Kidney CT Classification

Alice Drozd, Kara Christensen, Richard Oldham

adrozd62701@berkeley.edu, kara.christensen@berkeley.edu, oldha006@berkeley.edu

MIDS W281 Fall 2024

Abstract

In this project, our goal was to build a classifier that would determine the affliction on the kidney from CT scan images. Our data was unique in that it contained two views and multiple images per patient, so we developed classifiers for both image types and created a custom train-test split. We designed simple features such as Histogram of Intensities, Fourier Transform, GLCM, and HOG, as well as complex features like ResNet50 outputs and segmentation masks. For segmentation, we implemented a three-stage process: (1) training a custom U-Net model to generate kidney masks, (2) fine-tuning the U-Net model to create high-confidence pseudo-labels for our dataset using iterative training and feedback loops, and (3) leveraging these generated masks to support a CNN for feature extraction and classification. We performed PCA reduction to compare two different feature vectors per image type per classifier and ran experiments across these sets. Hyperparameter searches were conducted to optimize results. The Coronal model with the best performance was an XGBoost classifier on the entire feature dataset, with a test accuracy of 67.87%. The Axial model with best performance was an XGBoost classifier on the PCA-applied feature dataset, with a test accuracy of 73.07%. When comparing accuracy and efficiency, we found that most of our models were highly efficient, with improvements over our baseline in almost all cases.

1 Introduction

Early, accurate detection of kidney disease states, including kidney cysts, stones, and tumors, is important for giving patients the best treatment and outcomes. In a world where technology plays a pivotal role in everyday life, it is key for us to leverage new technologies to address medical challenges. Machine learning powered algorithms combined can be trained on curated features derived from CT and X-ray images to detect abnormalities and assist radiologists to make more accurate and effective decisions. The objective of our project is to create a ML model that will take CT image data as input and predict the state of the kidneys in the image. The possible outcomes are kidney tumor, cyst, stone, or normal findings.

2 Data

We leveraged a dataset from Kaggle for training and testing our model[4]. The dataset has 12,446 grayscale images, 11,929 after removing duplicates, each with a specified label of tumor, cyst, stone, or normal. The ground truth labels were verified by a radiologist and medical technologist. All images were collected from different hospitals in Bangladesh, and patient information has been removed from all images. We noted that in most cases, sequential series of varying lengths of images belonged to the same patient. This led us to develop a custom train test splitting strategy, further discussed in the Generalizability section.

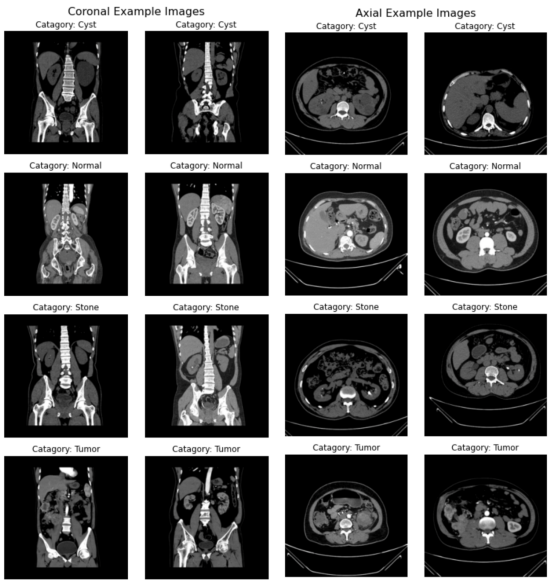

Figure 1: Examples of each class and image type

There were two types of images in the dataset, coronal and axial (Figure 1). We made two sets of classifiers, one set for each type of image, and trained them separately. The dataset has 4737 coronal and 7709 axial images, both with similar distributions and slight class imbalance (Figure 2).

3 Features

3.1 Histogram of Intensities

A histogram of intensities is a feature set representing the distribution of pixel intensity values in an image, ranging from 0 (black) to 255 (white) for grayscale images. It creates a frequency distribution of intensities by counting the number of pixels in predefined bins, forming a feature vector that characterizes the image's brightness and contrast. This simple feature works well for object detection and texture analysis, but lacks spatial information about pixel arrangement. We built this feature then did a correlation heatmap to visualize the overall visual understanding of the heatmap (Figure 4). We essentially saw the symmetrical

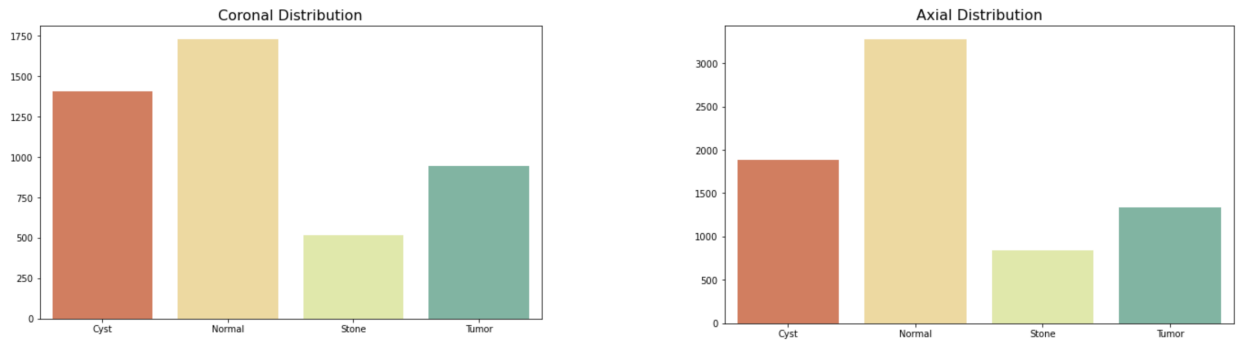

Figure 2: Distribution of Coronal and Axial Images by Class

Coronal	Axial	Total	Type
1803	3274	5077	Normal
1462	2247	3709	Cyst
529	848	1377	Stone
943	1340	2283	Tumor

Figure 3: Count of Images by Plane and Class

nature of the kidney, which led us to hypothesize that the histogram of intensities could potentially pick out sections of the image that was asymmetric, such as tumors, cysts, or stones.

3.2 Fourier Transform

Fourier transform features represent an image in the frequency domain by transforming it from the spatial domain into the frequency domain, decomposing the image into sinusoidal components. This produces the magnitude spectrum, which shows the intensity of different frequency components (Figure 5). We computed statistical descriptors like mean, standard deviation, skewness, and kurtosis from the magnitude spectrum to summarize the frequency distribution. These features capture texture, periodicity, and global patterns, making them robust to spatial transformations. While they lack localized spatial details, fourier transform features can effectively highlight differences in frequency content. We found that the magnitude spectrum could vary slightly, with especially the cyst images, both axial and coronal.

