第九章 深度学习

目前国内与国际上主流的深度学习框架是Tensorflow与Pytorch，个人选择Tensorflow学习的原因有几个：一是同时支持Python与R语言，适应面广。二是配套的Keras易上手易用，并且有很好的中译版教材，适合小白入门。三是前面研究的发票货劳名称识别模型，使用的HanLP基于Tensorflow的预训练模型。虽然HanLP2.0增加了对Pytorch的支持，但Pytorch不支持R语言(注：Rstudio已经增加了对Torch的支持，R语言原生的实现，不是对Pytorch的封装，详见[参考资料1](https://blogs.rstudio.com/ai/posts/2020-09-29-introducing-torch-for-r/)，[参考资料2](https://www.csdn.net/tags/NtDaAgxsMzcwODctYmxvZwO0O0OO0O0O.html)，只支持cuda10.2及11.1，我装的是cuda10.1，以后有时间再升级了解一下)。四是对主流的GPU支持较好。五是各大云平台都支持。

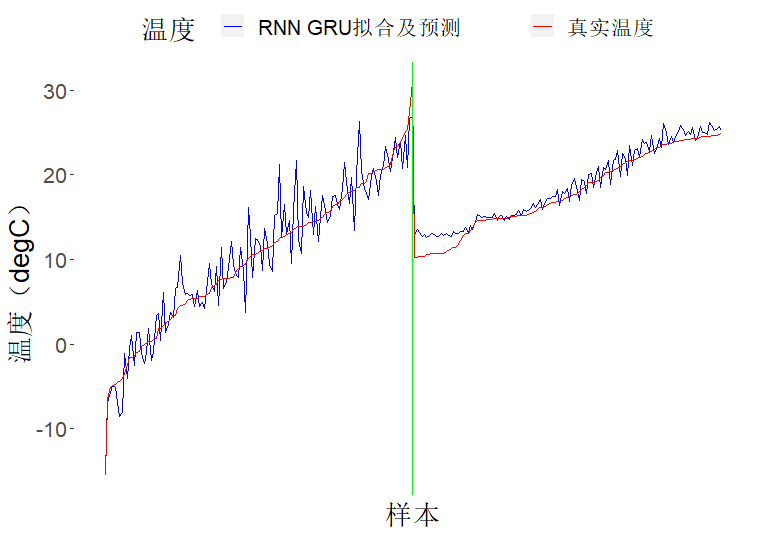
希望国内有志于此的厂商们早日搞出能与之比肩的国产框架，我一定会支持。有时间有精力也会测试一下国产的其它优秀框架。

所以本章用R语言RNN和Python CNN各学习一个例子，以演示两种语言下的使用，上一章回归分析中已经预先介绍了一个Python DNN房价预测的例子。除了发票货劳名称识别的NLP应用模型外，我还没有做出其它有价值的税收数据分析落地应用，所以本章是纯学习贴，但引用注明出处，应该不会侵犯原著《Python深度学习》与《R语言深度学习》的版权。

Tensorflow及Keras在Python与R语言上的安装在附录中再讲，Tensorflow及Keras都是基于Python的，R语言支持是通过接口包封装了对Python的调用，所以要先安装Python版的。

第一节 循环神经网络RNN时间序列预测

卷积神经网络CNN适合学习多维空间数据复杂的局部模式，比如图片识别等，而循环神经网络RNN适合学习时间（序列）数据中复杂的局部模式，比如NLP等。因为税收数据有很强的季节周期性（时间特征），前面已有章节尝试用季节ARIMA模型来分析，现在了解一下深度学习RNN在该方向的应用。虽然RNN在NLP等分类任务中的效果很好，但我更感兴趣的是它在回归等连续数值变量预测中的效果，尤其是房价等有明显的空间与时间复合特征。先看看气温预测拟合与预测的效果，左边拟合是在20万个时点的训练集中随机抽样的128个时点，右边是在测试集10万个时点中连续抽样的128个时点，接近一整天，拟合与预测的效果不错。为了突出拟合效果，两组样本分别按真实气温重新排了序。



一、数据集

这是[《Python深度学习》](https://github.com/fchollet/deep-learning-with-python-notebooks)与[《R语言深度学习》](https://www.manning.com/books/deep-learning-with-r)中的实验。数据是德国 Jena市马克斯.普朗克生物地球化学研究所气象站记录的2009~2016年天气时间序列数据，包括了摄氏温度、气压、湿度、风向等14项气象数据，每10分钟采样一次，每天144条，共42万条。实验的任务是根据过去10天的数据，预测未来一天气温的变化。

先作图观察，可以看到气温有明显的年周期。

# 下载数据 ------------------------------------------------------------------------

# 德国 jena市马克斯。普朗克生物地球化学研究所气象站记录的天气时间序列数据， 2009~2016年

dir.create("~/Downloads/jena\_climate", recursive = TRUE)

download.file(

"https://s3.amazonaws.com/keras-datasets/jena\_climate\_2009\_2016.csv.zip",

"~/Downloads/jena\_climate/jena\_climate\_2009\_2016.csv.zip"

)

unzip(

"~/Downloads/jena\_climate/jena\_climate\_2009\_2016.csv.zip",

exdir = "~/Downloads/jena\_climate"

)

# 加载并观察数据 ------------------------------------------------------------------

# 数据每10分钟采集一次，一天144次。

library(tibble)

library(readr)

library(keras)

library(ggplot2)

library(gtable)

library(grid)

data\_dir <- "D:/temp/data/jena\_climate"

fname <- file.path(data\_dir, "jena\_climate\_2009\_2016.csv")

data <- read\_csv(fname)

# 数据结构，共15个数据项，转置横向显示

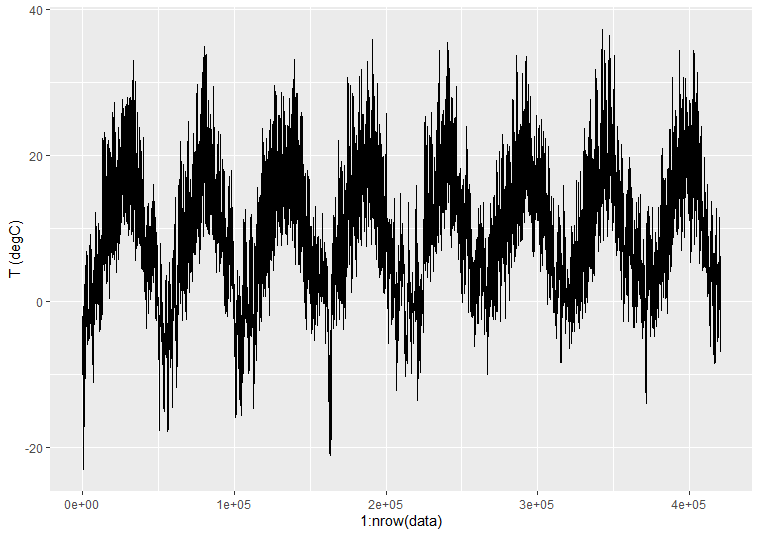
glimpse(data)

# 所有数据作图，有明显的年周期

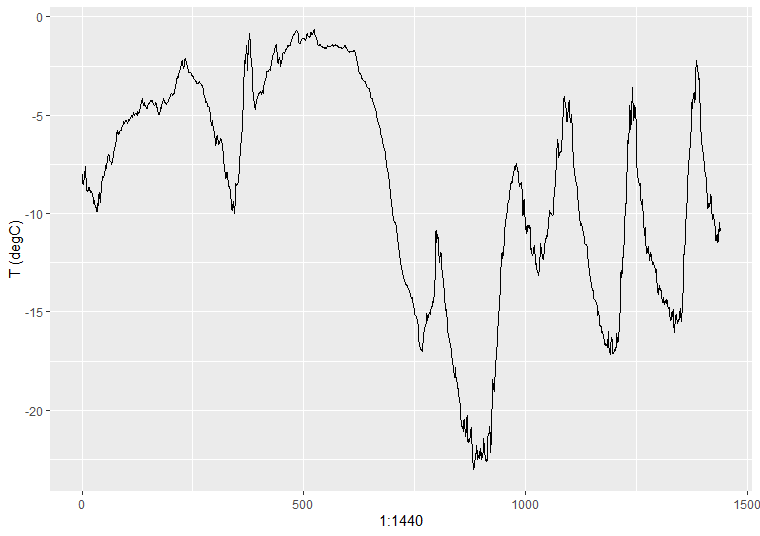
ggplot(data, aes(x = 1:nrow(data), y = `T (degC)`)) + geom\_line()

