**工作日志**

**日期**：4.18

**姓名**：韦璐

**项目名称**：散斑图像的神经网络处理

**本周主要工作内容：**论文撰写，代码调试，数据集生成

1. 对于传统散斑图像还原方法进行了调研，具体来说，常用的传统散斑图像还原方法主要有：  
   (a)透射矩阵方法，这种方法的缺陷在于，我们需要对不同介质具体地去测量投射矩阵，并且很容易受到散斑退关联的影响，可能散射介质的很小的扰动会对整个成像过程造成较大的影响。  
   (b)波前调制方法，这部分目前仍在调研中。  
   (c)光学相位共轭方法，这部分目前也仍在调研中。
2. 对神经网络图像处理算法进行了调研和总结，并撰写了论文中对应的章节。主要撰写ResNet，Unet以及DenseNet方法。
3. 详细调试和分析了代码，我计划使用的图像处理代码是基于Unet和Dense block结构的代码，具体的模型代码如下：  
   （a）调用的主要是下面几个模块  
   from \_\_future\_\_ import print\_function  
   from keras.models import Model  
   from keras.layers import Input, MaxPooling2D, UpSampling2D, Dropout, Conv2D, Concatenate, Activation  
   from keras.layers.normalization import BatchNormalization  
   from keras.regularizers import l2  
   （b）我们首先定义重要的卷积部分  
   def conv\_factory(x, concat\_axis, nb\_filter,  
    dropout\_rate=None, weight\_decay=1E-4):  
    x = BatchNormalization(axis=concat\_axis,  
    gamma\_regularizer=l2(weight\_decay),  
    beta\_regularizer=l2(weight\_decay))(x)  
    x = Activation('relu')(x)  
    x = Conv2D(nb\_filter, (5, 5), dilation\_rate=(2, 2),  
    kernel\_initializer="he\_uniform",  
    padding="same",  
    kernel\_regularizer=l2(weight\_decay))(x)  
    if dropout\_rate:  
    x = Dropout(dropout\_rate)(x)  
   return x  
   （c）在构造中我们要使用dense block  
   def denseblock(x, concat\_axis, nb\_layers, growth\_rate,  
    dropout\_rate=None, weight\_decay=1E-4):  
    list\_feat = [x]  
    for i in range(nb\_layers):  
    x = conv\_factory(x, concat\_axis, growth\_rate,  
    dropout\_rate, weight\_decay)  
    list\_feat.append(x)  
    x = Concatenate(axis=concat\_axis)(list\_feat)  
    return x  
   （d）具体的网络构造我们用的是使用了dense block的Unet结构  
   def get\_model\_deep\_speckle():  
    inputs = Input((256, 256, 1))

print("inputs shape:", inputs.shape)

conv1 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(inputs)

print("conv1 shape:", conv1.shape)

db1 = denseblock(x=conv1, concat\_axis=3, nb\_layers=4, growth\_rate=16, dropout\_rate=0.5)

print("db1 shape:", db1.shape)

pool1 = MaxPooling2D(pool\_size=(2, 2))(db1)

print("pool1 shape:", pool1.shape)

conv2 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool1)

print("conv2 shape:", conv2.shape)

db2 = denseblock(x=conv2, concat\_axis=3, nb\_layers=4, growth\_rate=16, dropout\_rate=0.5)

print("db2 shape:", db2.shape)

pool2 = MaxPooling2D(pool\_size=(2, 2))(db2)

print("pool2 shape:", pool2.shape)

conv3 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool2)

print("conv3 shape:", conv3.shape)

db3 = denseblock(x=conv3, concat\_axis=3, nb\_layers=4, growth\_rate=16, dropout\_rate=0.5)

print("db3 shape:", db3.shape)

pool3 = MaxPooling2D(pool\_size=(2, 2))(db3)

print("pool3 shape:", pool3.shape)

conv4 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool3)

print("conv4 shape:", conv4.shape)

db4 = denseblock(x=conv4, concat\_axis=3, nb\_layers=4, growth\_rate=16, dropout\_rate=0.5)

print("db4 shape:", db4.shape)

pool4 = MaxPooling2D(pool\_size=(2, 2))(db4)

print("pool4 shape:", pool4.shape)

conv5 = Conv2D(1024, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool4)

print("conv5 shape:", conv5.shape)

db5 = denseblock(x=conv5, concat\_axis=3, nb\_layers=4, growth\_rate=16, dropout\_rate=0.5)

print("db5 shape:", db5.shape)

up5 = Conv2D(512, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(

UpSampling2D(size=(2, 2))(db5))

print("up5 shape:", up5.shape)

merge5 = Concatenate(axis=3)([db4, up5])

print("merge5 shape:", merge5.shape)

conv6 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge5)

print("conv6 shape:", conv6.shape)

db6 = denseblock(x=conv6, concat\_axis=3, nb\_layers=3, growth\_rate=16, dropout\_rate=0.5)

print("db5 shape:", db6.shape)

up6 = Conv2D(256, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(

UpSampling2D(size=(2, 2))(db6))

print("up6 shape:", up6.shape)

merge6 = Concatenate(axis=3)([db3, up6])

print("merge6 shape:", merge6.shape)

conv7 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge6)

print("conv7 shape:", conv7.shape)

db7 = denseblock(x=conv7, concat\_axis=3, nb\_layers=3, growth\_rate=16, dropout\_rate=0.5)

print("db7 shape:", db7.shape)

up7 = Conv2D(128, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(

UpSampling2D(size=(2, 2))(db7))

print("up7 shape:", up7.shape)

merge7 = Concatenate(axis=3)([db2, up7])

print("merge7 shape:", merge7.shape)

conv8 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge7)

print("conv8 shape:", conv8.shape)

db8 = denseblock(x=conv8, concat\_axis=3, nb\_layers=3, growth\_rate=16, dropout\_rate=0.5)

print("db8 shape:", db8.shape)

up8 = Conv2D(64, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(

UpSampling2D(size=(2, 2))(db8))

print("up8 shape:", up8.shape)

merge8 = Concatenate(axis=3)([db1, up8])

print("merge8 shape:", merge8.shape)

conv9 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge8)

print("conv9 shape:", conv9.shape)

db9 = denseblock(x=conv9, concat\_axis=3, nb\_layers=3, growth\_rate=16, dropout\_rate=0.5)

print("db9 shape:", db9.shape)

conv10 = Conv2D(32, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(db9)

print("conv10 shape:", conv10.shape)

conv11 = Conv2D(2, 1, activation='softmax')(conv10)

print("conv11 shape:", conv11.shape)

1. 生成散斑图代码也进行了调试。

散斑图生成样图，原图数字7，来自于mnist数据集：



**下周的工作计划**：

1. 下周开始我将继续详细撰写学位论文的前面一部分，首先是传统散斑图还原方法，其次是首先是卷积神经网络以及他的各种图像处理的变形结构，比如典型的全连接卷积神经网络，VGG，Resnet，Unet等，最后是我们所使用的有dense block的Unet。
2. 下周我将优化训练数据的模拟生成。
3. 下周我将继续调试图像处理的神经网络代码，并在华为云上面进行运行测试