# 人工智能上机实验报告

|  |  |  |
| --- | --- | --- |
| 课程名称：人工智能 | 班级：计科1902 | 实验日期：2021 |
| 姓名： 徐超信 | 学号：1935010102 | 指导教师： 竺乐庆 |
| 实验序号： 1 | | 实验成绩： |
| 一、实验名称  深度学习模型初探 | | |
| 二、实验目的及要求 1、了解Keras,熟悉使用Keras时的深度学习基础概念；  2、了解Keras的主要层，创建核心模型；  3、掌握基于深度学习实现基本应用的流程。 | | |
| 三、实验环境  tensorflow+ keras  **pip install numpy**  **pip install pandas**  pip install tensorflow -i https://pypi.tuna.tsinghua.edu.cn/simple  pip install keras | | |
| 四、实验内容 1根据给定的例子程序，搭建LeNet、AlexNet、GoogleNet、VGG16等模型，  对手写数字数据集MNIST进行验证性实验；  2要求完成模型的搭建、训练、验证，并用自己的画图板上写的20个数字的手写数字样本进行测试，  3从识别精度、训练时间、识别时间等方面对3个以上网络进行比较。  4展示训练过程可视化效果，与传统全连接BP网络进行性能比较，以及给出数据增强前后的性能比较。 | | |
| 五、算法描述及实验步骤  **卷积神经网络原理：**  卷积神经网络能过参数复用，使卷积层的参数数量比全连接层的参数数量大幅度减小，从而使搭建更深的神经网络成为可能。将大的问题分解成形状结构相同的子问题，并且不断递归地分解，直到子问题规模小到可以直接求解。  **经典CNN结构描述：**  **1. LeNet-5**  LeNet-5是一种用于手写体字符识别的非常高效的卷积神经网络。LeNet神经网络由深度学习三巨头之一的Yan LeCun提出，他同时也是卷积神经网络 (CNN，Convolutional Neural Networks)之父。LeNet主要用来进行手写字符的识别与分类，并在美国的银行中投入了使用。LeNet的实现确立了CNN的结构，现在神经网络中的许多内容在LeNet的网络结构中都能看到，例如卷积层，Pooling层，ReLU层。虽然LeNet早在20世纪90年代就已经提出了，但由于当时缺乏大规模的训练数据，计算机硬件的性能也较低，因此LeNet神经网络在处理复杂问题时效果并不理想。虽然LeNet网络结构比较简单，但是刚好适合神经网络的入门学习。  网络解析（一）：LeNet-5详解  **2. AlexNet**  AlexNet是2012年[ImageNet](https://baike.baidu.com/item/ImageNet/17752829)竞赛冠军获得者Hinton和他的学生Alex Krizhevsky设计的。也是在那年之后，更多的更深的神经网络被提出。AlexNet将LeNet的思想发扬光大，把CNN的基本原理应用到了很深很宽的网络中。AlexNet主要使用到的新技术点如下：  （1）成功使用ReLU作为CNN的激活函数，并验证其效果在较深的网络超过了Sigmoid，成功解决了Sigmoid在网络较深时的梯度弥散问题。虽然ReLU激活函数在很久之前就被提出了，但是直到AlexNet的出现才将其发扬光大。  （2）训练时使用Dropout随机忽略一部分神经元，以避免模型过拟合。Dropout虽有单独的论文论述，但是AlexNet将其实用化，通过实践证实了它的效果。在AlexNet中主要是最后几个全连接层使用了Dropout。  （3）在CNN中使用重叠的最大池化。此前CNN中普遍使用平均池化，AlexNet全部使用最大池化，避免平均池化的模糊化效果。并且AlexNet中提出让步长比池化核的尺寸小，这样池化层的输出之间会有重叠和覆盖，提升了特征的丰富性。  （4）提出了LRN层，对局部神经元的活动创建竞争机制，使得其中响应比较大的值变得相对更大，并抑制其他反馈较小的神经元，增强了模型的泛化能力。  （5）使用CUDA加速深度卷积网络的训练，利用GPU强大的并行计算能力，处理神经网络训练时大量的矩阵运算。AlexNet使用了两块GTX 580 GPU进行训练，单个GTX 580只有3GB显存，这限制了可训练的网络的最大规模。因此作者将AlexNet分布在两个GPU上，在每个GPU的显存中储存一半的神经元的参数。因为GPU之间通信方便，可以互相访问显存，而不需要通过主机内存，所以同时使用多块GPU也是非常高效的。同时，AlexNet的设计让GPU之间的通信只在网络的某些层进行，控制了通信的性能损耗。  （6）数据增强，随机地从256\*256的原始图像中截取224\*224大小的区域（以及水平翻转的镜像），相当于增加了2\*(256-224)^2=2048倍的数据量。如果没有数据增强，仅靠原始的数据量，参数众多的CNN会陷入过拟合中，使用了数据增强后可以大大减轻过拟合，提升泛化能力。进行预测时，则是取图片的四个角加中间共5个位置，并进行左右翻转，一共获得10张图片，对他们进行预测并对10次结果求均值。同时，AlexNet论文中提到了会对图像的RGB数据进行PCA处理，并对主成分做一个标准差为0.1的高斯扰动，增加一些噪声，这个Trick可以让错误率再下降1%。    实验步骤：  1、载入数据(训练数据用于训练网络模型，测试数据用于测试网络模型；)  2、搭建网络模型；  3、设置训练参数，如损失、优化算法、学习率等；  4、设置批大小、训练轮次进行模型训练；  5、评估模型性能和测试；  6、第一步增加数据增强后比较性能，比较*使用不同损失函数、不同优化算法、是否使用dropout层*的差别；  7、用tensorboard或Jupyter notebook对训练过程进行可视化 | | |
| 六、调试过程及实验结果  详细记录程序在调试过程中出现的问题及解决方法。  记录程序执行的结果。  VGGNet/AlextNet的损失曲线的收敛    手写测试: 用**windows自带的画图软件**，鼠标手写数字，并保存成png图片 | | |
| 七、总结  对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。 | | |
| 八、附录  源程序（核心代码）清单或使用说明书，可另附页。   * Alex 模型   from keras.models import Sequential  from keras.layers import Dense, Flatten, Dropout  from keras.layers.convolutional import Conv2D, MaxPooling2D  from keras.utils.np\_utils import to\_categorical  import numpy as np  seed = 7  np.random.seed(seed)  # 创建模型序列  model = Sequential()  #第一层卷积网络，使用96个卷积核，大小为11x11步长为4， 要求输入的图片为227x227， 3个通道，不加边，激活函数使用relu  model.add(Conv2D(96, (11, 11), strides=(1, 1), input\_shape=(28, 28, 1), padding='same', activation='relu',  kernel\_initializer='uniform'))  # 池化层  model.add(MaxPooling2D(pool\_size=(3, 3), strides=(2, 2)))  # 第二层加边使用256个5x5的卷积核，加边，激活函数为relu  model.add(Conv2D(256, (5, 5), strides=(1, 1), padding='same', activation='relu', kernel\_initializer='uniform'))  #使用池化层，步长为2  model.add(MaxPooling2D(pool\_size=(3, 3), strides=(2, 2)))  # 第三层卷积，大小为3x3的卷积核使用384个  model.add(Conv2D(384, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel\_initializer='uniform'))  # 第四层卷积,同第三层  