CNN高级实训任务一实验报告

1.任务描述

- 卷积神经网络（CNN）因为其能够自动抽取图像的浅层到深层的特征，所以在近几年有许多应用。
- 我们尝试用一种CNN来对一个数据集进行图像分类。
- 需要提交博客报告以及GitHub代码仓库。
- 可选的CNN：AlexNet、GoogleNet、ResNet（三选二）。
- 可选的数据集：MNIST手写数据集、CIFAR-10彩色图像数据集。
- 可选深度学习框架：Tensorflow、PyTorch、Keras。
- 提交结果：项目报告、答辩幻灯片、相关代码和测试用例。

2.本次实验工具环境配置准备：

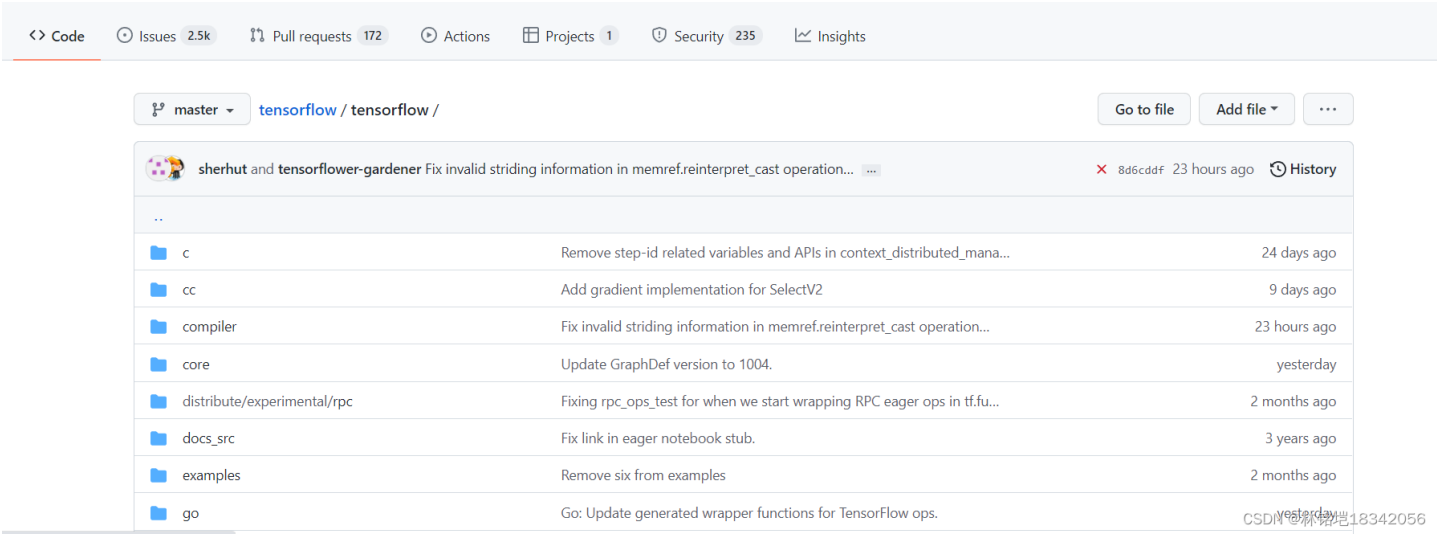
（1）任务选择

```
CNN：AlexNet，ResNet  ；
数据集：MNIST手写数据集；
深度学习框架：Tensorflow
```

（2）安装VS ide,同时配置安装TensorFlow

1.VS安装教程：教程

2.去GitHub的TensorFlow首页下载安装：Tensorflow



(3)下载数据集

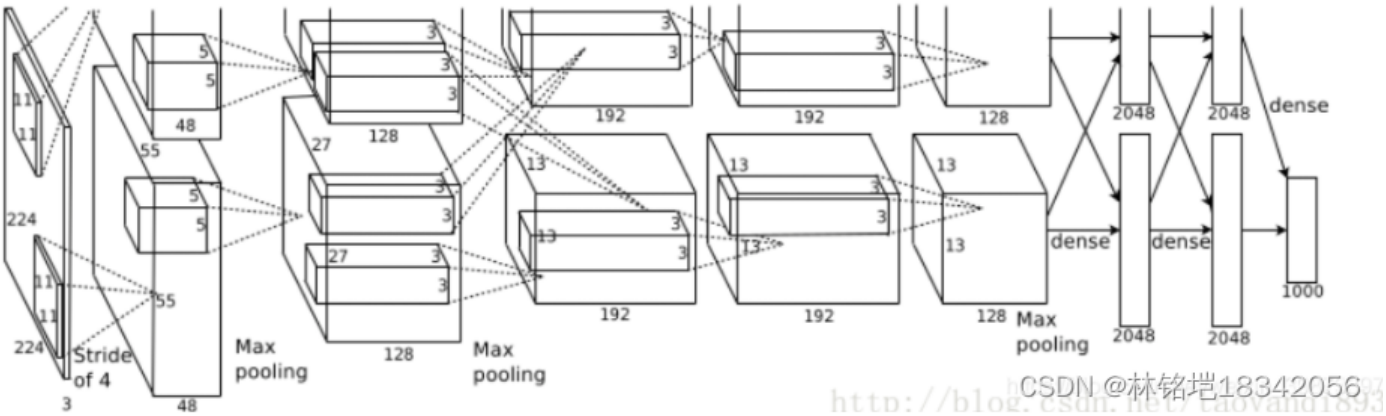
通过TensorFlow库里自带的教程函数进行下载。

```
from tensorflow.examples.tutorials.mnist import input_data
```

