高级实训任务二U-Net图像分割实验报告

1.任务描述

● 将卷积神经网络（CNN）应用在图像分割任务上，我们需要对网络结构进行设计。 ● 需要提交博客报告以及GitHub代码仓库。 ● 可选的任务：图像实例分割、语义分割、医疗图像分割。 ● 可选的网络结构：YOLO v3、Mask R-CNN、U-Net。 ● 可选的数据集：参考下文： ● https://zhuanlan.zhihu.com/p/50925449 ● 可选深度学习框架：Tensorflow、PyTorch、Keras。 ● 提交结果：项目报告、答辩幻灯片、相关代码和测试用例。

2.任务开始准备

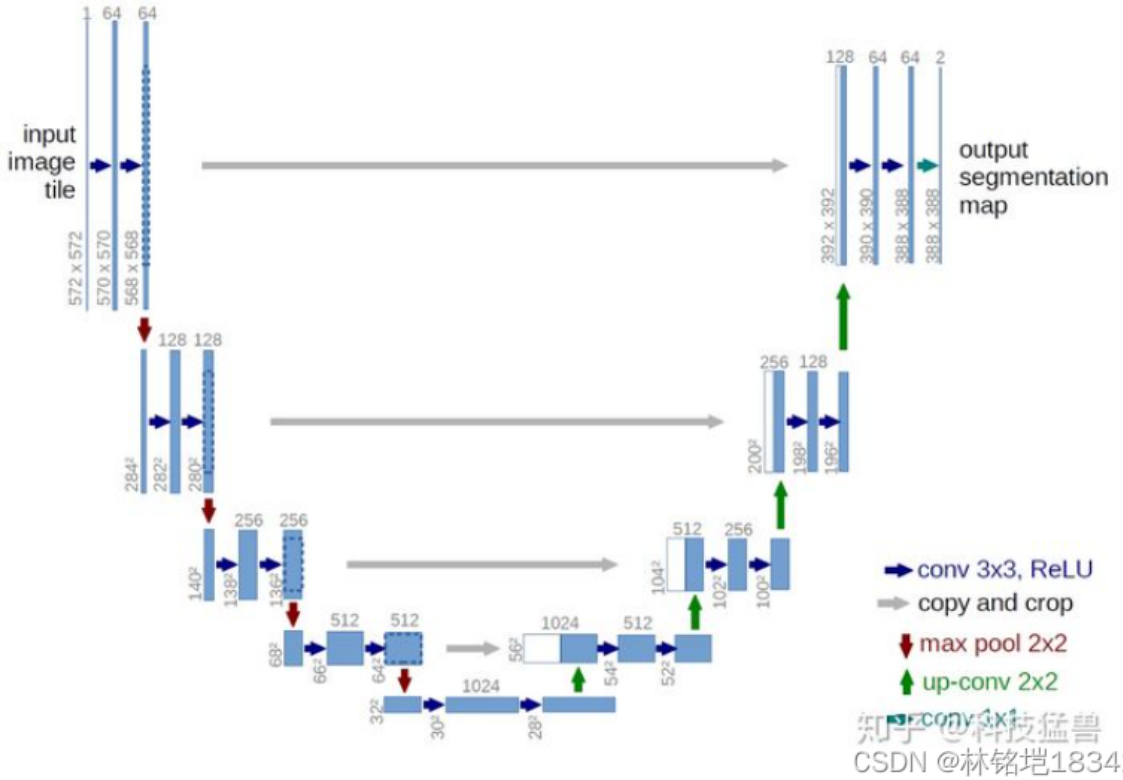
U-Net原理：

1.Unet 介绍

Unet 发表于 2015 年，属于 FCN 的一种变体。Unet 的初衷是为了解决生物医学图像方面的问题，由于效果确实很好后来也被广泛的应用在语义分割的各个方向，比如卫星图像分割，工业瑕疵检测等。Unet 跟 FCN 都是 Encoder-Decoder 结构，结构简单但很有效。Encoder 负责特征提取，你可以将自己熟悉的各种特征提取网络放在这个位置。由于在医学方面，样本收集较为困难，作者为了解决这个问题，应用了图像增强的方法，在数据集有限的情况下获得了不错的精度。