3.3 Grey Level Co Occurrence Matrix (GLCM)

When examining images for the various classes, we noticed that there were differences in the textured of the kidneys with different afflictions. For example, kidneys with cysts tended to be mostly solid, with a dark spot where the cyst was, while kidneys with tumors had

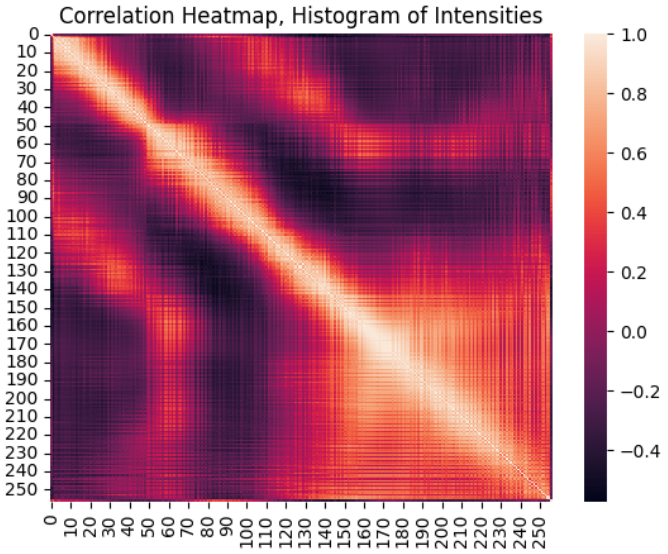

Figure 4: Correlation Heatmap of Histogram of Intensities Features, Showing Symmetry of Kidney

texture throughout due to the irregular shape of the tumor. Because of this, we decided to use features derived from a Grey Level Co Occurrence Matrix, of GLCM, to attempt to capture these differences. A GLCM looks at how the grayscale intensity of a pixel in the image is similar or different to another pixel in the image. We specifically had it so each pixel was compared at three different lengths, 1 pixel, 5 pixels, and 15 pixels, and four different directions, 0, pi/2, pi, and 3pi/2, for a total of 12 different configurations. Then, from these resulting matrices, we derived six statistics that would summarize the entire image at each of these configurations:

Contrast: The intensity contrast of the pixels over the image

Correlation: The correlation of the intensities throughout the image

Dissimilarity: The difference in intensity levels

Homogeneity: The similarity in intensity levels

Energy: The sum of the squared value of all entries in the GLCM

Angular Second Momentum: The uniformity in intensity levels

We found that correlation showed the most significant separation between classes (Figure 6). We took the 12 values for each of the 6 statistics, and flattened them to form one feature vector per image with length 72. You can see that the distribution for the axial and coronal images vary, even though both plots are showing the correlation statistic.

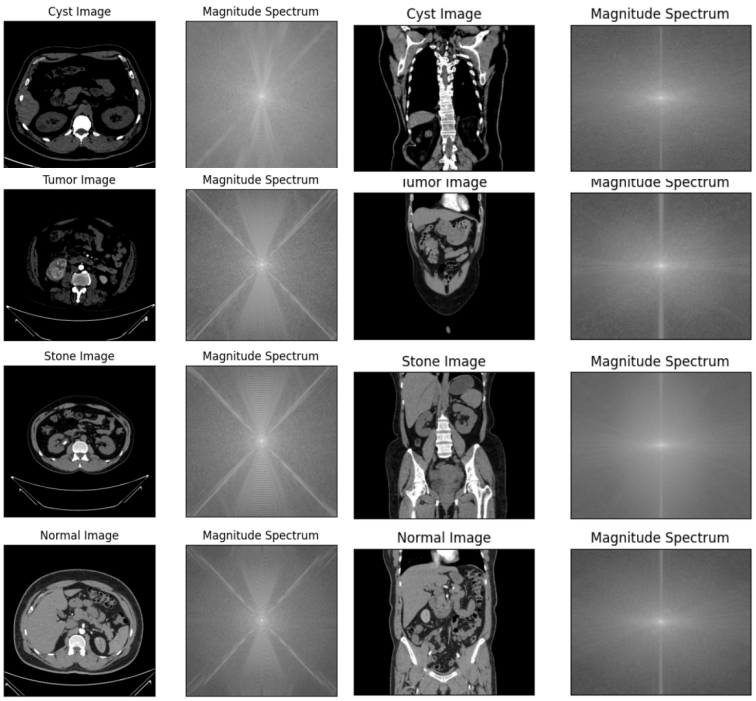

Figure 5: Fourier Transform Magnitude Spectrum For Sample of Images

3.4 Histogram of Gradients

Another thing we noticed when examining the images was that some afflictions, specifically cysts and tumors, caused the kidneys to have an irregular shape. Cysts are often a round protrusion from the otherwise regularly shaped kidney, while tumors can cause dramatic warping of the shape of the kidney. We decided to use Histogram of Gradients (HOG) to attempt to quantify the shapes in the CT scans, to hopefully capture these variations in shape. HOG looks at small areas of pixels, in our case sections of 10 pixels by 10 pixels, and catalogues the counts of gradients by direction. This can therefore identify an aggregated view of what direction the lines in an area tend to be moving, which can collectively identify the general shape. HOG can output both a vector to use as a feature, and a visualization (Figure 7). We utilized the feature vector that was output from HOG in our models, which had a total of 14,440 features.

3.5 ResNet50

When we were considering what complex feature to use, the first direction we decided to go was to utilize a well rounded and established image classification model. We chose this model because it was trained on a large amount of varied data, meaning it should have effective filters that would be applicable even in our case. We also thought that the depth of 50 would be a good balance between complexity and efficiency. We used ResNet50[3] to create complex features by pulling the model, removing the final classification layer, and 'predicting' on our images. We made sure that our images were size 244 by 244, which was the size of the images the model was trained on. The output of this was a feature vector with

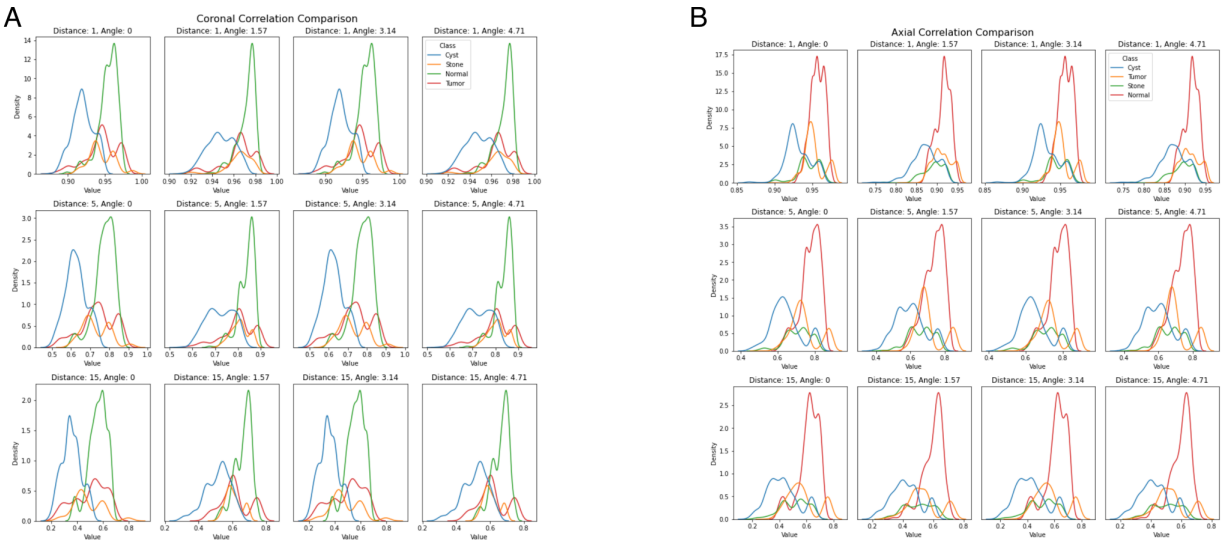

Figure 6: Coronal (left) and Axial (right) Gray Level Co Occurrence Matrices

100,352 features per image, holding what ResNet50 would have used to classify the image. Because this is a complex feature, we do not have example images that correspond with it. However, in the tSNE section, information will be provided on how this feature generally groups the data.