3.实验任务：

（1）AlexNet

1. AlexNet网络结构



AlexNet是2012年ILSVRC比赛的冠军，它的出现直接打破了沉寂多年的图片识别领域（在1998年出现LeNet-5网络一直占据图片识别的领头地位），给该领域带来了新的契机，并一步步发展至今，甚至打败了人类的识别精确度，可惜的是2017年的ILSVRC举办方宣布从2018年起将取消该比赛，因为目前的神经网络精确度已经达到跟高的程度了。但深度学习的步伐不会停止，人们将在其他方面进行深入的研究。

AlexNet是神经网络之父Hinton的学生Alex Krizhevsky开发完成，它总共有8层，其中有5个卷积层，3个全链层，附上最经典的AlexNet网络架构图，如下。Alex在他的论文中写到，他在处理图片的时候使用了两个GPU进行计算，因此，从图中看出，在卷积过程中他做了分组的处理，但是由于硬件资源问题，我们做的Alex网络是使用一个CPU进行计算的，但原理和他的一样，只是计算速度慢一点而已，对于大多数没有性能优良的GPU的人来说，用我们搭建好的网络，完全可以使用家用台式机进行训练。

2 .Tensorflow实现AlexNet

- 初始化学习率，每轮学习多少次， #进行的学习周期数

```
learning_rate = 0.02
batch_size = 100
epochs = 10
dropout = 0.8
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])
keep_prob = tf.placeholder(tf.float32)
```

- 生成卷积,池化操作,归一化

```
def Convolution(_name, _input, _w, _b):
    return tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(_input, _w, strides=[1,1,1,1], padding="SAME'
def Pool(_name, _input, k):
    return tf.nn.max_pool(_input, ksize=[1,k,k,1], strides=[1,k,k,1], padding="SAME", name=_nam
def Normalize(_name, _input, lsize=4):
    return tf.nn.lrn(_input, lsize, bias=1.0, alpha=0.001 / 9.0, beta=0.75, name=_name)
```

- 构建AlexNet模型

```
# AlexNet网络结构
<p class="mume-header " id="alexnet网络结构"></p>

class MineAlexNet(nn.Module):
    def __init__(self, num_classes=2):
        super(MineAlexNet, self).__init__()

        self.features=nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3,stride=2),
        )
        self.avgpool=nn.AdaptiveAvgPool2d((6,6))
        sefl.classifier=nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )
        #softmax
        self.logsoftmax = nn.LogSoftmax(dim=1)
    def forward(self, x):
        x=self.features(x)
        x=self.avgpool(x)
        x=x.view(x.size(0), 256*6*6)
        x=self.classifier(x)
        x=self.logsoftmax(x)
        return x
```

- 构建模型，并锁定输出

```
pred = alexnet(x, weights, bias, keep_prob)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=pred))
optimizer = tf.train.AdadeltaOptimizer(learning_rate).minimize(cost)
```

3.运行结果：

epochs = 5：

```
To enable them in other operations, rebuild TensorFlow with the
第 0 轮的熵损失:226463.148949
第 1 轮的熵损失:154068.247983
第 2 轮的熵损失:133833.039972
第 3 轮的熵损失:116202.628466
第 4 轮的熵损失:103827.130639
命中率： 0.21875
```

epochs = 10：

```
 Network Library (oneDNN) to use the following CPU instructions
To enable them in other operations, rebuild TensorFlow with the
第 0 轮的熵损失:245511.173097
第 1 轮的熵损失:153720.493509
第 2 轮的熵损失:124578.362812
第 3 轮的熵损失:103685.710085
第 4 轮的熵损失:87857.232635
第 5 轮的熵损失:76368.801626
第 6 轮的熵损失:67479.012955
第 7 轮的熵损失:60067.526065
第 8 轮的熵损失:54745.562972
第 9 轮的熵损失:50419.425792
命中率： 0.58203125
PS D:\python\Python37_64>
```

