ECSE 415 Final Project: Medical Imaging

Asma Gandour
asma.gandour@mail.mcgill.ca
260989101
Antonio Lopez
antonio.lopez@mail.mcgill.ca
261129754
Mathieu-Joseph Magri
mathieu-joseph.magri@mail.mcgill.ca
260928498
Mohammed Ridwan Mahbub
mohammed.mahbub@mail.mcgill.ca
260909180
Alban Moreon
alban.moreon@mail.mcgill.ca
260791985

School of Computer Science
McGill University
Canada
April 12th 2023

1 Introduction

In the field of computer vision, many different types of systems and computers are used to extract important information and data over a set of images or even videos. Computer vision allows us to then make much better analysis in the decision making process which in turn allows us to make better and more informed decisions in many fields of study such as the medical industry, especially for medical imaging. Indeed, medical Imaging is an ever developing field of computer vision that has gained in great importance over the last few years due to its potential to change the medical world. Overall, medical imaging is a fascinating and important field that has the potential to transform the way we diagnose and treat medical conditions. However, it is a difficult field due to the complexity of the human body, and many technical challenges. Despite these challenges, medical imaging still plays a critical role in modern healthcare. In the scope of this current project, medical imaging will grant us the possibility to automatize labelling and recognition of different types of nuclei in a given medical image. This is an extremely important and valuable task as it allows medical specialists to segment and detect given medical images with much greater efficiency without having to do it manually themselves.

2 Datasets

The dataset used is available here. The dataset contains medical images from different patients. For each image, a csv file and a mask are provided.

2.1 Structure

The folder organization of the data is the following:

```
ECSE 415
├── csv
├── mask
└── rgb
```

- **rgb**: Input images. All images are at 0.2 microns-per-pixel. All coordinate values in tables are pixel units at 0.2 microns-per-pixel.
- **csv**: Each image has a corresponding csv file which contains the coordinates of the bounding boxes/polylines that surround the nuclei. For each nucleous, there are three different types of classification.
- **mask**: Masks for the input image. The first channel in the mask encodes the label value. The product of the second and third channels encode the unique instance label for each nucleus. The fov area (gray) is included in the first channel of the mask.

The naming convention used is in the following format:

- **Example**: `TCGA-A1-A0SP-DX1_id-[..]_left-[a]_top-[b]_bottom-[c]_right-[d]`
- The first four strings uniquely identify a patient.
- The second string identifies the hospital, in this example `A1`.
- The third and fourth string identify the patient id in the hospital `A0SP-DX1`.
- `a,b,c,d` are coordinates.

The dataset contains multiple samples of the same subject, this means that multiple input images have the same prefix that, as we saw, identifies a patient.

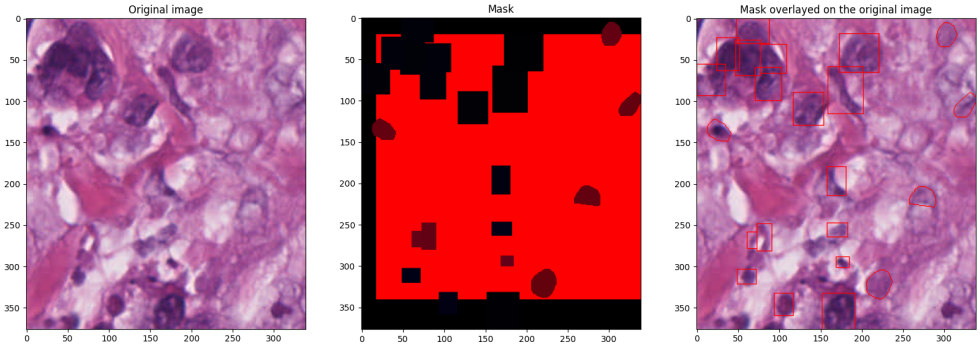


Figure 1: The rgb image, mask and the mask applied to the original image

2.2 Preprocessing

The main preprocessing steps are as follows:

- **Sample Grouping**: For each patient, there are several samples to aggregate.
- **Dataset Split**: The data was splitted according to the suggested division in the csv files.
- **Data Visualization**: An input image of a given patient, the corresponding mask and the overlay of the last two are displayed in *Figure*1.

