Computational Biology - TIL Score Prediction

Leon Chipchase

May 25, 2024

For this assignment, we implement a complete pathology workflow. We aim to estimate the TIL score by analysing whole-slide images (WSIs) of H&E stained breast cancer tissue slides.

I Tissue Segmentation

Our first step in this process is to segment the tissue, separating the glass from the tissue itself. This reduces the complexity of the analysis task since we will have identified where the tissue itself is as well as where we can focus on it. For this task, we use the TIAToolbox inbuilt function *reader.tissue˙mask()*, where the *reader* is an object of type *WSIReader*. This returns a x by y numpy array with a boolean value where yellow means tissue and blue means glass.

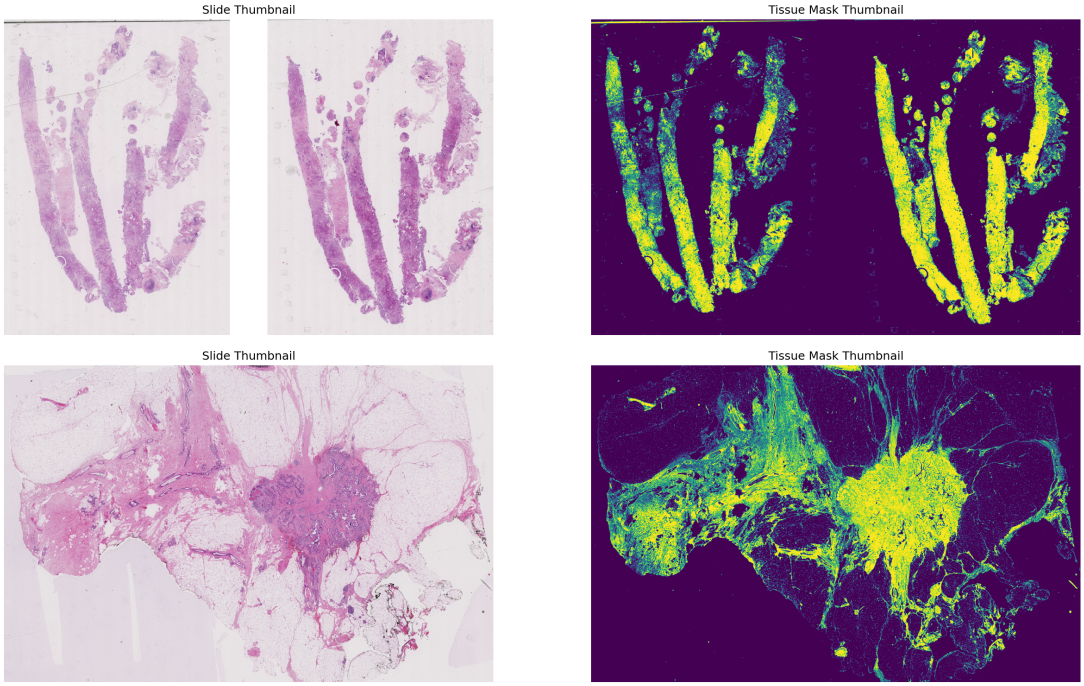

Figure 1: Segmented Images

II TILs Scoring Algorithm

After segmenting the tissue from the glass, we can begin our analysis. This is a very involved multiple-step process. Our approach was to utilise transfer learning from the ResNet50 neural network. According to [Mascarenhas and Agarwal(2021)], this network performs best (highest accuracy) on image classification tasks when compared to almost all others.