# 头10天的数据作图

ggplot(data[1:1440,], aes(x = 1:1440, y = `T (degC)`)) + geom\_line()



画前10天的气温图，可以看到有些日子有明显的日周期，符合生活的常识。



二、准备数据

1、丢掉时间列，固定间隔采样可以用位置索引代替。

2、数据标准化为均值为0，标准差为1，TensorFlow用小值数据，统一量纲。

3、降低采样频率，每小时采样一个数据点进行训练、验证与测试。通过定义一个generator()函数从原始数据中下采样完成，具体见下图。Keras可以通过这样的生成器函数无限循环供应训练、测试与验证的张量。完成后每个批次的张量格式是(128,240,14)，每个批次在GPU中并行计算。因为每个数据点都要回溯10天的数据，会有大量的重复，使用生成器函数从原始数据中动态产生有效避免了这种情况。

# 简单生成器函数示例 ---------------------------------------------------------------

# keras将使用类似的生成器从原始数据中生成每小时的时序数据，包括训练集、验证集与测试集。

sequence\_generator<- function(start){

value<-start-1

function(){

value<<-value+1

value

}

}

# 调用生成器

gen<-sequence\_generator(10)

for(i in 1:10) { cat(paste(gen(),"\n"))}

# 准备数据 -----------------------------------------------------------------------

# 丢掉第一列时间列，用行索引代替时间序列顺序，其它参数将作为张量传递给keras。

data <- data.matrix(data[,-1])

# 标准化训练数据

train\_data <- data[1:200000,]

mean <- apply(train\_data, 2, mean)

std <- apply(train\_data, 2, sd)

data <- scale(data, center = mean, scale = std)

# keras 生成器函数，用于从原始数据生成训练集、验证集与测试集，每次返回一个batch的数据(张量)

# data: 原始浮点数组（张量）

# lookback：输入数据应回溯多少时间步，1440即10天，根据过去10天的天气预测未来的天气

# delay：离目标还有多少时间步，即往前多少步预测，144步即预测未来1天的天气

# min\_index，max\_index：data中提取时间步的开始与结束索引，用于划分训练集、验证集与测试集

# shuffle：是否置乱样本或按时间顺序提取

# batch\_size：每批样本数量，用于GPU并行处理

# step：数据采样周期，以时间步为单位， 6步即1小时，每隔1小时从原始数据采样1次。

generator <- function(data, lookback, delay, min\_index, max\_index,

shuffle = FALSE, batch\_size = 128, step = 6) {

# 如果没有指定，最后一个数据点要由行数减去向前预测的步数

if (is.null(max\_index))

max\_index <- nrow(data) - delay - 1

# 当前采样的数据点从最小索引+回溯步数开始

i <- min\_index + lookback

function() {

if (shuffle) { # 置乱样本，在最大最小索引间随机采样一批数据，大小是batch\_size

rows <- sample(c((min\_index+lookback):max\_index), size = batch\_size)

} else { # 顺序采样, 连续采样batch\_size个数据点，这样数据比较集中于一个小的区间

# if (i + batch\_size >= max\_index) # 如果开始采样位置+batch\_size已经超过最大索引，则重置开始采样位置

# i <<- min\_index + lookback # 这个采样逻辑会丢掉最后一块不足batch\_size的数据块，这两行似乎应该在最后

rows <- c(i:min(i+batch\_size-1, max\_index)) # 最后一批取到max\_index为止，i+batch\_size-1从一开始就可能>max\_index

# cat(i); cat(":"); cat(i+batch\_size-1); cat("\n") # 原来程序有个小bug，i+batch\_size，没有减1, R语言序列包括终点。

i <<- i + length(rows) # 更新下标

if (i + batch\_size-1 >= max\_index) # 如果开始采样位置i+batch\_size已经超过最大索引，则重置开始采样位置

i <<- min\_index + lookback

}

# 采样数据与目标数据都预置为0，初始化返回结果张量维数，因为最后一个batch可能不够，所以先填零补够一个batch。

samples <- array(0, dim = c(length(rows), # 样本数，batch\_size

lookback / step, # 每个样本的数据点数， 1440/6=240,即用过去10天240个数据点预测未来一个点

dim(data)[[-1]])) # 列数， 参数个数，已丢掉时间列

targets <- array(0, dim = c(length(rows)))

# 对每一批次数据，

for (j in 1:length(rows)) {

indices <- seq(rows[[j]] - lookback, rows[[j]], # 每个数据点回溯lookback个点采样, 1440

length.out = dim(samples)[[2]]) # 以步辐step采样lookback / step个点，1440/6 = 240

samples[j,,] <- data[indices,] # 一个采样数据点的张量(lookback / step,变量数)

targets[[j]] <- data[rows[[j]] + delay,2] # 预测delay步后的值

}

list(samples, targets) # 返回1个batch\_size的张量(batch\_size,lookback / step,变量数)

}

}

4、定义训练、验证、测试数据集生成器函数。前20万条数据为训练集，中间10万条为验证集，后10万条为测试集。回溯1440步即过去10天，向前144步即预测1天，步幅为6即每60分钟下采样1次，这样每个数据点是(240,14)的张量（每个数据点包含了过去10天的数据），每批128个数据点，这样每批的张量维数是(128,240,14)。验证与测试是顺序采样，shuffle = FALSE，还要计算每次迭代中的步数（批数），训练是随机采样，指定适当步数（500）即可。

lookback <- 1440

step <- 6

delay <- 144

batch\_size <- 128

train\_gen <- generator(

data,

lookback = lookback,

delay = delay,

min\_index = 1,

max\_index = 200000,

shuffle = TRUE,

step = step,

batch\_size = batch\_size

)

val\_gen = generator(

data,

lookback = lookback,

delay = delay,

min\_index = 200001,

max\_index = 300000,

step = step,

batch\_size = batch\_size

)

test\_gen <- generator(

data,

lookback = lookback,

delay = delay,

min\_index = 300001,

max\_index = 400000,

step = step,

batch\_size = batch\_size

)

# for(i in 1:10) {test\_data\_sample<- test\_gen()}

# 需要验证的批次

val\_steps <- (300000 - 200001 - lookback) / batch\_size

# 需要测试的批次

test\_steps <- (nrow(data) - 300001 - lookback) / batch\_size

三、准备一个常识基线模型以备与深度学习模型比较效果

根据生活常识，基线模型可以假定明天与今天同一时间的气温相等，然后用平均绝对误差(MAE)来评估这个模型的效果。这个模型的MAE是0.2779875，乘以标准差8.852521还原就是2.46摄氏度（效果不错，胜过很多AI，有时常识就是那么简单可靠），机器学习模型要比它好才有用。

# 这是一个常识性的非机器学习基线预测：---------------------------------------------

# 明天同一时间的温度等于今天的，用于比较机器学习模型的效果

evaluate\_naive\_method <- function() {

batch\_maes <- c()

for (step in 1:val\_steps) {

c(samples, targets) %<-% val\_gen() # delay <- 144，24小时后的

preds <- samples[,dim(samples)[[2]],2] # 基线预测是24小时后的温度等于现在的温度

mae <- mean(abs(preds - targets)) # 计算基线预测的一个批次的平均绝对误差

batch\_maes <- c(batch\_maes, mae) # 记录所有批次的平均绝对误差

}

print(mean(batch\_maes)) # 返回基线模型的总体平均绝对误差

}

mae<- evaluate\_naive\_method() # 基线模型的总体平均绝对误差，0.2779875

celsius\_mae<- mae \* std[[2]] # 第2项是温度，乘以标准差就是摄氏温度的总体平均绝对误差，2.46089

print(celsius\_mae)

四、准备一个简单的深度学习模型以备比较

这里用一个2个稠密层的完全连接神经网络完成，训练迭代20次，每次训练跑500步(批)数据。可以看到，训练集的损失(MAE)接近0.2，好于上面的常识基线模型，但验证集的损失大于0.3，比不上常识基线模型。稠密层是不考虑数据的时序的。

