# 《企业实训》深度学习作业提交

大学：内蒙古大学 学院：计算机学院 专业： 软件工程

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 李纪元 | 学号 | 0191121383 | 班级 | 软工二班 |
| 实验任务名称 | 机器学习实验一： 使用卷积神经网络实现对cifar10数据集的分类识别任务 | | | | |
| 实验内容 | 包括cifar10数据集的下载、认识和处理cifar10数据集、卷积神经网络结构的设计（卷积层、池化层、分类器）、训练卷积神经网络、评估训练结果。 | | | | |
| 实验代码和结果 | # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  import matplotlib.pyplot as plt  x\_train = X\_train.astype(np.float32) / 255.  # 看第4张图片  plt.imshow(x\_train[4,:,:,:])  plt.show()    plt.figure(figsize = (12,12))  ROWS = 10  COLUMNS = 10  for i in range(ROWS \* COLUMNS):  plt.subplot(ROWS, COLUMNS, i + 1)  plt.xticks([])  plt.yticks([])  plt.imshow(x\_train[i,:,:,:])    from keras import models  from keras import layers  model = models.Sequential()  model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.Flatten())  model.add(layers.Dense(64, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()    model.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  X\_train = X\_train.reshape((50000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32, 32,3))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  history = model.fit(X\_train, Y\_train, epochs=10, batch\_size=128,validation\_split = 0.2)    import matplotlib.pyplot as plt  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 acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  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() | | | | |
| 实验任务名称 | 机器学习实验二： 解决卷积神经网络识别cifar10数据集分类任务的过拟合问题 | | | | |
| 实验内容 | 包括划分验证集做简单的留出验证、K折验证和带有打乱数据的重复K折验证、减小网络容量、添加权重正则化、添加dropout正则化等几种常用的过拟合解决方法，以及使用在卷积神经网络模型上的数据增强方法来解决过拟合问题。 | | | | |
| 实验代码和结果 | 留出验证：  # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  import matplotlib.pyplot as plt  x\_train = X\_train.astype(np.float32) / 255.  # 看第4张图片  plt.imshow(x\_train[4,:,:,:])  plt.show()    plt.figure(figsize = (12,12))  ROWS = 10  COLUMNS = 10  for i in range(ROWS \* COLUMNS):  plt.subplot(ROWS, COLUMNS, i + 1)  plt.xticks([])  plt.yticks([])  plt.imshow(x\_train[i,:,:,:])    from keras import models  from keras import layers  model = models.Sequential()  model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.Flatten())  model.add(layers.Dense(64, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()    model.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  X\_train = X\_train.reshape((50000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32, 32,3))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  history = model.fit(X\_train, Y\_train, epochs=10, batch\_size=128,validation\_split = 0.2)    import matplotlib.pyplot as plt  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 acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  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()    K折验证：  # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train,Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  import matplotlib.pyplot as plt  x\_train = X\_train.astype(np.float32) / 255.  # 看第4张图片  plt.imshow(x\_train[4,:,:,:])  plt.show()    import matplotlib.pyplot as plt  %matplotlib inline  plt.figure(figsize=(12,12))  ROWS=10  COLUMNS=10  for i in range(ROWS\*COLUMNS):  plt.subplot(ROWS,COLUMNS,i+1)  plt.xticks([])  plt.yticks([])  plt.imshow(x\_train[i,:,:,:])    from keras import models  from keras import layers  def build\_model():  model = models.Sequential()  model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.Flatten())  model.add(layers.Dense(64, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.compile(optimizer='rmsprop',loss='categorical\_crossentropy',metrics=['accuracy'])  return model  X\_train = X\_train.reshape((50000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32,32,3 ))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train= to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  