BMED 4783 - Medical Image Processing
Undergraduate Guided Project

Cancer Clinical Decision Support Enabled by Medical Image Processing of Histopathological Images from Cancer Patient

Nathan Chan, Kewal Khatiwala, Miles Macero, Kent Yamamoto

Module 1 - Image Pre-processing

We were presented with 300 original images. There were 100 images of each category: Necrosis, Stroma and Tumor. To increase the size of the data set, we first normalized the images using Reinhard's method. Image normalization was necessary because the color type, intensity and brightness would vary from image to image even if recorded through the same microscope by the same pathologist at the same time frame. This would reduce the error in our feature extraction process as the normalized image would have the same color distribution range and wouldn't fluctuate image to image.

1.1: Reinhard's color normalization

Reinhard's method needed an RGB image to be converted to LAB image format. Then the following formulas will be applied to each pixel of corresponding layer:

$$l_{mapped} = \frac{l_{original} - \bar{l}_{original}}{\hat{l}_{original}} \hat{l}_{target} + \bar{l}_{target} \quad (1)$$

$$\alpha_{mapped} = \frac{\alpha_{original} - \bar{\alpha}_{original}}{\hat{\alpha}_{original}} \hat{\alpha}_{target} + \bar{\alpha}_{target} \quad (2)$$

$$\beta_{mapped} = \frac{\beta_{original} - \bar{\beta}_{original}}{\hat{\beta}_{original}} \hat{\beta}_{target} + \bar{\beta}_{target} \quad (3)$$

Where $\bar{l}$,$\bar{\alpha}$, and $\bar{\beta}$ are the channel means and $\hat{l}$,$\hat{\alpha}$, and $\hat{\beta}$ are the channel standard deviations (calculated over all pixels in the image).

Figure 1-1. Reinhard's equations for color normalization [1]

The final image is the normalized form of the original image. Before we applied this formula to all the images, we needed a target image so we chose "Necrosis_97.png" as our target image because it was a well balanced image in terms of color, content coverage and light intensity with distinct staining for nucleus and cytoplasm.

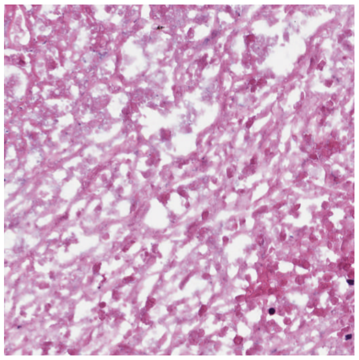


Figure 1-2. Target image for color normalization (Necrosis 97)

1.2: Image augmentation

We applied 3 kinds of augmentation to the normalized images to expand our dataset. The first method of augmentation was rotation of each normalized image by 3 different angles of 90°, 180°, 270° which made a new set of 900 images. Next the original normalized images were flipped horizontally and vertically making 600 new images. And lastly, the original normalized images which were 512 x 512 pixels were given random crops of size 224 x 224 pixels and 4 of those crops from each normalized image became part of the dataset, all shown in Table 1-1:

Table 1-1: Augmentation breakdown

Sr #	Augmentation	Variations	Image count
1	Original Images	None	300
2	Normalization	None	300
3	Rotation	90°, 180°, 270°	900
4	Flip	Horizontal, Vertical	600
5	Random crop	4 randoms	1200
	Total		**3300**

1.3: Dataset breakdown

We made a folder for each original image and placed all the normalized and augmented images in each folder. This was to prevent repetition of images while creating the model. While breaking down the dataset, we broke 100 images of the three categories randomly to maintain a balance for prediction of each type correctly and not skew the model to favour prediction of one category like tumor or necrosis. The dataset was broken into a 60-20-20 format for the Deep Neural Network model and 80-20 for KNN and SVM where cross validation was applied. To stay consistent, each image and its derived augmentations were kept in the same type of dataset (Table 1-2).

Table 1-2: Dataset breakdown according to model

Model	Dataset Breakdown
SVM (Linear Kernel), KNN	80/20 - Training/Testing
Deep Neural Network	60/20/20 - Training/Testing/Validating

Module 2 - Feature Extraction and Selection

An extensive literature review of the various features that are commonly used in image classification was done, both referring to the provided references in the project outline as well as external journal publication research. Gurcan et al. provide an extensive overview of the process of classifying histopathological image analysis, mentioning that there are generally four types of features: size and shape, radiometric and densitometric, textures, and chromatin-specific [2]. These four types were categorized and the algorithm for each type of feature extraction method was thoroughly described in Boucheron et al. [3].

Three feature categories were decided on to look into further: color, texture, and morphological features. Table 2.1 represents the breakdown of the number of features extracted from each category, with a total of 134 features:

Table 2-1: Number of Features Extracted per Category

Feature Category	# of Features
Color	72
Texture	44
Morphological	18
Total	**134**

2.1: Color Features

Tabesh et al. provided a color feature extraction process that was incorporated in our feature extraction process [4]. It was reported that seeing the color change within an image can potentially imply that there is malignancy present in the image (epithelial nuclei staining blue invades stroma which stain red). With that said, 16-bin histogram channels were created per color layer for each image in the RGB scale, which provided 48 features. Separately, 8-bin histogram channels were created per layer for each image in the HSV (hue, saturation, value) scale, which provided 24 additional features. This results in a total of 72 features categorized as "color features" (please refer to "mod2_kento.m" for implementation).

2.2: Textural Features

Boucheron et al. mentioned the use of textural features in pathological image classification using the Matlab function *graycomatrix* to create a co-occurrence matrix after grey-scaling the images. A symmetrical co-occurrence matrix with an offset of 2 was created and the function *GLCM_Features1* created by Avinash Uppuluri was used to extract 22 features per offset; some of the textural features extracted include: contrast, cluster prominence, cluster shade, energy, entropy, homogeneity, sum of squares, etc. (full list can be found in function documentation) [5]. We decided to use this function rather than the standard *greycoprops* function in the Matlab Image Processing Toolbox (which includes contrast, correlation, energy, and homogeneity), because Uppuluri had successfully integrated parameters from Haralick et al. which dives deeper into the various textural features that are commonly extracted for classification (also known as the Haralick Features) [6]. These features have been frequently used to distinguish between healthy tissue and cancerous tumors as mentioned in Caruso et al.'s study on how Haralick Features impact the classification of colorectal cancer cells [6]. Of the 14 features they calculated, the 4 most significant features were: Energy (uniformity of image), Contrast (local fluctuation), Correlation (linear dependencies of image grey levels), and Inverse Difference Moment (uniformity of affinity in co-occurrence grey levels). We have made sure to implement these four features in our feature extraction process as well.

