AI识万物：从0搭建和部署手语识别系统

原创 韩信子@ShowMeAI ShowMeAI研究中心 2022-07-18 08:15 发表于北京

收录于合集
#AI实战系列

3个

本文对超长代码段进行了删减，确保能顺畅读完图文并get重点。学习 完整代码 可点击文末『阅读全文』，或访问下方链接。

> http://www.showmeai.tech/article-detail/292

据北京听力协会预估数据，我国听障人群数量已过千万。而在全球范围内有4.66亿人患有残疾性听力损失，约占全世界人口的5%。聋哑人士很特殊，他们需要使用手语进行交流，其他与常人无异，我国存在特殊教育水平在各城市中发展力度具有较大差异，国家通用手语推广程度浅，但不懂手语，与听力障碍者交流会非常困难。

在本篇内容中，ShowMeAI 借助深度学习与神经网络技术，针对这个问题从 0 构建 1 个应用程序，检测手语，并将其翻译给其他人进而打破手语隔阂。

搭建和部署完成后，你可以通过摄像头，轻松测试模型，如下图所示，快来一起试试吧。这个动图中的手势代表的单词，见文末哦！

手语介绍

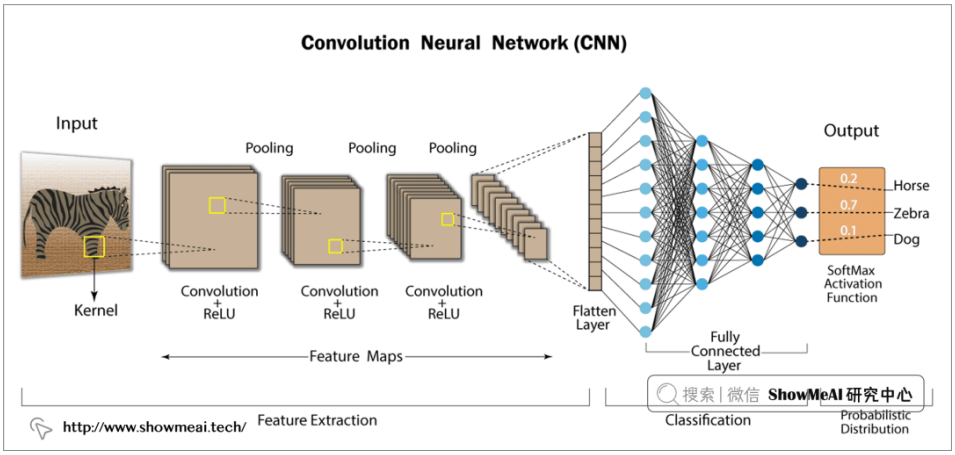
我们先来简单了解一下手语，它由 3 个主要部分组成：

- 手指拼写 ：这是一种手动的交流方式，用双手和手指拼写单词。每个字母都用指定的手位置表示。
- 单词级符号词汇 ：这是一个大型视频数据集，用于识别单词或字母的整个手势。
- 非手部特征 ：包括任何面部表情、嘴巴、舌头或身体姿势。

在本文中，我们先解决第①个部分的问题。我们准备使用的解决方案是基于视觉数据的神经网络。

深度学习与计算机视觉

人工智能和计算机视觉的最典型的模型是卷积神经网络（CNN），它在典型的计算机视觉应用中（如图像识别、目标检测等）应用广泛。我们在本次应用的核心技术也将采用 CNN。



CNN 网络有着如上图所示的网络结构，典型的结构包括卷积层、池化层、激活层、全连接层等，对于输入图像，可以有效抽取图像内容表征，并进行分类或其他处理。卷积层等特殊结构，可以在控制参数量的前提下，保证良好的图像特征提取能力。

> 关于卷积神经网络的详细知识可以参考ShowMeAI下述教程（链接见文末）
>
> - 深度学习教程 | 吴恩达专项课程 · 全套笔记解读 中的文章 **卷积神经网络解读**
> - 深度学习与计算机视觉教程 中的文章 卷积神经网络详解