2.Unet 网络结构

- Encoder
  - **Encoder**



如上图，Unet 网络结构是对称的，形似英文字母 U 所以被称为 Unet。整张图都是由蓝/白色框与各种颜色的箭头组成，其中，蓝/白色框表示 feature map；蓝色箭头表示 3x3 卷积，用于特征提取；灰色箭头表示 skip-connection，用于特征融合；红色箭头表示池化 pooling，用于降低维度；绿色箭头表示上采样 upsample，用于恢复维度；青色箭头表示 1x1 卷积，用于输出结果。其中灰色箭头copy and crop中的copy就是concatenate而crop是为了让两者的长宽一致。Encoder 由卷积操作和下采样操作组成，文中所用的卷积结构统一为 3x3 的卷积核，padding 为 0 ，striding 为 1。没有 padding 所以每次卷积之后 feature map 的 H 和 W 变小了，在 skip-connection 时要注意 feature map 的维度(其实也可以将 padding 设置为 1 避免维度不对应问题)。 更多可以参考：Unet详解

常见的图像分割损失函数有交叉熵，dice系数，FocalLoss等。今天我将分享图像分割FocalLoss损失函数及Tensorflow版本的复现。

3.图像分割unet_Tensorflow入门

FocalLoss介绍

FocalLoss思想出自何凯明大神的论文《Focal Loss for Dense Object Detection》，主要是为了解决one-stage目标检测中正负样本比例严重失衡的问题。

FocalLoss是在交叉熵函数的基础上进行的改进，改进的地方主要在两个地方

(1)、改进第一点如下公式所示。

$$L_{fl} = \begin{cases} -(1-y')^{\gamma} log y' & , \quad y = 1 \\ -y'^{\gamma} \log(1-y') & , \quad y = 0 \end{cases}$$

首先在原有交叉熵函数基础上加了一个权重因子，其中gamma>0，使得更关注于困难的、错分的样本。比如：若 gamma = 2，对于正类样本来说，如果预测结果为0.97，那么肯定是易分类的样本，权重值为0.0009，损失函数值就会很小了；对于正类样本来说，如果预测结果为0.3，那么肯定是难分类的样本，权重值为0.49，其损失函数值相对就会很大；对于负类样本来说，如果预测结果为0.8，那么肯定是难分类的样本，权重值为0.64，其损失函数值相对就会很大；对于负类样本来说，如果预测结果为0.1，那么肯定是易分类的样本，权重值为0.01，其损失函数值就会很小。而对于预测概率为0.5时，损失函数值只减少了0.25倍，所以FocalLoss减少了简单样本的影响从而更加关注于难以区分的样本。

(2)、改进第二点如下公式所示。

$$L_{fl} = \begin{cases} -\alpha(1-y')^{\gamma} log y' & , \quad y = 1 \\ -(1-\alpha) y'^{\gamma} \log(1-y') & , \quad y = 0 \end{cases}$$

FocalLoss还引入了平衡因子alpha，用来平衡正负样本本身的比例不均匀。alpha取值范围0~1，当alpha>0.5时，可以相对增加y=1所占的比例，保证正负样本的平衡。