2.3: Morphological Features

Fukuma et al. reported the accuracy of various features extracted from brain histopathological images, two of which are nuclear density of a given window and perimeter of the nucleus, reporting accuracies of 99.87% and 98.76% respectively [7]. Nuclear density is a very logical feature that will provide us with how many nuclei there are in a given window. For cancerous cells that cannot regulate cell division, we will expect to see significantly more nuclei. Zheng et al. designed a convolutional neural network for breast lesion classification and implemented cell nucleus appearance, surrounding cytoplasm, and distribution of nuclei as features [8]. The justification for the above features are present by pathologists - some examples include a nucleus with folds and indentations, enlarged nucleoli, and dispersed heterochromatin [9].

To implement the morphological feature extraction mentioned above, we first linearized the images for better contrast, then incorporated a Canny edge detection algorithm. This will provide us with a mask that represents the edges of the nuclei. We then ran the Matlab function *regionprops* and obtained the following features: area, major axis length, minor axis length, circularity, eccentricity, orientation, convex area, filled area, euler number, equivalent diameter, solidity, extent, and perimeter. The function provided a *N x 18* matrix in which each column represented a feature extracted from the image, thus 18 features.

2.4: Principal Component Analysis (PCA)

Once the features were extracted and concatenated horizontally to a *N x 134* matrix, a principal component analysis was done to determine the number of principal components that will sum to at least 90% while still considering computation power - a balance must be considered. More principal components can be used to create classification models, but that may result in unnecessary computational power used if a lower number of principal components can be utilized to score in a similar range. The features were first normalized to obtain better results, and then the *pca* function in Matlab was used (default centering algorithm used is singular value decomposition, SVD). After running, the principal component number and the eigenvalues were plotted against each other to find the number of principal components required to reach a variance of at least 90%. This is done by looking at the area under the curve of the eigenvalue graph. Through analysis, the first 16 principal components had allowed us to reach the target variance that we were looking for, as shown in Figure 2.1. A three-dimensional plot with the top three principal components were also plotted to show how the features and classes clustered.

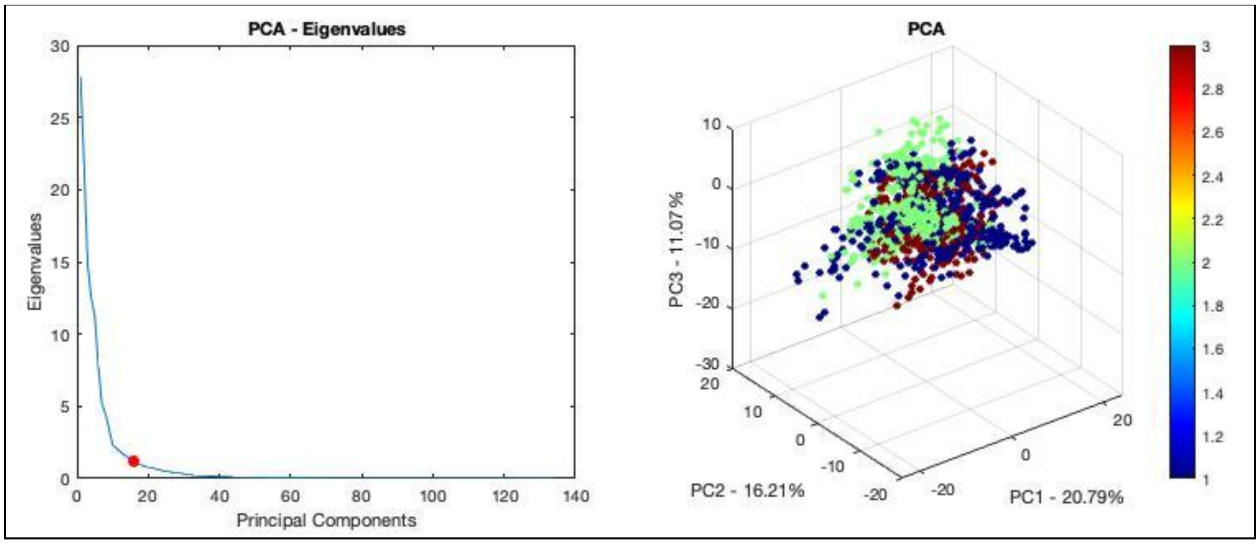


Figure 2-1: a) Eigenvalues v.s. principal components graph to determine the top 90% variance; b) Top 3 principal components plotted against each other.

From this initial principal component analysis, our team hypothesized that we will probably see better results with a non-linear classifier, as the data does not look as if it will be linearly separable.

Module 3 - Prediction Modeling

3.1: Literature Survey

KNN

The K-Nearest Neighbor classifier, or KNN for short, is a linear classifier that classifies an object based on a majority vote of its nearest neighbors. The amount of nearest neighbors is specified by K, such that if K=5, then each object will have its class determined by the most common class among its 5 nearest neighbors. The distance between each object is calculated by a Euclidean distance, which is calculated by the equation in Figure 3-1.

$$D(a,b)=\sqrt{\sum_{i=1}^{n}(b_i-a_i)^2}$$

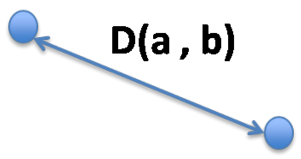


Figure 3-1: Formula and representation of Euclidean Distance

The papers from Guru et al. and Sharma et al. both utilized KNN for image classification, with Euclidean Distance being the basis of distance measurement [10][11]. The first paper used texture features from GLCM to classify flower images, and had a performance of 90.18%, with K=12. The second paper classified cancer cells from their morphological features with an accuracy of 82.9%, but this paper did not specify the amount of nearest neighbors. These two papers show that KNN can be a simple yet an effective tool in classifying images.

SVM

Support Vector Machine (SVM) is a linear classifier which works by making an optimal boundary with supporting vectors as the margin. The main optimal boundary which separates the two categories is called a hyperplane. It is a line in 2D and a plane in 3D and the best possible hyperplane is formed by having maximum distant margins to best classify the data. When the data is not significantly distinguishable, the 2D plot can become 3D until a hyperplane with optimal support vectors that form margins is obtained. The margins lie on the points closest to the hyperplane so the more distant the margins, the better the model. The paper from Le et al. merges an artificial neural network (ANN) with a support vector machine for enhanced image classification [12]. Firstly, they separated the image into many sub-images based on the features of images. Each sub-image is classified into the responsive class by an ANN. Finally, the SVM compiled all the classified results of ANN. The combined model gave 86% precision.