model.add(Conv2D(384, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel\_initializer='uniform'))  # 第五层卷积使用的卷积核为256个，其他同上  model.add(Conv2D(256, (3, 3), strides=(1, 1), padding='same', activation='relu', kernel\_initializer='uniform'))  model.add(MaxPooling2D(pool\_size=(3, 3), strides=(2, 2)))  model.add(Flatten())  model.add(Dense(4096, activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(4096, activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(10, activation='softmax'))  model.compile(loss='categorical\_crossentropy', optimizer='sgd', metrics=['accuracy'])  model.summary()   * VGGNet 模型   from keras import Sequential  from keras.layers import Conv2D,MaxPooling2D,Flatten,Softmax,Activation,Dense  from keras.utils.np\_utils import to\_categorical  from keras.datasets import mnist  from sklearn.metrics import recall\_score,f1\_score,precision\_score  步骤二：加载数据集  data=mnist.load\_data()  (X\_train,Y\_train),(X\_test,Y\_test)=data  X\_train=X\_train.reshape(-1,28,28,1) #这里我们使用的是黑白图片  X\_test=X\_test.reshape(-1,28,28,1)  Y\_train=to\_categorical(Y\_train,num\_classes=10)  Y\_test=to\_categorical(Y\_test,num\_classes=10)  1  2  3  4  5  6  步骤三：搭建模型  def VGG(X,Y):  model=Sequential()  #layer\_1  model.add(Conv2D(64,(3,3),strides=(1,1),input\_shape=X.shape[1:],padding='same',data\_format='channels\_last',activation='relu',kernel\_initializer='uniform'))  model.add(Conv2D(64,(3,3),strides=(1,1),padding='same',data\_format='channels\_last',kernel\_initializer='uniform',activation='relu'))  model.add(MaxPooling2D((2,2)))  #layer\_2  model.add(Conv2D(128,(3,3),strides=(1,1),padding='same',data\_format='channels\_last',activation='relu',kernel\_initializer='uniform'))  model.add(Conv2D(128,(2,2),strides=(1,1),padding='same',data\_format='channels\_last',activation='relu',kernel\_initializer='uniform'))  model.add(MaxPooling2D((2,2)))  #layer\_3  model.add(Conv2D(256,(3,3),strides=(1,1),padding='same',data\_format='channels\_last',activation='relu'))  model.add(Conv2D(256,(3,3),strides=(1,1),padding='same',data\_format='channels\_last',activation='relu'))  model.add(Conv2D(256, (1, 1), strides=(1, 1), padding='same', data\_format='channels\_last', activation='relu'))  model.add(MaxPooling2D((2,2)))  #layer\_4  model.add(Conv2D(512,(3,3),strides=(1,1),padding='same',data\_format='channels\_last',activation='relu'))  model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', data\_format='channels\_last', activation='relu'))  model.add(Conv2D(512, (1,1), strides=(1, 1), padding='same', data\_format='channels\_last', activation='relu'))  model.add(MaxPooling2D((2,2)))  #layer\_5  model.add(Conv2D(512,(3,3),strides=(1,1),padding='same',data\_format='channels\_last',activation='relu'))  model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', data\_format='channels\_last', activation='relu'))  model.add(Conv2D(512, (1,1), strides=(1, 1), padding='same', data\_format='channels\_last', activation='relu'))  model.add(MaxPooling2D((2,2)))  model.add(Flatten()) #拉平  model.add(Dense(4096,activation='relu'))  model.add(Dense(4096,activation='relu'))  model.add(Dense(1000,activation='relu'))  model.add(Dense(10,activation='softmax'))  model.summary()  model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['accuracy'])  return model   * GoogleNet 模型   import tensorflow as tf  import numpy as np  from tensorflow import keras  import os  class ConvBNRelu(keras.Model):    def \_\_init\_\_(self, ch, kernelsz=3, strides=1, padding='same'):  super(ConvBNRelu, self).\_\_init\_\_()    self.model = keras.models.Sequential([  keras.layers.Conv2D(ch, kernelsz, strides=strides, padding=padding),  keras.layers.BatchNormalization(),  keras.layers.ReLU()  ])    def call(self, x, training=None):  x = self.model(x, training=training)    return x      # Inception Block 模块。  