3.6 Segmentation

Given the overall goal of predicting kidney conditions, we felt it was reasonable to create segmentation[1] masks for our dataset to support classification by using the generated masks to help guide a CNN model to the portions of the images in our dataset that contain the kidney[5]. Our image segmentation approach consists of three stages:

Stage 1: Training a Custom U-Net Model for Mask Generation
We trained our custom U-Net model using a labeled dataset from The Cancer Imaging Archive (TCIA), which consists of CT scan imagery of 140 patients in the Neuroimaging Informatics Technology Initiative (NIfTI) format. The labels in the dataset are image masks. Since the data included scans of other organs, the TCIA data was filtered for all samples of class 4, which corresponds to kidney scans/masks. The unpacked and filtered NIfTI files produced 7,777 viable kidney image/mask pairs for training. The model was set up as a binary classifier (kidney or non-kidney), and the model output is a mask image (Figure 8).

Pretrained models were produced using various image sizes: 128 x 128, 256 x 256, and 512 x 512. We tuned all three models using various hyperparameter settings in a trial-and-error fashion to overcome the biggest challenge during training: stagnant losses. The most impactful tuning came from the implementation of a cyclic learning rate[6], which helped the model escape what we suspected were local minima. Our

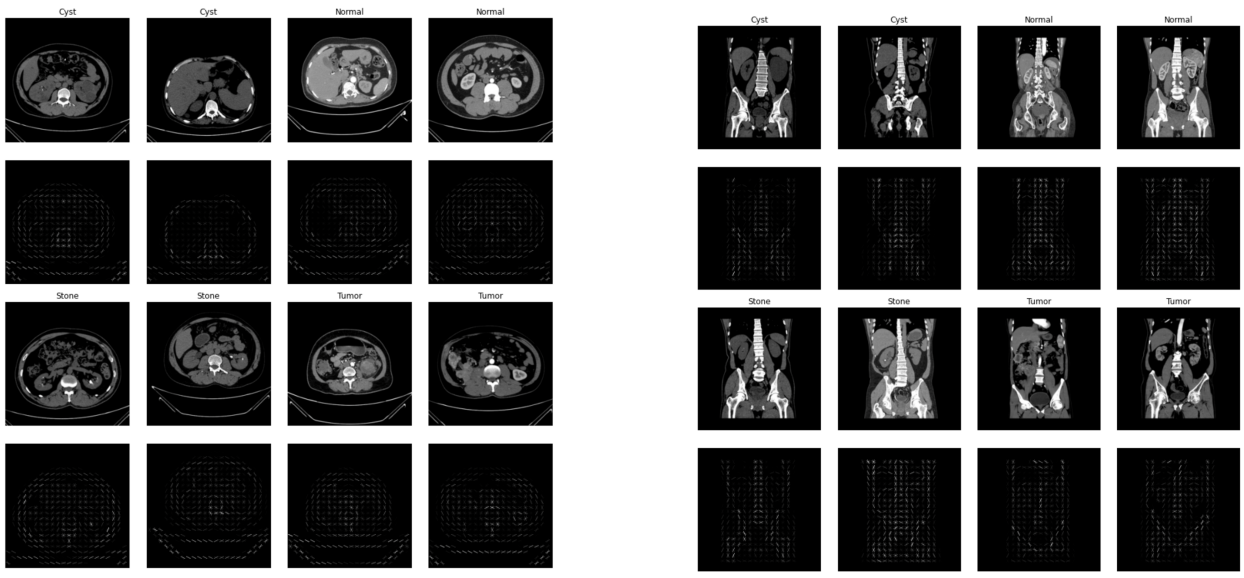

Figure 7: Histogram of Gradients for Sample of Axial (left) and Coronal (right) Images

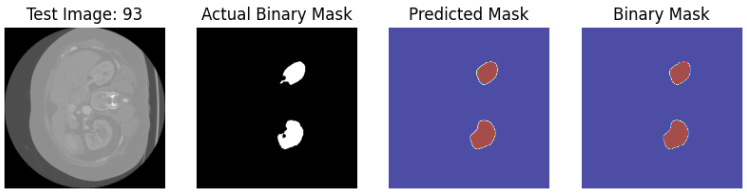

Figure 8: Kidney Image/Mask Pairings

learning rate algorithm cycles through the intermediate values between an initial rate (e.g., 1e-5) and a maximum rate (e.g., 1e-3). When losses stall, the rate is repeatedly adjusted by a configurable step until losses improve, or the maximum value is reached and reset to the initial rate. The combination of low losses (as low as 0.01) and high-quality masks for test data were the criteria we used to save the models for the next stage in the pipeline.

Stage 2: Fine-tuning the Pretrained Model to Our Dataset

Using the pretrained U-Net model, we created a label generation pipeline to generate what we pseudo labels[7] for our project dataset. We experimented with fine-tuning the pretrained model with just our project data as well as with a combination of our project data and the TCIA data. The latter option proved more effective. Using only the project data, the masks were either blank or otherwise not viable. Once we mixed in the TCIA data, the pretrained model was able to generate masks, albeit of low quality.

An inordinate amount of time was spent refining the code in this stage to improve mask quality, which was measured by a value we called `pseudo_label_confidence`. This value represents the mean pixel intensity of a mask, where values near 0 indicate

a blank image. The value increases as more kidney regions are rendered. Initial refinements included region filtering to remove noise, spatial constraints to focus on the kidney's expected positions, and morphological operations[2] to enhance mask quality. The improvements allowed the model to isolate the kidney region more efficiently, setting the conditions required to incrementally "grow" masks for the inputs.

Each round trains the model in a loop for n iterations at 5 epochs per iteration (Figures 9, 10). The first round yielded initial confidence of 0.0004 and final confidence of 0.0011, and the second round with a warm start yielded an initial confidence of 0.0011 and final confidence of 0.0026.

Figure 9: Round 1 Pseudo Confidence levels for 1 iteration

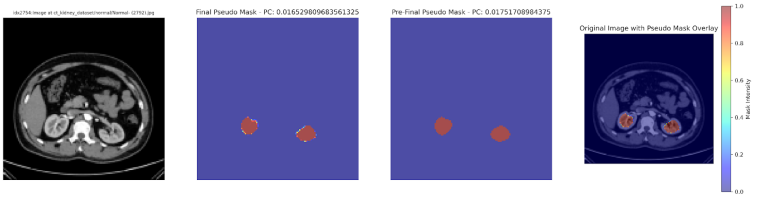

Figure 10: Round 2 Pseudo Confidence levels for 2 iterations

Final refinements of this stage included implementing an iterative process and adding a feedback mechanism. For the iterative process, pseudo-labels were generated for training data as follows:

1. Generate the first set of pseudo-labels.
2. Train the model for five epochs.
3. Use the updated model to generate a new label set and save the updated model.
4. Repeat until all iterations are completed.
5. Continue the process for another round with the updated model and variables in scope or start over completely.

The iterative process combined with the feedback mechanism allowed for manual validation of generated masks, with viable masks being retained and reused in subsequent iterations to enhance model learning. Leveraging visual inspection and pseudo-confidence values, we skipped mask generation for previously validated samples. While this was not the most ideal solution, there was a noticeable uptick in mask quality with each iteration. These refinements elevated our mask quality significantly. However, we acknowledge that the process is time-consuming, requires tuning numerous parameters, and demands constant attention for feedback.