import numpy as np  k = 4  num\_val\_samples = len(X\_train) // k  num\_epochs = 20  accuracy\_histories = []  average\_accuracy\_history = 0  all\_scores = []  all\_loss = []  # K折验证  for i in range(k):  print('processing fold #', i)  # 准备验证数据：第k个分区的输入数据和目标/输出数据  val\_data = X\_train[i \* num\_val\_samples: (i + 1) \* num\_val\_samples]  val\_targets=Y\_train[i\*num\_val\_samples:(i + 1) \* num\_val\_samples]  # 准备训练数据：其他所有分区的数据  partial\_train\_data=np.concatenate([X\_train[:i\*num\_val\_samples],X\_train[(i+1)\*num\_val\_samples:]], axis=0)  partial\_train\_targets=np.concatenate([Y\_train[:i\*num\_val\_samples],Y\_train[(i+1)\*num\_val\_samples:]], axis=0)  # 构建Keras模型（已编译）  model = build\_model()  # 训练模型（静默模式，verbose=0）  history=model.fit( partial\_train\_data, partial\_train\_targets,  validation\_data = (val\_data,val\_targets),epochs=num\_epochs, batch\_size=128,verbose = 0)  accuracy\_history=history.history['val\_accuracy']  accuracy\_histories.append(accuracy\_history)  val\_loss,val\_score=model.evaluate(val\_data,val\_targets)  all\_scores.append(val\_score)  all\_loss.append(val\_loss)    average\_accuracy\_history=[np.mean([x[i] for x in accuracy\_histories]) for i in range (num\_epochs)]  np.mean(average\_accuracy\_history)    all\_scores    np.mean(all\_scores)    import matplotlib.pyplot as plt  plt.plot(range(1,len(average\_accuracy\_history)+1),average\_accuracy\_history)  plt.xlabel('Epochs')  plt.ylabel('Validation accuracy')  plt.show()    # 打乱数据的K折验证  # 注：由于我的电脑内存不足，无法使用完整的数据集跑打乱数据的K折验证，所以这里只用了很小一部分数据，精度不高，仅作展示  # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train,Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  import matplotlib.pyplot as plt  x\_train = X\_train.astype(np.float32) / 255.  # 看第4张图片  plt.imshow(x\_train[4,:,:,:])  plt.show()    import matplotlib.pyplot as plt  %matplotlib inline  plt.figure(figsize=(12,12))  ROWS=10  COLUMNS=10  for i in range(ROWS\*COLUMNS):  plt.subplot(ROWS,COLUMNS,i+1)  plt.xticks([])  plt.yticks([])  plt.imshow(x\_train[i,:,:,:])    from keras import models  from keras import layers  def build\_model():  model = models.Sequential()  model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.Flatten())  model.add(layers.Dense(64, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.compile(optimizer='rmsprop',loss='categorical\_crossentropy',metrics=['accuracy'])  return model  X\_train = X\_train[:10000]  Y\_train = Y\_train[:10000]  X\_train = X\_train.reshape((10000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32,32,3 ))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train= to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  import numpy as np  from sklearn.utils import shuffle  def K\_check():  k = 4  num\_val\_samples = len(X\_train) // k  num\_epochs = 10  accuracy\_histories = []  average\_accuracy\_history = 0  all\_scores = []  all\_loss = []    # K折验证  for i in range(k):  print('processing fold #', i)  # 准备验证数据：第k个分区的输入数据和目标/输出数据  val\_data =X\_train[i\*num\_val\_samples:(i+1)\* num\_val\_samples]  val\_targets=Y\_train[i\*num\_val\_samples:(i+1)\*num\_val\_samples]    # 准备训练数据：其他所有分区的数据  partial\_train\_data=np.concatenate([X\_train[:i\*num\_val\_samples], X\_train[(i + 1) \* num\_val\_samples:]], axis=0)  partial\_train\_targets=np.concatenate([Y\_train[:i\*num\_val\_samples],Y\_train[(i + 1) \* num\_val\_samples:]], axis=0)  # 构建Keras模型（已编译）  model = build\_model()  history=model.fit(partial\_train\_data,partial\_train\_targets,validation\_data=(val\_data,val\_targets),epochs=num\_epochs, batch\_size=1)  accuracy\_history=history.history['val\_accuracy']  accuracy\_histories.append(accuracy\_history)  val\_loss,val\_score=model.evaluate(val\_data,val\_targets)  all\_scores.append(val\_score)  