class InceptionBlk(keras.Model):    def \_\_init\_\_(self, ch, strides=1):  super(InceptionBlk, self).\_\_init\_\_()    self.ch = ch  self.strides = strides    self.conv1 = ConvBNRelu(ch, strides=strides)  self.conv2 = ConvBNRelu(ch, kernelsz=3, strides=strides)  self.conv3\_1 = ConvBNRelu(ch, kernelsz=3, strides=strides)  self.conv3\_2 = ConvBNRelu(ch, kernelsz=3, strides=1)    self.pool = keras.layers.MaxPooling2D(3, strides=1, padding='same')  self.pool\_conv = ConvBNRelu(ch, strides=strides)    def call(self, x, training=None):  x1 = self.conv1(x, training=training)    x2 = self.conv2(x, training=training)    x3\_1 = self.conv3\_1(x, training=training)  x3\_2 = self.conv3\_2(x3\_1, training=training)    x4 = self.pool(x)  x4 = self.pool\_conv(x4, training=training)    # concat along axis=channel 通道数  x = tf.concat([x1, x2, x3\_2, x4], axis=3)    return x      # Res Block 模块。继承keras.Model或者keras.Layer都可以  class Inception(keras.Model):    def \_\_init\_\_(self, num\_layers, num\_classes, init\_ch=16, \*\*kwargs):  super(Inception, self).\_\_init\_\_(\*\*kwargs)    self.in\_channels = init\_ch  self.out\_channels = init\_ch  self.num\_layers = num\_layers  self.init\_ch = init\_ch    self.conv1 = ConvBNRelu(init\_ch)    self.blocks = keras.models.Sequential(name='dynamic-blocks')    for block\_id in range(num\_layers):    for layer\_id in range(2):    if layer\_id == 0:    block = InceptionBlk(self.out\_channels, strides=2)    else:  block = InceptionBlk(self.out\_channels, strides=1)    self.blocks.add(block)    # enlarger out\_channels per block  self.out\_channels \*= 2    self.avg\_pool = keras.layers.GlobalAveragePooling2D()  self.fc = keras.layers.Dense(num\_classes)    def call(self, x, training=None):    out = self.conv1(x, training=training)    out = self.blocks(out, training=training)    out = self.avg\_pool(out)  out = self.fc(out)    return out   * ResNet   1.载入数据  import kerasfrom keras.datasets import mnist  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # 数据集会默认下载到C:\Users\xxx\.keras\datasets 中  2.将图片像素值范围缩小至0到1  x\_train = x\_train.astype('float32')  x\_test = x\_test.astype('float32')    # 将像素范围缩至0到1  x\_train /= 255  x\_test /= 255  3.将labels转换为稠密格式  y\_train = keras.utils.to\_categorical(y\_train, num\_classes) # 60000个  y\_test = keras.utils.to\_categorical(y\_test, num\_classes) # 10000个  4.构建模型  ① conv\_1（结合数据集对kernel size进行了调整）  # conv\_1 28x28→28x28x16  x = Conv2D(16,  kernel\_size=(3, 3),  activation='relu',  input\_shape=input\_shape,  padding='same'  )(inpt)  ②conv\_2 和 conv\_3  由于MNIST数据集的输入图像size较小，所以省略了maxpooling这一步骤，防止丢失太多信息  以下是每个block的实现，参数i用于判断当前是第几个block，在第一个block处取步长为2  def res\_block(x, channels, i):  if i == 1: # 第二个block  strides = (1, 1)  x\_add = x  else: # 第一个block  strides = (2, 2)  # x\_add 是对原输入的bottleneck操作  x\_add = Conv2D(channels,  kernel\_size=(3, 3),  activation='relu',  padding='same',  strides=strides)(x)    x = Conv2D(channels,  kernel\_size=(3, 3),  activation='relu',  padding='same')(x)  x = Conv2D(channels,  kernel\_size=(3, 3),  padding='same',  strides=strides)(x)  x = add([x, x\_add])  Activation(K.relu)(x)  return x  # conv\_2 28x28x16→14x14x16  for i in range(2):  x = res\_block(x, 16, i)    # conv\_3 14x14x16→7x7x32  for i in range(2):  x = res\_block(x, 32, i)  ## 训练  #-\*- coding: UTF-8 -\*-  import tensorflow as tf  import input\_data  mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True) # one\_hot 编码 [1 0 0 0]  sess = tf.InteractiveSession()  x = tf.placeholder("float", shape=[None, 784], name='x') # 输入  y\_ = tf.placeholder("float", shape=[None, 10], name='y\_') # 实际值  # 初始化权重  def