目录

[研究的问题 3](#_Toc74409884)

[问题的重要性和意义 3](#_Toc74409885)

[前人工作 4](#_Toc74409886)

[某个针对该问题设计的CNN 4](#_Toc74409887)

[VGG16 4](#_Toc74409888)

[解决方案 5](#_Toc74409889)

[修改前人方案使用Keras的MNIST数据集 5](#_Toc74409890)

[VGG16+全连接层 迁移学习 6](#_Toc74409891)

[分析比较模型结果 7](#_Toc74409892)

[总结 8](#_Toc74409893)

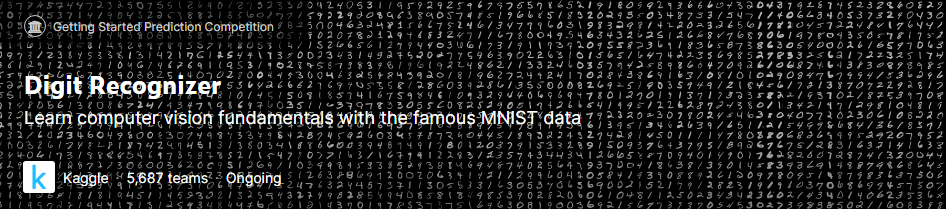
[用于验证方法的数据集 8](#_Toc74409894)

[MNIST 8](#_Toc74409895)

[参考文献 9](#_Toc74409896)

[组员名单 9](#_Toc74409897)

# 研究的问题



该问题来源于Kaggle平台上的一个经典案例Digit Recognizer [1]，目的是将数据集中的手写数字图片识别为数字。

在本次研究中，我简化了前人已有的模型，简要探究了迁移学习的基础方法，并将经典的预训练分类网络VGG16应用在数字识别中，对两者进行对比，以期达到相近的效果。

# 问题的重要性和意义

该问题相当经典，也是入门神经网络的推荐选题，以及不少神经网络方面教程进行讲解的常用例子。但是经过了解，我认为推荐原因主要在于训练目标非常直观，而且数据集较小，使用非专业的设备也能进行模型训练，但是其背后的神经网络设计经常被一笔带过，但实际上仍较复杂。通常，该问题属于深度学习范畴，使用卷积神经网络(CNN)来解决。

传统方案的问题在于模型结构、参数需要完全手工设计。深度学习的核心是特征学习，旨在通过分层网络获取分层次的特征信息，因而需要理解常见结构在具体任务中的作用，如卷积、池化、全连接等，进行对比实验体会不同结构、参数对神经网络性能的影响，这个过程需要大量的先验经验，也是不同神经网络性能瓶颈的主要所在。除此之外，在训练时，所有参数都需要从完全未知的状态开始训练，训练时间长，往往需要多个epochs才能达到预期中较高的准确率。

基于此，我简要探究了迁移学习的方法。迁移学习是将已训练好的模型参数迁移到新的模型来帮助新模型训练，考虑到大部分数据或任务是存在相关性的，所以通过迁移学习可以将已经学到的模型参数，通过某种方式来分享给新模型从而加快并优化模型的学习效率，而不用像通常的传统网络设计流程那样从零学习。

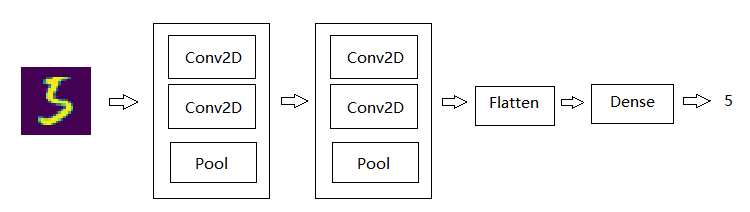
因而我认为，研究该问题的意义在于，当需要应用神经网络处理某些问题时，例如将数字识别从传统的特征提取转变为分类问题，是否可以利用一种通用的预训练分类模型来简化神经网络设计的工作量，加快研究进度的同时能取得可接受的效果。