小试牛刀，打通流程

我们来构建一个 CNN 识别的流程，会分成以下基础步骤：

- 数据读取与切分
- 数据可视化及预处理
- CNN网络构建与训练

① 导入相关库

我们在这里主要使用 TensorFlow 构建网络与训练，会使用 Numpy 做数据计算与处理，以及使用 Matplotlib 进行简单可视化。

> 对于这些工具库，ShowMeAI都制作了快捷即查即用的速查表手册，大家可以在文末获取链接：
>
> - Tensorflow 速查手册
> - Numpy 速查手册
> - Matplotlib 速查手册

我们先把这些工具库导入。

```
# 导入工具库
import string
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
from functools import partial
from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_img
```

② 读取数据集

本数据集为手语字母对应的数据集，图片 size 不大，所以也叫做 sign_mnist 数据集（类比手写数字数据集 mnist），部分示例图片如下：

数据集大家可以在 Kaggle平台对应数据集页面 下载，也可以通过ShowMeAI的百度网盘地址下载。

> 链接：https://pan.baidu.com/s/1dXKUZB0hyWtGiDOsHHrRuA
> 提取码：show

下面我们加载训练集与测试集并切分特征与标签：

```
# 读取数据
test = pd.read_csv("sign_mnist_test.csv")
train = pd.read_csv("sign_mnist_train.csv")

# 输出基本信息
print("训练集维度", train.shape)
print("测试集维度", train.shape)
```

```
# 输出标签信息
labels = train["label"].value_counts().sort_index(ascending=True)
labels
```

```
# 切分特征与标签
train_x = train.drop(labels = "label", axis = 1)
train_y = train["label"]
test_x = test.drop(labels = "label", axis = 1)
test_y = test["label"]
train_x.head()
```

```
# 数据预处理与可视化

# 存储标签数据
test_classes= test_y
train_clasees = train_y

# 特征转为numpy格式
train_x = train_x.to_numpy()
test_x = test_x.to_numpy()
```

```
# 把数据转为3维图像数据（图片数量*宽*高，这里如果是灰度图，颜色通道为1，省略）
train_x = train_x.reshape(-1,28,28)
test_x = test_x.reshape(-1,28,28)
```

```
# 在训练集中取样30张图片，做可视化查看
def plot_categories(training_images, training_labels):
    fig, axes = plt.subplots(3, 10, figsize=(16, 15))
    axes = axes.flatten()
    letters = list(string.ascii_lowercase)

    for k in range(30):
        img = training_images[k]
        img = np.expand_dims(img, axis=-1)
        img = array_to_img(img)
        ax = axes[k]
        ax.imshow(img, cmap="Greys_r")
        ax.set_title(f"{letters[int(training_labels[k])]}")
        ax.set_axis_off()

    plt.tight_layout()
    plt.show()

plot_categories(train_x, train_y)
```

③ 卷积神经网络CNN搭建

我们使用 TensorFlow 的 high level API（即keras）搭建一个简易CNN神经网络，并拟合一下数据。

```
def create_model():
    model = tf.keras.models.Sequential([
    # 卷积层
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
    # 池化层
    tf.keras.layers.MaxPooling2D(2,2),
    # 卷积层
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    # 池化层
    tf.keras.layers.MaxPooling2D(2,2),
    # 展平
    tf.keras.layers.Flatten(),
    # 全连接层
    tf.keras.layers.Dense(512, activation='relu'),
    # softmax分类
    tf.keras.layers.Dense(26, activation='softmax')])
```

```
    model.compile(
    optimizer='adam',  #优化器
    loss='sparse_categorical_crossentropy',  #损失函数
    metrics=['accuracy']) #评估准则

    return model
```

```
# 初始化模型
model = create_model()
# 拟合数据
history = model.fit(train_x, train_y, epochs=20, validation_data=(test_x, test_y))
```