weight\_variable(shape):  initial = tf.truncated\_normal(shape, stddev=0.1) # 产生正态分布 标准差0.1  return tf.Variable(initial)  # 初始化偏置  def bias\_variable(shape):  initial = tf.constant(0.1, shape=shape) # 定义常量  return tf.Variable(initial)  '''  tf.nn.conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, name=None)  input: 输入图像，张量[batch, in\_height, in \_width, in\_channels]  filter: 卷积核， 张量[filter\_height, filter \_width, in\_channels, out\_channels]  strides: 步长，一维向量，长度4  padding：卷积方式，'SAME' 'VALID'  '''  # 卷积层  def conv2d(x,W):  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')  '''  tf.nn.max\_pool(value, ksize, strides, padding, name=None)  value: 输入，一般是卷积层的输出 feature map  ksize: 池化窗口大小，[1, height, width, 1]  strides: 窗口每个维度滑动步长 [1, strides, strides, 1]  padding：和卷积类似，'SAME' 'VALID'  '''  # 池化层  def max\_pool\_2x2(x):  return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME') # 最大池化  # 第一层卷积 卷积在每个5\*5中算出32个特征  W\_conv1 = weight\_variable([5, 5, 1, 32])  b\_conv1 = bias\_variable([32])  x\_image = tf.reshape(x, [-1, 28, 28, 1])  h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)  h\_pool1 = max\_pool\_2x2(h\_conv1)  # 第二层卷积  W\_conv2 = weight\_variable([5, 5, 32, 64])  b\_conv2 = bias\_variable([64])  h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)  h\_pool2 = max\_pool\_2x2(h\_conv2)  # 密集连接层 图片尺寸缩减到了7\*7， 本层用1024个神经元处理  W\_fc1 = weight\_variable([7 \* 7 \* 64, 1024])  b\_fc1 = bias\_variable([1024])  h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7 \* 7 \* 64])  h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)  # dropout 防止过拟合  keep\_prob = tf.placeholder("float", name='keep\_prob')  h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)  # 输出层 最后添加一个Softmax层  W\_fc2 = weight\_variable([1024, 10])  b\_fc2 = bias\_variable([10])  y\_conv = tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2, name='y\_conv')  # 训练和评估模型  cross\_entropy = - tf.reduce\_sum(y\_ \* tf.log(y\_conv))  train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)  correct\_prediction = tf.equal(tf.argmax(y\_conv, 1), tf.argmax(y\_, 1))  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))  sess.run(tf.global\_variables\_initializer())  saver = tf.train.Saver()  for i in range(20000):  batch = mnist.train.next\_batch(50)  if i % 100 == 0:  train\_accuracy = accuracy.eval(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 1.0})  print("step %d, training accuracy %g"%(i, train\_accuracy))  train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})  print(accuracy.eval(feed\_dict={x: mnist.test.images, y\_: mnist.test.labels, keep\_prob: 1.0}))  # 保存模型  saver.save(sess, "E:/MyPython/02MNIST\_NN/mnist/minst\_cnn\_model.ckpt")  from PIL import Image  import tensorflow as tf  def imageprepare():  file\_name = 'number28\_28/5\_270\_270.png'  myimage = Image.open(file\_name)  myimage = myimage.resize((28, 28), Image.ANTIALIAS).convert('L') #变换成28\*28像素，并转换成灰度图  tv = list(myimage.getdata()) # 获取像素值  tva = [(255-x)\*1.0/255.0 for x in tv] # 转换像素范围到[0 1], 0是纯白 1是纯黑  return tva  result = imageprepare()  init = tf.global\_variables\_initializer()  saver = tf.train.Saver  with tf.Session() as sess:  sess.run(init)  saver = tf.train.import\_meta\_graph('minst\_cnn\_model.ckpt.meta') # 载入模型结构  saver.restore(sess, 'minst\_cnn\_model.ckpt') # 载入模型参数  graph = tf.get\_default\_graph() # 加载计算图  x = graph.get\_tensor\_by\_name("x:0") # 从模型中读取占位符张量  keep\_prob = graph.get\_tensor\_by\_name("keep\_prob:0")  y\_conv = graph.get\_tensor\_by\_name("y\_conv:0") # 关键的一句 从模型中读取占位符变量  prediction = tf.argmax(y\_conv, 1)  predint = prediction.eval(feed\_dict={x: [result], keep\_prob: 1.0}, session=sess) # feed\_dict输入数据给placeholder占位符  print(predint[0]) # 打印预测结果 | | |