all\_loss.append(val\_loss)  average\_accuracy\_history=[np.mean([x[i] for x in accuracy\_histories]) for i in range (num\_epochs)]  return np.mean(average\_accuracy\_history)  def shuffle\_data(X,Y):  X,Y = shuffle(X,Y, random\_state=1337)  return X,Y  P = 3#进行P次K折验证  scores = []  for i in range(P):  X\_train,Y\_train = shuffle\_data(X\_train,Y\_train)  score = K\_check()  scores.append(score)    import matplotlib.pyplot as plt  plt.plot(range(1, len(scores) + 1), scores)  plt.xlabel('Epochs')  plt.ylabel('Validation accuracy')  plt.show()    改变网络容量：  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  import matplotlib.pyplot as plt  x\_train = X\_train.astype(np.float32) / 255.  # 看第4张图片  plt.imshow(x\_train[4,:,:,:])  plt.show()    plt.figure(figsize = (12,12))  ROWS = 10  COLUMNS = 10  for i in range(ROWS \* COLUMNS):  plt.subplot(ROWS, COLUMNS, i + 1)  plt.xticks([])  plt.yticks([])  plt.imshow(x\_train[i,:,:,:])    X\_train = X\_train.reshape((50000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32, 32,3))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  from keras import models  from keras import layers  model\_1 = models.Sequential()  model\_1.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model\_1.add(layers.MaxPooling2D((2, 2)))  model\_1.add(layers.Conv2D(64, (3, 3), activation='relu'))  model\_1.add(layers.MaxPooling2D((2, 2)))  model\_1.add(layers.Conv2D(64, (3, 3), activation='relu'))  model\_1.add(layers.Flatten())  model\_1.add(layers.Dense(64, activation='relu'))  model\_1.add(layers.Dense(10, activation='softmax'))  model\_1.summary()    model\_1.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  model\_1.fit(X\_train,Y\_train,epochs=20,batch\_size=128,validation\_split = 0.2)    from keras import models  from keras import layers  #r容量更小  model2= models.Sequential()  model2.add(layers.Conv2D(16, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model2.add(layers.MaxPooling2D((2, 2)))  model2.add(layers.Conv2D(32, (3, 3), activation='relu'))  model2.add(layers.MaxPooling2D((2, 2)))  model2.add(layers.Conv2D(32, (3, 3), activation='relu'))  model2.add(layers.Flatten())  model2.add(layers.Dense(64, activation='relu'))  model2.add(layers.Dense(10, activation='softmax'))  model2.summary()    model2.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  model2.fit(X\_train,Y\_train,epochs=20,batch\_size=128,validation\_split = 0.2)    from keras import models  from keras import layers  #r容量更大  model3= models.Sequential()  model3.add(layers.Conv2D(64, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model3.add(layers.MaxPooling2D((2, 2)))  model3.add(layers.Conv2D(128, (3, 3), activation='relu'))  model3.add(layers.MaxPooling2D((2, 2)))  model3.add(layers.Conv2D(128, (3, 3), activation='relu'))  model3.add(layers.Flatten())  model3.add(layers.Dense(64, activation='relu'))  model3.add(layers.Dense(10, activation='softmax'))  model3.summary()    model3.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  model3.fit(X\_train,Y\_train,epochs=20,batch\_size=128,validation\_split = 0.2)    import matplotlib.pyplot as plt  plt.rcParams['font.sans-serif'] = ['SimHei']    model1\_val\_loss = model\_1.history.history['val\_loss']  model2\_val\_loss = model2.history.history['val\_loss']  model3\_val\_loss = model3.history.history['val\_loss']  epochs = range(1, len(model1\_val\_loss) + 1)  model1\_loss = model\_1.history.history['loss']  model2\_loss = model2.history.history['loss']  model3\_loss = model3.history.history['loss']  plt.plot(epochs, model1\_val\_loss, 'b', label='Original model\_val\_loss')  plt.plot(epochs, model1\_loss, 'b+', label='Original model\_loss')  plt.plot(epochs, model2\_val\_loss, 'g', label='Smaller model\_val\_loss')  plt.plot(epochs, model2\_loss, 'go', label='Smaller