2、FocalLoss公式推导

在github上已经可以找到很多FocalLoss的实现，如下二分类的FocalLoss实现。实现其实不是很难，但是在实际训练时会出现NAN的现象。

```
def focal_loss_sigmoid(labels,logits,alpha=0.25,gamma=2):
    """
    Computer focal loss for binary classification
    Args:
      labels: A int32 tensor of shape [batch_size].
      logits: A float32 tensor of shape [batch_size].
      alpha: A scalar for focal loss alpha hyper-parameter. If positive samples number
      > negtive samples number, alpha < 0.5 and vice versa.
      gamma: A scalar for focal loss gamma hyper-parameter.
    Returns:
      A tensor of the same shape as `lables`
    """
    y_pred=tf.nn.sigmoid(logits)
    labels=tf.to_float(labels)
    L=-labels*(1-alpha)*((1-y_pred)*gamma)*tf.log(y_pred)-\
      (1-labels)*alpha*(y_pred**gamma)*tf.log(1-y_pred)
    return L
```


3、FocalLoss代码实现

按照上面导出的表达式FocalLoss的伪代码可以表示为：

$$focalloss = tf.\log 1p(tf.\exp(-x)) * (weight\_a + weight\_b) + x * weight\_b$$

其中，

$$weight\_a = alpha * tf.pow((1 - y\_pred), gamma) * y\_gt$$

$$weight\_b = (1 - alpha) * tf.pow(y\_pred, gamma) * (1 - y\_gt)$$

$$x = \log(\frac{y\_pred}{})$$

从这里可以看到1-y_pred项可能为0或1，这会导致log函数值出现NAN现象，所以好需要对y_pred项进行固定范围值的截断操作。最后在TensorFlow1.8下实现了该函数。

```
import tensorflow as tfdef focal_loss(y_true, y_pred, alpha=0.25, gamma=2):
epsilon = 1e-5    y_pred =
tf.clip_by_value(y_pred, epsilon, 1 - epsilon)    logits = tf.log(y_pred / (1 -
y_pred))    weight_a = alpha * tf.pow((1
- y_pred), gamma) * y_true    weight_b = (1 - alpha) * tf.pow(y_pred, gamma) * (1
- y_true)    loss = tf.log1p(tf.exp(-
- logits)) * (weight_a + weight_b) + logits * weight_b    return
tf.reduce_mean(loss)
```

3.任务实现方法

工具选择 语言：python 框架选择：tensorflow，keras

此外还用了absl库函数。用absl库定义可重复的代码段，定义训练的各个参数（比如学习率，周期数，每个周期的训练轮数等）。

- tensorflow定义

```
'''
参数设定：eopch=4，step=100，batch_size=2，learning_rate=0.0002
'''
inputs = tf.keras.layers.Input((512, 512, 1))
        # Contracting part
        conv1 = tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(inputs)
        conv1 = tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv1)
        assert conv1.shape[1:] == (512, 512, 64)
        pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv1)
        assert pool1.shape[1:] == (256, 256, 64)
        conv2 = tf.keras.layers.Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(pool1)
        conv2 = tf.keras.layers.Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv2)
        assert conv2.shape[1:] == (256, 256, 128)
        pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv2)
        assert pool2.shape[1:] == (128, 128, 128)
       ng='same', kernel_initializer='he_normal')(
            tf.keras.layers.UpSampling2D(size=(2, 2))(drop5))
        assert up6.shape[1:] == (64, 64, 512)
        merge6 = tf.keras.layers.concatenate([drop4, up6], axis=3)
        assert merge6.shape[1:] == (64, 64, 1024)
        conv6 = tf.keras.layers.Conv2D(512, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(
            merge6)
        conv6 = tf.keras.layers.Conv2D(512, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv6)
        assert conv6.shape[1:] == (64, 64, 512)

        up7 = tf.keras.layers.Conv2D(256, 2, activation='relu', padding='same',
kernel_initializer='he_normal')(
            tf.keras.layers.UpSampling2D(size=(2, 2))(conv6))
        assert up7.shape[1:] == (128, 128, 256)
        merge7 = tf.keras.layers.concatenate([conv3, up7], axis=3)
        assert merge7.shape[1:] == (128, 128, 512)
        conv7 = tf.keras.layers.Conv2D(256, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(
            merge7)
        conv7 = tf.keras.layers.Conv2D(256, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv7)
        assert conv7.shape[1:] == (128, 128, 256)

        up8 = tf.keras.layers.Conv2D(128, 2, activation='relu', padding='same',
```

```
kernel_initializer='he_normal')(
            tf.keras.layers.UpSampling2D(size=(2, 2))(conv7))
        assert up8.shape[1:] == (256, 256, 128)
        merge8 = tf.keras.layers.concatenate([conv2, up8], axis=3)
        assert merge8.shape[1:] == (256, 256, 256)
        conv8 = tf.keras.layers.Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(
            merge8)
        conv8 = tf.keras.layers.Conv2D(128, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv8)
        assert conv8.shape[1:] == (256, 256, 128)

        up9 = tf.keras.layers.Conv2D(64, 2, activation='relu', padding='same',
kernel_initializer='he_normal')(
            tf.keras.layers.UpSampling2D(size=(2, 2))(conv8))
        assert up9.shape[1:] == (512, 512, 64)
        merge9 = tf.keras.layers.concatenate([conv1, up9], axis=3)
        assert merge9.shape[1:] == (512, 512, 128)
        conv9 = tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(merge9)
        conv9 = tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv9)
        assert conv9.shape[1:] == (512, 512, 64)
        conv9 = tf.keras.layers.Conv2D(2, 3, activation='relu', padding='same',
kernel_initializer='he_normal')(conv9)
        assert conv9.shape[1:] == (512, 512, 2)
        conv10 = tf.keras.layers.Conv2D(num_classes, 1, activation='sigmoid')
(conv9)
        assert conv10.shape[1:] == (512, 512, num_classes)

        model = tf.keras.Model(inputs=inputs, outputs=conv10)
```