Stage 3: Using the Generated Masks (Pseudo Masks) to Support a CNN
While we refined our Stage 2 code, we also began implementing the Convolutional Neural Network (CNN) that Stages 1 and 2 were designed to support. Our theory was that once the pseudo-masks achieved acceptable quality, the CNN would utilize the masks to focus on kidney regions, improving feature extraction for classification. The baseline CNN was supposed to serve as the foundation for integrating mask-guided learning into the classification task. However, problems with data leakage rendered the results of our initial model invalid, and we ran out of time before we could address this problem or experiment with mask-supported classification. Another major complication was the error-prone experience of trying to extract features from any of these custom models. We were unable to determine why the extraction process repeatedly failed.

4 Feature Analysis

4.1 tSNE

t-Distributed Stochastic Neighbor Embedding (tSNE) is a statistical method that allows for the visualization of high dimensional features, mapping high dimensional data down to, in our case, a two dimensional plane. We performed tSNE on our features in order to visualize the feature information, colored by class, to give an idea of how effectively the features cluster and separate the classes.

In our data, all tSNE plots help visualize not only the classes, but also an underlying feature of our data that we mentioned briefly in the Data section. When observing our data, we could see stretches of images that looked almost identical to each other. Then, we remembered how CT scans work. In a CT scan, many images are taken, then 'stacked' on top of each other to construct a 3D image. Therefore, our data has multiple images per patient, each at a slightly different height or depth. However, each patient has a different number of images, and they are not labeled. We dealt with this via a customized train test split we will discuss in the generalizability section, but it can be seen in the tSNE visualizations. When looking at the images, you will see 'strings' of data points, where the 'strings' are distinct from each other but the points within a 'string' are very close together. Each 'string' represents one patient's images. It is more visible in some features than others.

Looking at the tSNE plots for the coronal training set, we can see that some features do cluster certain classes together. For example, Histogram of Intensities and Fourier transform both cluster Cyst images together, while HOG and ResNet50 do a better job clustering Normal and Stone images. None of them have perfectly separable classes, which is not surprising. This is also the first visualization we have gotten of ResNet 50, and we can see it clusters some classes together more than others, but separates each patient's images rather effectively (Figure 11).

For the axial images, we can see that the results are different (Figure 12). First of all, in ResNet 50, the 'strings' are a lot curvier. We are not sure exactly why this happens, but it may have to do with the difference between the shape of the body in each type of scan, compared to the regular usage of ResNet. In terms of groupings, the Fourier feature shows

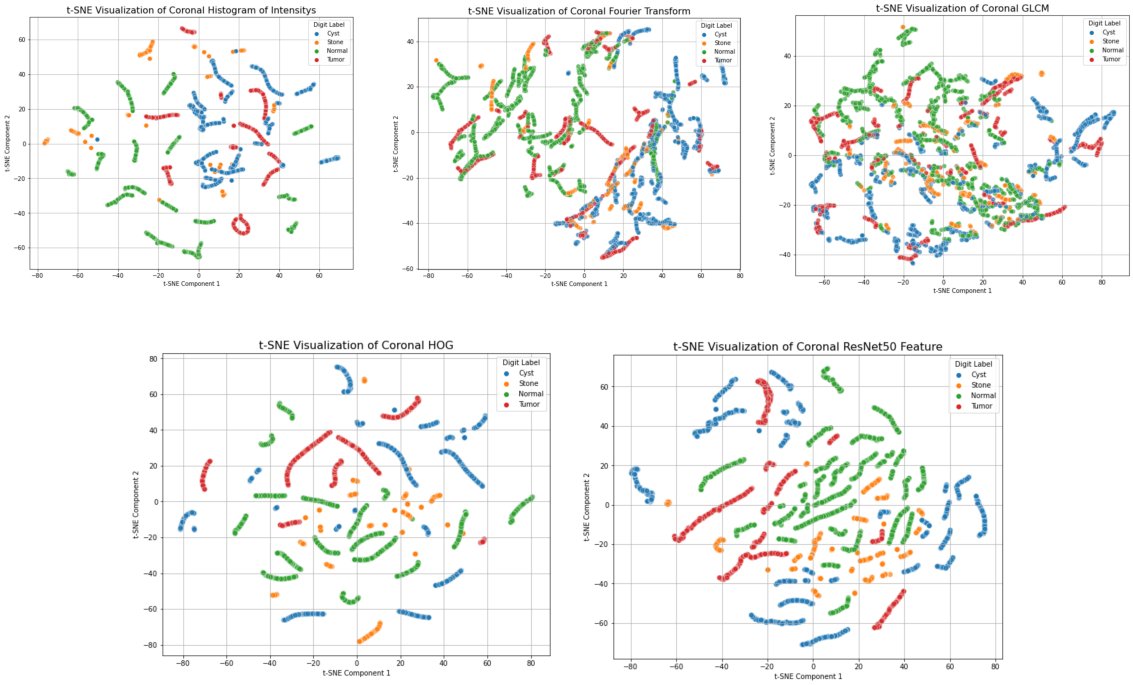

Figure 11: Coronal tSNE Analysis

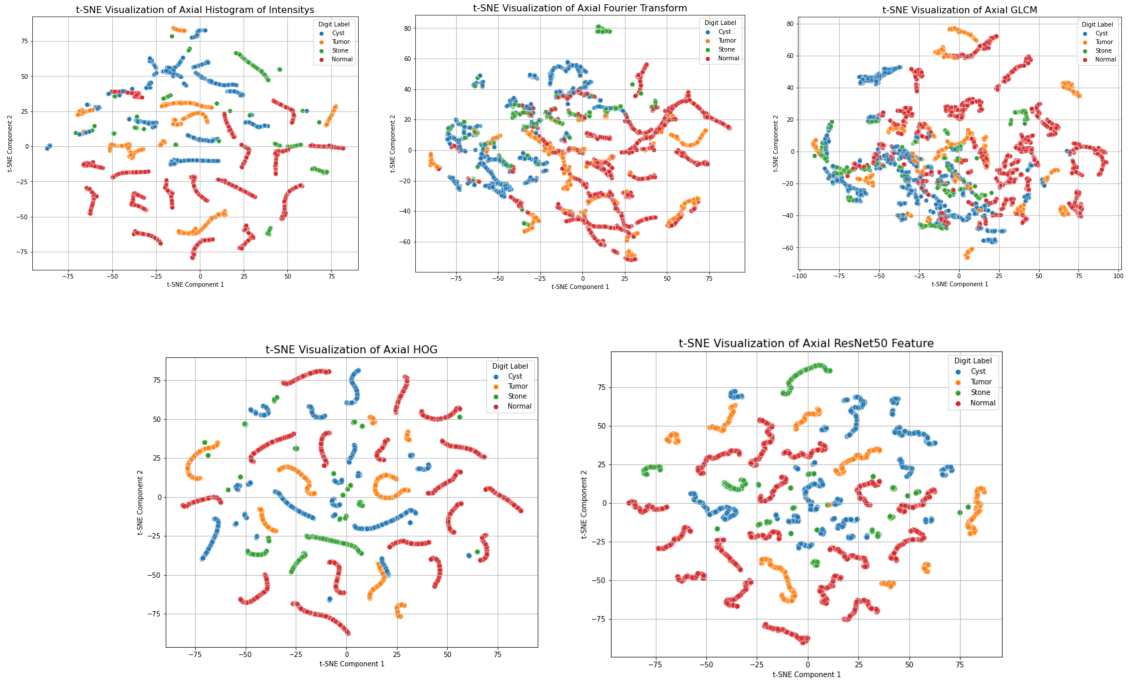

Figure 12: Coronal tSNE Analysis

some promise in separating Normal and Tumor images, while GLCM has clustered many of the Cyst images together.