Furthermore, training and validation sets were created by looping over all the images, obtaining distinct subjects. A training and validation split following the recommended 80/20 split was applied, meaning that 80% of patients were put in a training directory while the remaining 20% of data was put in a validation directory. Following this split, there should be approximately 100 unique subjects in the training set.

3 Segmentation

In this section we will apply different segmentation techniques to separate the medical images from the background generating a binary mask. Each image has a ground truth mask that will be used as term of comparison.

3.1 Unsupervised Learning

Two unsupervised segmentation methods were evaluated: K-means segmentation and Quick Shift. The following section will explain how these methods were implemented and discuss the results obtained. Finally, the two methods will be compared based on quantitative and qualitative results.

3.1.1 K-means Segmentation

Method The performance of K-means segmentation was evaluated for many different number of clusters K. Average DICE coefficients (averaged over all training images) were computed for each value of K from 2 to 6. Two masks were generated to compute DICE coefficients: the ground-truth mask and the binary mask obtained from K-means segmentation. The first mask was obtained by drawing white polylines (with *fillPoly* from OpenCV) based on CSV files. The second mask was obtained by applying a threshold (with threshold from OpenCV) on the output of K-means segmentation. It should be noted that this segmentation is done by using the function kmeans from OpenCV. Once the two masks are generated, the DICE coefficient is computed with the following formula.

$$DICE = \frac{2TP}{TP + TN + FP + FN} [1]$$

TP is the number of pixels that are white in both masks and TN is the number of pixels that are black in both masks. FP is the number of pixels that are white in the K-means mask and black in the ground truth mask. Finally, FN is the number of pixels that are black in the K-means mask and white the ground truth mask.

Results and Discussion Table 1 shows DICE coefficients obtained for each number of clusters K. The highest DICE coefficient is observed at K=6. Overall, varying K does not affect the DICE coefficients very much because binary masks obtained from K-means segmentation are relatively the same for different values of K (see figure 2 below).

Table 1: Average DICE coefficient for each value of K

K	DICE Coefficient
2	0.16914643351874373
3	0.17003148779558294
4	0.16625633261467382
5	0.16789588234566372
6	0.17119668953873968

Figure 2: Binary masks for K from 2 to 6

The above figure 2 shows a sample set of binary masks for values of K from 2 to 6. Even if the segmentation becomes more and more precise as K increases, the binary mask is relatively the same for K=3, K=5 and K=6 because of the threshold. At K=4, the binary mask is different because the threshold is too low. Hence, to improve the performance of K-means segmentation, the threshold should be specific to the number of clusters.

Based on DICE results obtained above, the value of 6 was chosen for K. The performance of K-means segmentation on RGB images was compared to that of grayscale images. Table 2 shows the results for three different images. Using grayscale images instead of RGB images for K-means segmentation does slightly improve DICE

Table 2: Average DICE coefficient for three RGB and grayscale images at K=6

Image	RGB	Grayscale
1	0.19765987025023168	0.19950571516836577
2	0.36672966247359734	0.3726045203054584
3	0.16937805804320563	0.17257520357515058

coefficients. This might be due to the way the function *kmeans* deals with input images with a single color channel.

3.1.2 Quick Shift

Method The second method used is quick shift [5], a non parametric unsupervised segmentation method. Quick shift is an image segmentation technique used in computer vision and image processing. It involves grouping similar pixels together to form regions or segments based on color, texture, and spatial proximity. Quick shift works by treating the image as a weighted graph, where each pixel is connected to its neighbors. The algorithm then iteratively shifts each pixel towards its most similar neighbor, eventually converging to form coherent segments. The process is computationally efficient and faster if compared to mean shift [2]. One crucial parameter in this algorithm is the **cut-off** value, an higher value means fewer cluster since two cluster are divided accordingly to this threshold value.

Results and Discussion In Table 3 we can see the DICE score for the two cut-off values chosen. The highest score is achieved with a cut-off of 100. Overall, varying the cut-off is not affecting the DICE coefficient very much. The segmented image appears to be coherent with the original image while the binary mask is missing some zones of interest. The binary mask is created using a threshold value to separate the background from the foreground.