## 全连接网络源码：

from keras.utils import np\_utils

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Activation

(X\_train,Y\_train),(X\_test,Y\_test)=mnist.load\_data()

X\_train=X\_train.reshape(60000,784).astype('float32')/255.0

X\_test=X\_test.reshape(10000,784).astype('float32')/255.0

Y\_train=np\_utils.to\_categorical(Y\_train)

Y\_test=np\_utils.to\_categorical(Y\_test)

model=Sequential()

model.add(Dense(units=64,input\_dim=28\*28,activation='relu'))

model.add(Dense(units=10,activation='softmax'))

model.compile(loss='categorical\_crossentropy',optimizer='sgd',metrics='accuracy')

model.fit(X\_train,Y\_train,epochs=5,batch\_size=32)

loss\_and\_metrics=model.evaluate(X\_test,Y\_test, batch\_size=32)

print ('loss\_and\_metrics:'+str(loss\_and\_metrics))

## Jupyter notebook可视化

import keras

from keras.utils import np\_utils

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense,Activation

from keras.callbacks import EarlyStopping

import numpy as np

np.random.seed(3)

(X\_train, Y\_train),(X\_test,Y\_test)=mnist.load\_data()

X\_val=X\_train[50000:]

Y\_val=Y\_train[50000:]

X\_train=X\_train[:50000]

Y\_train=Y\_train[:50000]

X\_train=X\_train.reshape(50000,784).astype('float32')/255.0

X\_val=X\_val.reshape(10000,784).astype('float32')/255.0

X\_test=X\_test.reshape(10000,784).astype('float32')/255.0

train\_rand\_idxs=np.random.choice(50000,700)

val\_rand\_idxs=np.random.choice(10000,300)