DNN

A Deep Neural Network (DNN) is a type of Artificial Neural Networks (ANN), which are both nonlinear classifiers. The main difference between the two is the amount of hidden layers; DNN typically has more layers than a typical Neural Network. The paper from Sladojevic et al. used a Deep Convolutional Neural Network to recognize 13 types of plant diseases from leaf images [13]. The authors used a database with over 30,000 images, and then they performed data augmentation, which included affine transformations, perspective transformations and image rotations, in order to help prevent overfitting. The model used 8 learning layers, 5 convolutional layers, and 3 fully connected layers. The dataset was split up into training and testing, with 30880 images used for training and 2589 images used for validation. 10-fold cross validation was used to evaluate the accuracy of the model. After tuning the parameters of the hidden layers and hyperparameters of the network, they achieved an overall accuracy of 96.3%.

Random Forest

The Random Forest (RF) classifier is a nonlinear classifier that utilizes a majority vote among a collection of decision trees to classify its objects. The paper by Mirmohammadi et al. used Random Forest to classify microscopic cell images to recognize acute lymphoblastic leukemia and lymphocytes cell subtypes [14]. A total of 312 images were taken from blood samples of 7 normal subjects and 14 patients. Image enhancement was performed to normalize the lightness of the images by transforming the data set from RGB to HSV, and using a histogram equalization method on the V channel of HSV color space. The authors then segmented the nuclei in the images and performed feature extraction on the segmented nuclei, categorizing them into geometrical and statistical features. When training the RF model, each tree is only given a limited portion of the dataset in order to prevent overfitting. After tuning hyperparameters and performing 10-fold cross-validation, the model achieved a classification accuracy of 98.22%.

3.2: Implementation

From the four classifiers identified in the previous section, three were chosen for this project: KNN, SVM and DNN. The 2 linear classifiers (KNN and SVM) would test how well the image data could be linearly separated. DNN would count as a nonlinear classifier, as the creation of weighted layers does not create a linear decision boundary. The implementation of these 3 classifiers all started with the features extracted in Module 2.

The KNN and SVM implementation were nearly identical in implementation and only required changing a few lines of code in Matlab. First, the features (principal components) and labels were imported into Matlab and 20% of the data was split off to act as a testing set for later. Next, the data was fit to a model in Matlab - for multiclass SVM the team used *fitcecoc*, and for KNN, *fitcknn*. These models are designed specifically for classification tasks involving multiple classes. For KNN, the number of neighbors has to be specified here as a parameter (which will be discussed in a later section). The model was fit and then further trained using cross validation error. In this process, the training data was split into k-folds, and the model was trained iteratively with 1 fold left out. After the cross validation, the trained model was used to predict the labels of the initial 20% validation test set. From these predictors the team extracted the metrics of accuracy, F1 score, and area under the receiver operating curve (AUROC).

The implementation of the deep neural network was slightly different. The network was created following Matlab's guide to creating a pattern recognition neural network. The network architecture consists of a two-layer feedforward network, with a sigmoid transfer function in the hidden layer, and a softmax transfer function in the output layer [15]. A sample architecture is shown in Figure 3-2:

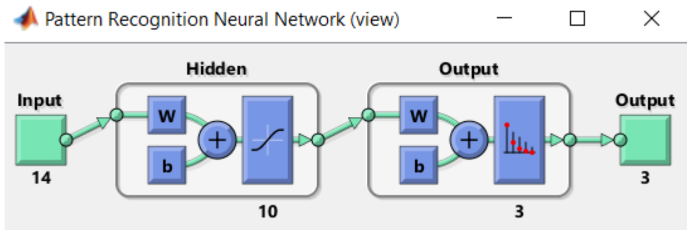


Figure 3-2: Architecture of a Neural Network

The 10 represents 10 hidden layers with sigmoid functions and there are 3 output layers to match the 3 classes (Necrosis, Tumor, Stroma). In addition, a generic max function must be implemented since the network only outputs weights for each class (e.g. weights of 0.8 0.1 0.1 are classified as class 1). The neural network has internal validation as well, meaning that the data was split into 60% training, 20% testing, and 20% validation. The neural network uses the 20% as reference to lower the training error with each epoch. The final 20% acts as validation after the training similar to the 20% from the KNN and SVM from which metrics can be extracted. The training type selected was a scaled conjugate gradient backpropagation algorithm.

3.3: Model Hyperparameter Optimization

After setting up the models, the next important step was to optimize their hyperparameters. These hyperparameters control how the models are run, and changing them will change their outputs. This process was done through iteratively running the models and changing selected hyperparameters in each run.

KNN Hyperparameters

For the KNN model, the number of nearest neighbors and the number of folds in k-fold cross-validation were the selected hyperparameters that were tuned. The number of nearest neighbors ranged from 1 to 23, and the number of folds ranged from 2 to 24, with 2 being the minimum since at least 2 folds are needed for cross validation to occur, and 24 being the upper limit, because it would result in 110 images per fold with a data set of 3300 images. The range of 1 to 23 was chosen for the nearest neighbors to get a wide range of values and to make a square matrix. The results of the KNN hyperparameter analysis can be shown below in Figure 3-3, with the 3 target metrics being accuracy, F1 score, and ROC AUC. From these 3 heat maps, the optimal values for the hyperparameters were identified as 3 nearest neighbors and 12 folds.

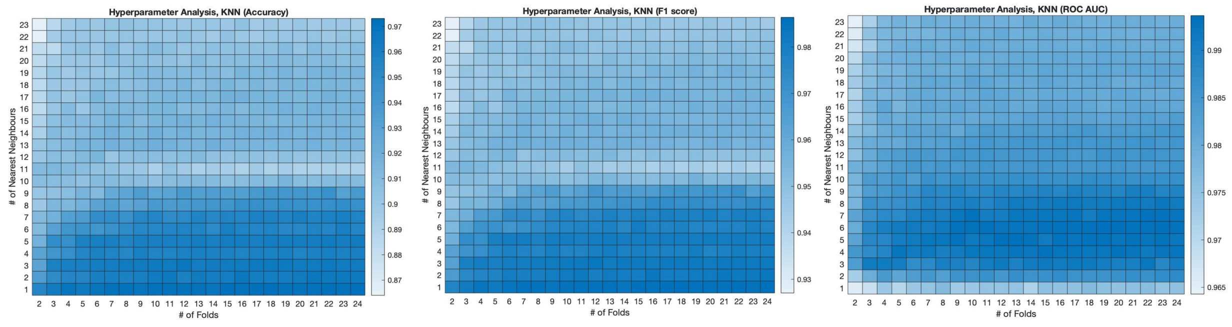


Figure 3-3: Heatmaps for KNN Hyperparameter tuning from 3 target metrics: a) Accuracy; b) F1 score; c) ROC AUC

SVM (Linear Kernel) Hyperparameters

The number of folds for k-fold cross-validation and the number of features were chosen as the hyperparameters for the SVM model. As shown in the SVM heatmaps, the results above 15 features remained the same. Principal Component Analysis that was run in Module 2 indicated that the top 16 features made up over 90% of the variance, so evaluating the number of features as a hyperparameter in the SVM model helped further validate the results of PCA. As shown on the X axis of Figure 3-4, the number of cross-validation folds had no impact on the results of the model. From these heat maps, it was determined that the optimal parameters would be 16 features and 2 folds, in order to maximize results while not using excess computational power.