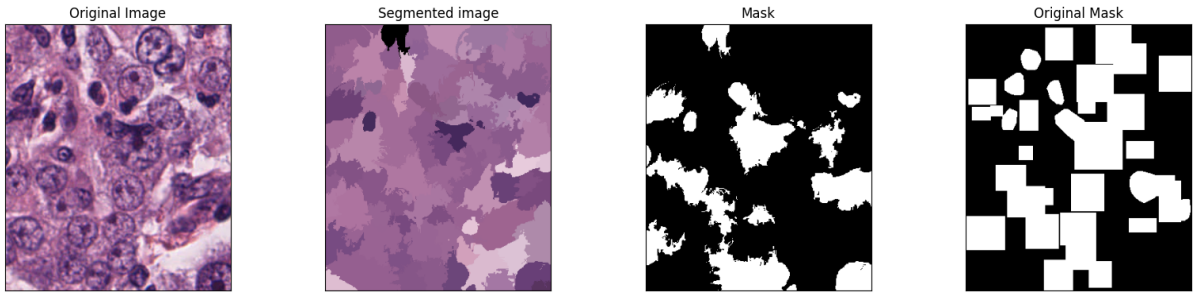


Figure 3: Example of segmentation using quick shift on a random image

Table 3: Average DICE coefficient for each cut-off value

Cut-off	DICE Coefficient
50	0.124102
100	0.138799
150	0.124087

3.1.3 Comparison

As shown in section 3.1.1, binary masks can be easily generated by thresholding the output of K-means segmentation. However, to obtain good performance, the threshold must be specific to the number of clusters K. Moreover, the sensibility of the segmentation can be easily adjusted by varying the K parameter. On the other hand, it is hard to generate the relative binary mask from the output of quick shift and, like K-means segmentation, the cut threshold is a crucial parameter. The main advantage of quick shift is that it is faster than mean shift and a variable number of clusters can be used which is not the case for K-means segmentation.

3.2 Supervised Learning

Supervised learning segmentation was performed by training a Random Forrest Classifier to segment the medical images. The training data set provided had labels for every pixel which means the data can be organized as data and labels, formatted in *x-train* and *y-train* respectively.

3.2.1 Approach

Data Selection The data set was first organized and pre-processed to provide the classifier with sufficient data to be trained effectively however the total dataset posed memory constraints during training. Using the training and testing split method from section 2, the program loops through each subject and picks a certain number of samples at random. This is because not all subjects in the data set contain an equal number of samples, meaning if samples were taken at random, the training data set might be biased towards some types of nuclei groups more than others. The label for each pixel was also obtained from the `.csv` file for each corresponding pixel.

Supervised Method The train and validation split was trained using both Random Forest Classifier and SVM. However, as there are multiple output classes, Random Forest Classifier outperformed SVM in terms of computational intensity and accuracy. In order to find the optimal hyperparameters for the Random Forest Classifier, the `RandomizedSearchCV` method was used where multiple iterations with different parameters were tried to train the model. The final parameters chosen for the classifier were `max_depth=10, n_estimators=100 class_weight='balanced'`.

3.2.2 Performance on Training and Validation Images

Metrics Using the train and validation split from section 2, metrics were obtained for the model. The accuracy of the training data set is 49.0% and the validation data set is 46%. Note that as the Random Forest model was trained on a limited data set, some classes had limited to no examples for the model to pick. This can be seen from the confusion matrix present in the Colab. The classifier struggles with some classes which had few examples to look at. To further investigate the performance of the model, more metrics are displayed below.

class	precision	recall	f1-score	support
0	85%	64%	73%	33696984
1	31%	15%	20%	1757853
2	28%	18%	22%	8442370
4	0%	9%	1%	42545
5	0%	33%	0%	43515
6	1%	10%	2%	55084
7	0%	19%	1%	164025
8	0%	0%	0%	352802
9	4%	1%	2%	668413
11	0%	0%	0%	2416957
12	0%	9%	0%	25734
accuracy			49%	47666282
macro avg	14%	16%	11%	47666282
weighted avg	66%	49%	56%	47666282