model\_loss')  plt.plot(epochs, model3\_val\_loss, 'r', label='Bigger model\_val\_loss')  plt.plot(epochs, model3\_loss, 'r<', label='Bigger model\_loss')  plt.xlabel('Epochs')  plt.ylabel('LOSS')  plt.title("网络容量LOSS对比")  plt.legend()  plt.show()    import matplotlib.pyplot as plt  plt.rcParams['font.sans-serif'] = ['SimHei']  model1\_val\_accuracy = model\_1.history.history['val\_accuracy']  model2\_val\_accuracy = model2.history.history['val\_accuracy']  model3\_val\_accuracy = model3.history.history['val\_accuracy']  epochs = range(1, len(model1\_val\_accuracy) + 1)  model1\_accuracy= model\_1.history.history['accuracy']  model2\_accuracy = model2.history.history['accuracy']  model3\_accuracy = model3.history.history['accuracy']  plt.plot(epochs, model1\_val\_accuracy, 'b', label='Original model\_val\_accuracy')  plt.plot(epochs, model1\_accuracy, 'b+', label='Original model\_accuracy')  plt.plot(epochs, model2\_val\_accuracy, 'g', label='Smaller model\_val\_accuracy')  plt.plot(epochs, model2\_accuracy, 'go', label='Smaller model\_accuracy')  plt.plot(epochs, model3\_val\_accuracy, 'r', label='Bigger model\_val\_accuracy')  plt.plot(epochs, model3\_accuracy, 'r<', label='Bigger model\_accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.title("网络容量Accuracy对比")  plt.legend()  plt.show()    # 添加权重正则化：  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  import matplotlib.pyplot as plt  x\_train = X\_train.astype(np.float32) / 255.  # 看第4张图片  plt.imshow(x\_train[4,:,:,:])  plt.show()    plt.figure(figsize = (12,12))  ROWS = 10  COLUMNS = 10  for i in range(ROWS \* COLUMNS):  plt.subplot(ROWS, COLUMNS, i + 1)  plt.xticks([])  plt.yticks([])  plt.imshow(x\_train[i,:,:,:])    from keras import regularizers  from keras import models  from keras import layers  model = models.Sequential()  model.add(layers.Conv2D(128, (3, 3), activation='relu', input\_shape=(32, 32, 3),kernel\_regularizer=regularizers.l1\_l2(l1 = 0.001,l2 = 0.001)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu',kernel\_regularizer=regularizers.l1\_l2(l1 = 0.001, l2 = 0.001)))  model.add(layers.Flatten())  model.add(layers.Dense(64, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()    model.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  X\_train = X\_train.reshape((50000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32, 32,3))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  history = model.fit(X\_train, Y\_train, epochs=20, batch\_size=512,validation\_split = 0.2)    model.evaluate(X\_test, Y\_test)    import matplotlib.pyplot as plt  loss = history.history['loss']  val\_loss = history.history['val\_loss']  epochs = range(1, len(loss) + 1)  plt.plot(epochs, loss, 'bo', label='Training loss')  plt.plot(epochs, val\_loss, 'b', label='Validation loss')  plt.title('Training and validation loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()    plt.clf()  acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  plt.plot(epochs, acc, 'bo', label='Training acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  plt.title('Training and validation accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  plt.show()    # 添加Dropout正则化：  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  import matplotlib.pyplot as plt  x\_train = X\_train.astype(np.float32) / 255.  # 看第4张图片  plt.imshow(x\_train[4,:,:,:])  plt.show()    plt.figure(figsize = (12,12))  ROWS = 10  COLUMNS = 10  for i in range(ROWS \* COLUMNS):  plt.subplot(ROWS, COLUMNS, i + 1)  plt.xticks([])  plt.yticks([])  plt.imshow(x\_train[i,:,:,:])    from keras import regularizers  from keras import models  from keras import layers  model = models.Sequential()  model.add(layers.Conv2D(128, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Dropout(0.5))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.Flatten())  model.add(layers.Dropout(0.5))  model.add(layers.Dense(64, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()    model.compile(optimizer='rmsprop',  loss='categorical\_crossentropy',  metrics=['accuracy'])  X\_train = X\_train.reshape((50000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32, 32,3))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  history = model.fit(X\_train, Y\_train, epochs=20, batch\_size=512,validation\_split = 0.2)    model.evaluate(X\_test, Y\_test)    import matplotlib.pyplot as plt  loss = history.history['loss']  val\_loss = history.history['val\_loss']  epochs = range(1, len(loss) + 1)  plt.plot(epochs, loss, 'bo', label='Training loss')  plt.plot(epochs, val\_loss, 'b', label='Validation loss')  plt.title('Training and validation loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()    plt.clf()  acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  plt.plot(epochs, acc, 'bo', label='Training acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  plt.title('Training and validation accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  plt.show()    # 数据增强：  # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  from keras import models  from keras import layers  model = models.Sequential()  model.add(layers.Conv2D(256, (3, 3), activation='relu',input\_shape=(32, 32, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(128, (3, 3),activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), padding = 'same',activation='relu'))  model.add(layers.Conv2D(64, (3, 3),activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()    model.compile(optimizer='adam',  loss='categorical\_crossentropy',  metrics=['accuracy'])  X\_train = X\_train.reshape((50000, 32, 32, 3))  X\_train = X\_train.astype('float32') / 255  X\_test = X\_test.reshape((10000, 32, 32,3))  X\_test = X\_test.astype('float32') / 255  from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  from keras.preprocessing.image import ImageDataGenerator  datagen = ImageDataGenerator(rotation\_range=90,shear\_range=1.3)  xt = X\_train[:40000]  xv = X\_train[40000:]  yt = Y\_train[:40000]  yv = Y\_train[40000:]  history = model.fit\_generator(datagen.flow(xt, yt, batch\_size=128),validation\_data=(xv, yv), steps\_per\_epoch=len(xt) // 512,epochs=40)    result = model.evaluate(X\_test, Y\_test)  print(result)    import matplotlib.pyplot as plt  loss = history.history['loss']  val\_loss = history.history['val\_loss']  epochs = range(1, len(loss) + 1)  plt.plot(epochs, loss, 'bo', label='Training loss')  plt.plot(epochs, val\_loss, 'b', label='Validation loss')  plt.title('Training and validation loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  plt.show()    plt.clf()  acc = history.history['accuracy']  val\_acc = history.history['val\_accuracy']  plt.plot(epochs, acc, 'bo', label='Training acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  plt.title('Training and validation accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  plt.show() | | | | |
| 实验任务名称 | 机器学习实验三： 使用预训练模型进行迁移学习提升cifar10数据集分类识别准确度 | | | | |
| 实验内容 | 包括基于vgg16预训练模型的特征提取、基于vgg16预训练模型的带有数据增强的特征提取、基于vgg16预训练模型的finetune精调三种常用的迁移学习方法提升对cifar10数据集的分类识别准确度。 | | | | |