# ----------------------------------------------------------------------------

# 第一个神经网络机器学习模型，只使用稠密连接层，不使用RNN网络，简单的机器学习模型用于比较参考

# 需要展平数据为(None,lookback / step,变量数)维张量

# 回归问题没有激活函数，编译时用平均绝对误差mae作为损失函数

model <- keras\_model\_sequential() %>%

layer\_flatten(input\_shape = c(lookback / step, dim(data)[-1])) %>%

layer\_dense(units = 32, activation = "relu") %>%

layer\_dense(units = 1)

model %>% compile(

optimizer = optimizer\_rmsprop(),

loss = "mae"

)

history <- model %>% fit\_generator(

train\_gen,

steps\_per\_epoch = 500, # batch\_size=128, 200000/128=1562.5, shuffle = TRUE, 因为样本是随机选取的，每次迭代500步已经足够

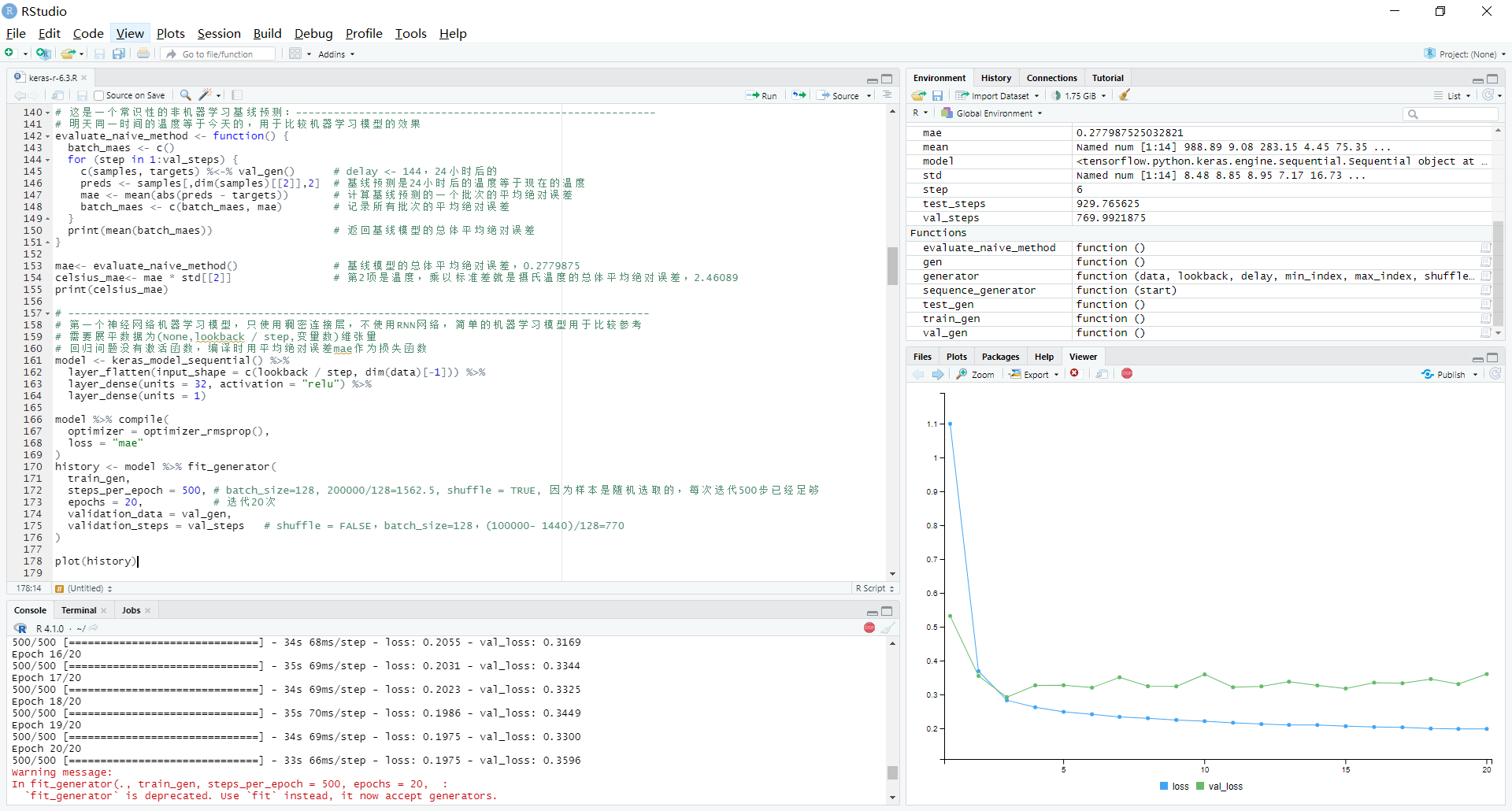
epochs = 20, # 迭代20次

validation\_data = val\_gen,

validation\_steps = val\_steps # shuffle = FALSE，batch\_size=128，(100000- 1440)/128=770

)

plot(history)



五、普通循环神经网络模型

这里使用一个门控循环装置(GRU)层完成。可以看到，在迭代5次以后，验证损失大于训练损失，开始出现过拟合，在迭代15次以后，验证损失>0.275，开始比不上常识基线模型了，但<0.3，仍然好于上面的稠密层模型。循环神经网络的训练比较慢，即使有GPU也需要点耐心。

# ---------------------------------------------------------------------------------

# 第一个循环神经网络机器学习模型，使用GRU层，迭代速度明显慢了很多，迭代与验证一次要576秒，差不多10分钟

model <- keras\_model\_sequential() %>%

layer\_gru(units = 32, input\_shape = list(NULL, dim(data)[[-1]])) %>%

layer\_dense(units = 1)

model %>% compile(

optimizer = optimizer\_rmsprop(),

loss = "mae"

)

history <- model %>% fit\_generator(

train\_gen,

steps\_per\_epoch = 500, # batch\_size=128, 200000/128=1562.5, shuffle = TRUE, 因为样本是随机选取的，每次迭代500步已经足够

epochs = 20, # 迭代20次

validation\_data = val\_gen,

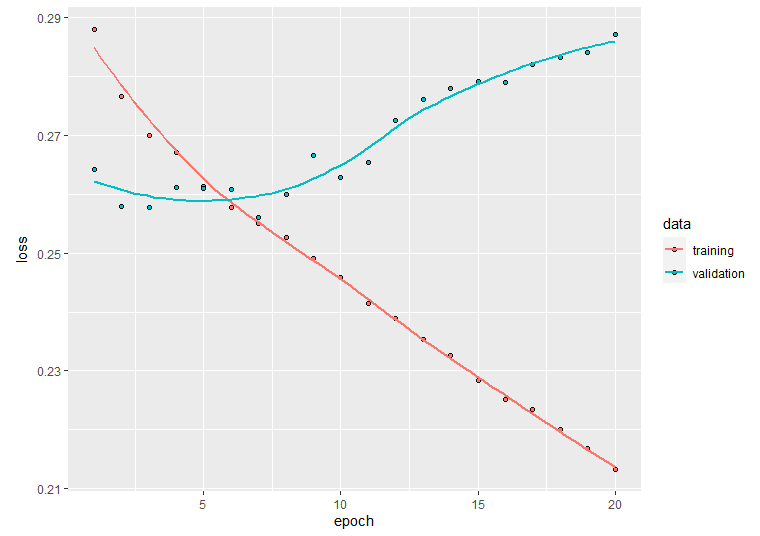
validation\_steps = val\_steps # shuffle = FALSE，batch\_size=128，(100000- 1440)/128=770

)

plot(history)

# 保存模型以便后面使用

save\_model\_hdf5(model,file.path(data\_dir, "pre\_trained\_gru\_model.h5"))



六、使用循环dropout来克服过拟合

使用循环dropout正则化后，可以看到迭代10多轮以后才出现过拟合，在第25轮以前，损失都小于0.28，即比常识基线模型要好，之后就比它差了。

# -------------------------------------------------------------------------------

# 使用循环dropout来克服过拟合，拉平了验证曲线，有所改善。

model <- keras\_model\_sequential() %>%

layer\_gru(units = 32, dropout = 0.2, recurrent\_dropout = 0.2,

input\_shape = list(NULL, dim(data)[[-1]])) %>%

layer\_dense(units = 1)

model %>% compile(

optimizer = optimizer\_rmsprop(),

loss = "mae"