(a) Metrics for Training Dataset

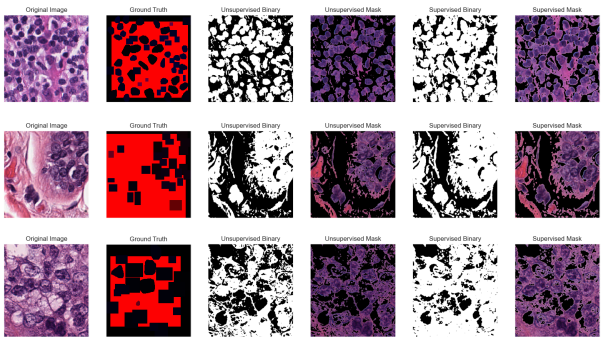
class	precision	recall	f1-score	support
0	83%	61%	70%	31600343
1	32%	14%	19%	1815314
2	25%	17%	20%	8052650
3	0%	0%	0%	2415
4	0%	4%	0%	53077
5	1%	24%	1%	166679
6	1%	4%	1%	76360
7	0%	15%	0%	121959
8	0%	0%	0%	296835
9	8%	1%	2%	1281184
10	0%	0%	0%	53575
11	0%	0%	0%	1910478
12	0%	0%	0%	0
accuracy			46%	45430869
macro avg	12%	11%	9%	45430869
weighted avg	63%	46%	53%	45430869

(b) Metrics for Validation Dataset

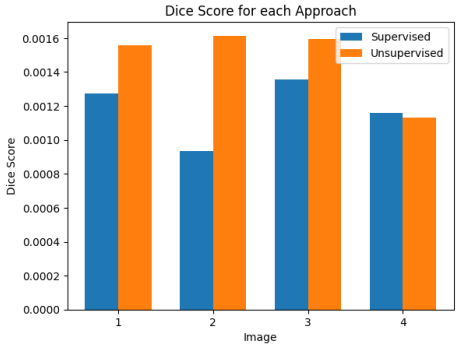
Figure 4: Metrics for the Random Forest Classifier

Looking at the metrics 4, it can be seen that prediction for `class 0` is highest for both cases. This is as expected as it was the unlabelled class and occurred the most. The implications of an uneven data set can also be seen in the difference in classes for both predictions. As the model was trained on a smaller subset of samples from the training split, the classifier encountered fewer variations compared to the whole validation split. To counter the imbalance in the frequency of class occurrence, the hyperparameter `class_weight='balanced'` was included during training which improved the model predictions moderately.

3.2.3 Performance of Supervised and Unsupervised Approach



(a) Example of Supervised and Unsupervised Approach



(b) DICE score for different approaches

Figure 5: Comparison between supervised and unsupervised

Differences As can be seen above 5, both supervised and unsupervised approaches work comparably similarly to each other. Qualitatively, the unsupervised approach works better when it comes to segmenting small cluster-like nucleotides. On the other hand, the supervised approach works well for larger, even textured samples. The random forest classifier struggles quite a bit when it comes to ambiguous structures where there isn't much variation in texture and color. Quantitatively, the DICE score was calculated for each approach, and it can be seen that it varies based on the type of image. As mentioned above, the unsupervised approach outperforms the supervised method when the samples have small circular nucleotides giving a better DICE score. Conversely, the supervised approach provides a better dice score when structures are larger and more textured.

Conclusion When comparing the results of the supervised approach by itself, from the different nuclei groups, the random forest classifier was able to better segment individual structures from each other when pixel intensity was greater and the texture variation was higher as can be seen below 6. This is likely because the classifier struggles with details that do not have sharp edges making boundaries between classes harder to detect. The limited training sample also means some classes are harder to classify. The unbalanced class weights make the classifier more likely to be biased toward some classes.

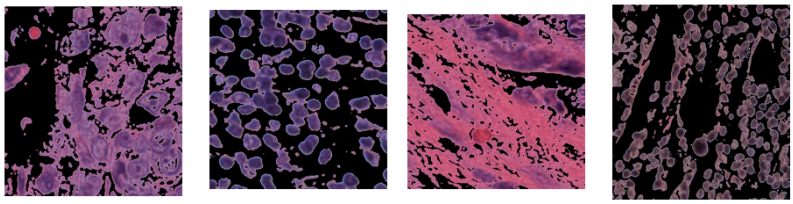


Figure 6: Example of segmentation using Supervised Approach

4 Classification

In this section, a supervised learning framework using an SVM was used to detect and count tumors. We will first discuss the processing and training pipeline with the choice of parameters. Finally, we will discuss the performance and results obtained.