# 前人工作

## 某个针对该问题设计的CNN



我以Kaggle平台上最热门的开源解决方案为基础，其基于Tensorflow使用Keras搭建神经网络[2]，模型结构如下



# In -> [[Conv2D->relu]\*2 -> MaxPool2D -> Dropout]\*2 -> Flatten -> Dense -> Dropout -> Out

model = Sequential()

model.add(Conv2D(filters=32, kernel\_size=(5, 5), padding='Same',

activation='relu', input\_shape = (28, 28, 1)))

model.add(Conv2D(filters=32, kernel\_size=(5, 5), padding='Same',

activation='relu'))

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), padding='Same',

activation='relu'))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), padding='Same',

activation='relu'))

model.add(MaxPool2D(pool\_size=(2, 2), strides=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation="relu"))

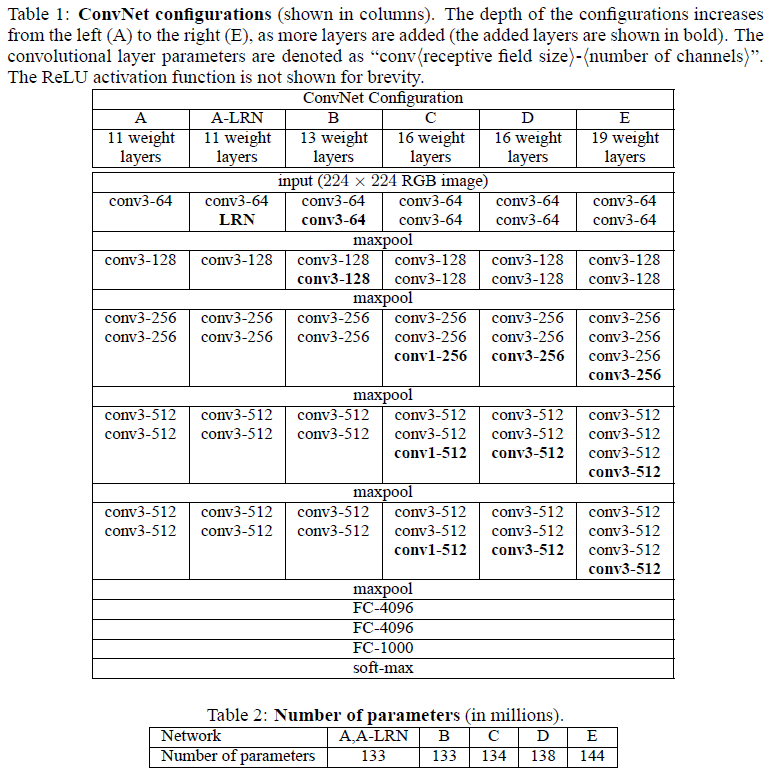
model.add(Dropout(0.5))

model.add(Dense(10, activation="softmax"))

## VGG16

常见的预训练分类网络有牛津的 VGG 模型、谷歌的 Inception模型、微软的 ResNet 模型等，他们都是预训练的用于分类和检测的卷积神经网络(CNN)。

本次选用的是VGG16模型[4]，是一个在ImageNet数据集上预训练的模型，分类性能优秀，对其他数据集适应能力优秀。



上图为原论文中对VGG16模型内部结构的介绍，可以看出相当复杂，但在本次的研究中，并没有准备对该结构进行任何调整，而选择冻结其中所有的预训练参数，仅对这之后的几个必要层进行训练。

# 解决方案

## 修改前人方案使用Keras的MNIST数据集

前人分享的代码使用的是Kaggle提供的csv格式的数据集，将图片以像素为列，存储像素的灰度值。为了简化代码和方便验证两模型的准确率，统一使用Keras包中提供的数据集，训练集和测试集的获取方法如下

from keras.datasets import mnist

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

除此之外，原作者还设计了数据增强部分，在原数据集的基础上随机旋转、平移、缩放、产生噪音，从而更好地聚焦于数字特征的提取，而不是数据集本身。但受限于机器性能，为了缩减模型的训练时间，我删减了该部分功能。