)

history <- model %>% fit\_generator(

train\_gen,

steps\_per\_epoch = 500,

epochs = 40, # dropout正则化需要更长的时间才能完全收敛，训练两倍的轮数，40轮，时间太长了，代价太高。

validation\_data = val\_gen,

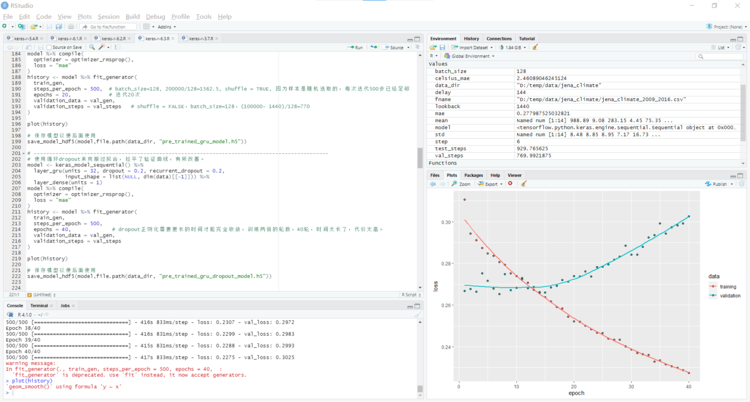
validation\_steps = val\_steps

)

plot(history)

# 保存模型以便后面使用

save\_model\_hdf5(model,file.path(data\_dir, "pre\_trained\_gru\_dropout\_model.h5"))



七、用训练的模型预测

根据上面的情况，重新训练一个迭代20次的循环dropout正则化模型（也许13次更好一点），然后用训练集与测试集的各一个批次(128,240,14)的张量进行预测，作图看看拟合与预测的效果。

# ---------------------------------------------------------------------------------

# 作图显示RNN模型拟合及预测情况

# 训练、验证、测试数据

train\_data\_sample<- train\_gen()

test\_data\_sample<- test\_gen()

val\_data\_sample <- val\_gen()

# 重新装入模型

model<- load\_model\_hdf5(file.path(data\_dir, "pre\_trained\_gru\_dropout\_model.h5"))

summary(model)

# 训练数据拟合情况

preds\_train<- model %>% predict(train\_data\_sample[1])

target\_train<- as.array(train\_data\_sample[2], dim=c(batch\_size))

res\_train <- data.frame(train\_data\_sample[2],preds\_train)

names(res\_train)<-c("target","predict")

res\_train<- res\_train\*std[2]+mean[2]

res\_train<- res\_train[order(res\_train$target),]

# 测试数据拟合情况

preds\_test<- model %>% predict(test\_data\_sample[1])

target\_test<- as.array(test\_data\_sample[2], dim=c(batch\_size))

res\_test <- data.frame(test\_data\_sample[2],preds\_test)

names(res\_test)<-c("target","predict")

res\_test<- res\_test\*std[2]+mean[2]

res\_test<- res\_test[order(res\_test$target),]

origin<- c(res\_train$target, res\_test$target)

fits<- c(res\_train$predict, res\_test$predict)

temp1<- data.frame(origin)

names(temp1)<- c("T\_degC")

temp1$sample<- as.integer(row.names(temp1))

temp1$type<-"真实温度"

temp2<- data.frame(fits)

names(temp2)<- c("T\_degC")

temp2$sample<- as.integer(row.names(temp2))

temp2$type<-"RNN GRU拟合及预测"

temp<- rbind(temp1, temp2)

temp$type<- as.factor(temp$type)

colors<-c("blue","red")

p <- ggplot(data = temp,aes(x=sample,y=T\_degC,group=type,colour=type))+

geom\_line()+

geom\_vline(xintercept = 128, colour = "green")+

# geom\_text(aes(label=round(price,2),vjust=0),size=5)+

theme(legend.position="top",text=element\_text(size=20),

# axis.text.x = element\_text(angle = 30, hjust = 0),

axis.ticks.x = element\_blank(),

axis.line.x = element\_blank(),

axis.text.x = element\_blank(),

#axis.text.y = element\_text(colour = "red"),

#axis.title.y = element\_text(colour = "red"),

panel.background = element\_rect(fill = NA),

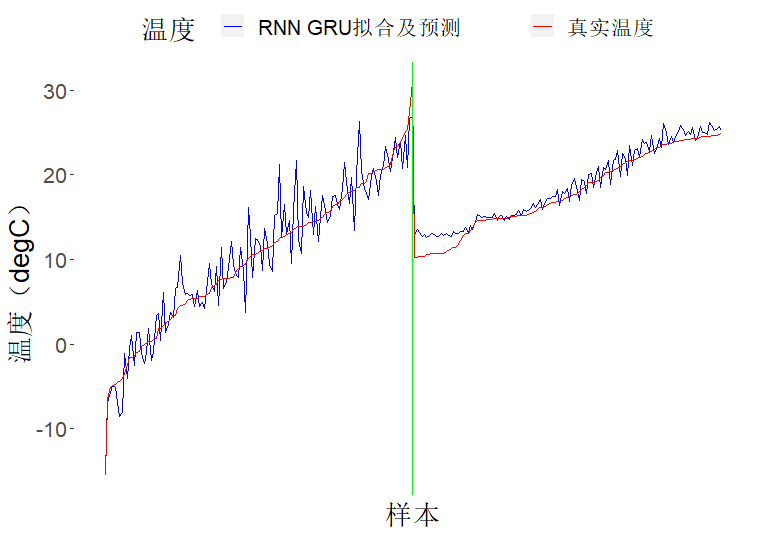
panel.grid = element\_blank())+

xlab("样本")+ylab("温度（degC）")+

scale\_colour\_manual(name = "温度",values=colors)

print(p)

这样看起来预测的效果比训练的效果还好。训练集与测试集的一个批次都是随机抽样的，总体效果还是要看MAE，MAE每降低0.1，乘以标准差准确度就提高0.885摄氏度。从上面迭代曲线可以看到，迭代13次左右大概可以达到这个要求。



这个实验练习了使用Keras+TensorFlow 循环神经网络RNN对时间序列数据作连续变量回归预测的整个流程，也许实验的结果还不够好，没有经典的卷积神经网络CNN猫狗照片识别实验那么让人眼前一亮，也许还有其它的方法可以继续提高，也许有些问题呢，简单常识的基线模型才是最好的（因为一年中气温变化的缓慢，这个常识基线模型极具统计上的洞察力，简单高效，其实是比较极端的），书中也说，参数学习有时可能无法找到简单问题的简单解决方案，这是机器学习一个相当重要的限制。

另外，RNN的计算量非常大，本实验训练每迭代一次要近7分钟，稠密层模型只用了30秒左右，没有GPU是很难玩转的，这也是一个要考虑的因素。相对来说，传统的时间序列分析如ARIMA等性能要好很多，各有各的优势和适用场景。

所以，实验的主要目标还是掌握循环神经网络RNN深度学习的方法论与相关工具。

八、用卷积神经网络加快序列数据的处理

书中提出了用一维卷积神经网络CNN对序列数据预处理以提高循环神经网络RNN性能的组合方案。时间可以看成是数据空间的一个维度，这样就可以应用一维卷积网络，为文本分类等简单任务提供RNN的快速替代方案，准确度差不多，效果不错。当全局顺序，比如单词在句子中出现的位置不是决定性的因素时，这个方法是有用的： 在情感分类等应用中，对于很多句子来说，一些关键词出现在句子的开头或结尾，对分类的影响不大。而在气温预测等全局顺序影响比较大的应用中，效果不则太好，下面这个CNN与RNN的组合模型损失接近0.3，甚至还不如常识基线模型，但迭代一次的时间由7分钟减少到171秒，快了一倍多。