Data Distribution: First, to examine the distribution of our training labels, we create the following plot. Since the number of samples we have is very small (40 labelled samples). It is important to understand that the model may not be able to capture all the relevant data effectively. Figure 2 shows the distribution of the TILs scores. Evidently, there is a much higher mass towards the lower TILs scores. This may mean

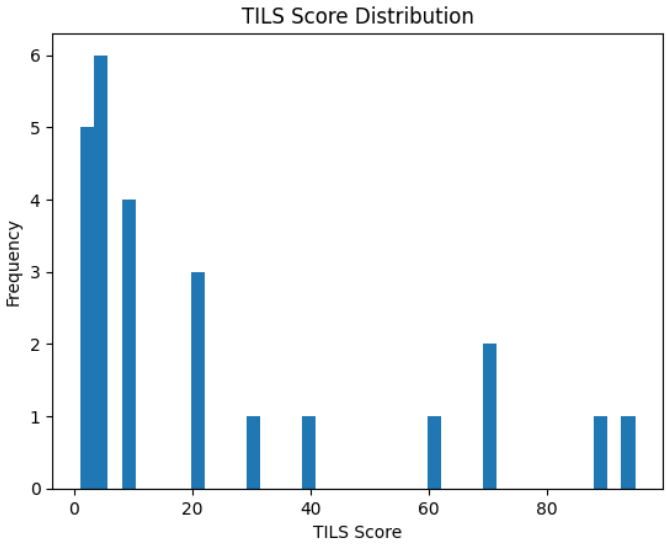

Figure 2: TILs Score Distribution

that the model we train could have a bias towards lower TIL scores in our predictions. **Preprocessing:** Our first step is to extract patches from the WSIs which have a high proportion of tissue. This step is very simple, and since the tissue mask consists of only boolean values, we can very simply calculate a threshold consisting of 70% or more of tissue as follows.

$$tissue\_proportion = np.mean(tissue\_mask)$$

An example of ten patches extracted from the image above are shown in Figure 3.It is clear that this method effectively extracts the sections of the image which are likely to be of use to us in our prediction task.

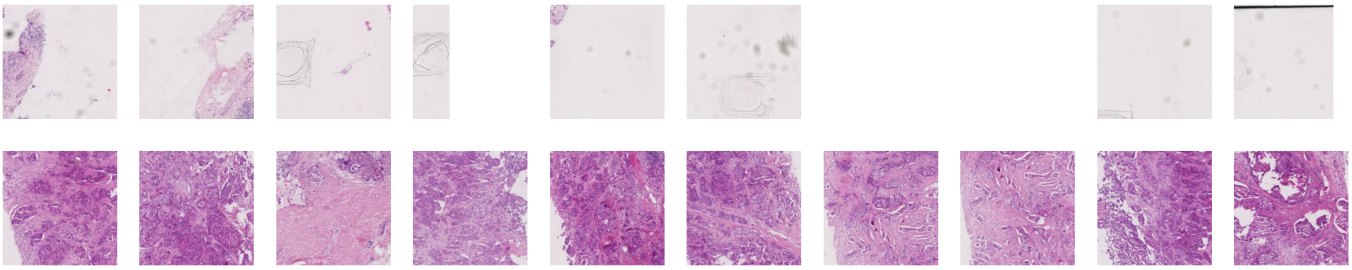

Figure 3: Low Density (top) and High Density (bottom) Patches Extracted from Figuire 1

Before this step, we use stain normalisation to normalise the images. This allows us to highlight important areas of the tissue as well as increase the tissue contrast. The normalisation we used was the TIAToolbox - ReinhardNormalizer().