4.2 PCA

Another way we modified our data was by using Principal Component Analysis (PCA), which is the process of changing the basis of the data to a basis with minimal variance along one axis, then collapsing down and removing that axis. PCA can be used to visualize data, but we used it to form feature vectors that encoded similar information with fewer dimensions, to increase efficiency.

For both coronal and axial, we will show PCA decomposition charts, which demonstrate how much of the total variance can be explained at each decreased dimension.

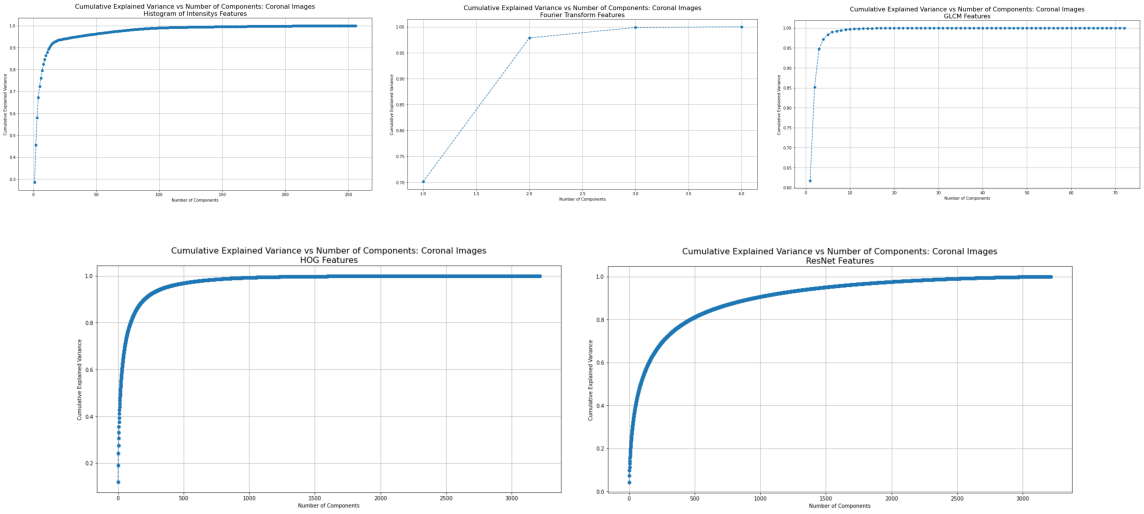

Figure 13: Coronal PCA Analysis

Looking at the coronal feature PCA charts, we see the variety both in the number of features, and in how many dimensions are needed to encode them (Figure 13). Fourier transform only starts with 4 dimensions, because we utilized summary statistics, and almost all of its explained variance can be maintained with only two dimensions. ResNet50, on the other hand, not only starts much larger but requires many more dimensions to encode the data, which you can see by comparing the grade of that curve to the others. For part of our experiments, we used versions of our feature vectors that had gone through PCA, to be reduced to the smallest dimension possible that still explained at least 95% of the variance. For our coronal feature vectors, this affected the size of the features in the following way:

Histogram of Intensities: started with 256 features, reduced to 35 features

Fourier Transform: started with 4 features, reduced to 2 features

GLCM: started with 72 features, reduced to 4 features

HOG: started with 14,400 features, reduced to 357 features

ResNet50: started with 100,352 features, reduced to 1,491 features

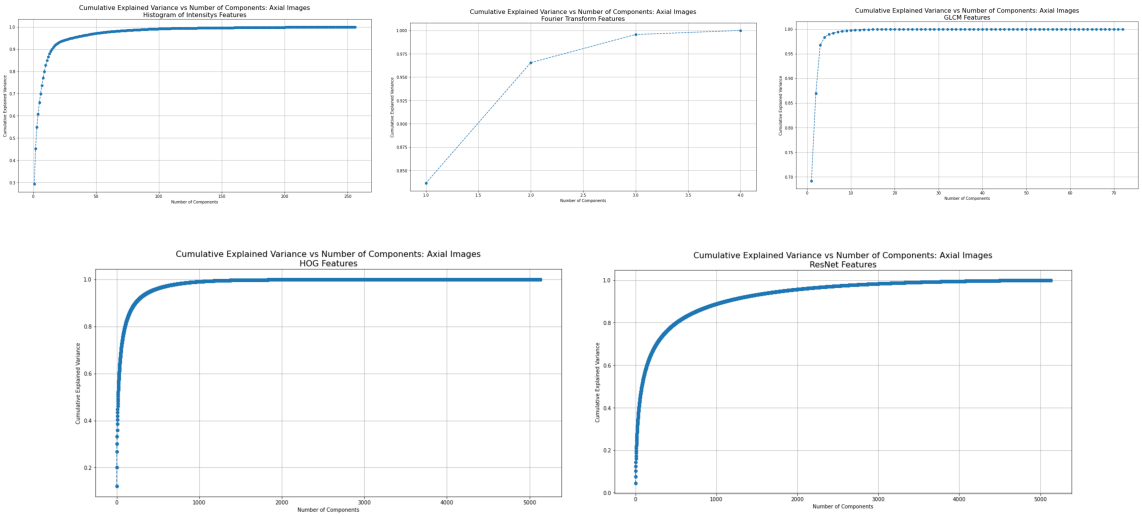

Figure 14: Axial PCA Analysis

In most cases, the axial PCA plots look similar to the coronal ones (Figure 14). However, when we used PCA to reduce the dimension, for many of the features, the number of dimensions needed to encode the information while maintaining 95% of the variance was either slightly higher, indicating a higher level of complexity, or slightly lower, indicating that the features were more able to be consolidated.

Histogram of Intensities: started with 256 features, reduced to 32 features

Fourier Transform: started with 4 features, reduced to 2 features

GLCM: started with 72 features, reduced to 3 features

HOG: started with 14,400 features, reduced to 411 features

ResNet50: started with 100,352 features, reduced to 1,836 features

5 Classification

5.1 Data Preprocessing

Once we defined our features, we were able to begin experimenting. Because our data contained multiple images per patient, we had to develop a unique methodology for splitting our data into train, validation, and test sets. We refer the reader to the Generalizability section later in our report for a description of it.

Once we defined our train, validation, and test sets, one of each for the coronal and axial images, we had to define our features. We concatenated the following features together in order to make one feature vector per image: Histogram of Intensities, Fourier Transform, GLCM, HOG, and ResNet50. This left us with one feature vector per image with a length

of 115,084. This was then scaled with standard scaling, so each feature has a mean of 0 and a standard deviation of one. This then created the 'scaled' set of data.

We created one more set of data to experiment on, based on the data that had been put through PCA. We concatenated the results of PCA together to create this data, which was left with 1,889 features per image in the coronal data, and 2,284 features per image in the axial data. This created the PCA set of data. Once we had these sets, we were able to start experimenting with models.

5.2 Logistic Regression

We chose a logistic regression model as our baseline model due to its simple nature. Although it is not capable of complicated spatial recognition, it provides an interpretable model for this complicated task. We evaluated this model on four separate sets, the scaled version and the PCA version of both the coronal and axial data. For the model, we used standard gradient descent as an optimization function, and sparse categorical cross entropy as a loss function. When evaluated on the Scaled Coronal dataset, the model achieved 100% train accuracy, as well as 62.0% validation accuracy and 50.5% test accuracy. For the Scaled Axial dataset, the model again achieved 100% train accuracy, 55.4% validation accuracy and 66.9% test accuracy. When evaluated on the PCA feature dataset, it achieved a 100%, 69.3%, and 61.9% accuracies on train, validation, test, respectively, for the Coronal PCA feature datasets. On the Axial PCA features datasets, it achieved 100%, 54.4%, and 64.5% accuracies on train, validation, and test, respectively. You will notice that these models, though they were intended to be baselines, already overfit on the training set. We will discuss why we think this is happening in the Generalizability section.