# This was previously set to 6 (one point per hour).

# Now 3 (one point per 30 min).

step <- 3 # 采样加密，每半小时一次

lookback <- 720 # Unchanged # 总体回溯数据点不变， 720/3 = 240

delay <- 144 # Unchanged # 预测1天以后的气温

train\_gen <- generator(

data,

lookback = lookback,

delay = delay,

min\_index = 1,

max\_index = 200000,

shuffle = TRUE,

step = step

)

val\_gen <- generator(

data,

lookback = lookback,

delay = delay,

min\_index = 200001,

max\_index = 300000,

step = step

)

test\_gen <- generator(

data,

lookback = lookback,

delay = delay,

min\_index = 300001,

max\_index = NULL,

step = step

)

val\_steps <- (300000 - 200001 - lookback) / 128

test\_steps <- (nrow(data) - 300001 - lookback) / 128

k\_clear\_session()

model <- keras\_model\_sequential() %>%

layer\_conv\_1d(filters = 32, kernel\_size = 5, activation = "relu", # 先用一维卷积层做预处理

input\_shape = list(NULL, dim(data)[[-1]])) %>%

layer\_max\_pooling\_1d(pool\_size = 3) %>% # 一维池化层

layer\_conv\_1d(filters = 32, kernel\_size = 5, activation = "relu") %>% # 堆叠一个一维卷积层以看到更长的序列

layer\_gru(units = 32, dropout = 0.1, recurrent\_dropout = 0.5) %>% # 2个一维卷积层提炼特征后，循环神经网络GRU层处理

layer\_dense(units = 1) # 最后是一个稠密层输出

summary(model)

model %>% compile(

optimizer = optimizer\_rmsprop(),

loss = "mae"

)

history <- model %>% fit\_generator(

train\_gen,

steps\_per\_epoch = 500,

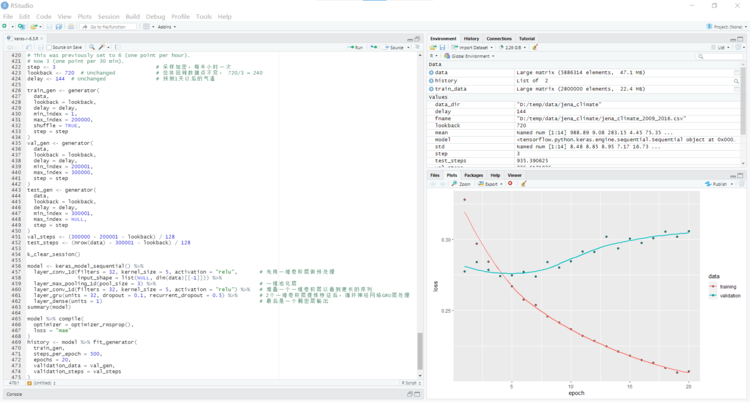
epochs = 20,

validation\_data = val\_gen,

validation\_steps = val\_steps

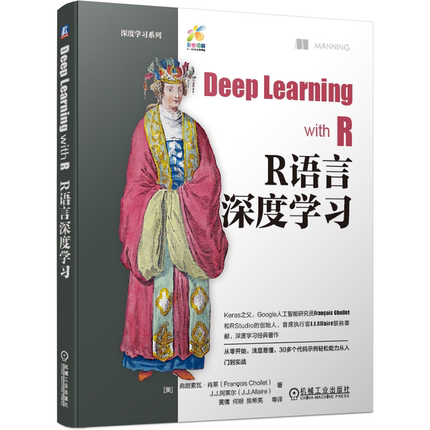
)

plot(history)



最近跑跑[《Python深度学习》](https://github.com/fchollet/deep-learning-with-python-notebooks)中上述实验等的Python版，该书第二版的英文版计划于2021年11月出版，在出版商的网站上可以免费在线看到预印版，对Keras与Tensorflow作了更深入的介绍，实例有增删，中文版大概还要等一段时间。因为Keras与Tensorflow版本功能的变化，代码大部分也都重写了，所以读读第二版最新的代码以了解技术的更新，是有好处的。

深度学习作为本系统数据挖掘可能涉及的最后一个板块，还不能说掌握了，只能说在门口略窥一二吧。



第二节 二维卷积神经网络CNN照片分类

前面的章节中可以看到，有些问题，在回归分析的技术路线上，以我的能力，91%是难以跨越的极限。因此可以考虑尝试一下其它的技术路线，比如深度学习，在猫狗等照片识别的实例中，卷积神经网络CNN证明了善于发现空间中的局部结构，如果把时间看作多维空间中的一维，也许可以尝试一下卷积神经网络的效果。

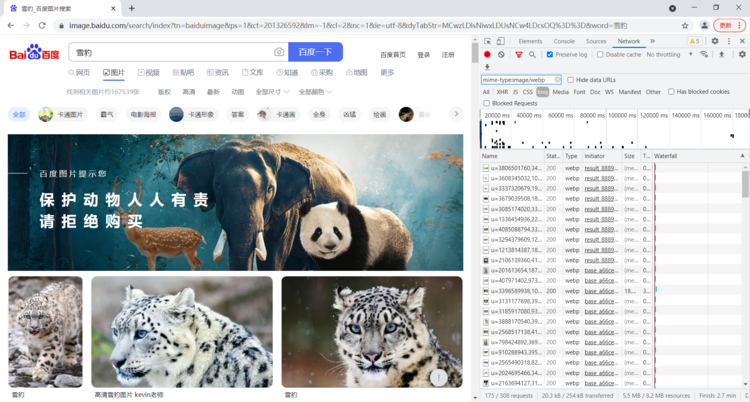
[《Python深度学习》](https://www.manning.com/books/deep-learning-with-python-second-edition?a_aid=keras&amp;a_bid=76564dff)与[《R语言深度学习》](https://www.manning.com/books/deep-learning-with-r)这两本书中猫狗识别的实验非常好，但正如[这篇文章](https://zhuanlan.zhihu.com/p/54103805)所说，复制官网的例子就有神奇的效果，而当我们开始尝试解决自己面临的问题时，可能就会觉得非常困难。IT界最有效的经验是learn by example，因此通过VGG16模型雪豹识别这个相似例子来练习一下CNN的使用，是比较合适的。一张照片是或者不是雪豹，[这是个与猫狗分类一样的二分类问题](https://groups.google.com/g/keras-users/c/iSFPXIDxb_g)，把书中实例的照片换成雪豹与非雪豹的两类动物照片训练模型就可以了。

[VGG16](https://www.jiqizhixin.com/graph/technologies/1d2204cb-e4cf-47c5-bfa2-691367fe2387)是牛津大学开发的照片分类模型，非常有名，kaggle上的顶尖模型之一，可以分辨1000类对象，主要是动物，非常适合于野外红外相机视频AI自动处理等落地场景的应用，在VGG16的基础上，按书上的例子针对雪豹的照片作迁移学习训练，则可以分辨动物是否为雪豹。这样，通过2遍扫描，第一遍用VGG16识别出有动物的片段（如果连续n帧都识别为某动物，比如半秒内的12帧，则可认为是监测到了该动物的活动），第二遍用迁移学习模型扫描提取的动物活动片段，就可以判断是否为雪豹，完成整个自动处理过程。

一、获得雪豹图集

这是测试与训练的基础。VGG16是在斯坦福大学[ImageNet图集](https://image-net.org/)上训练的，该图集有2万多类1400万张照片，是计算机视觉界的标准图集，通常是在它的子集[ISLVRC 2014](https://image-net.org/challenges/LSVRC/2014/browse-synsets.php)（ImageNet Large Scale Visual Recognition Challenge比赛）上训练模型，大概有128万张照片。从该子集的类别清单上可以看到“n02128757: snow leopard, ounce, Panthera uncia”，即有雪豹的图集，该模型可以识别雪豹。但从[这篇文章](https://zhuanlan.zhihu.com/p/42696535)可知，斯坦福大学只允许.edu后缀电邮地址的教育界用户申请下载，所以要另觅来源。这里要委屈一下小度，以小度图片为例说明一下怎样用合法合规的方式收集一些免费的图集。VGG16模型输入照片的规格是224\*224，这样收集的低像素照片刚好合用，训练VGG16模型并不需要高画质的照片。

1、在Google Chrome中打开devtools中的networks，记录网络流量，输入目标网址刷新网页，在filter中过滤，小度图片中返回的搜索结果是image/webp类型，输出保存成.har文件。具体可以参考[这篇文章](https://www.jianshu.com/p/471950517b07)。



2、Spyder中读入上面保存的json格式.har文件，提取照片，去重，并输出到硬盘目录。

# -\*- coding: utf-8 -\*-

"""