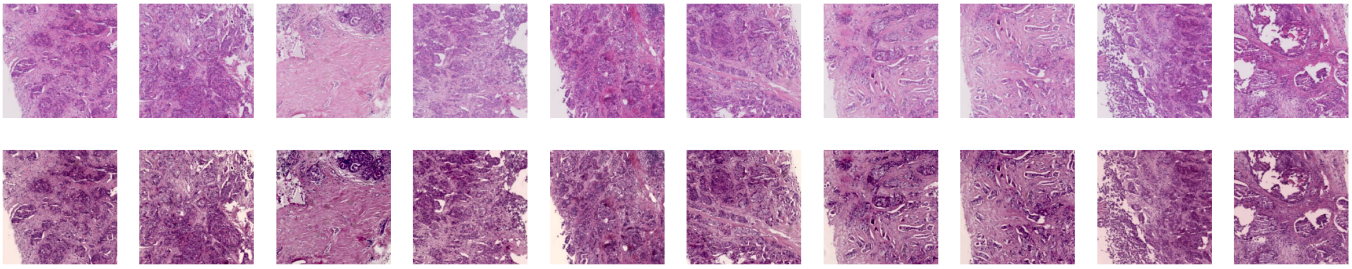

Figure 4: Original (Top) vs Stain Normalised Images

Creating our Dataset Once we were able to extract regions of the image with high tissue densities, we then selected 30 $(224, 224)$ size patches at 3 mpp. This seemed a reasonably sufficient level of detail to

extract images from, showing a large number of nuclei per image. Some examples of these images are shown in Figure 5 below.

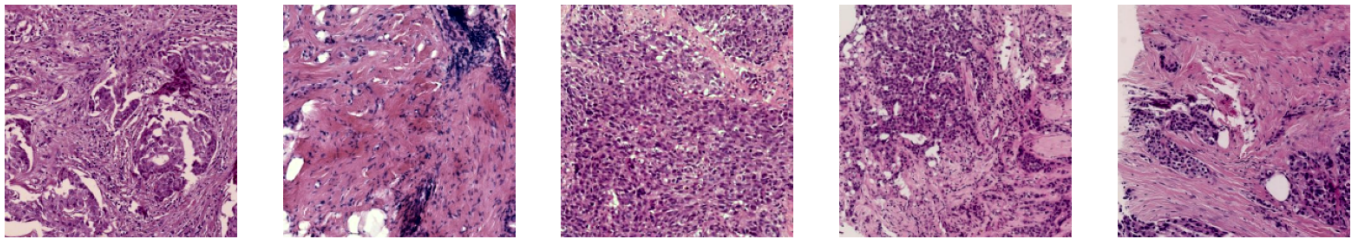

Figure 5: 5 Random Training Images

Transfer Learning: We use transfer learning in our approach for two reasons. Firstly, we have a very limited dataset consisting of only 40 images; therefore, training a new neural network on purely these images would likely result in a very ineffective model. Secondly, designing an effective neural network - especially a deep NN. For this reason, we opted to use ResNet50, a model which performs extremely well on image classification tasks. For this reason, we assume that the model may perform most effectively in these tasks.

Transforming Our Data: One of the first things we do before training a model is to define a transformer. We use the same default transformer as ResNet50, which we define as:

```
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((patch_size, patch_size)), # Resize to 224 by 224 - default resnet
    transforms.ToTensor(), # Load to Tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Additional Pre-Processing: Due the complexity of the data in our models, as well as to avoid overfitting, we add some noise to the images. This accounts for the difference between the training images and the test images we may come across. This is performed using the following method:

```
patch = np.load(f"data/{img_path}") # Load the patch
noise = np.random.normal(0, 20, patch.shape) # Create an array of random numbers
blur = cv2.GaussianBlur(noise, (7, 7), 2) # Add a blur
noise = np.clip(blur, 0, 255).astype(np.uint8) # Convert to the right format
patch = patch + noise # update our patch with the blurred noise
```

Model Architecture: In Figure 7, we see the original ResNet50 Architecture.

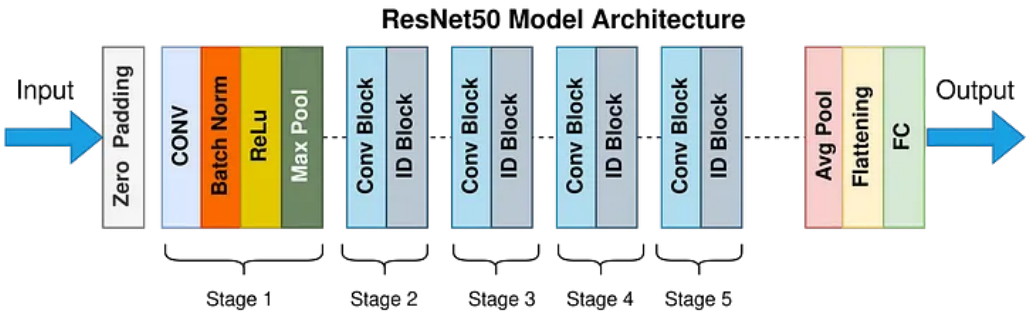

Figure 6: Original ResNet50 Model Architecture (Source [Kundu(2023)])

To adapt our model, we first unfreeze the first and last two layers. Firstly, this allows our input data to adapt to the ResNet50 model, ensuring that the loss can adapt the previous layers to accommodate for any

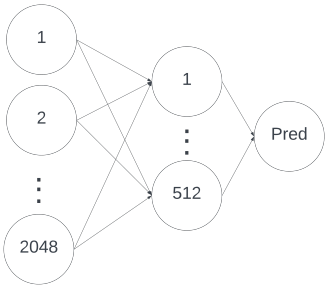

Figure 7: Updated Final Layer

differences in our dataset. then we unfreeze the last two layers, and add on a final layer of the following structure:

This includes a ReLU function between the layers with 2048 and 512 layers; this introduces non-linearity to the final layers of the model. Additionally.

Hyper-Parameter	**Value**	**Reasoning**
epochs	10	An epoch of 10 allowed for sufficient training with minimal overfitting.
batch_size	32	This keeps memory usage low, whilst allowing for sufficient training. speed. As well as capturing the nuances in each batch.
learning rate	0.0005	Through various testing, we determined that $lr = 0.0005$ was the optimal value since it ensured the model converged most effectively.
Loss	MSELoss	**(Chosen)** MSELoss was considered due to us optimising our model for a regression task for values between 0 and 1.
Loss	L1Loss	Whilst not selected, this was considered due to the model initially struggling to accommodate small differences in TILs scores, and MSELoss reduces its error when in the decimal range.

Table 1: Model Hyper Parameters

Our Results: To assess our model, we used Leave One Out Cross Validation. We iterated through each of the WSIs in our training set, exclude that WSI from the training, then predicted its TILs score. The results for this are shown in Figure 8. Table 2 shows our predicted results for the test images.

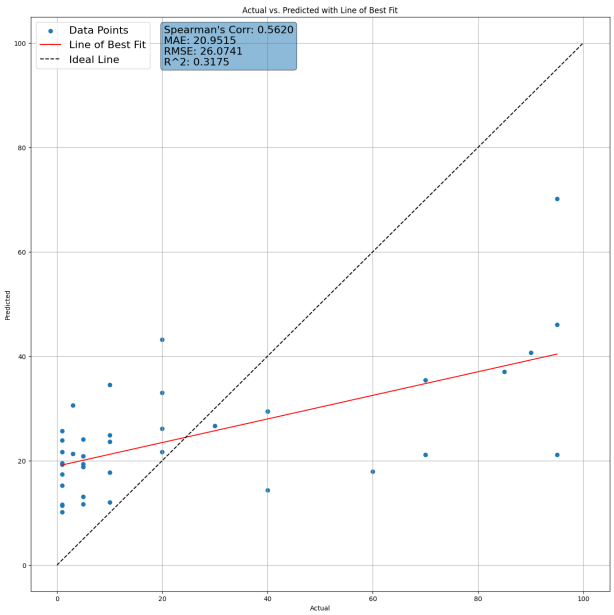

Figure 8: Leave One Out - Cross Validated Metrics

Image-Id	Predicted TILs Score)
TC_S01_P000055_C0001_B101	45.84
TC_S01_P000058_C0001_B101	35.47
TC_S01_P000061_C0001_B101	59.97
TC_S01_P000063_C0001_B101	27.93
TC_S01_P000064_C0001_B101	11.46
159S	18.78
160B	24.86