- data

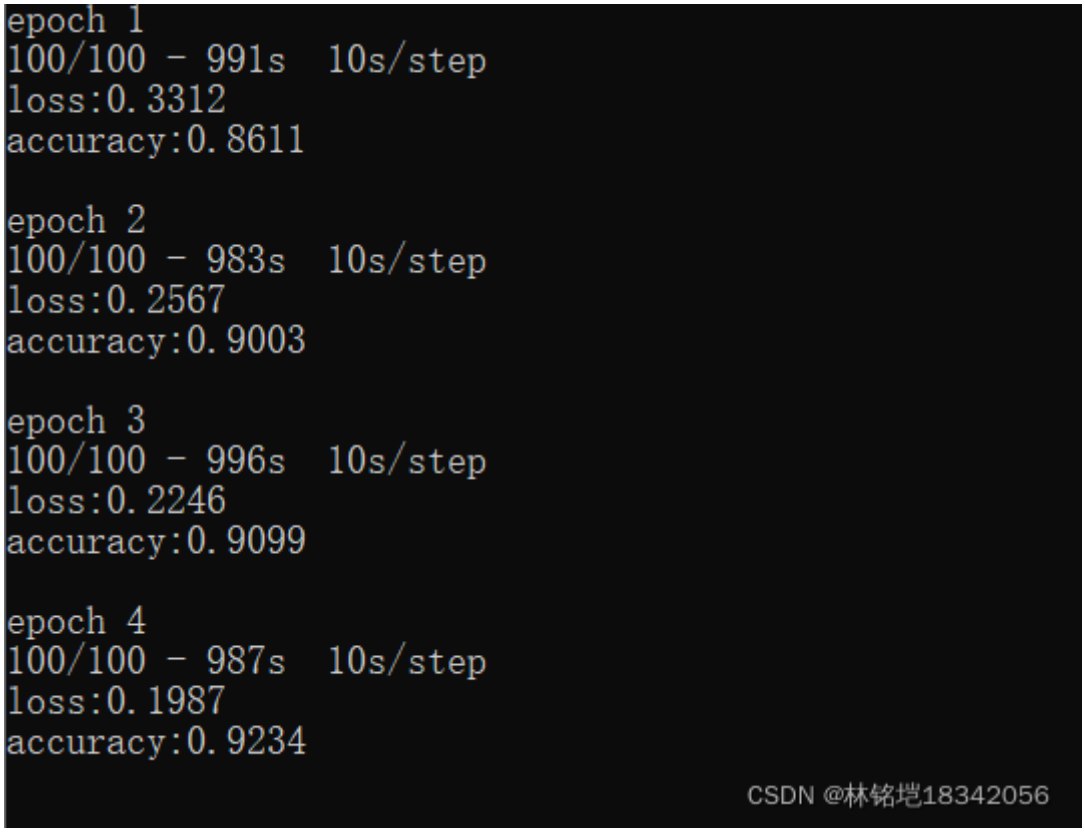
```
import os
import tensorflow as tf
import numpy as np
import scipy.misc

class Read_TFRecords(object):
    def __init__(self, filename, batch_size=64,
        image_h=256, image_w=256, image_c=1, num_threads=8, capacity_factor=3,
min_after_dequeue=1000):
        self.filename = filename
        self.batch_size = batch_size
        self.image_h = image_h
        self.image_w = image_w
        self.image_c = image_c
        self.num_threads = num_threads
        self.capacity_factor = capacity_factor
        self.min_after_dequeue = min_after_dequeue
```

```
    def read(self):
        reader = tf.TFRecordReader()
        filename_queue = tf.train.string_input_producer([self.filename])
        key, serialized_example = reader.read(filename_queue)
        features = tf.parse_single_example(serialized_example,
            features={
                "image_raw": tf.FixedLenFeature([], tf.string),
                "image_label": tf.FixedLenFeature([], tf.string),
            })

        image_raw = tf.image.decode_jpeg(features["image_raw"],
channels=self.image_c,
            name="decode_image")
        image_label = tf.image.decode_jpeg(features["image_label"],
channels=self.image_c,
            name="decode_image")
        if self.image_h is not None and self.image_w is not None:
            image_raw = tf.image.resize_images(image_raw, [self.image_h,
self.image_w],
                method=tf.image.ResizeMethod.BICUBIC)
            image_label = tf.image.resize_images(image_label, [324, 324],
                method=tf.image.ResizeMethod.BICUBIC)