Created on Sun Oct 17 09:48:57 2021

@author: Jean

"""

import json

import base64

# from PIL import Image

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import time

import os, shutil, pathlib

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.preprocessing import image\_dataset\_from\_directory

from keras.applications.vgg16 import VGG16

from keras import backend as K

from keras.preprocessing import image

from keras.applications.vgg16 import preprocess\_input, decode\_predictions

# 限制单进程GPU内存占用上限， 方法二

# =============================================================================

gpu\_options = tf.compat.v1.GPUOptions(per\_process\_gpu\_memory\_fraction=0.85)

sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(gpu\_options=gpu\_options))

K.clear\_session()

# 从google chrome抓图保存的HAR文件中提取雪豹的图集

with open("D:/temp/data/snow\_leopard/snow\_leopard.har", 'r',encoding='utf-8') as f:

har = json.loads(f.read())

f.close()

entries = har['log']['entries']

urls = []; size=[]; text=[]

j=1

for entry in entries:

if entry['response']['content']['mimeType']=='image/jpeg':

url = entry['request']['url']

try:

t1 = entry['response']['content']['text']

s1 = entry['response']['content']['size']

text.append(t1)

urls.append(url)

size.append(s1)

j=j+1

except (KeyError):

print(url,'\n')

# 根据url去重

df = pd.DataFrame({"url":urls, "size":size, "image":text})

df.drop\_duplicates(["url"],inplace=True)

# 输出图集用于训练，1840张，人工检查，去掉误分类的照片，剩下1520张。

# 这里注释掉是避免再运行一次，以免覆盖掉人工检查标签的结果。

j=1

for img in df["image"]:

img2 = base64.b64decode(img)

fname = 'D:/temp/data/snow\_leopard/original/SnowLeopard.'+str(j)+'.jpg'

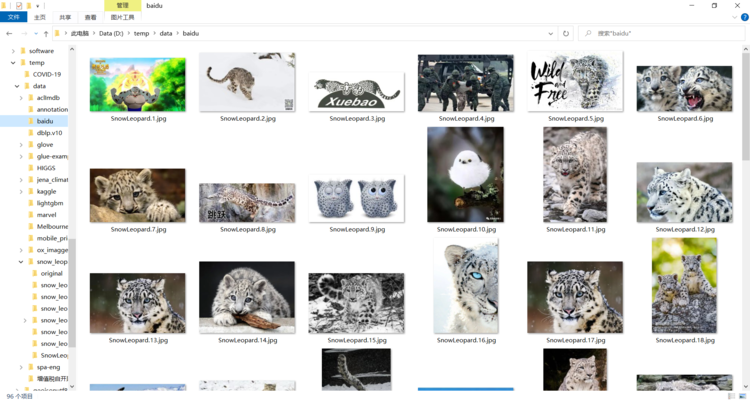
file = open(fname,'wb')

file.write(img2)

file.close()

j=j+1

3、这样收集的照片中有一些干扰照片，它们不是雪豹。



二、测试VGG16预训练模型

1、先用VGG16模型识别一下。Keras已经内置了VGG16模型，第一次调用时，会下载预训练模型的数据，大约540M，以及1000个类别的元数据。这里收集了1840张照片，因为动物的名称都是专业名词，用小度翻译做了个中英对照字典，中英文都列在上面。

# 装入所有抓取的图片，用VGG16预训练模型先分类一次

basedir = 'D:/temp/data/snow\_leopard/original/'

files = os.listdir(basedir)

images = []

for file in files:

image\_path = basedir+file

# `img` is a PIL image of size 224x224

img = image.load\_img(image\_path, target\_size=(224, 224))

images.append(img)

# 装入预训练的VGG16模型

model = VGG16(weights='imagenet')

t1 = time.time()

predicts = []; rate = []; label = []

for i in range(len(files)):

# `x` is a float32 Numpy array of shape (224, 224, 3)

x = image.img\_to\_array(images[i])

# We add a dimension to transform our array into a "batch" of size (1, 224, 224, 3)

x = np.expand\_dims(x, axis=0)

# Finally we preprocess the batch (this does channel-wise color normalization)

x = preprocess\_input(x)

preds = model.predict(x)

pre = decode\_predictions(preds, top=3)[0][0]

predicts.append(pre[1])

rate.append(pre[2])

label.append(pre[0])

t2=time.time()

print(str(i),' File: ',files[i],' Predicted:', pre[1]," Elapse",str(round(t2-t1,3)),'\n')

# 翻译为中文

with open('C:/Users/Jean/.keras/models/imagenet\_class\_index\_zh.json', 'r') as f:

trans = json.loads(f.read())

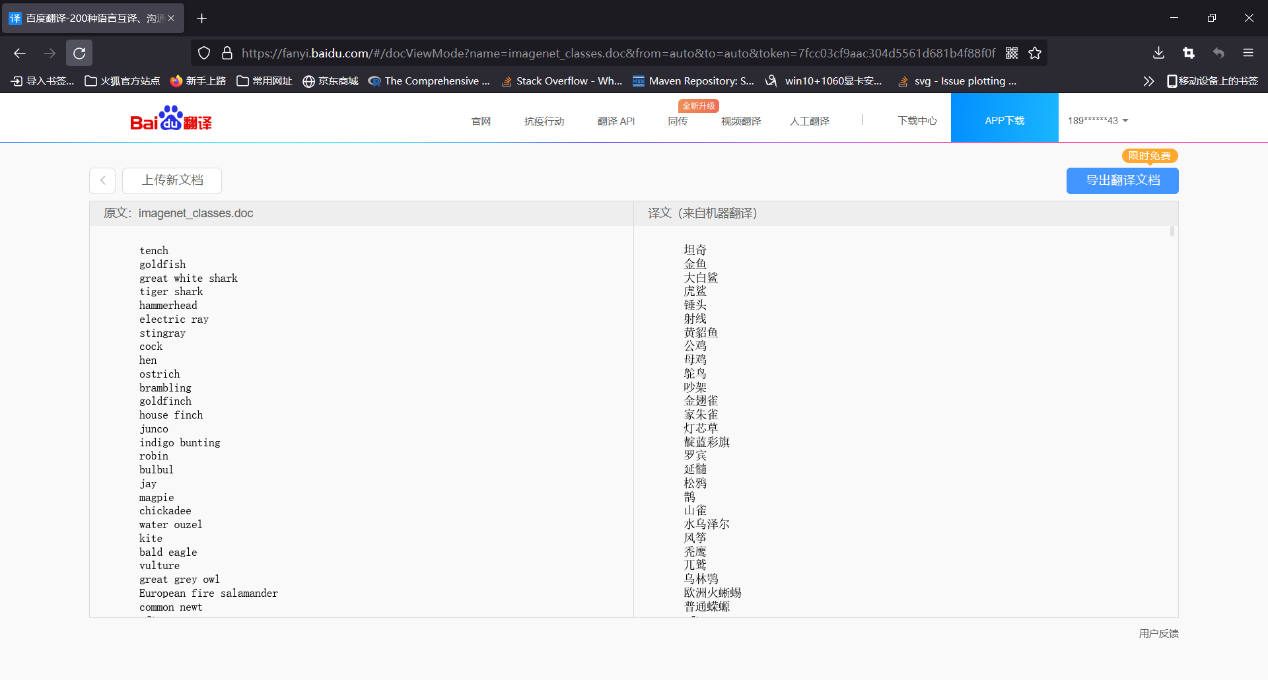
f.close()

predicts\_zh = []

for i in range(len(predicts)):

predicts\_zh.append(trans[predicts[i]])

小度翻译的中英文对照。



生成并载入分类名称英中对照字典。



2、对识别结果人工核验分类。

1840张照片，识别为雪豹的1303张，其中错误的1张，正确的1302张。识别为不是雪豹的537张，其中是雪豹的184张，不是雪豹的313张，照片质量有问题的40张。只按雪豹的照片计算，正确率=1302/(1302+184+1)=87.6%，全口径计算的正确率=(1302+313)/(1302+184+1+313)=89.7%，相当不错。

# 找出不是雪豹的图片

df2 = pd.DataFrame({"image":files, "predict":predicts, "predict\_zh":predicts\_zh, "label":label,"prob":rate})

nb = [ int(fname.split('.')[1]) for fname in df2['image']]

df2['findex']=nb

df2.sort\_values(by=['findex'], inplace=True)

df2 = df2.reset\_index(drop=True)

df2['predict'].value\_counts()

df3 = df2[df2['predict']!='snow\_leopard']

df3 = df3.reset\_index(drop=True)

df4= df2[df2['predict']=='snow\_leopard']

df4 = df4.reset\_index(drop=True)