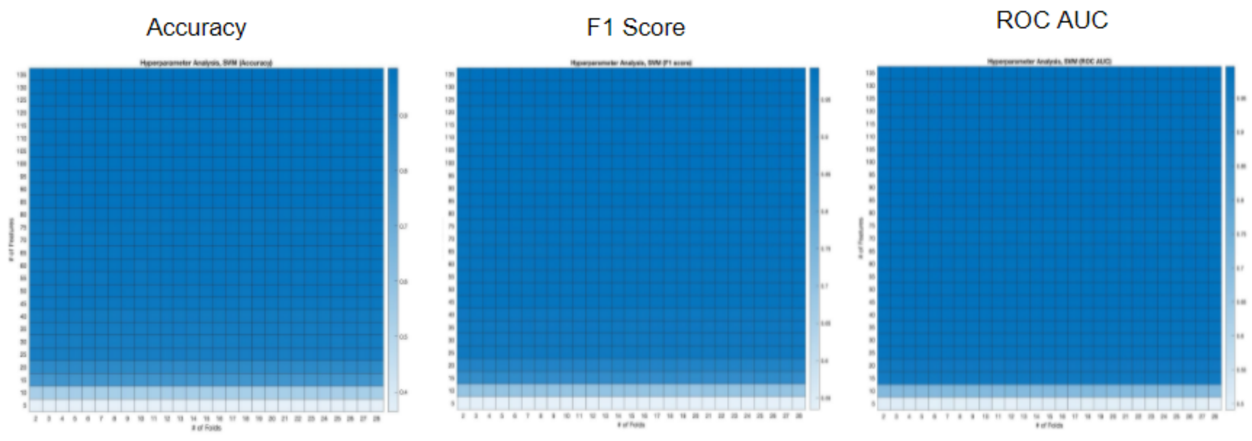


Figure 3-4: Heatmaps for SVM Hyperparameter tuning from 3 target metrics: a) Accuracy; b) F1 score; c) ROC AUC

Deep NN Hyperparameters

The neural network was implemented with # of features and # of hidden layers as hyperparameters. Unfortunately, the NN performed extremely poorly with the 16 principal components, so all the hyperparameter tuning was done with all 134 features. This made sense, as each feature creates an additional input layer for the neural network, allowing more fine-tuning to take place. Fortunately, this increase in features did not seem to be an overbearing load for the training sequence, with each net being trained in approximately 5 seconds or less. Varying the hyperparameter of the # of hidden layers from 1-1000 yielded the interesting results shown below:

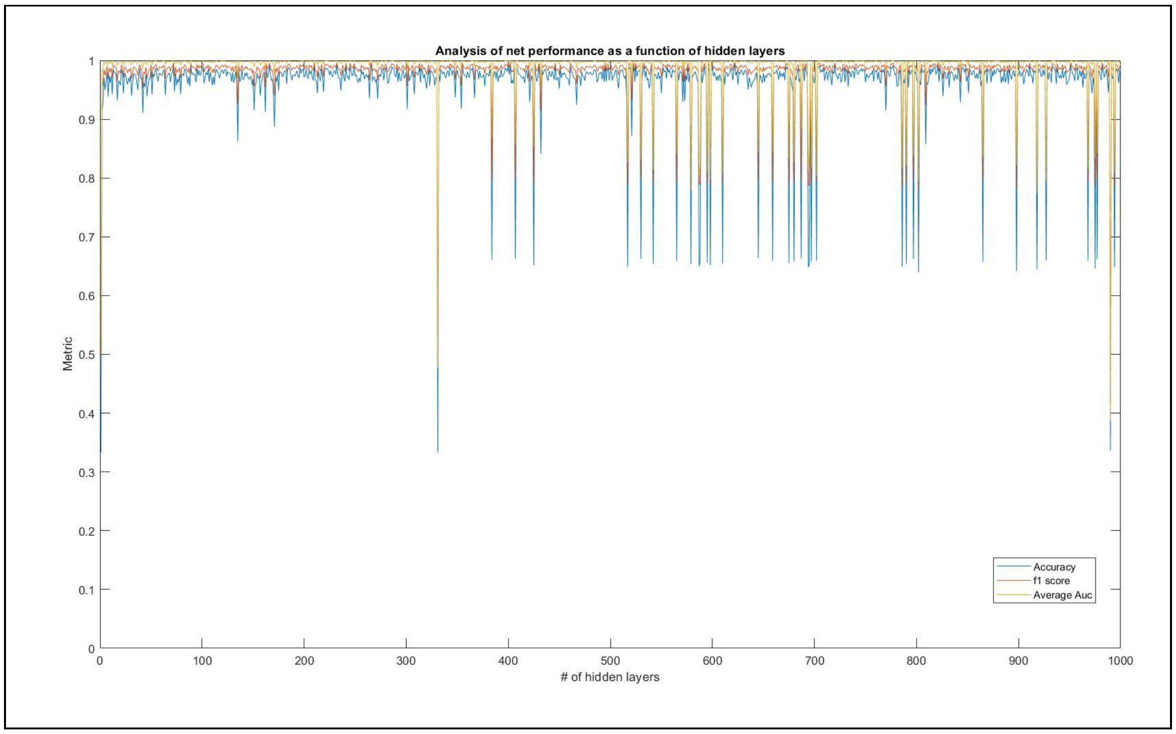


Figure 3-5: # of hidden layer hyperparameter tuning results

After further analysis, this was probably an excessive range to perform tuning upon. One can see that after 300 hidden layers, the performance metrics experience random spikes in performance (probably due to overfitting). In addition, the team saw that there is little benefit to increasing the number of hidden layers. As such, the team selected a model with fewer than 300 hidden layers (e.g. 250 layers), which is a stable region that consistently scores > 0.9 on all three metrics.

Table 3-1 depicts our finalized proposed, tuned hyperparameters for the three classifiers based off of our hyperparameter optimization analysis. We then implemented these hyperparameters, trained our model, and ran our testing dataset. To assess which model is better, there is a need to calculate standardized performance metrics.

