人工智能与商业创新期末报告

Project 1、病理图像识别

姓 名: 孟 念 Muennighoff Niklas
学 号: 1800092850
班 号: 02839210

人工智能与商业创新
(秋季, 2021)

北京大学
光华管理学院
彭一杰老师
高华西助教

2021 年 12 月 15 日

北京大学
PEKING UNIVERSITY

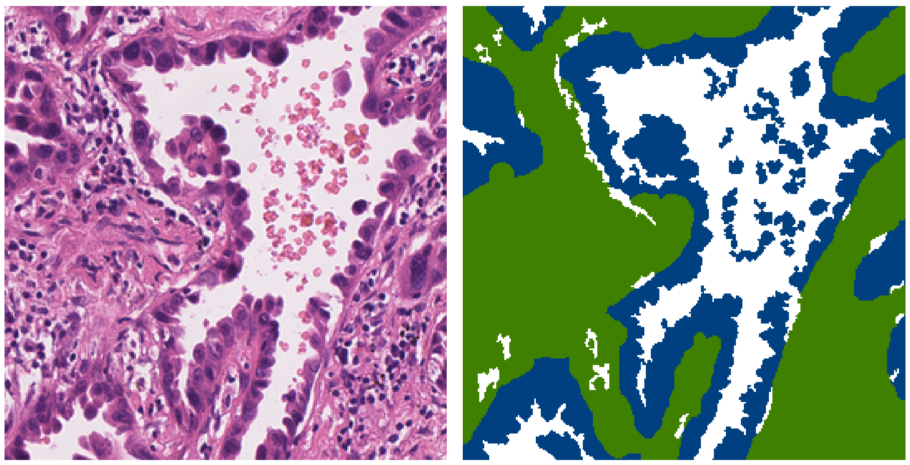
目 录

1 Introduction

This work constitutes the final report for the WSSS4LUAD Challenge dataset. The dataset consists of 10,000 images. The task as outlined in the course materials can be split into two parts:

- **Image Classification**: In this module the goal is to classify each image as to whether it contains a tumor epithelial tissue, tumor stromal tissue or normal tissue or any combination of these.
- **Image Segmentation**: In this module, the different tissue types and the background need to be annotated on a pixel-level basis for each image.

My solution to both of these approaches has been presented in the Mid-Term report. This final report looks at some additional examples from the data and the code. The code is also provided alongside the report.



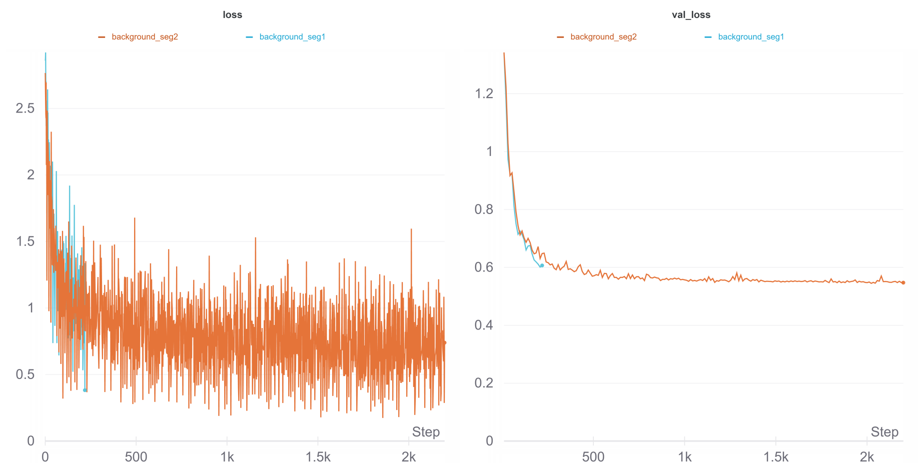
(a) Ground truth image (b) Ground truth mask

图 1: An example image and mask from the validation set. For the mask, white (255, 255, 255) corresponds to non-labeled background, turquoise (64, 128, 0) to stroma, dark blue (0, 64, 128) to tumor. The label is hence **T+S** (Tumor + Stroma).

2 Training progress

Figure 2 shows the loss curves of the segmentation model. Overall we can see that both training and validation loss decrease over time. The model hence seems to be learning something.

The accuracy curves in Figure 3 show a consistent increase in model accuracy over time. Note that accuracy here is measured on a pixel level, i.e. for each pixel whether the model did the correct prediction. Both training and validation accuracy converge to around 90%. As it’s the same for both, the model does not seem to overfit to the training set. This is thanks to the use of dropout and augmentation.



(a) Training loss (b) Validation loss

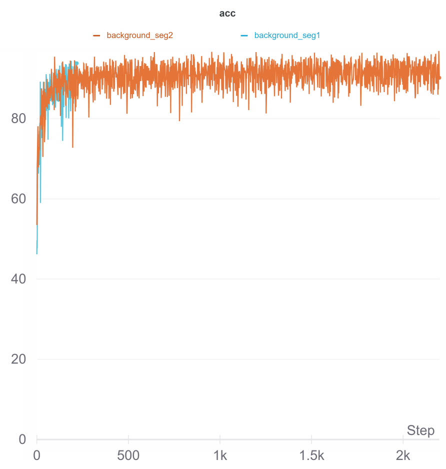
图 2: Training and validation loss tracked in Weights and Biases

3 Samples

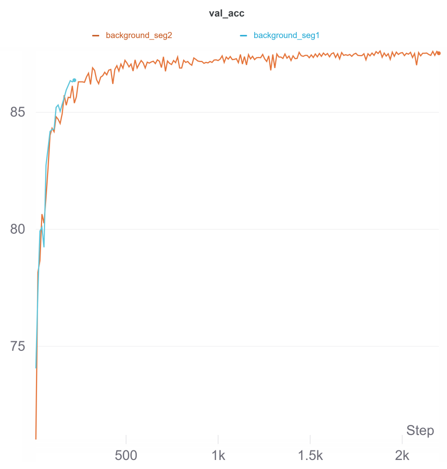
In this section we will look at some additional output predictions from the model.