4.1 Processing Pipeline

The processing pipeline starts by converting a subject's image to grayscale. Once done, the coordinates of a region of interest are obtained from the CSV data and a mask is created. The mask is applied on the grayscale image to only retain the region The masked image is resized to a constant predefined size in order to extract HoG features. The masked image is also labeled as per its class. This process is performed for all regions of interest per image. The set of features and labels for all regions are then fed to the SVM for fitting. We used the SDGClassifier from the scikit-learn library as it enables to partially fit data and avoid storing the features for all images in RAM before fitting as we where running into RAM space issues attempting the latter. Once the SVM is fitted with all the data from the training set, it can be used to predict the pathology of a new region by applying the same processing pipeline beforehand. The count of tumors per image can be obtained by predicting the class of each region and summing the regions predicted as tumors.

The parameters for the different step of the processing pipeline where obtained by attempting to maximise the prediction accuracy and minimise the RMSE of the tumor count per image. The final parameters chosen are the following:

- Image resized to 100 x 100 pixels.
- HoG cell size of 3 x 3 pixels
- HoG block size of 2 x 2 cells
- Number of orientation bins: 8.

4.2 Performance

The performance of the classifier was computed by running the same processing pipeline on each image of the specified dataset and using the extracted region's features to predict its pathology. The pathology was divided in two classes: "tumors" and "non-tumors" (all the other pathologies). The results are presented in Table 4. We notice that the classifier has very low false positives rates but high false negatives rates. This could be considered quite problematic in a real-world scenario as some tumors would be missed in the detection. The RMSE was only computed on the validation dataset.

Table 4: Performance metrics per dataset

Dataset	Prediction Accuracy	False Positives	False Negatives	RMSE
Training dataset	70.19%	1.57%	28.23%	-
Validation dataset	62.64%	1.73%	35.63%	19.78

4.3 Performance on unmasked images

In order to train the classifier on unmasked images, the pipeline was slightly altered to remove the masking step. The class types were unchanged but if an image contains at least 1 tumor blob, it is labeled as such. The performance metrics in Table 5 were obtained using the same methods as the previous section. We can notice a significant decrease in accuracy when the classifier is trained with unmasked images. As such the classifier is now used to detect if an image contains one or more tumors but would not be able to classify the pathology of a specific region of interest.

Table 5: Performance metrics per dataset for unmasked images

Dataset	Prediction Accuracy	False Positives	False Negatives
Training dataset	55.16%	3.16%	41.68%
Validation dataset	42.06%	3.44%	54.48%

4.4 Performance for other pathological structures

For this section, five pathology classes were used: tumor, fibroblast, lymphocyte, plasma cell and macrophage. The same pipeline was used. The prediction accuracy per class is presented in Table 6 below were we can see that the lymphocyte pathology has the highest prediction accuracy. This could be explained by the fact that lymphocytes have a dark color which results in a high constrast blob in the images, making it more easily recognizable.

Table 6: Prediction accuracy per pathology class

Pathology	Prediction Accuracy
Tumor	27.97%
Fibroblast	33.44%
Lymphocyte	66.80%
Plasma Cell	7.36%
Macrophage	3.84%

5 Bonus Question

In this section we will use a pre-trained deep learning architecture to perform the segmentation task that we saw in section 3. Since the model that we will use is not designed to our use case we will fine tune it to take advantage of the previous knowledge embedded in the network and specialize it to our case.

5.1 Architecture

Over the last few years, deep learning models have introduced a new approach of image segmentation models with remarkable performance improvements. Deep Learning based image segmentation models often achieve the best accuracy rates on popular benchmarks, resulting in a paradigm shift in the field. One of the popular architectures used for image segmentation is U-Net [3]. It is a fully convolutional network composed by an encoder and a decoder. It consists of the following:

- **Contracting path**:

- 3×3 convolution (with no padding).
- A ReLU as activation.
- 2×2 max pooling with a stride of 2.
- At each down-sampling step we double the number of feature channels.

- **Expansive path**:
  - Up-sampling of the features.
  - 2×2 convolution that halves the number of the channels.
  - Concatenation with the corresponding features of the encoder.
  - Two 3×3 convolutions, each followed by a ReLU.