| 实验代码和结果 | 快速特征提取：  # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  from keras import models  from keras import layers  import keras  from keras.applications.vgg16 import VGG16  conv\_base = VGG16(weights='imagenet',  include\_top=False,  input\_shape=(32, 32, 3))  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  val\_samples = 20000  x\_val = X\_train[:20000]  partial\_x\_train = X\_train[20000:]  y\_val = Y\_train[:20000]  partial\_y\_train = Y\_train[20000:]  from keras.preprocessing.image import ImageDataGenerator  datagen = ImageDataGenerator(rescale=1./255)  batch\_size = 64  def extract\_features(data,sample\_count):  features = np.zeros(shape=(sample\_count, 1, 1, 512))  labels = np.zeros(shape=(sample\_count,10))  generator = datagen.flow(  data[0],data[1],  batch\_size=batch\_size,  )  i = 0  for inputs\_batch, labels\_batch in generator:  features\_batch = conv\_base.predict(inputs\_batch)  features[i \* batch\_size : (i + 1) \* batch\_size] = features\_batch  labels[i \* batch\_size : (i + 1) \* batch\_size] = labels\_batch  i += 1  if i \* batch\_size >= sample\_count:  break  return features, labels  train\_features, train\_labels = extract\_features((partial\_x\_train,partial\_y\_train),30000)  validation\_features, validation\_labels = extract\_features((x\_val,y\_val),20000)  test\_features, test\_labels = extract\_features((X\_test,Y\_test),10000)    train\_features = np.reshape(train\_features, (30000, 1 \* 1 \* 512))  validation\_features = np.reshape(validation\_features, (20000, 1 \* 1 \* 512))  test\_features = np.reshape(test\_features, (10000, 1 \* 1 \* 512))  from keras import models  from keras import layers  model = models.Sequential()  model.add(layers.Dense(64, activation='relu', input\_dim=1 \* 1 \* 512))  model.add(layers.Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.compile(optimizer='rmsprop',  loss='binary\_crossentropy',  metrics=['acc'])  history = model.fit(train\_features, train\_labels,  epochs=100,  batch\_size=64,  validation\_data=(validation\_features,validation\_labels)  )    result = model.evaluate(test\_features,test\_labels)  print(result)    import matplotlib.pyplot as plt  acc = history.history['acc']  val\_acc = history.history['val\_acc']  loss = history.history['loss']  val\_loss = history.history['val\_loss']  epochs = range(1, len(acc) + 1)  plt.plot(epochs, acc, 'bo', label='Training acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  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()    带有数据增强的特征提取：  import cv2  # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  X\_train = [cv2.resize(i,(64,64)) for i in X\_train]  X\_test = [cv2.resize(i,(64,64)) for i in X\_test]  X\_train = np.concatenate([arr[np.newaxis] for arr in X\_train] ).astype('float32')  X\_test = np.concatenate([arr[np.newaxis] for arr in X\_test] ).astype('float32')  X\_train = X\_train/255  X\_test = X\_test/255  from keras.applications.vgg16 import VGG16  conv\_base = VGG16(weights='imagenet',  include\_top=False,  input\_shape=(64, 64, 3))  conv\_base.summary()    from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  from keras import models  from keras import layers  from tensorflow import optimizers  model = models.Sequential()  model.add(conv\_base)  model.add(layers.Flatten())  model.add(layers.Dense(4096, activation='relu'))  model.add(layers.Dense(4096, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()    print('trainable weights: ', len(model.trainable\_weights))  conv\_base.trainable = False  print('trainable weights: ', len(model.trainable\_weights))    xt = X\_train[:40000]  yt = Y\_train[:40000]  xv = X\_train[40000:]  yv = Y\_train[40000:]  from keras.preprocessing.image import ImageDataGenerator  import os  train\_datagen = ImageDataGenerator(  rotation\_range=40,  width\_shift\_range=0.2,  height\_shift\_range=0.2,  shear\_range=0.2,  zoom\_range=0.2,  horizontal\_flip=True,  fill\_mode='nearest',  )  batchsize = 100  val\_datagen = ImageDataGenerator()  train\_generator = train\_datagen.flow(  xt,yt,  batch\_size = batchsize,  shuffle = False  )  validation\_generator = val\_datagen.flow(  xv,yv,  batch\_size = batchsize,  shuffle = False  )  import keras  callbacks\_list = [  keras.callbacks.ReduceLROnPlateau( # 不再改善时降低学习率  monitor='loss',  factor=0.5,  patience=2,  ),  