161S	32.72
162S	31.73
165B	19.31

Table 2: Predicted Scores for Test Set

Though the Spearman Correlation Coefficient and the $R2$ values were not very high, the model performed well overall, it is clear that it has some struggle differentiating the higher value TILs scores. This was expected given the reduced number of samples of higher TILs scores in the original dataset. However, regardless, this is somewhere the model could perform better on.

If this was attempted another time, some areas we could improve or adapt are using a smaller number of microns per pixel to get higher detail. To utilise a UNet Architecture [Bajwa(2023)] to segment the image further before beginning analysis, as well as explore various approaches and compare different staining methods to determine which gave the best results.

III Visualising Results

The process of visualising our results first meant we iterated in 224 by 224 chunks through the image. We got the mask, the original patch, and the stained patch for prediction. From here, we checked if the tissue coverage was above 0.3; if so, we would predict the TILs score, or else we would assign a value of zero. From here, we would have an array of predictions - for each of the tiles in the image. To reduce the harshness of the boundaries, we essentially split predictions and averaged them between each of its neighbours, which smooths the transition between one tile and another. Finally, we applied a blur to smooth the image further. Figure 9 shows the visualisations of our algorithm on the Tissue. Clearly, the model captures this information well, since the right image has a TILs score of 0.95, whereas the left image has a TILs score of 0.1.

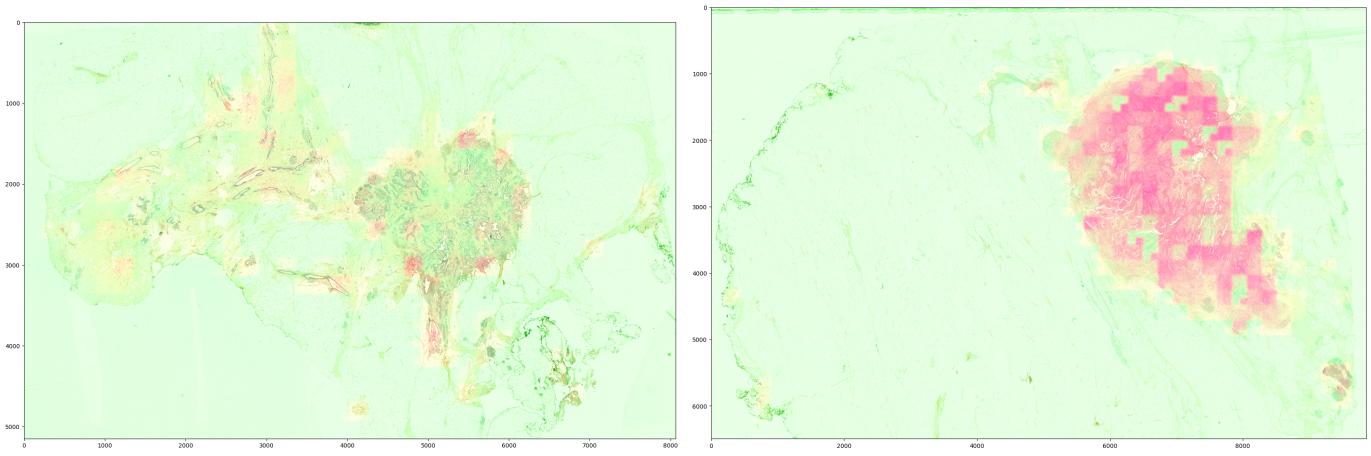

Figure 9: Visualisations of Two WSIs. Left has TILs Score 0.1, Right has TILs Score (0.95).

References

[Bajwa(2023)] Angad Bajwa. Image segmentation with u-net, Nov 2023. URL https://www.analyticsvidhya.com/blog/2022/10/image-segmentation-with-u-net/.

[Kundu(2023)] Nitish Kundu. Exploring resnet50: An in-depth look at the model architecture and code implementation, Jan 2023. URL https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f.

[Mascarenhas and Agarwal(2021)] Sheldon Mascarenhas and Mukul Agarwal. A comparison between vgg16, vgg19 and resnet50 architecture frameworks for image classification. In *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, volume 1, pages 96–99, 2021. doi: 10.1109/CENTCON52345.2021.9687944.