Classifying Digital Pathology Images Using Machine Learning Models

Jouri Ghazi
Department of Electrical and Computer Engineering, Temple University
jouri.ghazi@temple.edu

Introduction: Digital pathology is the process of producing digital images from tissue sections, enabling efficient analysis and sharing tissue samples among medical professionals. The digitized tissue samples allows for the automation of the diagnostic process using machine learning applications. In this project, a dataset of DCT coefficients in the form of three CSV files, train, development and evaluation will be used to implement non-neural and neural network model to classify the images. The train subset has 10,067 images, development includes 5,959 images, and evaluation has 6,260 images. The first column of these CSVs contains the class value, there are 9 classes labeled from 0 to 8. The models will be trained on the train set and evaluated on the development set. The evaluation set will be used as a blind test, with the class labels set to 0. The non-neural method implemented for this project is Random Forest, and the Neural Network method is a Feed-Forward Network (FFN).

Before implementing the methods, the DCT data was processed. The CSV contains 3×1024 coefficients, the first 1024 correspond to the red channel, the next 1024 values correspond to the green channel, and the last 1024 to the blue channel. The values in each channel are reshaped to 32×32 matrices, in each matrix the first row represents the horizontal frequencies, and the column represents the vertical frequencies. The $(1,1)$ value represents the overall brightness of the image. The intersection of the values correspond to a combination of the horizontal and vertical frequencies. The top-left corner represents low frequency values, quantifying smooth color gradients, while the bottom-right contains high frequency value, quantifying textures and changes in pixel colors. For both models, the top left $N \times N$ values of the submatrices were extracted and flattened into a vector and used for training and evaluation.

Random Forest Description: Random Forest is a non-neural network implemented, this method was selected due to its performance in capturing non-linear decision boundaries. This model trains M decision trees random on subsets of the data and classifies based on a majority vote across the trees. First, the top-left $N \times N$ DCT coefficients from the red, green and blue channels were trained on and evaluated separately to determine if a single color space would provide good performance, potentially cutting the required computation by one-third. However, training and evaluating on all 3 color spaces consistently yielded higher accuracy for all N values. As N increased, the accuracy gradually decreased, which is likely caused by the lower concentration of low frequency values, demonstrating the importance of selecting the most ideal subset of data. The value of $N = 4$ was determined to be ideal, next the models hyperparameters were tuned. These parameters included the number of estimators, maximum tree depth, minimum number of samples needed to split a node, minimum number of samples per leaf, maximum number of features, and the usage of bootstrap sampling, these values were varied to observe its impact on the model's performance.

The number of estimators defines how many individual trees are included within the model. A larger number of trees would result in a better fitting to the training data set but requires more computation and increased time to train. Increasing this value generally led to higher accuracy on the development set, which eventually converged. The maximum depth determines how deep a tree is allowed to grow, limiting the depth would prevent the model from overfitting, while trees that are too shallow underfitted the data and resulted in poor performance. The minimum samples split parameter controls the minimum number of samples required to split an internal node. Higher values would reduce the model's complexity by requiring more data to grow and develop more complicated patterns, while a low value would result in overfitting to the train set dataset, also hindering performance. The minimum samples leaf variable determines the minimum number of samples needed to be a leaf node, a higher value would ensure that a leaf node would prevent in overfitting to the data. The maximum features determines the number of features considered

when looking for the best split within each node. Lower values such as 'sqrt' and 'log2' would increase the randomness and reduce the correlation between the trees. While setting it to None would allow all the features to be considered, this value provided a higher accuracy when evaluating. The Bootstrap variable determines whether sampling with replacement is used when creating the trees. The usage of bootstrapping allows the trees to be more diverse and reduced overfitting. Table 1 shows the values selected for each of these hyperparameters, the optimized model yielded an accuracy of 97.51% on the train and 58.67% on the development.

Parameters	Value chosen
N Estimators	82
Max Depty	None
Min Samples Split	None
Max Leaf Nodes	998
Max Features	None
Bootstrap	TRUE

Table 1. Random Forest Parameters

Feed-Forward Network Description: The FFN was the neural network method implemented for this task. An FFN processes data in a singular direction without a feedback loop, from the input to output. This model is made of three layers, the input, hidden and output layers. The input layer receives the feature vector, while the hidden layer would extract the complex patterns, and the output layer would produce the class predictions. The size of the input layer corresponds to the number of features, and the output layer has 9 units, corresponding to the possible image classes. The number of hidden layers determines the model's capacity to learn more complicated patterns. Deeper models with more hidden layers are more likely to overfit, whereas a shallow network may not be able to pick up on the patterns as well. It was observed that the more the model was overfit to the training, the worse it would generalize to the development data.

The parameters tuned for this model include the number of epochs, activation function, optimization algorithm, learning rate and the dropout rate. Epochs refer to the number of times the training dataset goes through the model, more epochs can allow the model to better fit the data but lead to overfitting. The activation function introduces non-linearity to the model and enables it to learn complex patterns. The GELU activation function provided the best accuracy for both train and development across a variety of epochs. The optimizer controls how the model's weight are updated to minimize the loss value. Adaptive Moment Estimation (ADAM) provided a high level of generalizability to the development and was chosen for this model. The learning rate determines the step size for weight updates per iteration. If it is too high, the model may overshoot the ideal value, and a value that is too low would hinder the learning progress. The learning rate was varied and the value of 10^{-3} resulted in the highest accuracy. The dropout value is used to prevent overfitting by randomly dropping a portion of the neurons during the training process, disabling dropout resulted in the highest accuracy. Table 2 shows the hyperparameter values used for the FFN. This model achieved 90.94% accuracy on the training set and 39.87% accuracy on the development set.

Parameters	Value chosen
Number of Layers	100
Learning Rate	10^{-3}
Activation Function	GELU
Epochs	115
Optimization	ADAM
Dropout	0

Table 2. FFN Parameters

Conclusions: This project implemented a neural and non-neural model to classify digital pathology images using the extracted DCT coefficients. The Random Forest model provided a baseline and generalized better on unseen data compared to the FFN model, both models achieved an accuracy of at least 35%. Table 3 summarizes the accuracies and error score for the train and development for each of the methods implemented. For future work it would be useful to explore the inverse DCT to reconstruct and analyze the original image content and implement a Convolutional Neural Network (CNN). Additionally implementing the CNN on the DCT data could allow the model to capture the relationship between the frequencies present within the dataset.

		Data Set	
	Algorithm	**Train**	**Dev**
Accuracies	RNF	97.51%	58.67%
	FFN	90.94%	39.87%
Error Score	RNF	2.57%	61.65%
	FFN	3.66%	76.64%

Table 3. Accuracies & Error Score of Train and Dev