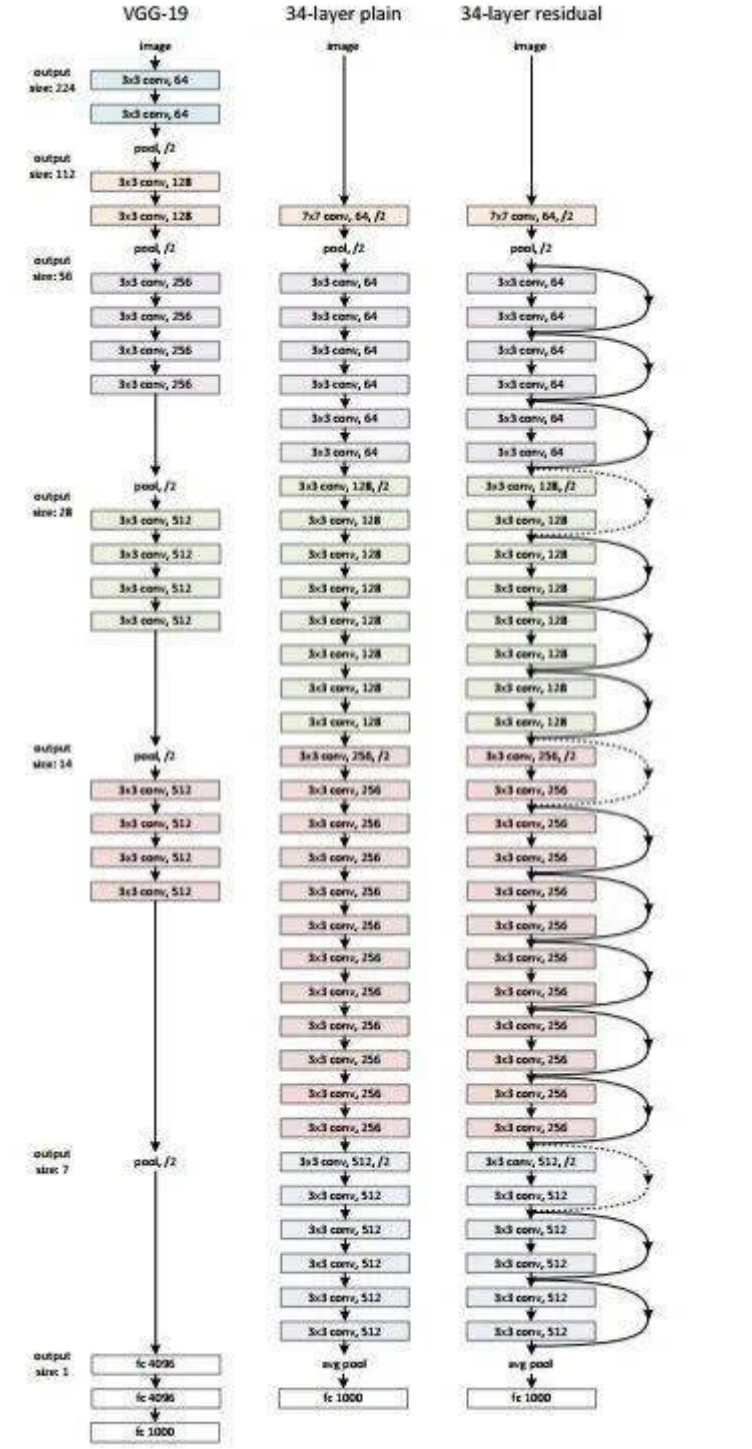

每一轮的熵损失逐渐变小，轮数增加，准确率上升.

（2）Resnet

1.ResNet

- 残差网络（ResNet）是著名的深度学习模型之一，由任少清、何开明、孙健和张翔宇在他们的论文中引入。ResNet 模型是迄今为止广泛流行和最成功的深度学习模型之一。
- 残差块
  随着这些残差（Residual）块的引入，训练非常深的网络时面临的问题得到了缓解，ResNet 模型由这些块组成。随着这些残差块的引入，训练非常深的网络时面临的问题得到了缓解，ResNet 模型由这些块组成。在上图中，我们可以注意到的第一件事是跳过模型的某些层的直接连接。这种连接称为“跳过连接”，是残差块的核心。由于存在这种跳过连接，输出是不相同的。如果没有跳过连接，输入‘X 将乘以层的权重，然后添加一个偏置项。然后是激活函数 f，我们得到输出为 H(x)。H(x)=f(wx+b) 或 H(x)=f(x)现在引入了新的跳过连接技术，输出 H(x) 更改为H(x)=f(x)+x.但是输入的维度可能与输出的维度不同，这可能发生在卷积层或池化层中。因此，这个问题可以用这两种方法来处理：用跳过连接填充零以增加其维度。
- ResNet 的架构