- **Final layer**: 1×1 convolution used to map the 64-component feature vector to the desired number of classes.

The implementation used in this project is the one provided by "Segmentation Models". This library provides several architectures for image segmentation written using python and PyTorch. All these models can be trained from scratch or can be fine tuned (our case).

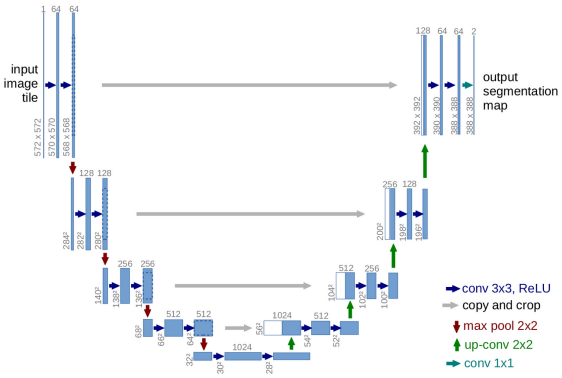


Figure 7: U-Net architecture [3]

5.2 Pre processing

Before fine tuning our network we need to preprocess our data according to the architecture requirements. The steps are as follows:

- **For the rgb images (input)**:
  - Convert to grayscale.
  - Resize to $(320, 320)$.
  - Convert to tensor.

- **For the mask (target)**:
  - Generate the binary mask using the bounding box coordinates (1 channel image).
  - The black pixels must have value 0 and the white one value 1.
  - Resize the mask.
  - Convert to tensor.

Since loading all the data set in memory is unfeasible the data is dynamically loaded and preprocessed.

5.3 Training

For training the network we set the following parameters:

- **Adam optimizer** with learning rate of 0.0001
- **Loss function**: DICE loss [4].
- **Metrics**: IoU score $\frac{|A \cap B|}{|A \cup B|}$.
- **Epochs**: 100.

At the end of the training the best model is saved.

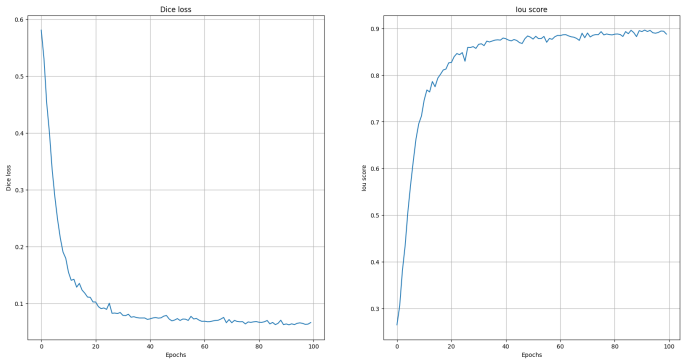
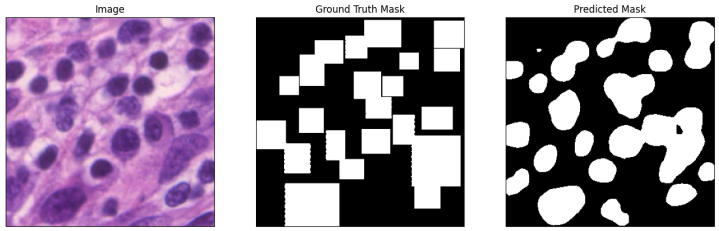


Figure 8: Loss and IoU score over epochs

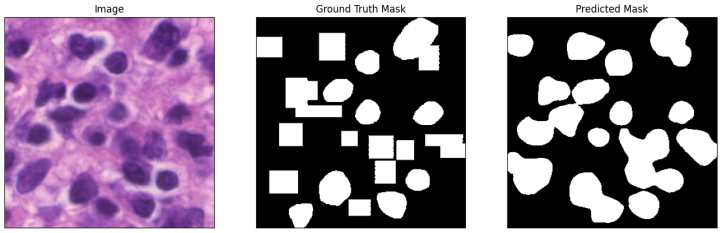
5.4 Results and Discussion

Once the model is trained, we test it over our test images with the following results:

- **IoU score**: 0.6327
- **DICE score**: 0.8487



(a) Test sample 1



(b) Test sample 2

Figure 9: In order: rgb image, real mask, predicted mask

As we can see, also with a limited number of training samples (50), the model presents good qualitative 9 and quantitative results. Most areas are correctly segmented and represented in the predicted mask. Comparing this

model to others, it was possible to notice a significant difference in terms of DICE score and quality of the predicted masks. This is consistent with what we expected, deep learning models have outperformed normal methods based on manual features of extraction. It is also significant to see how using pretrained models and fine tuning them for a specific task brings great results. The amount of training data and time used is drastically reduced if we compare them to a normal model trained from scratch.