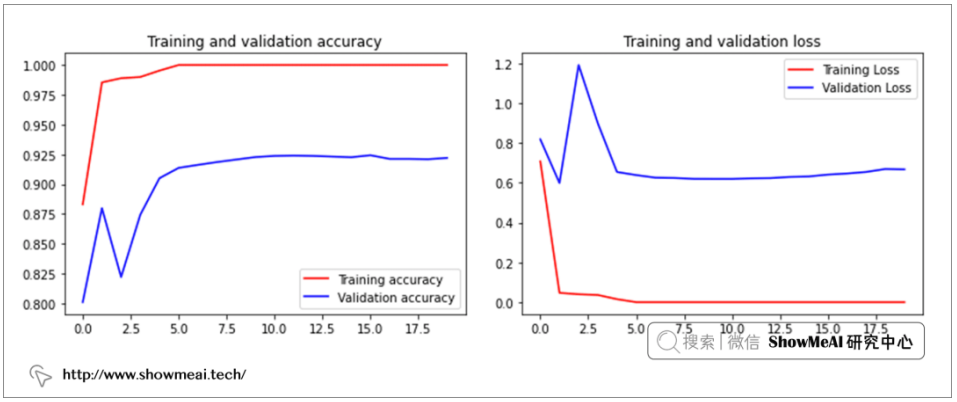
我们这里在全量数据集上迭代20个轮次，结果如下：

```
Epoch 1/20
858/858 [==============================] - 29s 32ms/step - loss: 0.7078 - accuracy: 0.8830 - val_loss: 0.8186 - val_accuracy: 0.8010
Epoch 2/20
858/858 [==============================] - 27s 32ms/step - loss: 0.0469 - accuracy: 0.9854 - val_loss: 0.5989 - val_accuracy: 0.8799
Epoch 3/20
858/858 [==============================] - 28s 32ms/step - loss: 0.0397 - accuracy: 0.9889 - val_loss: 1.1920 - val_accuracy: 0.8221
Epoch 4/20
858/858 [==============================] - 28s 33ms/step - loss: 0.0361 - accuracy: 0.9899 - val_loss: 0.8995 - val_accuracy: 0.8744
Epoch 5/20
858/858 [==============================] - 28s 33ms/step - loss: 0.0150 - accuracy: 0.9953 - val_loss: 0.6543 - val_accuracy: 0.9050
Epoch 6/20
858/858 [==============================] - 29s 34ms/step - loss: 3.2234e-05 - accuracy: 1.0000 - val_loss: 0.6388 - val_accuracy: 0.9137
Epoch 7/20
858/858 [==============================] - 28s 33ms/step - loss: 1.5477e-05 - accuracy: 1.0000 - val_loss: 0.6259 - val_accuracy: 0.9162
Epoch 8/20
858/858 [==============================] - 28s 33ms/step - loss: 9.4971e-06 - accuracy: 1.0000 - val_loss: 0.6243 - val_accuracy: 0.9186
Epoch 9/20
858/858 [==============================] - 28s 33ms/step - loss: 6.4234e-06 - accuracy: 1.0000 - val_loss: 0.6197 - val_accuracy: 0.9207
Epoch 10/20
858/858 [==============================] - 28s 33ms/step - loss: 4.4300e-06 - accuracy: 1.0000 - val_loss: 0.6194 - val_accuracy: 0.9228
Epoch 11/20
858/858 [==============================] - 28s 33ms/step - loss: 3.1067e-06 - accuracy: 1.0000 - val_loss: 0.6194 - val_accuracy: 0.9237
Epoch 12/20
858/858 [==============================] - 28s 32ms/step - loss: 2.1514e-06 - accuracy: 1.0000 - val_loss: 0.6223 - val_accuracy: 0.9240
Epoch 13/20
858/858 [==============================] - 28s 32ms/step - loss: 1.5093e-06 - accuracy: 1.0000 - val_loss: 0.6236 - val_accuracy: 0.9237
Epoch 14/20
858/858 [==============================] - 28s 32ms/step - loss: 1.0179e-06 - accuracy: 1.0000 - val_loss: 0.6294 - val_accuracy: 0.9232
Epoch 15/20
858/858 [==============================] - 27s 32ms/step - loss: 6.8312e-07 - accuracy: 1.0000 - val_loss: 0.6317 - val_accuracy: 0.9226
Epoch 16/20
858/858 [==============================] - 27s 32ms/step - loss: 4.7814e-07 - accuracy: 1.0000 - val_loss: 0.6410 - val_accuracy: 0.9243
Epoch 17/20
858/858 [==============================] - 27s 32ms/step - loss: 3.1418e-07 - accuracy: 1.0000 - val_loss: 0.6466 - val_accuracy: 0.9212
Epoch 18/20
858/858 [==============================] - 28s 32ms/step - loss: 2.1092e-07 - accuracy: 1.0000 - val_loss: 0.6544 - val_accuracy: 0.9212
Epoch 19/20
858/858 [==============================] - 28s 33ms/step - loss: 1.4387e-07 - accuracy: 1.0000 - val_loss: 0.6693 - val_accuracy: 0.9209
Epoch 20/20
858/858 [==============================] - 29s 34ms/step - loss: 9.4373e-08 - accuracy: 1.0000 - val_loss: 0.6673 - val_accuracy: 0.9221
```