Table 3-1: Optimized Hyperparameters for each Classification Model

Classification Model	Tuned Hyperparameters
SVM (Linear Kernel)	# of PC's: 16, 2-Fold CV
KNN	# of Nearest Neighbors: 3, 12-Fold CV
Deep NN	# of Hidden Layers 10

3.4 Performance Metrics

The metrics the team ultimately selected for quantifying model performance were accuracy (1), F1 score (2), and AUC of the ROC curve, AUROC. These are considered standard metrics for evaluating performance and can be calculated using the following formulae:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \qquad (1)$$

$$F_1 = \frac{2}{\text{recall}^{-1}+\text{precision}^{-1}} = 2\cdot\frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}} = \frac{\text{tp}}{\text{tp}+\frac{1}{2}(\text{fp}+\text{fn})} \qquad (2)$$

Where
TP: True Positive
TN: True Negative
FP: False Positive
FN: False Negative

Area under the curve requires plotting the false positive rate against the true positive rate and finding the area under the resulting curve, like so:

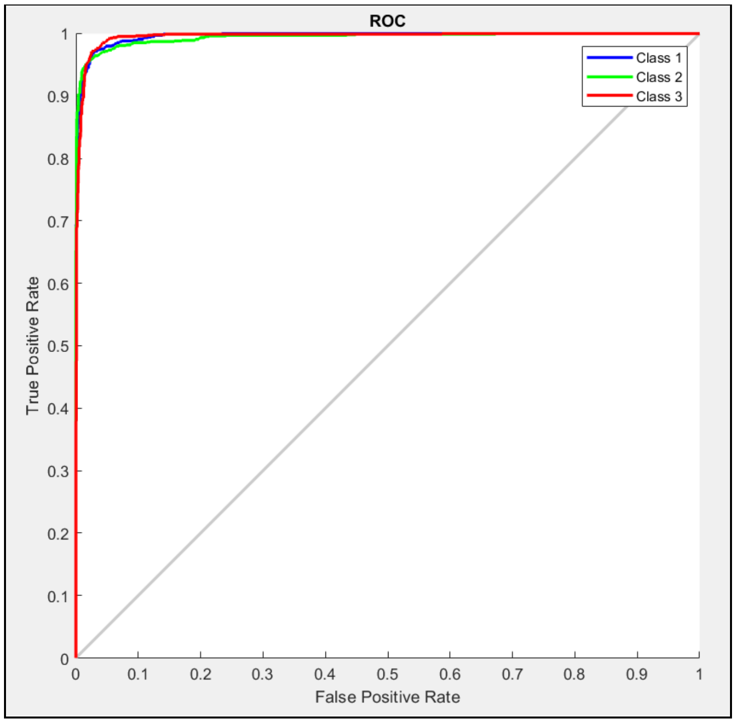


Figure 3-6: ROC Curve

A higher area as seen in Figure 3-6 implies that the model identifies true positives correctly far more often than making an incorrect false positive prediction. As such, higher AUC values are desirable. The same goes for accuracy and F1 score. The team chose these performance metrics because they are scale invariant - they can be applied to any model and are great evaluators of classifiers.

Table 3-2 summarizes our results from the confusion matrices and ROC curves that we plotted (presented in Appendix).

Table 3-2: Performance Metric Results of KNN, SVM and Deep NN

	KNN	**SVM**	**Deep NN**
Accuracy	0.9394	0.8894	0.9721
F1 Score	0.9688	0.9415	0.9859
AUC	Necrosis: 0.9615 Stroma: 0.9538 Tumor: 0.9463	Necrosis: 0.9552 Stroma: 0.9487 Tumor: 0.9411	Stroma: 0.9890 Necrosis: 0.9843 Tumor: 0.9856

3.5: Final Model Selection, Graphical User Interface (GUI), Future Work

Final Model Selection

After optimizing the three classification methods we pursued, a predictivity plot for the linear classifiers was created where the internal accuracy was plotted against the external accuracy. The internal accuracy was calculated from the K-fold Cross Validation loss and the external accuracy was represented by the accuracy values mentioned above. The deep neural network result was not implemented in the predictivity plot, as we did not conduct cross validation for the deep neural net. Figure 3-7 illustrates the predictivity plot.

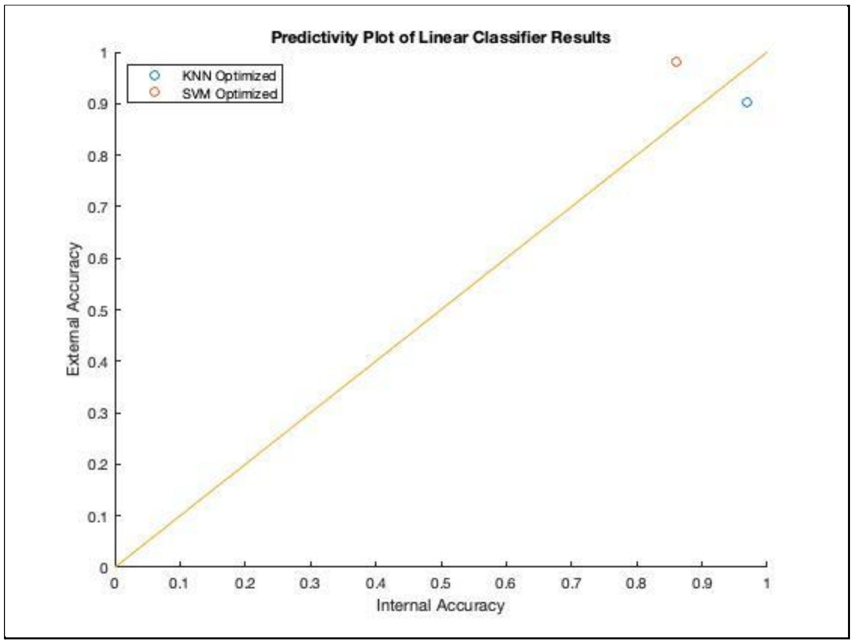


Figure 3-7: Predictivity plot of optimized linear classifiers.

From the results we can see that although the optimized SVM model has a greater external accuracy, it does not have as great of an internal accuracy compared to the optimized KNN model. From the linear line shown, we can see that overall the optimized KNN model performs better, as it is closer to the diagonal line. From this result, we can conclude that out of the linear classifiers that we had implemented, the optimized KNN model with K = 3 nearest neighbors and K-Fold cross validation of 12 performs best.