6 Conclusion

In this following report, an analysis of a self-implemented image segmentor was completed, analysing the results of many different types of learning and segmentation methods. When considering unsupervised learning, the first method was K-means segmentation. For K-means, the highest DICE coefficient is observed at K=6, where the coefficient is equal to 0.17119668953873968. Furthermore, it was possible to notice that varying K does not affect the DICE coefficients very much because binary masks obtained from K-means segmentation are relatively the same for different values of K. To improve the performance of K-means segmentation, the threshold should be specific to the number of clusters present. Additionally, the performance of K-means segmentation on RGB images was compared to that of greyscale images. It was possible to notice that using greyscale images instead of RGB images for K-means segmentation does slightly improve DICE coefficients which can be due to the way the function *kmeans* deals with input images with a single color channel. To continue, Quick-Shift segmentation was also analysed, where the highest DICE score achieved is with a a cut-off of 100. Overall, varying the cut-off does not seem to affect the DICE coefficient. The segmented image appears to be coherent with the original image while the binary mask is missing some zones of interest. When comparing both K-means and Quick Shift segmentation, the main advantage of quick shift is that it is faster than mean shift and a variable number of clusters can be used which is not the case for K-means segmentation.

When considering supervised learning segmentation methods, Random Forrest Classifier was used. The train and validation split was trained using both Random Forest Classifier and SVM. However, as there are multiple output classes, Random Forest Classifier outperformed SVM in terms of computational intensity and accuracy. The optimal hyperparameters for the Random Forest Classifier were `max_depth=10, n_estimators=100`.

When considering the classification procedure, where a supervised learning framework using an SVM was used to detect and count tumors, the final parameters which were used were an image size of 100x100, an HoG cell size of 3 x 3 pixels, an HoG block size of 2 x 2 cells, and a 8 orientation bins. The classifer built has a very low false positive rate but a high false negative rate. This could be considered quite problematic in a real-world scenario as some tumors would be missed when trying to detect them. The prediction accuracy on masked images on the training dataset was 70.61% while for validation images, it was 63.11%. The RMSE on the validation set was 19.24. On unmasked images, the prediction accuracy on the training dataset was 55.16% while for validation images, it was 42.06%. When considering other pathological structures, tumors were detected 27.97% of the time, fibroblast 33.44% of the time, lymphocyte 66.80% of the time, plasma Cell 7.36% of the time, and macrophage 3.84% of the time. The lymphocyte pathology has the highest prediction accuracy. This could be explained by the fact that lymphocytes have a dark color which results in a high contrast blob in the images, making it more easily recognizable.

In the bonus section, a pretrained U-NET network was used to accomplish the classification task. Once the model was trained, it obtained and IoU score of 0.6327, and a DICE score of 0.8487. With a limited number of training samples (50), most areas are correctly segmented and represented in the predicted mask. It was possible to notice a significant difference in terms of DICE score and quality of the predicted masks when comparing with other methods used in this report. This is consistent with our expectations, as deep learning models should outperform normal methods. Furthermore, the amount of training data and time used is drastically reduced if we compare them to a normal model trained from scratch.

7 Bibliography

References

[1] Tal Arbel. *Lecture 16 - Segmentation (part 2)*. Lecture Notes. Apr. 2023.

[2] Dorin Comaniciu and Peter Meer. "Mean shift analysis and applications". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. IEEE. 1999, pp. 1197–1203.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241.

[4] Carole H Sudre et al. "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations". In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*. Springer. 2017, pp. 240–248.

[5] Andrea Vedaldi and Stefano Soatto. "Quick shift and kernel methods for mode seeking". In: *Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part IV 10*. Springer. 2008, pp. 705–718.