我们可以看到，这里的数据并不特别复杂，在自己从头搭建的 CNN 模型上，经过训练可以达到训练集 100% 验证集 92% 的准确率。

我们再对训练过程中的「准确率」及「损失函数」变化值进行绘制，以了解模型状态。

```
# 获取准确率与损失函数情况
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

# matplotlib绘制训练过程中指标的变化状况
epochs = range(len(acc))

plt.plot(epochs, acc, 'r', 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, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



问题与优化

① 深度网络与梯度消失

一般来说，随着 CNN 网络层数变深，模型的学习能力会变强，也能学到更多的信息。但训练深度CNN存在梯度消失的问题。

> 梯度消失和梯度爆炸部分内容也可以参考ShowMeAI的对吴恩达老师课程的总结文章 **深度学习教程 | 深度学习的实用层面** （链接见文末）。

非常深的神经网络的梯度会很快变为零（反向传播的梯度连乘带来的问题），这最终会使整个梯度下降变慢。有一些特殊结构的神经网络，可以大程度缓解这个问题，比如最著名的 ResNet，当然，大家可以借助 ResNet 预训练模型快速迁移学习应用在我们当前的手语识别问题上，为了让大家对ResNet 细节更清晰，我们在这里手动搭建 ResNet-50（即50层的ResNet网络）来训练和做效果对比。

> ResNet的详细讲解也可以参考ShowMeAI的 **深度学习教程 | 吴恩达专项课程 · 全套笔记解读** 中的文章 **经典CNN网络实例详解**（链接见文末）。

② ResNet 模型简介

ResNet 是 Residual Networks 的简称，是迄今为止我们看到的最流行和最成功的深度学习模型之一。ResNets 由残差块组成，残差块的核心组件是『跳跃连接/skip-connection』。

跳跃连接，也称为快捷连接，让神经网络跳过某些层并将一层的输出馈送到神经网络中另一层的输入。它能帮助模型避免乘以中间跳过的那些层的权重，从而有助于解决梯度消失的问题。

然而，使用 ResNet 和跳跃连接，由于中间有卷积层和池化层，一层输出的维度可能与另一层的输出维度不同。为了解决这个问题，可以使用两种方法：

- 快捷连接填充多个零实体以增加其维度
- 添加 1X1 卷积层来匹配维度。

但是，对于第二种方法，我们需要在输出中添加一个额外的参数，而第一种方法不需要。

③ ResNet为何有效

ResNet的效果核心有2点：

- ① 它使用我们上面提到的跳跃连接，它跳过层来解决梯度消失的问题。
- ② 它通过让模型学习恒等函数来确保最高层的性能至少与最低层一样好。

④ 构建ResNet-50

下面我们参考 keras 官方 ResNet 构建方式，构建一个 ResNet-50，如下所示，我们先构建基本模块，再组装成最终的网络。