We also wanted to find a method to compare our deep neural network results with our two linear models, so we extracted the three target metrics we had calculated from each model's performance and drew a linear bar chart to see how the three models compare as shown in Figure 3-8. As predicted, the deep neural network out-performed the two linear classifiers in all three target metrics. This enforces our hypothesis that the dataset can be classified better with a non-linear classifier.

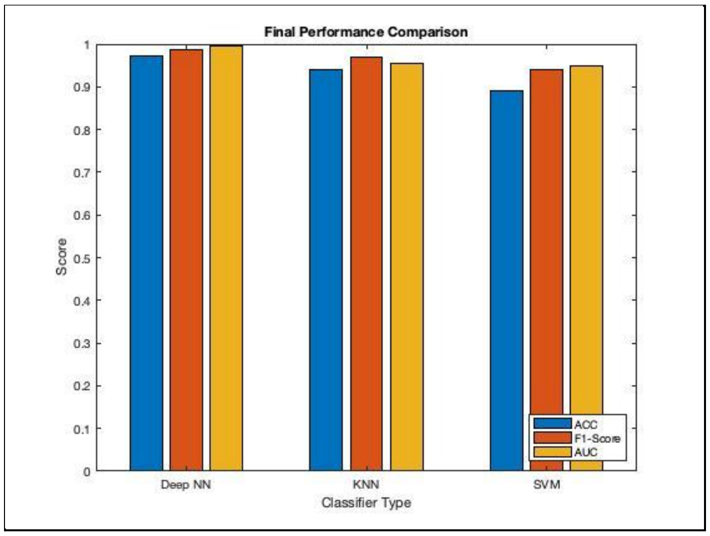
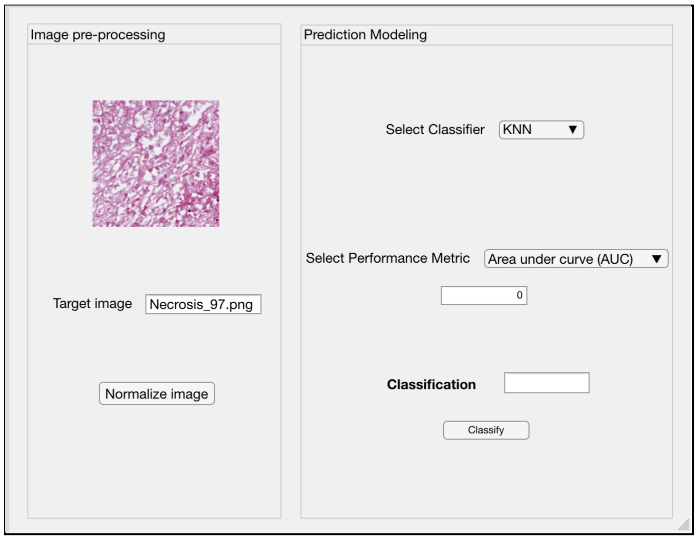


Figure 3-8: Bar graph with three classifiers and respective metric scores for comparison

Graphical User Interface (GUI)

After obtaining our results, we worked on a graphical user interface design (GUI) that will better streamline the pathological identification process if our proposed classification pipeline is implemented. The GUI design is shown in Figure 3-9, with two main components: 1) it will first implement the color normalization we integrated into our project in Module 1, and it will show the pathologist the output. Then, the user will choose the classifier they would like to use as well as the performance metric. The score will show in a value box as well as the classification that the selected classifier outputs. It's a very simple, straightforward, GUI that will not confuse any pathologists who may not be familiar with the technical concepts of machine learning.



Target image: Necrosis_97.png

Select Classifier: KNN ▼

Select Performance Metric: Area under curve (AUC) ▼

0

Classification

Figure 3-9: GUI design with image pre-processing and prediction modeling implemented

Future Work

In conclusion, we have experienced the entire process of classifying histopathological images - from image pre-processing to feature extraction to hyperparameter tuning and final model selection! This was a very informative, hands-on, application-heavy project that taught us what it takes to create a machine learning algorithm for image classification. If we had more time, we would do additional work that would improve our work. First, we would like to see how the performance of our models will improve with a greater dataset - we would love to explore greater image augmentation methods to increase the size of the dataset. Second, now that we have confirmed that linear classifiers are not the most ideal with this specific dataset, we would test other non-linear classification methods (such as the Random Forest algorithm that we have done a literature review on). Finally, we would love to continue work on our GUI so that it is fully functional. All in all, this was a great project for us to really learn how to incorporate image processing and machine learning applications to the development of an image classifier for classifying histopathological images from cancer patients.

References

[1] D. Magee, D. Treanor, D. Crellin, M. Shires, K. Smith, K. Mohee, et al., "Colour Normalisationin Digital Histopathology Images," Proc. Optical Tissue Image analysis in Microscopy,Histopathology and Endoscopy (MICCAI Workshop), pp. 100-111, 2009.
[2] M. N. Gurcan, L. E. Boucheron, A. Can, A. Madabhushi, N. M. Rajpoot and B. Yener, "Histopathological Image Analysis: A Review," in IEEE Reviews in Biomedical Engineering, vol. 2, pp. 147-171, 2009, doi: 10.1109/RBME.2009.2034865.
[3] L. Boucheron, "Object-and spatial-level quantitative analysis of multispectral histopathology images for detection and characterization of cancer," PhD thesis, University of California, Santa Barbara, 2008.
[4] A. Tabesh *et al.*, "Multifeature Prostate Cancer Diagnosis and Gleason Grading of Histological Images," in *IEEE Transactions on Medical Imaging*, vol. 26, no. 10, pp. 1366-1378, Oct. 2007, doi: 10.1109/TMI.2007.898536.
[5] A. Uppuluri, "GLCM texture features", Mathworks File Exchange, https://www.mathworks.com/matlabcentral/fileexchange/22187-glcm-texture-features
[6] R. M. Haralick, K. Shanmugam and I. Dinstein, "Textural Features for Image Classification," in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610-621, Nov. 1973, doi: 10.1109/TSMC.1973.4309314.
[7] K. Fukuma, V. B. S. Prasath, H. Kawanaka, B. J. Aronow, and H. Takase, “A study on nuclei segmentation, feature extraction and disease stage classification for human brain histopathological images,” *Procedia Computer Science*, vol. 96, pp. 1202–1210, 2016.
[8] Y. Zheng, Z. Jiang, F. Xie, H. Zhang, Y. Ma, H. Shi, and Y. Zhao, “Feature extraction from histopathological images based on nucleus-guided convolutional neural network for breast lesion classification,” *Pattern Recognition*, vol. 71, pp. 14–25, 2017.
[9] D. Zink, A. H. Fischer, and J. A. Nickerson, “Nuclear structure in cancer cells,” *Nature Reviews Cancer*, vol. 4, no. 9, pp. 677–687, 2004.
[10] D. S. Guru, Y. H. Sharath, S. Manjunath, “Texture features and KNN in classification of flower images”, *RTIPPR,* 2010.
[11] M. Sharma, S. Kumar Singh, P. Agrawal, and V. Madaan, “Classification of Clinical Dataset of Cervical Cancer using KNN,” *Indian Journal of Science and Technology*, vol. 9, no. 28, 2016.
[12] L. H. Thai, T. S. Hai, and N. T. Thuy, “Image Classification using Support Vector Machine and Artificial Neural Network,” *International Journal of Information Technology and Computer Science*, vol. 4, no. 5, pp. 32–38, 2012.
[13] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, “Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification,” *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1–11, 2016.
[14] P. Mirmohammadi, M. Ameri, and A. Shalbaf, “Recognition of acute lymphoblastic leukemia and lymphocytes cell subtypes in microscopic images using random forest classifier,” *Physical and Engineering Sciences in Medicine*, 2021.
[15] Mathworks, “Classify patterns with shallow neural network”, https://www.mathworks.com/help/deeplearning/gs/classify-patterns-with-a-neural-network.html