# VGG16分类结果处理

wrong\_dir ='D:/temp/data/snow\_leopard/snow\_leopard\_wrong' # 537

for file in df3['image']:

shutil.copyfile(src=basedir +'/'+ file, dst= wrong\_dir +'/'+ file)

for file in df3['image']:

os.remove(basedir +'/'+ file)

# 然后从basedir剪切到D:/temp/data/snow\_leopard/snow\_leopard/，这是识别为snow\_leopard的,1303

# 然后从其中核验出1个错误的，剪切到D:/temp/data/snow\_leopard/snow\_leopard\_wrong2

# 对snow\_leopard\_wrong目录人工核验分类，分出snow\_leopard\_No 313张，snow\_leopard\_qual 4张



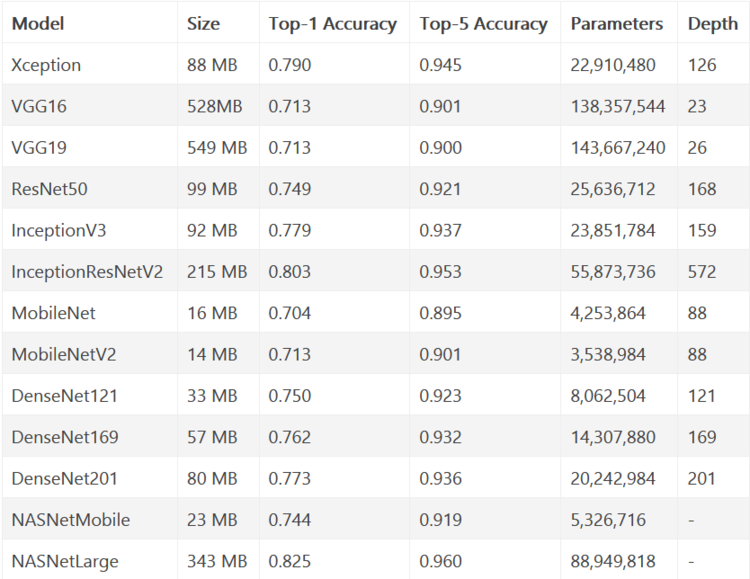


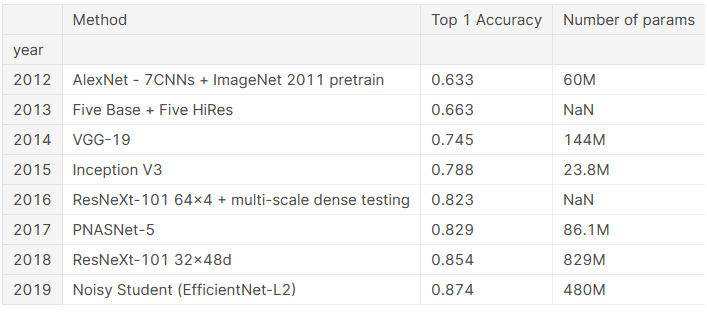
3、如果心里不踏实，VGG16模型也可以从零开始自己训练，因为深度学习框架、模型与ImageNet都是开源的，可以参考[这篇文章](https://zhuanlan.zhihu.com/p/54103805)，他山之石，可以攻玉。个人的观点我们可以把外国先进的科技公司与政客们区分开来，他们的利益与价值观显然有很多差异。不过在ImageNet数据集上训练VGG16非常耗时耗资源，4块Nvidia GPU卡大概要2~3周才能完成，所以用预训练模型是首选。

自己从头训练的一个好处是，可以选择放多少类，哪些类到预训练模型中，以及用哪些照片来训练，因为每个落地应用场景都不同，比如国内野外用的就可以不选斑马的照片。如果预训练模型已经满足要求了，就不需要后面的迁移学习模型了。

这里有[百度实现的版本](https://www.paddlepaddle.org.cn/modelbasedetail/vgg)，阿里、腾讯、华为等云平台上也可以用tensorflow等深度学习框架自行训练。还有清华计图、旷视天元等深度学习平台都可以训练VGG16模型。

下表列出了现在比较流行的深度学习神经网络模型，具体可以参考[这篇文章](https://www.jiqizhixin.com/graph/technologies/1d2204cb-e4cf-47c5-bfa2-691367fe2387)。





三、生成训练集、验证集与测试集  
1、把上面1302+184=1486张确认为雪豹的照片合并，然后划分为1000、243、243张的训练集、验证集与测试集，在snow\_leopard\_small下分别建立train、validation、test三个对应目录，数据有限，训练数据为重。

# 合并目录snow\_leopard与snow\_leopard\_wrong（剩下的都是正确的了）到SnowLeopard，用于后面训练

# 把图片划分为训练集，验证集与测试集中，分别为1000、243、243张。

# 拷贝猫狗二分类小数据集中cat类的训练集，验证集与测试集，并将验证集与测试集照片数量删减至与雪豹数据集相同，

# 以形成均衡的二分类数据集。

original\_dir = pathlib.Path("D:/temp/data/snow\_leopard/SnowLeopard")

new\_base\_dir = pathlib.Path("D:/temp/data/snow\_leopard/snow\_leopard\_small")

files = os.listdir(original\_dir)

# Copying images to training, validation, and test directories

def make\_subset(subset\_name, start\_index, end\_index):

category = "SnowLeopards"

dir = new\_base\_dir / subset\_name / category

os.makedirs(dir)

#fnames = [f"{category}.{i}.jpg" for i in range(start\_index, end\_index)]

fnames = [files[i] for i in range(start\_index, end\_index)]

for fname in fnames:

shutil.copyfile(src=original\_dir / fname,

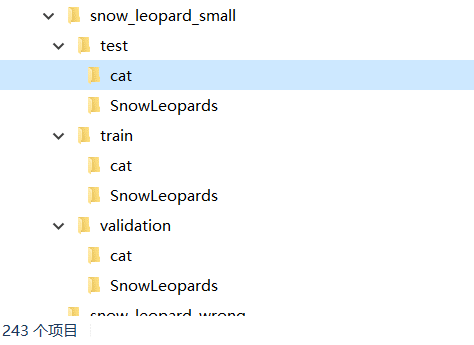
dst=dir / fname)

make\_subset("train", start\_index=0, end\_index=1000)

make\_subset("validation", start\_index=1000, end\_index=1243)

make\_subset("test", start\_index=1243, end\_index=1486)

2、把《Python深度学习》猫狗分类实验中small目录下cat类相应的目录拷贝过来，选cat因为它与雪豹很像，可以提高模型的准确率，但这样会影响泛化能力，因为这类对应的应该是“非雪豹”的动物照片，本应该非常多样化。把cat类的validation与test目录下的照片都删减至与雪豹相应目录下的数量相同，以使数据集平衡。



3、加载数据，预处理成tensorflow batch格式。

# Data preprocessing

# Using image\_dataset\_from\_directory to read images from directories

new\_base\_dir = pathlib.Path("D:/temp/data/snow\_leopard/snow\_leopard\_small")

train\_dataset = image\_dataset\_from\_directory(

new\_base\_dir / "train",

image\_size=(180, 180),

batch\_size=32)

validation\_dataset = image\_dataset\_from\_directory(

new\_base\_dir / "validation",

image\_size=(180, 180),

batch\_size=32)

test\_dataset = image\_dataset\_from\_directory(

new\_base\_dir / "test",

image\_size=(180, 180),

batch\_size=32)

四、直接训练一个DNN二分类模型

这个模型不使用VGG16的卷积基，用作对比。测试集准确率95.9%，已经很高了。比VGG16预训练模型提高了不少，不过请记住它们的用途是不同的，VGG16的任务是从照片中认出有没有动物，这个模型的任务是回答动物是不是雪豹，前者显然要强大很多。

