相关知识，感谢百度 aistudio 让我能够每天免费使用 10 小时 gpu。

参考文献

[1]. He KM, Zhang XY, Ren SQ, et al. Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV, USA. 2016. 770–778.

[2]. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[3]. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 2017, 60(6): 84-90.

[4]. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. Computer Science, 2014, 1-14.

[5]. Szegedy C, Liu W, Jia YQ, et al. Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition. Boston, MA, USA. 2015. 1–9.

[6]. Tan M, Le Q V. Efficientnet: Rethinking model scaling for convolutional neural networks[J]. arXiv preprint arXiv:1905.11946, 2019.

[7]. Huang G, Li Y, Pleiss G, et al. Snapshot ensembles: Train 1, get m for free[J]. arXiv preprint arXiv:1704.00109, 2017.

[8]. Heo, B., Chun, S., Oh, S. J., Han, D., Yun, S., Uh, Y., & Ha, J. W. (2020). Slowing down the weight norm increase in momentum-based optimizers. arXiv preprint arXiv:2006.08217.