Appendix

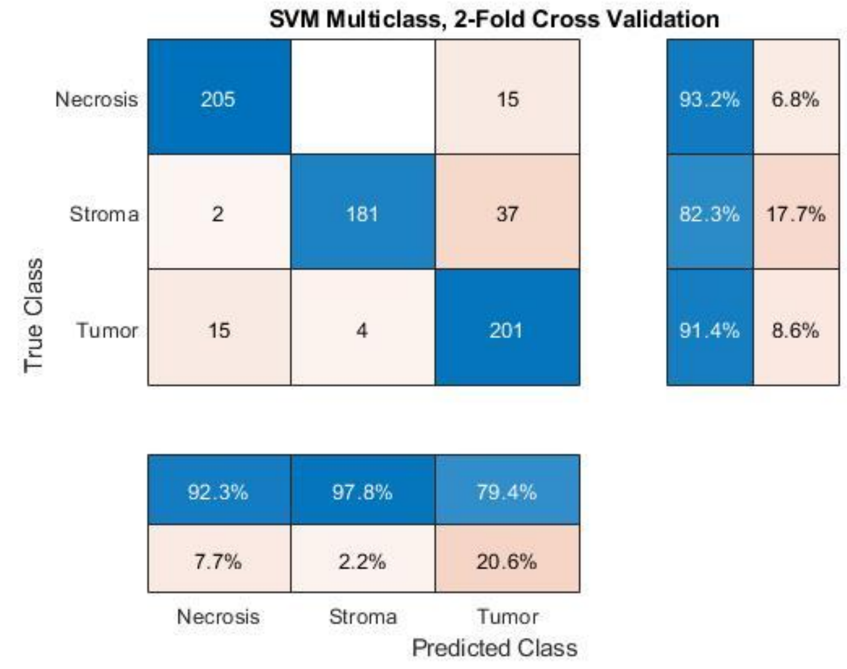


Figure A-1: Optimal SVM confusion matrix (2-Fold CV, 16 Principal Components)

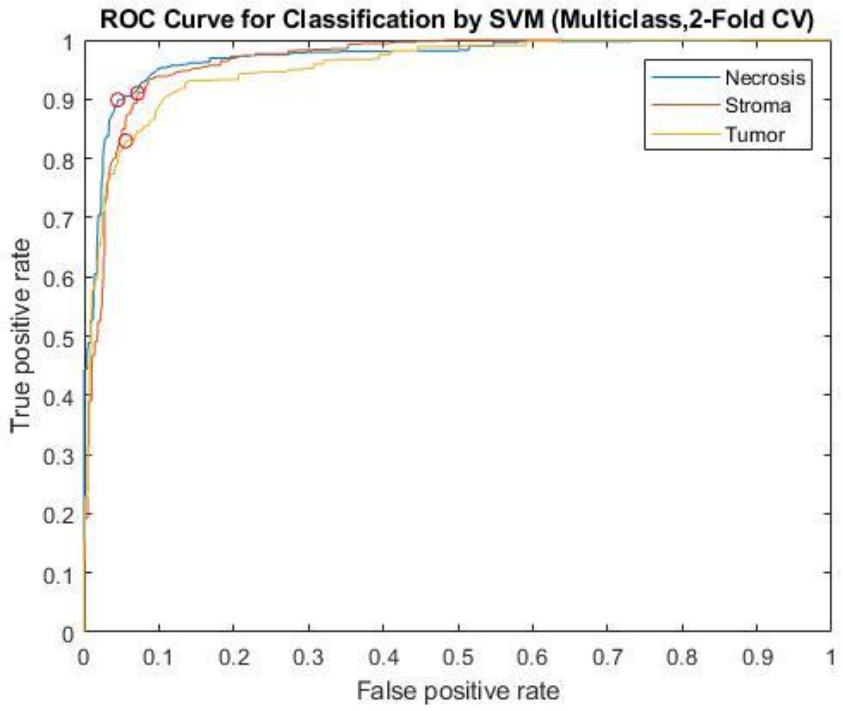


Figure A-2: Optimal SVM ROC curve (2-Fold CV, 16 Principal Components)

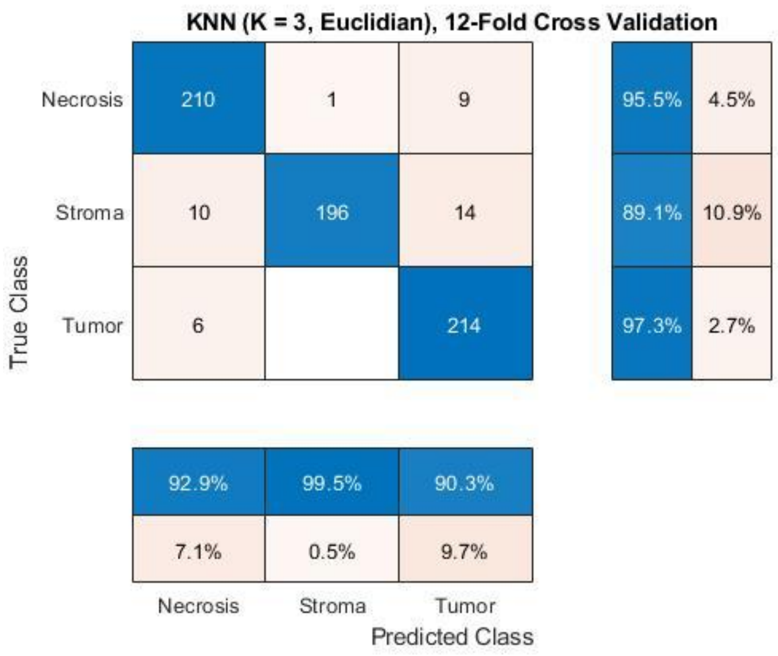


Figure A-3: Optimal KNN confusion matrix (12-Fold CV, K = 3 Nearest Neighbors)