```
# Defining the identity block of the Resnet-50 Model.
def identity_block(X, f, filters, training=True):
    # filter of the three convs
    f1,f2,f3 = filters
    X_shortcut = X

    # First Component
    X = tf.keras.layers.Conv2D(filters = f1, kernel_size = 1, strides = (1,1), padding = 'val
    X = tf.keras.layers.BatchNormalization(axis = 3)(X, training = training) # Default axis
    X = tf.keras.layers.Activation('relu')(X)
```

```
# Defining the Convolution Block of the Resnet-50 Model.
def convolutional_block(X, f, filters, s=2,training=True):
    # filter of the three convs
    f1,f2,f3 = filters
    X_shortcut = X

    # First Component
    X = tf.keras.layers.Conv2D(filters = f1, kernel_size = 1, strides = (1,1), padding = 'val
    X = tf.keras.layers.BatchNormalization(axis = 3)(X, training = training) # Default axis
    X = tf.keras.layers.Activation('relu')(X)
```

```
# Defining a modified Resnet-50 Model using the Identity and Convolution Blocks.
def ResNet50(input_shape = (28, 28, 1), classes = 26):

    # Defining the input as a tensor with shape input_shape
    X_input = tf.keras.Input(input_shape)

    # Zero-Padding
    X = tf.keras.layers.ZeroPadding2D((3, 3))(X_input)
```

> 查看完整代码 showmeai.tech/article-detail/292

⑤ 训练ResNet-50

下面我们在数据集上，使用 ResNet-50 网络进行训练。

```
# 初始化模型
model = ResNet50()

# 编译
model.compile(optimizer="adam",metrics=["accuracy"],loss = "sparse_categorical_crossentropy")

# 训练
history = model.fit(train_x, train_y, validation_data = (test_x, test_y), epochs =10)
```

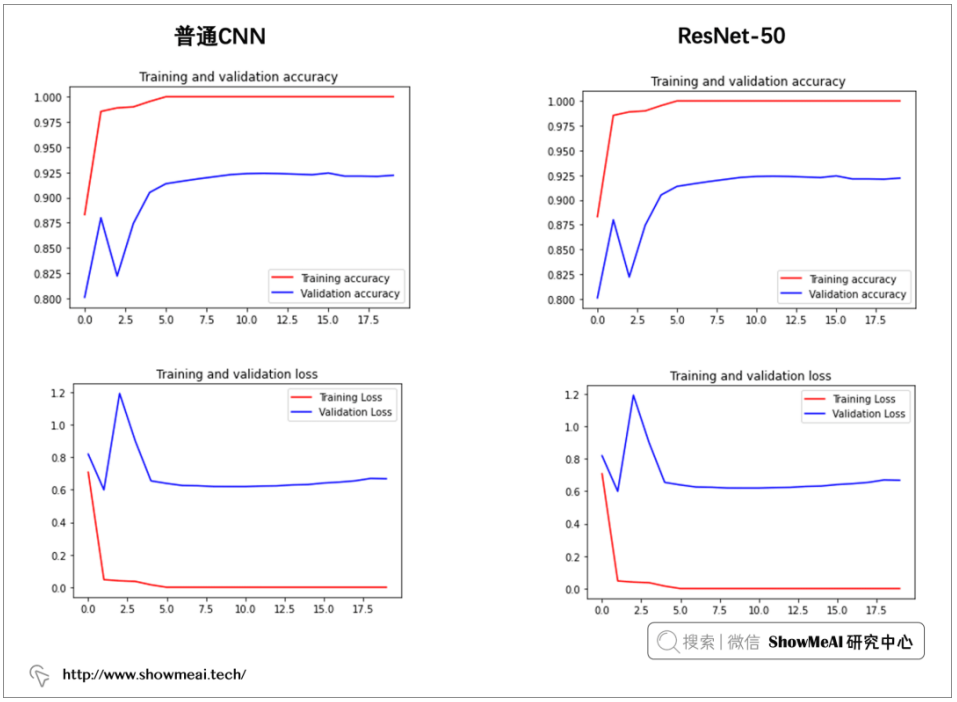
得到如下结果：

优化效果对比

我们对ResNet-50也绘制训练过程中准确率和损失函数的变化，如下：