架构中有一个 34 层的普通网络，其灵感来自 VGG-19，其中添加了快捷连接或跳过连接。这些跳过连接或残差块将架构转换为残差网络，如下图所示。



- 将 ResNet 与 Keras 结合使用：

Keras 是一个开源深度学习库，能够在 TensorFlow 上运行。让我们从零开始构建 ResNet：

2.构建 ResNet网络。

最核心的部分在__call__函数中包括：

- 为了加速GPU运算，将输入由NHWC转换成NCHW。
- 首次卷积运算。

- 根据ResNet版本判断是否要做batch norm。
- 首次pooling。
- 堆叠block。
- 最终的pooling（代码中用表现更好的reduce_mean替代）。
- 最终的全连接层。

用于输入处理的函数：

```
def process_record_dataset(dataset, ...)
def get_synth_input_fn(height, width, num_channels, num_classes, ...)
def image_bytes_serving_input_fn(image_shape, dtype=tf.float32)
def override_flags_and_set_envars_for_gpu_thread_pool(flags_obj)
```

用于运行模型的培训/评估/验证循环的函数：

输入处理

```
def learning_rate_with_decay(...)
def resnet_model_fn(features, labels, mode, model_class, ...)
def resnet_main(...)
def define_resnet_flags(resnet_size_choices=None)

dataset = dataset.apply(
    tf.contrib.data.map_and_batch(
        lambda value: parse_record_fn(value, is_training, dtype),
        batch_size=batch_size,
        num_parallel_batches=num_parallel_batches,
        drop_remainder=False))
```

运行循环：
数据输入已经确定，模型也已经定义完成，但是我们离模型训练还差关键的一步：loss的定义。文件中的函数resnet_model_fn则刚好包含了这一实现。具体的实现为：

获取模型输出：

```
logits = model(features, mode == tf.estimator.ModeKeys.TRAIN)
```

计算交叉熵：

```
cross_entropy = tf.losses.sparse_softmax_cross_entropy(
      logits=logits, labels=labels)
```

计算l2 loss：

```
l2_loss = weight_decay * tf.add_n(
      [tf.nn.l2_loss(tf.cast(v, tf.float32)) for v in tf.trainable_variables()
       if loss_filter_fn(v.name)])
```

获得最终loss：

loss = cross_entropy + l2_loss

函数resnet_model_fn的作用其实是构建EstimatorSpec，loss其实是EstimatorSpec的一部分。而estimator的实例化则在函数resnet_main内部，这也是真正的训练与验证过程的所在。在Cifar10的例子中，也是直接调用了resnet_main。

Cifar10：

```
result = resnet_run_loop.resnet_main(
    flags_obj, cifar10_model_fn, input_function, DATASET_NAME,
    shape=[HEIGHT, WIDTH, NUM_CHANNELS])
```

在resnet_main函数内可以找到训练、验证与导出的调用代码：

训练：

```
classifier.train(input_fn=lambda: input_fn_train(num_train_epochs),
                                  hooks=train_hooks,max_steps=flags_obj.max_train_steps)
```

验证：

```
eval_results = classifier.evaluate(input_fn=input_fn_eval,steps=flags_obj.max_train_steps)
```

导出：

```
classifier.export_savedmodel(flags_obj.export_dir, input_receiver_fn,
                                    strip_default_attrs=True)
```

运行结果：

epochs = 1：

```
1epoch train loss : 19163.32
1epoch test loss : 20175.613
2022-01-08 18:58:31.523905: W tensorflow/core/framework/cpu_alloca
测试精度 : 0.8677
PS C:\Users\LMK\Desktop>
```

epochs = 3：

```
0epoch train loss : 18730.3
0epoch test loss : 19778.318
1
2
2022-01-08 18:48:14.825744: W tensorflow/core/framew
测试精度 : 0.9347
```

epochs = 6：

```
1epoch train loss : 17396.656
1epoch test loss : 21637.652
1
2
3
4
5
2022-01-08 19:37:58.296997: W tensorflow/core/framework/cpu_allocator_impl.cc:82]
测试精度 : 0.9541
PS C:\Users\LMK\Desktop>
```


可见周期越多，准确率越大。

Alexnet与Resnet比较

在训练相同周期数量的情况下，Resnet的准确率比Alexnet高，但是与之相对，Resnet的单轮运行时间，比Alexnet长。