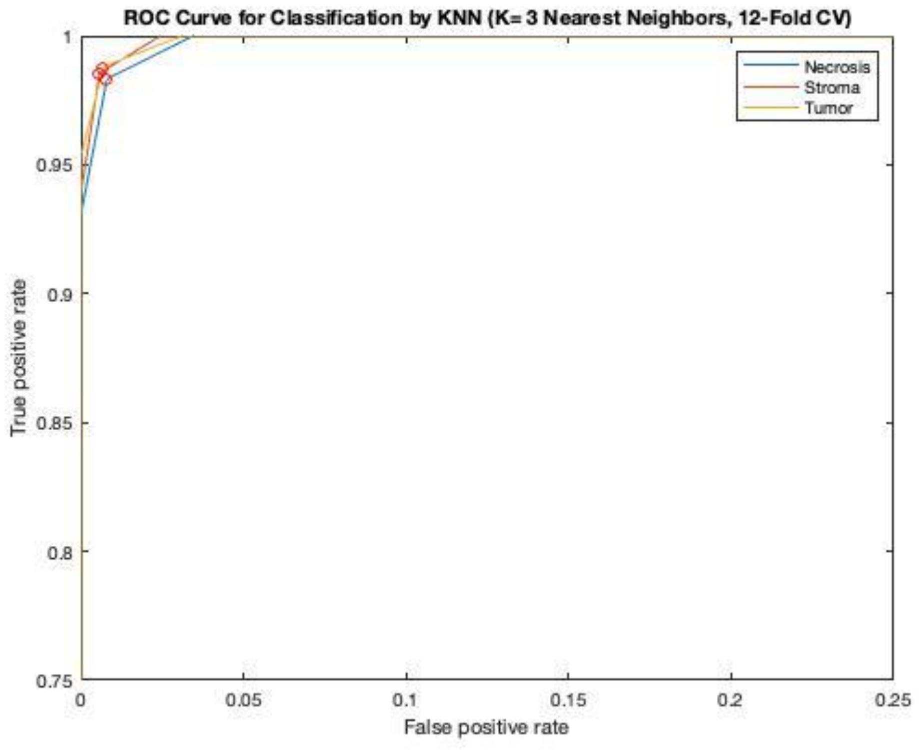


Figure A-4: Optimal KNN ROC curve (12-Fold CV, K = 3 Nearest Neighbors)

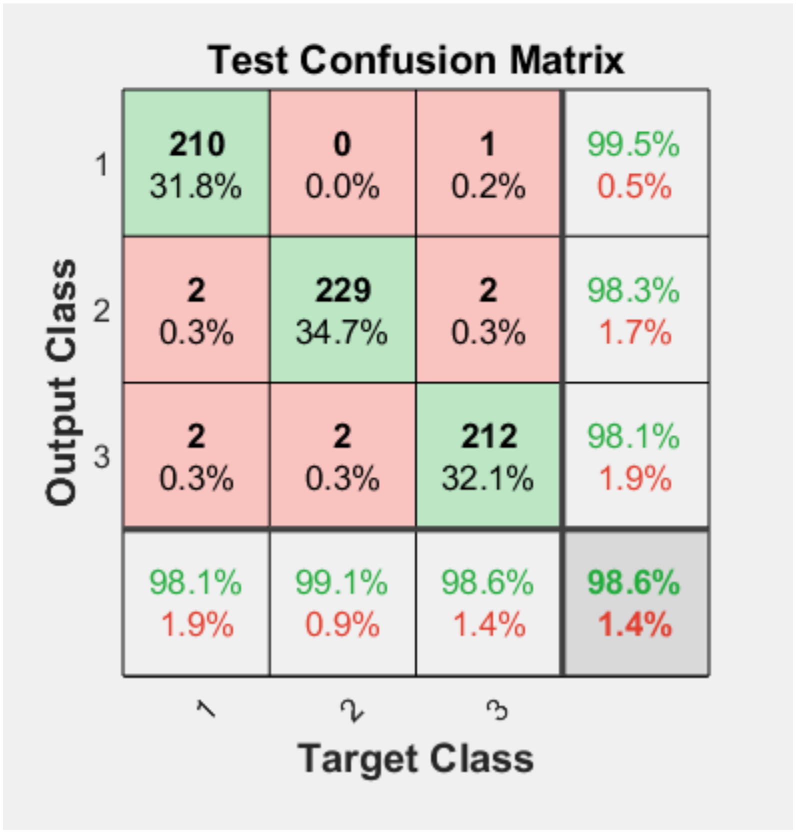


Figure A-5: Optimal DNN confusion matrix (10 Hidden Layers)

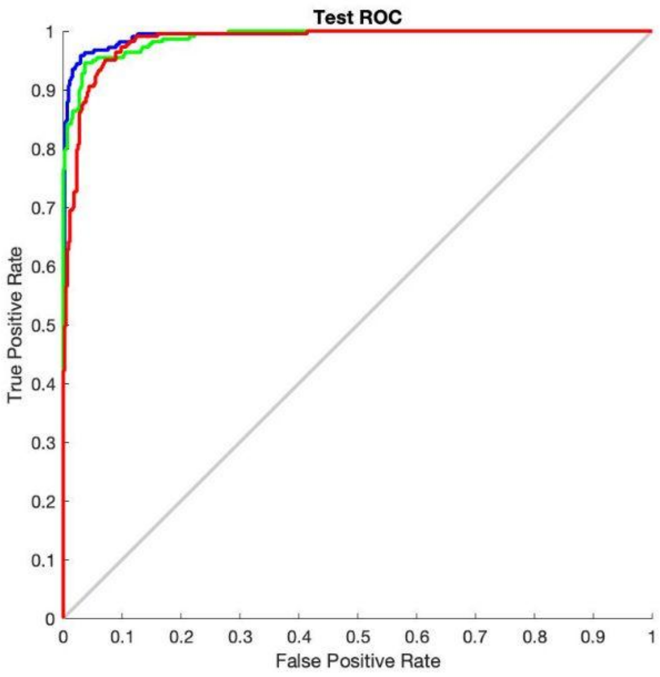


Figure A-6: Optimal DNN ROC curve (10 Hidden Layer)