keras.callbacks.EarlyStopping( # 不再改善时中断训练  monitor='loss',  patience=5,  )  ]  from tensorflow import optimizers  model.compile(optimizer=optimizers.RMSprop(lr=1e-5),  loss='categorical\_crossentropy',  metrics=['acc'])  history = model.fit\_generator(  train\_generator,  steps\_per\_epoch=100,  epochs=200,  validation\_data=validation\_generator,  validation\_steps=100,  callbacks = callbacks\_list)    model.evaluate(X\_test,Y\_test)    import matplotlib.pyplot as plt  acc = history.history['acc']  val\_acc = history.history['val\_acc']  loss = history.history['loss']  val\_loss = history.history['val\_loss']  epochs = range(1, len(acc) + 1)  plt.plot(epochs, acc, 'bo', label='Training acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  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()    微调模型：  import cv2  # -\*- coding: utf-8 -\*-  from tensorflow.keras import datasets  import numpy as np  # 加载数据  cifar10 = datasets.cifar10  (X\_train, Y\_train), (X\_test, Y\_test) = cifar10.load\_data()  X\_train = [cv2.resize(i,(64,64)) for i in X\_train]  X\_test = [cv2.resize(i,(64,64)) for i in X\_test]  X\_train = np.concatenate([arr[np.newaxis] for arr in X\_train] ).astype('float32')  X\_test = np.concatenate([arr[np.newaxis] for arr in X\_test] ).astype('float32')  X\_train = X\_train/255  X\_test = X\_test/255  from keras.applications.vgg16 import VGG16  conv\_base = VGG16(weights='imagenet',  include\_top=False,  input\_shape=(64, 64, 3))  conv\_base.summary()    from keras.utils.np\_utils import to\_categorical  Y\_train = to\_categorical(Y\_train)  Y\_test = to\_categorical(Y\_test)  from keras import models  from keras import layers  from tensorflow import optimizers  model = models.Sequential()  model.add(conv\_base)  model.add(layers.Flatten())  model.add(layers.Dense(4096, activation='relu'))  model.add(layers.Dense(4096, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()    print('trainable weights: ', len(model.trainable\_weights))  conv\_base.trainable = True  f = False  for layer in conv\_base.layers:  if layer.name == 'block3\_conv1':  f = True  if f == True:  layer.trainable = True  else:  layer.trainable = False  print('trainable weights: ', len(model.trainable\_weights))    xt = X\_train[:40000]  yt = Y\_train[:40000]  xv = X\_train[40000:]  yv = Y\_train[40000:]  from keras.preprocessing.image import ImageDataGenerator  import os  train\_datagen = ImageDataGenerator(  rotation\_range=40,  width\_shift\_range=0.2,  height\_shift\_range=0.2,  shear\_range=0.2,  zoom\_range=0.2,  horizontal\_flip=True,  fill\_mode='nearest',  )  batchsize = 100  val\_datagen = ImageDataGenerator()  train\_generator = train\_datagen.flow(  xt,yt,  batch\_size = batchsize,  shuffle = False  )  validation\_generator = val\_datagen.flow(  xv,yv,  batch\_size = batchsize,  shuffle = False  )  import keras  callbacks\_list = [  keras.callbacks.ReduceLROnPlateau( # 不再改善时降低学习率  monitor='loss',  factor=0.5,  patience=2,  ),  keras.callbacks.EarlyStopping( # 不再改善时中断训练  monitor='loss',  patience=5,  )  ]  from tensorflow import optimizers  model.compile(optimizer=optimizers.RMSprop(lr=1e-5),  loss='categorical\_crossentropy',  metrics=['acc'])  history = model.fit\_generator(  train\_generator,  steps\_per\_epoch=100,  epochs=200,  validation\_data=validation\_generator,  validation\_steps=100,  callbacks = callbacks\_list)    model.evaluate(X\_test,Y\_test)    import matplotlib.pyplot as plt  acc = history.history['acc']  val\_acc = history.history['val\_acc']  loss = history.history['loss']  val\_loss = history.history['val\_loss']  epochs = range(1, len(acc) + 1)  plt.plot(epochs, acc, 'bo', label='Training acc')  plt.plot(epochs, val\_acc, 'b', label='Validation acc')  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() | | | | |