        image_raw = tf.cast(image_raw, tf.float32) / 255.0 # convert to float32
        image_label = tf.cast(image_label, tf.float32) / 255.0 # convert to
float32

        input_data, input_masks = tf.train.shuffle_batch([image_raw,
image_label],
            batch_size=self.batch_size,
            capacity=self.min_after_dequeue + self.capacity_factor *
self.batch_size,
            min_after_dequeue=self.min_after_dequeue,
            num_threads=self.num_threads,
            name='images')

        return input_data, input_masks
```

- train

```
  # train
        if self.is_training:
            self.training_set = tf_flags.training_set
            self.sample_dir = "train_results"

            # makedir aux dir
            self._make_aux_dirs()
            # compute and define loss
            self._build_training()
            # logging, only use in training
            log_file = self.output_dir + "/Unet.log"
```

```
            logging.basicConfig(format='%(asctime)s [%(levelname)s] %(message)s',
                                filename=log_file,
                                level=logging.DEBUG,
                                filemode='w')
            logging.getLogger().addHandler(logging.StreamHandler())
        else:
            # test
            self.testing_set = tf_flags.testing_set
            # build model
            self.output = self._build_test()
```



运行结果： 由运行结果可以看出来周期数的推进，训练准确度不断提高。但是由于没有服务器，只能进行4轮的训练，但是也能看出随着周期数增加，准确率不断提升的。

4.个人总结

通过本次实验，我也进一步了解了用Tensorflow定义神经网络的相关操作，以及通过论文的阅读，资料的查阅，以及开源代码的学习，能够学习到Unet 网络在图像处理，图像分割上的基本应用。不仅个人编程技能上取得进步，同时也增强了资料查询，论文阅读能力，学会自己动手学习新的知识，克服畏难心理。