It is worth noting already that these results are better than randomly guessing the majority class, which would result in a 38% accuracy for the coronal data, and 45% accuracy for the axial data (Figure 15).

5.3 Neural Network

We then implemented a second model we discussed in class, a Neural Network with one hidden layer with 32 neurons. For this model, we used Adam as an optimizer, and sparse categorical cross entropy as the loss function. We chose this model because we initially thought the additional complexity would help our model increase accuracy. This, however, did not end up being the case. Logistic regression overfit, so the additional complexity only exacerbated the issue. On the Scaled Coronal dataset, the Neural Network model achieved 100% train accuracy, 64.8% validation accuracy and 52.0% test accuracy, while on the Scaled Axial dataset, it had a train accuracy of 100%, with validation and test accuracy of 46.6% and 61.8%. For the PCA Coronal dataset, the model achieved a train accuracy of 100%, a validation accuracy of 63.8% and a test accuracy of 48.3%. Finally, on the PCA Axial dataset, the model achieved a train, validation, and test accuracy of 100%, 41.2% and 40.5% (Figure 16).

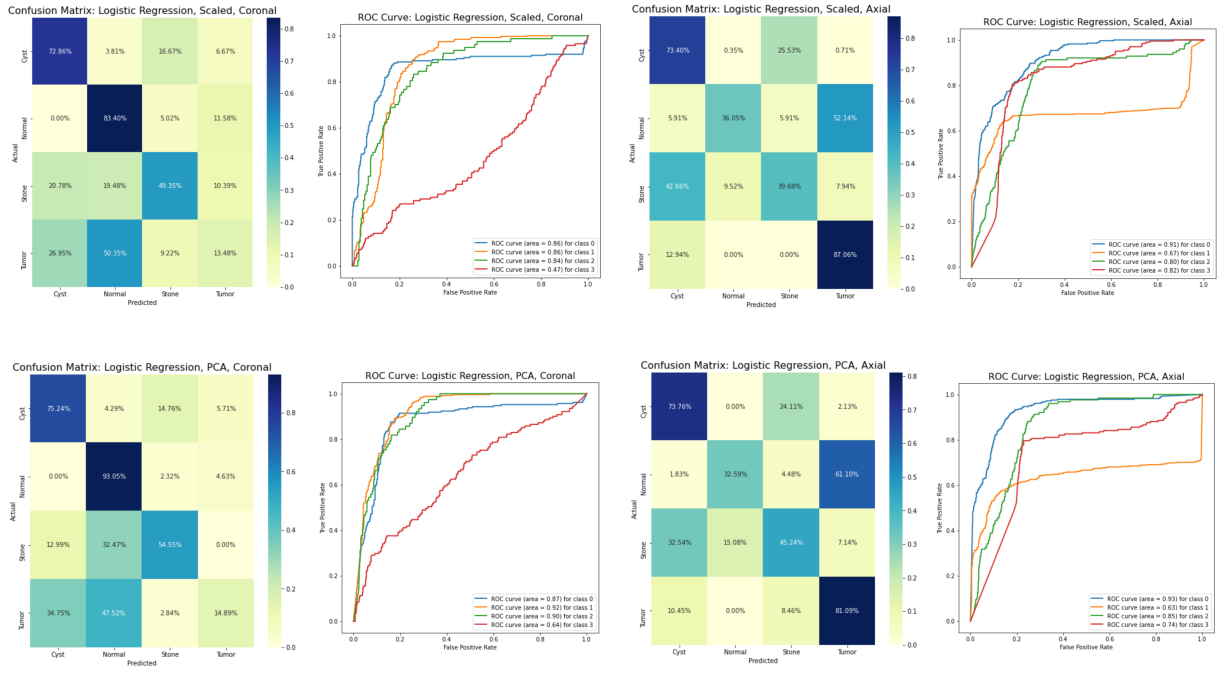

Figure 15: Confusion Matrices and ROC Curves for Logistic Regression Models

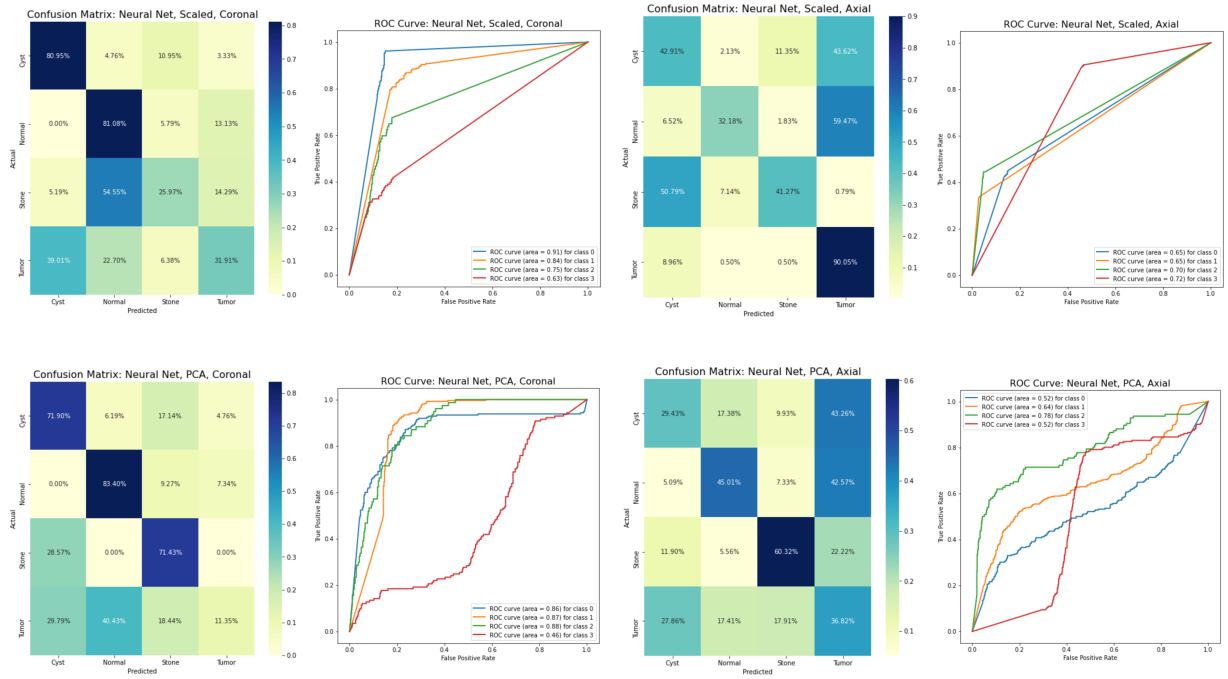

Figure 16: Confusion Matrices and ROC Curves for Neural Network Models

5.4 Neural Network with Regularization

After having issues with our other models overfitting, we decided to experiment with adding regularization parameters to our neural network. We thought this may improve the generalizability of our model. The base model had the same specs as our previous neural network, but to both the hidden layer and the output layer we added a kernel L1 regularizer with a value of 0.05. This means that to each layer, 0.05 times the sum of the absolute value of the weights. This incentivises low weight values, which helps avoid overfitting. Not only did this model decrease overfitting, it also improved performance. The Neural Network with Regularization model achieved 80.8% training accuracy on the Scaled Coronal dataset, along with 52.3% validation accuracy and 58.6% test accuracy. On the Scaled Axial dataset, the model achieved 86.3% accuracy on the training set, as well as 55.1% on the validation set and 66.0% on the test set. Because the PCA data has less complexity, there was more overfitting on the PCA sets, but still not as much as the models without regularizers. On the PCA Coronal Dataset, the model had a train accuracy of 99.3%, with validation and test accuracy of 72.5% and 60.2%. On the Axial PCA features datasets, it achieved 99.2%, 64.6%, and 70.4% accuracies on train, validation, and test, respectively (Figure 17).