Figure 4 shows an example where the model correctly includes all classes. The segments align well with the ground truth mask. An accuracy somewhere around 90% as seen in 3 seems reasonable.

Figure 5 shows an example where the model adds an incorrect class to its predictions. The ground truth image in the figure shows a darker area where the mis-prediction occurs. From the ground truth image in the 1 section, we can see that the mis-predicted class (dark green) corresponds to

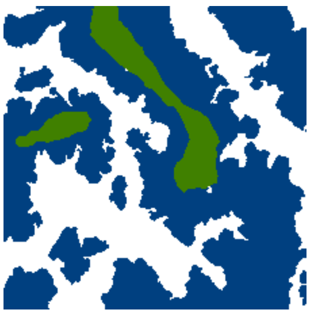


(a) Training accuracy

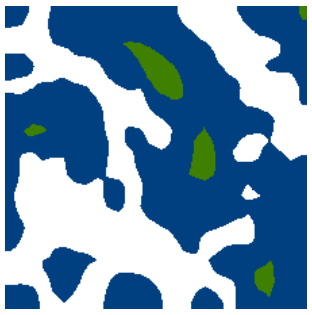


(b) Validation accuracy

图 3: Training and validation accuracy tracked in Weights and Biases



(a) Ground truth mask



(b) Predicted mask

图 4: Example label and prediction: The model does a decent job. About 90% accuracy seems reasonable

a darker area in the ground truth image. Taking this into account we can explain the mistake of the model.

4 Code

I used PyTorch for the entire project. I use the same loop structure for training and validation set. Only for training, gradient optimization is performed. After every X steps results are logged to weights and biases or printed to stdout. See the training code below:

```
import copy
import time
```

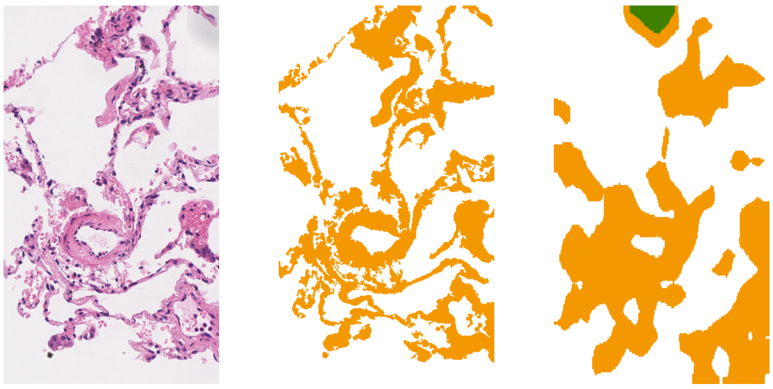

(a) Ground truth image (b) Ground truth mask (c) Predicted mask

图 5: Example label and prediction: The model makes an obvious mistake adding a class not present

```
def train_model(model, criterion, optimizer, scheduler=None, num_epochs=25,
report_step=50, use_wandb=False):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_loss = np.inf

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss, running_corrects = [], []

            # Iterate over data.
            for step, (inputs, masks) in enumerate(dataloaders[phase]):
                inputs = inputs.to(device)
                masks = masks.to(device).squeeze() # Remove the Grayscale dim

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    loss = criterion(outputs['out'].squeeze(), masks.long())
                    _, preds = torch.max(outputs['out'].detach(), 1)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                        # Scheduler will update every X steps as specified in its config
                        if scheduler: scheduler.step()

                # statistics
                running_loss.append(loss.item() * inputs.size(0))
                running_corrects.extend((preds ==  masks.data).flatten().tolist())

                if (phase == 'train') and (step % report_step == 0):
                    report_loss = sum(running_loss) / len(running_loss)
                    report_corrects = round(sum(running_corrects) / len(running_corrects), 4) * 100

                    if use_wandb:
                        wandb.log({"loss": epoch_clf_loss, "acc": epoch_corrects})
                    else:
                        print('{} Step: {} Loss: {:.4f} Acc: {:.2f}%'.format(phase,
                        step, report_loss, report_corrects))

                    running_loss, running_corrects = [], []

            if phase == 'val':
                epoch_loss = sum(running_loss) / len(running_loss)
                epoch_corrects = round(sum(running_corrects) / len(running_corrects), 4) * 100

                print('{} Step: {} Loss: {:.4f} Acc: {:.2f}%'.format(phase, step,
                epoch_loss, epoch_corrects))
                # deep copy the model
                if epoch_loss < best_loss:
                    best_loss = epoch_loss
                    best_model_wts = copy.deepcopy(model.state_dict())


    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val loss: {:4f}'.format(best_loss))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

5 Conclusion

In this report, I have provided additional info to the Mid-term report. I had already finished all tasks outlined in the project introduction including the optional segmentation part in the Mid-term report.