X\_val=X\_train[val\_rand\_idxs]

Y\_val=Y\_train[val\_rand\_idxs]

X\_train=X\_train[train\_rand\_idxs]

Y\_train=Y\_train[train\_rand\_idxs]

Y\_train=np\_utils.to\_categorical(Y\_train)

Y\_val=np\_utils.to\_categorical(Y\_val)

Y\_test=np\_utils.to\_categorical(Y\_test)

model=Sequential()

model.add(Dense(units=64,input\_dim=28\*28,activation='relu'))

model.add(Dense(units=10,activation='softmax'))

model.compile(loss='categorical\_crossentropy',optimizer='sgd',metrics=['accuracy'])

earlystop=EarlyStopping(patience=20)

tb\_hist=keras.callbacks.TensorBoard(log\_dir='./graph',histogram\_freq=0,write\_graph=True,write\_images=True)

hist=model.fit(X\_train,Y\_train,epochs=3000,batch\_size=10,validation\_data=(X\_val,Y\_val),callbacks=[earlystop])

loss\_and\_metrics=model.evaluate(X\_test,Y\_test,batch\_size=32)

print('loss\_and\_metrics:'+str(loss\_and\_metrics))

from IPython.display import SVG

from keras.utils.vis\_utils import model\_to\_dot

%matplotlib inline

import matplotlib.pyplot as plt

fig,loss\_ax=plt.subplots()

acc\_ax=loss\_ax.twinx()

loss\_ax.plot(hist.history['loss'],'y',label='train loss')

loss\_ax.plot(hist.history['val\_loss'],'r',label='val loss')

acc\_ax.plot(hist.history['accuracy'],'b',label='train acc')

acc\_ax.plot(hist.history['val\_accuracy'],'g',label='val acc')

loss\_ax.set\_xlabel('epoch')

loss\_ax.set\_ylabel('loss')

acc\_ax.set\_ylabel('accuracy')

loss\_ax.legend(loc='upper left')

acc\_ax.legend(loc='lower left')

plt.show()

## #数据增强例程

datagen = ImageDataGenerator(featurewise\_center=True, featurewise\_std\_normalization=True, rotation\_range=20,width\_shift\_range=0.2, height\_shift\_range=0.2, horizontal\_flip=True)

model.fit\_generator(datagen.flow(x\_train, y\_train, batch\_size=12),steps\_per\_epoch=…, epochs=10, verbose=1)

#输出模型结构

model.summary()

#保存模型

from keras.models import load model

model.save(‘filename.h5’)

#调用模型

model=load\_model(‘filename.h5’)

img = image.load\_img("xxx.jpg", target\_size=(28, 28))

input\_image = image.img\_to\_array(img)

input\_image /= 255.

　  # Add a 4th dimension for batch size (Keras)

input\_image = np.expand\_dims(input\_image, axis=0)

predictions = model.predict(input\_image)

y = model.predict\_classes(input\_image)

## Tensorboard可视化

keras使用tensorboard是通过回调函数来实现的，所以Tensorboard也是定义在keras.callbacks模块中的，通过构造一个Tensorboard类的对象，然后在训练的时候在fit里面指定callbacks参数即可，keras使用的一般格式为：

# 构造一个Tensorboard类的对象

tbCallBack = TensorBoard(log\_dir="./model",update\_freq='batch', histogram\_freq=0,write\_graph=True, write\_images=True,...)

# 在fit 里面指定callbacks参数

history=model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=2,

validation\_split=0.2,

callbacks=[tbCallBack])

verbose

日志展示，整数

0:为不在标准输出流输出日志信息

1:显示进度条

2:每个epoch输出一行记录

callbacks

其中的元素是keras.callbacks.Callback的对象。这个list中的回调函数将会在训练过程中的适当时机被调用，参考回调函数

validation\_split

浮点数0-1之间

用作验证集的训练数据的比例。

模型将分出一部分不会被训练的验证数据，并将在每一轮结束时评估这些验证数据的误差和任何其他模型指标。

验 证 数 据 是 混 洗 之 前 x 和 y 数 据 的 最 后 一 部 分 样 本 中 。 \color{red}{ 验证数据是混洗之前 x 和y 数据的最后一部分样本中。}验证数据是混洗之前x和y数据的最后一部分样本中。

# 打开TensorBoard方式

  Terminal中输入：tensorboard --logdir=…\logs