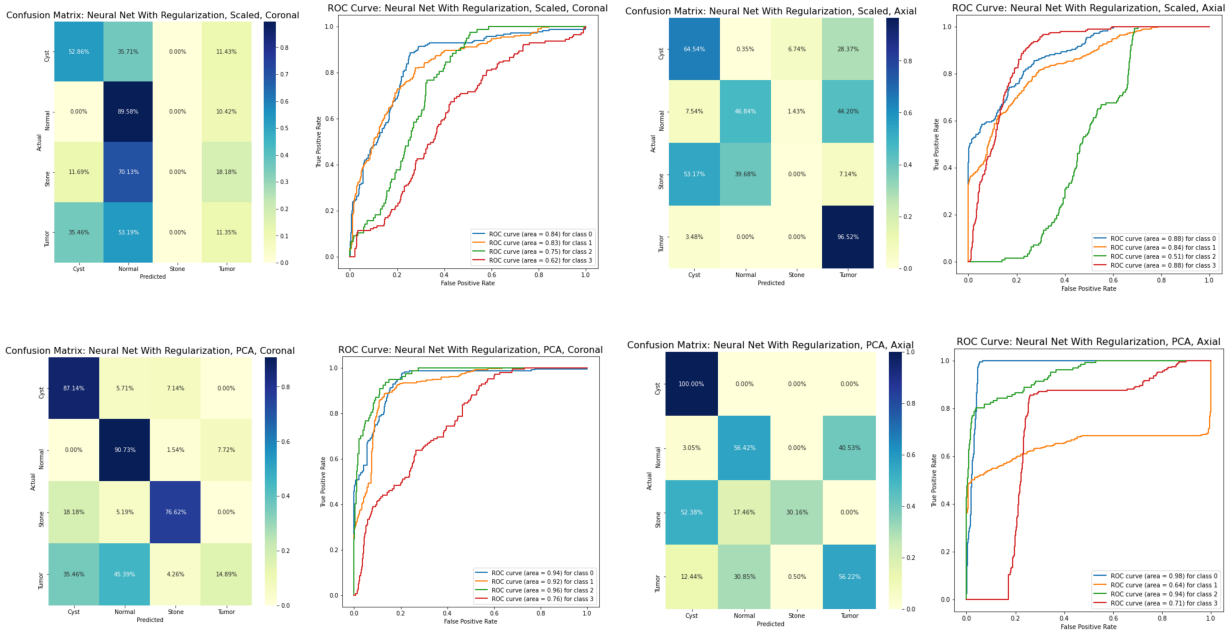

Figure 17: Confusion Matrices and ROC Curves for Regularized Neural Network Models

5.5 XGBoost

XGBoost is a machine learning algorithm based on gradient boosting, designed for supervised learning tasks such as classification. It builds an ensemble of decision trees sequentially, where each tree corrects the errors of the previous one by optimizing a specific loss function using gradient descent. In our case, the objective was a multiclass softmax. We choose XGBoost as

a model because it supports regularization to prevent overfitting and parallelizes computation to enhance efficiency and accuracy. It is also capable of supporting easy hyperparameter tuning, which we performed and will discuss in the next section. When evaluated on the PCA feature dataset, it achieved a 100%, 70.2%, and 64.1% accuracies on train, validation, test, respectively, for the Coronal PCA feature datasets. On the Axial PCA features datasets, it achieved 100%, 57.9%, and 75.0% accuracies on train, validation, and test, respectively. When evaluated on the Scaled Coronal dataset, the model achieved 100% train accuracy, as well as 74.1% validation accuracy and 68.1% test accuracy. For the Scaled Axial dataset, the model again achieved 100% train accuracy, 52.8% validation accuracy and 62.7% test accuracy. However, it is worth noting that these models took the longest to train, each taking over 20 minutes. We will discuss this in more detail in the Efficiency vs Accuracy section. Because the PCA features were less complex, they also were quicker to train, at around 30 seconds. For the PCA Coronal dataset, the model achieved a train accuracy of 100%, a validation accuracy of 69.9% and a test accuracy of 65.8%. Finally, on the PCA Axial dataset, the model achieved a train, validation, and test accuracy of 100%, 57.6% and 73.1% (Figure 18).

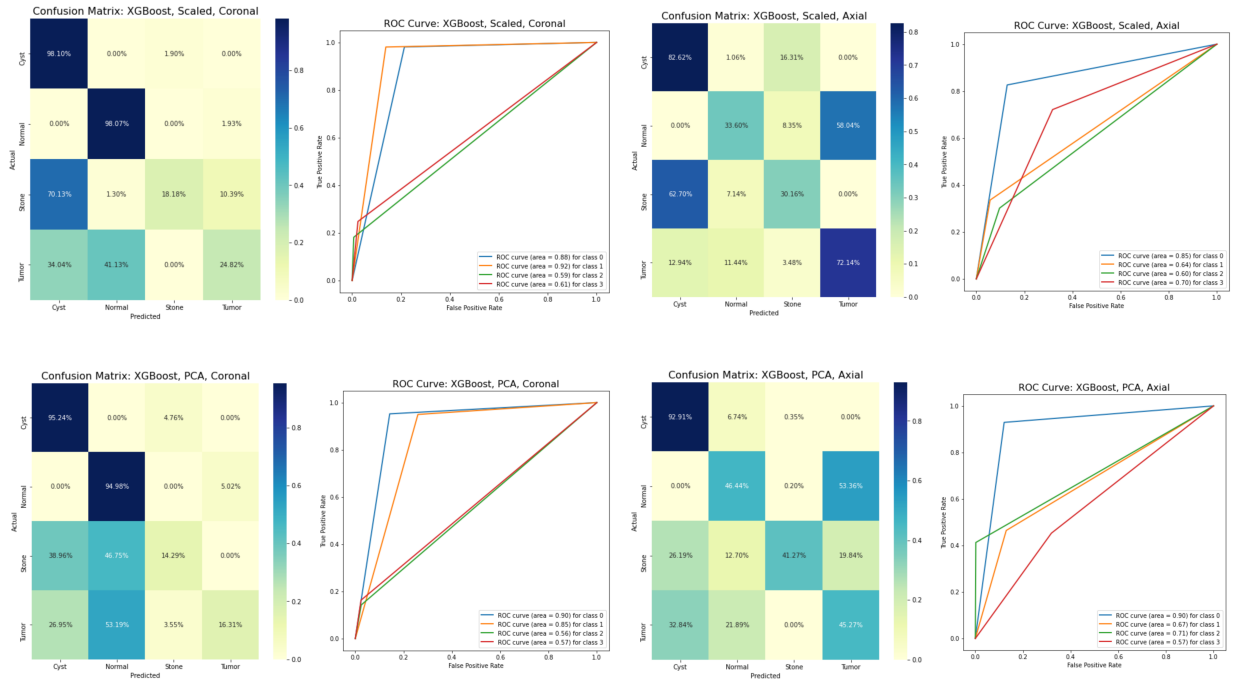

Figure 18: Confusion Matrices and ROC Curves for Regularized Neural Network Models

6 Generalizability

One complexity with this dataset was that there were multiple CT scans belonging to the same patient's CT scan session. This is because CT scans involve taking many images that can be constructed into a 3D visualization. Although this was helpful for the purpose of

bolstering our dataset with additional images, thereby increasing its size, this also meant that if we applied a standard shuffled train/test split, we would be inevitably causing data leakage in our models. This is because, since the images are near-identical for the same patient, having one of the patient's series in the train and another in the test set, would mean that we would be essentially testing the classifier on training data, and hence not actually evaluating the model's behavior on unseen data. The ideal method to deal with this issue would have been to group together images from the same patient, assign them an ID, then shuffle the dataset according to the ID of the patient. However, this was impractical for our project, as the dataset had been de-identified for privacy, and with 12,446 images, grouping these images by patient manually was impossible. If we had more time for this project, we could have developed an algorithm to group CT scans by their similarity to each other, then verify a sample manually. This would be a potential future improvement on this project.