```
# 获取准确率与损失函数情况
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

# matplotlib绘制训练过程中指标的变化状况
epochs = range(len(acc))

plt.plot(epochs, acc, 'r', 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, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

对比图如下：



我们观察到，从简单的 CNN 模型换到 ResNet 模型时，测试集的准确率从92% 到 97% 。也说明了，ResNet 的结构确实能够带来效果上的提升。

部署与实时测试

在这里我们做一个简单的测试，使用 OpenCV 的视频录制功能，通过 python 收集我们的摄像头的镜头采集的图像并进行实时预测。

> ShowMeAI给OpenCV工具库制作了快捷即查即用的 OpenCV 速查表手册 ，大家可以点击查看和下载。

具体的过程是，我们解析捕获的每一帧图像，将其处理为灰度图（类似于我们模型的训练集），在图像中心抓取一个 400*400 像素的正方形区域（参见 x0,x1,y0,y1），将正方形调整为我们最初的 28x28 大小并使用我们的模型进行测试（之前保存到 .h5 文件）。

```
# 导入工具库
import keras
import numpy as np
from PIL import Image
import string
import pandas as pd
import tensorflow as tf
```

```
# 导入OpenCV
import cv2
from matplotlib import pyplot

## 设定维度
dim = (28, 28) # 图像维度
letters = list(string.ascii_lowercase) # 识别的字母

x0 = 1920 // 2 - 400 # 400px left of center
x1 = 1920 // 2 + 400 # 400px right of center
y0 = 1080 // 2 - 400 # 400px right of center
y1 = 1080 // 2 + 400 # 400px right of center
```

> 查看完整代码 showmeai.tech/article-detail/292

为了更轻松地对预估结果查看，我们把将预测的字母显示在实时画面上（请参阅下面的 gif 以测试单词 `hello` ）。

参考资料

- 深度学习教程 | 吴恩达专项课程 · 全套笔记解读: http://www.showmeai.tech/tutorials/35
- 卷积神经网络解读: http://www.showmeai.tech/article-detail/221
- 深度学习的实用层面: http://www.showmeai.tech/article-detail/216
- 经典CNN网络实例详解: http://www.showmeai.tech/article-detail/222
- 深度学习与计算机视觉教程: http://www.showmeai.tech/tutorials/37
- 卷积神经网络详解: http://www.showmeai.tech/article-detail/264
- Tensorflow 速查手册: http://www.showmeai.tech/article-detail/109
- OpenCV 速查表手册: http://www.showmeai.tech/article-detail/112
- Numpy 速查手册: http://www.showmeai.tech/article-detail/100
- Matplotlib 速查手册: http://www.showmeai.tech/article-detail/103
- https://arxiv.org/ftp/arxiv/papers/1905/1905.05487.pdf
- https://www.analyticsvidhya.com/blog/2021/06/sign-language-recognition-for-computer-vision-enthusiasts

THE END

ShowMeAI研究中心
为AI硬核资料库(cool)而生！
90篇原创内容

公众号

点击『**阅读原文**』，学习**完整代码**！

收录于合集 #AI实战系列 3

下一篇 · 师妹问我怎么搭建神经网络。但这篇讲得也太全了...

阅读原文

喜欢此内容的人还喜欢

提示学习（Prompt Learning）

GAP Lab

SJARACNe-NetBID转录因子活性分析！（二）NetBID实操，GEO数据一站式处理

生信作曲家

【会员风采】渊亭科技企业知识中台打造知识大脑，助力企业智能化升级

厦门软协