通过如上修改，将该模型再与后续的基于VGG16的迁移学习模型进行比较，分析迁移学习得到的模型准确率水平。

## VGG16+全连接层 迁移学习

我使用了keras.applications.vgg16中的VGG16，在线获取已有的VGG16模型及参数，获取后冻结VGG16中的所有参数进行训练。

在这之后添加一层relu全连接以及用于多分类的softmax全连接，并插入卷积层到全连接层的过渡flatten层等，相较前人设计的CNN而言设计十分简要。

# In -> VGG16 -> Flatten -> Dense -> Dropout -> Dense -> Out

vgg16\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(48, 48, 3))

for layer in vgg16\_model.layers:

layer.trainable = False # freeze VGG16卷积层的参数

model = Sequential()

model.add(vgg16\_model)

model.add(Flatten(input\_shape=vgg16\_model.output\_shape[1:]))

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.4))

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

## 分析比较模型结果

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from keras.datasets import mnist

epochs = 1

test\_total = 10000

df\_CNN = pd.read\_csv("./epochs%d/CNN.csv" % epochs)

df\_VGG16 = pd.read\_csv("./epochs%d/VGG16.csv" % epochs)

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

X\_test\_data = X\_test\_data.astype('float32') / 255.0

X\_test = np.reshape(X\_test\_data, (-1, 28, 28, 1))

Y\_test = Y\_test\_data

err\_CNN = 0

err\_VGG16 = 0

for i in range(test\_total):

res\_CNN = df\_CNN["Label"][i]

res\_VGG16 = df\_VGG16["Label"][i]

if res\_CNN != res\_VGG16:

res\_correct = Y\_test[i]

if res\_CNN != res\_correct:

err\_CNN = err\_CNN + 1

if res\_VGG16 != res\_correct:

err\_VGG16 = err\_VGG16 + 1

plt.imshow(X\_test[i][:, :, 0])

plt.savefig("./epochs%d/%d\_%d\_%d\_%d.jpg" % (epochs, i, Y\_test[i], res\_CNN, res\_VGG16))

print(err\_CNN, err\_VGG16)

训练epochs=1后，通过以上代码，输出两模型对于分类结果预测不一致的测试样例，命名为”样例序号\_参考结果\_前人CNN预测结果\_VGG16预测结果.jpg”，并且输出10000个测试样例中的错误分类数，输出及部分样例如下





观察如上结果，在10000张测试样例中，前人CNN准确率为98.95%，应用了VGG16进行迁移学习的模型准确率为95.65%，虽然结果不及CNN，但是我认为这已经超过了我预期的结果。

## 总结

由于VGG16并不是针对该问题而设计的，而是一个基于ImageNET上1400万张1000类图片而预训练的模型，在我的工作中只添加了必要的全连接层等，就实现了95%以上准确率的分类效果，可以说是较为满意的。

通过观察如上错误样例，能够发现VGG16将一些形状十分类似于另一数字的图片分类成了另一数字，例如右半部分较短、下半部分较长的”4”分类成了”9”，将下半部分极窄的”8”分类成了”9”，能够感受到VGG16更多的是在将形状类似的图片分为一类，而并没有像CNN那样通过(5,5)、(3,3)的kernel聚焦于数字的特征，这在对于分类一些书写并不规范且特殊的数字而言是致命的，但对于正常数字识别而言是能够接受的，要解决该问题，可能需要调整VGG16中的内部结构。

# 用于验证方法的数据集

## MNIST

将数据集中的像素信息转化为图片，由于实际训练中不需要图片信息，在此我仅将测试集的一部分样例转化为图片进行演示，代码如下

for i in range(100):

plt.imshow(X\_test[i][:, :, 0])

plt.savefig("./test/%d.jpg" % i)



# 参考文献

[1]Kaggle.Digit Recognizer[EB/OL].https://www.kaggle.com/c/digit-recognizer,2012.

[2]Yassine Ghouzam.Introduction to CNN Keras - 0.997 (top 6%)[EB/OL]. https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6,2017.

[3]Wikipedia.MNIST database[EB/OL].https://en.wikipedia.org/wiki/MNIST\_database,2021.

[4]K Simonyan, A Zisserman.Very Deep Convolutional Networks for Large-Scale Image Recognition[J].arXiv,2014,1409:1556.

# 组员名单

|  |  |  |
| --- | --- | --- |
| 班级 | 姓名 | 学号 |
| 2018211305 | 索政铎 | 2018211279 |

共1人

非常感谢老师的实验设计和引导！