# --------------------------------------------------------------------------------

# Instantiating a small convnet for dogs vs. cats classification

inputs = keras.Input(shape=(180, 180, 3))

x = layers.experimental.preprocessing.Rescaling(1./255)(inputs)

x = layers.Conv2D(filters=32, kernel\_size=3, activation="relu")(x)

x = layers.MaxPooling2D(pool\_size=2)(x)

x = layers.Conv2D(filters=64, kernel\_size=3, activation="relu")(x)

x = layers.MaxPooling2D(pool\_size=2)(x)

x = layers.Conv2D(filters=128, kernel\_size=3, activation="relu")(x)

x = layers.MaxPooling2D(pool\_size=2)(x)

x = layers.Conv2D(filters=256, kernel\_size=3, activation="relu")(x)

x = layers.MaxPooling2D(pool\_size=2)(x)

x = layers.Conv2D(filters=256, kernel\_size=3, activation="relu")(x)

x = layers.Flatten()(x)

outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

model.summary()

# Configuring the model for training

model.compile(loss="binary\_crossentropy",

optimizer="rmsprop",

metrics=["accuracy"])

# Fitting the model using a Dataset

# save the model and metrics

callbacks = [

keras.callbacks.ModelCheckpoint(

filepath="./convnet\_from\_scratch.keras",

save\_best\_only=True,

monitor="val\_loss")

]

history = model.fit(

train\_dataset,

epochs=30,

validation\_data=validation\_dataset,

callbacks=callbacks)

# Displaying curves of loss and accuracy during training

accuracy = history.history["accuracy"]

val\_accuracy = history.history["val\_accuracy"]

loss = history.history["loss"]

val\_loss = history.history["val\_loss"]

epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, "bo", label="Training accuracy")

plt.plot(epochs, val\_accuracy, "b", label="Validation accuracy")

plt.title("Training and validation accuracy")

plt.legend()

plt.figure()

plt.plot(epochs, loss, "bo", label="Training loss")

plt.plot(epochs, val\_loss, "b", label="Validation loss")

plt.title("Training and validation loss")

plt.legend()

plt.show()

# Evaluating the model on the test set

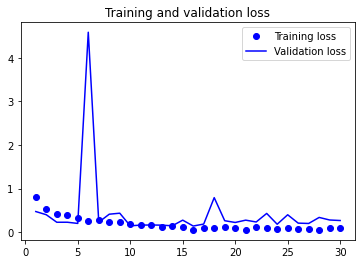
test\_model = keras.models.load\_model("./convnet\_from\_scratch.keras")

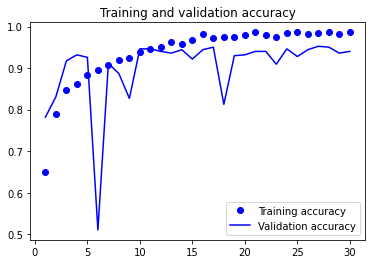
test\_loss, test\_acc = test\_model.evaluate(test\_dataset)

print(f"Test accuracy: {test\_acc:.3f}")

In [**9**]: print(f"Test accuracy: {test\_acc:.3f}")

Test accuracy: 0.955





五、在VGG16的卷积基上训练雪豹的二分类模型

相关源码，可以参考《Python深度学习》第二版第8章猫狗分类实验的源码：[chapter08\_intro-to-dl-for-computer-vision.ipynb](https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/chapter08_intro-to-dl-for-computer-vision.ipynb)。训练集与验证集的准确率都接近100%了。

# -------------------------------------------------------------------------------

# Feature extraction together with data augmentation

# Instantiating and freeze the VGG16 convolutional base

conv\_base = keras.applications.vgg16.VGG16(

weights="imagenet",

include\_top=False)

conv\_base.trainable = False

# Printing the list of trainable weights before and after freezing

conv\_base.trainable = True

print("This is the number of trainable weights "

"before freezing the conv base:", len(conv\_base.trainable\_weights))

conv\_base.trainable = False

print("This is the number of trainable weights "

"after freezing the conv base:", len(conv\_base.trainable\_weights))

# Adding data augmentation and a densely-connected classifier to the frozen convolutional base

data\_augmentation = keras.Sequential([

layers.experimental.preprocessing.RandomFlip("horizontal"),

layers.experimental.preprocessing.RandomRotation(0.1),

layers.experimental.preprocessing.RandomZoom(0.2),

])

inputs = keras.Input(shape=(180, 180, 3))

x = data\_augmentation(inputs)

x = conv\_base(x)

x = layers.Flatten()(x)

x = layers.Dense(256)(x)

x = layers.Dropout(0.5)(x)

outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs, outputs)

model.compile(loss="binary\_crossentropy",

optimizer="rmsprop",

metrics=["accuracy"])

callbacks = [

keras.callbacks.ModelCheckpoint(

filepath="./feature\_extraction\_with\_data\_augmentation.keras",

save\_best\_only=True,

monitor="val\_loss")]

history = model.fit(

train\_dataset,

epochs=50,

validation\_data=validation\_dataset,

callbacks=callbacks)

# Plotting the results

acc = history.history["accuracy"]

val\_acc = history.history["val\_accuracy"]

loss = history.history["loss"]

val\_loss = history.history["val\_loss"]

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, "bo", label="Training accuracy")

plt.plot(epochs, val\_acc, "b", label="Validation accuracy")

plt.title("Training and validation accuracy")

plt.legend()

plt.figure()

plt.plot(epochs, loss, "bo", label="Training loss")

plt.plot(epochs, val\_loss, "b", label="Validation loss")

plt.title("Training and validation loss")

plt.legend()

plt.show()

# Evaluating the model on the test set

test\_model = keras.models.load\_model(

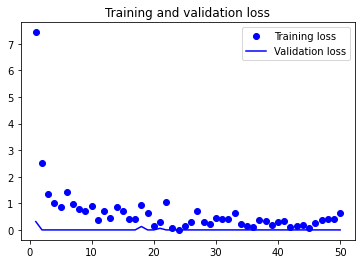
"./feature\_extraction\_with\_data\_augmentation.keras")

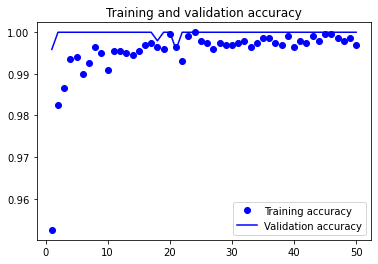
test\_loss, test\_acc = test\_model.evaluate(test\_dataset)

print(f"Test accuracy: {test\_acc:.3f}")

In [**12**]: print(f"Test accuracy: {test\_acc:.3f}")

Test accuracy: 1.000





从该书的第9章第3节[《卷积层学习可视化》](https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/chapter09_part03_interpreting-what-convnets-learn.ipynb)（第一版的第5章第4节）可知，卷积基各通道主要学习了照片各部分的轮廓、位置、纹理等特征，所以在卷积基上的迁移学习模型有更好的性能，这里还使用数据扩增技术增加了训练数据的差异，模型的泛化能力进一步加强。

六、参考资料：ILSVRC2017，最后一届。

1、[ImageNet Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)](https://image-net.org/challenges/LSVRC/2017/index.php)。该项竞赛有三项任务：

1）图片分类与定位。就是上述的1000个分类。

2）目标检测，检测图片中的所有对象，有200个分类。

3）视频目标检测，从视频中检测所有对象，包括了上述200个分类中的30个。

中文介绍可看[参考资料1](https://blog.csdn.net/xingwei_09/article/details/79148294)。

2、[该项竞赛Kaggle上的主页](https://www.kaggle.com/c/imagenet-object-localization-challenge/data)。上面可以下载任务的一百多万张照片，共166G。2020年3月已关闭，只发布比赛结果，没有发布什么代码，可以通过[kaggle搜索查找参考代码](https://www.kaggle.com/search?q=Imagenet+in%3Anotebooks)。[ImageNet-mini](https://www.kaggle.com/ifigotin/imagenetmini-1000)只有4G，1000类每类训练集几十张，验证集每类5张。

3、[ImageNet Winning CNN Architectures (ILSVRC)](https://www.kaggle.com/getting-started/149448)

在深度学习领域，我纯粹是只菜鸟，也许会有些想法，但知识与能力显然不足，所以有关各方真的不必太在意，怎么玩高兴随便我好了。这样的实验也可以说明，我们怎样可以好好的利用全球优秀的技术和资源去解决面临的问题。这并不妨碍我期望国内的大学与厂商们也做出优秀的产品来，去kaggle上揭揭榜，更快更好的实现科技自立自强。