As a result, we had to develop a custom train test split, with the following schema: First, we split the ordered dataset into a train set, validation set, and test set with a 70:15:15 ratio respectively. This kept the CT images from the same patients together, with only 2 patients' image series on the cusp of the train/validation and validation/test splits, potentially being leaked from the train to the validation set or from the validation to test set. Then, we shuffled each of the train, validation, and test datasets separately.

Further, we used the os library from python to read in our files, and learned that the `os.listdir` function, which we used to list all the images in the file directory, shuffles the file names according to the operating system order, which results in a textually-arbitrary order. We defined a numerical sort function (Figure 19), to sort the file names. Then, we were able to extract the ordered images and split the dataset with our custom train/test split method.

```
def num_sort(test_string):
    return list(map(int, re.findall(r'\d+', test_string)))[0]
```

Figure 19: Numerical Sorting to Resolve os.listdir Shuffling

We implemented a pipeline for tuning and evaluating the XGBoost classifier using a custom cross-validation strategy and hyperparameter optimization via grid search. The dataset was first stratified by class (Normal, Cyst, Tumor, Stone) and split without shuffling into training and test subsets, maintaining proportional representation of each class in both sets. Then, for each of the unshuffled four sets, a fold was created. Then, the four train sets were merged into one dataset, and similarly the four validation sets were merged into one dataset. Both train and validation sets were subsequently shuffled to ensure randomness, with proportionally balanced class distributions in both splits.

This method was integrated into a grid search procedure to systematically explore a range of hyperparameters for the XGBoost classifier, using cross-validation accuracy as the optimization criterion. The grid search identified optimal parameters for learning rate, maximum depth, minimum child weight, number of estimators, alpha regularization, and lambda regularization.

Although we did achieve relatively strong performance on the test set for our models,

with best-performing test accuracies of 75% on the PCA-axial set, and 67% on the non-PCA coronal set, we did find that the models consistently overfit from the train, to validation and test sets. We suspect that this was due to the false illusion of abundant data. Despite there being nearly 12,500 data points, from observation we found that groups of around 15 images were near-identical, likely belonging to the same patient. As a result, despite having more data, our models were likely not learning anything new past the estimated 800 overall unique images or patients. Further breaking this number down, this meant that across all the train, validation, and test sets, there were likely only around 300 unique patients for the coronal models, and approximately 500 unique patients for the axial models. We attribute some overfitting to the artificially exaggerated dataset size.

We tried to combat this unique characteristic of our dataset, but introducing alpha and lambda regularization, reducing the number of estimators, and generally adjusting our hyperparameters to accommodate models of smaller size. Unfortunately we were unable to avoid overfitting completely.

Even for a trained medical professional, determining through a CT scan whether a kidney contains stones, tumors, or cysts is difficult. Since the dataset we used is from kaggle, there were associated notebooks and papers with the dataset. We noticed that many of these papers yielded near-perfect accuracy, between 98-99.99%. We immediately flagged these papers to reassess after our project, as we did not believe their accuracy at first. After working through our project and learning about the potential of data leakage with using a standard train/test split on this dataset, we re-read those papers and notebooks, and realized that the vast majority of them fell victim to this data leakage. As a result, we believe that our model is actually vastly more generalizable than the models described in these notebooks. Although our model is not perfect, we were able to present honest performance on unseen data, which was a tricky problem for this particular dataset.

7 Results

We were able to improve upon the baseline logistic regression. The classifier that performed the best was XGBoost, followed relatively closely by the Neural Net with Regularization (Table 1). We believe that this has to do with the gradient boosting nature of XGBoost, and the increased generalizability of the model with regularization. Additionally, in these cases, the PCA features outperformed the scaled features. Our hypothesis is that in these cases, the slight loss of variance that comes with PCA provided noise that helped the model generalize better to unseen cases.

Within the confusion matrices referred to by the Classification section, we can see how performance of the models varied across the different experiments. As we mentioned at the beginning, we had a bit of a class imbalance in our data, with the Cyst and Normal classes being larger than the Tumor and Stone. Starting by looking at the results, we see that results generally were similar when comparing the Linear Regression results of the Scaled set to the PCA set. But, in other models, PCA showed general improvement over the Scaled set, normally creating a more balanced classifier.

Results were very different when comparing the coronal images to the axial images. Classifiers working on the axial images tended to do much better at classifying the tumor

Model	Image Type	Data Type	Train Acc	Val Acc	Test Acc	Train Time
Multiclass Logistic Regression	Coronal	Complete Scaled Feature Vector	100.0%	62.01%	50.51%	0.4885 minutes
		PCA Applied Feature Vector	100.0%	67.25%	56.71%	0.0486 minutes
	Axial	Complete Scaled Feature Vector	100.0%	55.36%	66.91%	0.7884 minutes
		PCA Applied Feature Vector	100.0%	53.45%	59.47%	0.0692 minutes
Neural Net	Coronal	Complete Scaled Feature Vector	100.0%	64.77%	51.95%	0.6371 minutes
		PCA Applied Feature Vector	100.0%	63.76%	48.34%	0.0532 minutes
	Axial	Complete Scaled Feature Vector	100.0%	46.55%	61.83%	1.0226 minutes
		PCA Applied Feature Vector	100.0%	41.27%	40.53%	0.0844 minutes
Neural Net With Regularization	Coronal	Complete Scaled Feature Vector	80.8%	52.26%	58.59%	0.8831 minutes
		PCA Applied Feature Vector	99.28%	72.49%	60.17%	0.0576 minutes
	Axial	Complete Scaled Feature Vector	86.34%	55.09%	66.0%	1.3433 minutes
		PCA Applied Feature Vector	99.16%	64.55%	70.35%	0.0808 minutes
XGBoost	Coronal	Complete Scaled Feature Vector	100.0%	74.09%	67.97%	22.54 minutes
		PCA Applied Feature Vector	100.0%	69.87%	65.8%	0.433 minutes
	Axial	Complete Scaled Feature Vector	100.0%	52.82%	62.74%	26.4131 minutes
		PCA Applied Feature Vector	100.0%	57.55%	73.07%	0.5892 minutes

Table 1: Overall Results Table for All Variations of Models

class than those working on the coronal images. This may have been because in the axial plane, the irregular nature of the tumor may have stood out more.

XGBoost tended to have very high accuracy on the majority classes, cyst and normal, and almost ignored the two other classes when looking at the coronal data. It was able to find more balance on the axial data.

We believe that our biggest limitation was lack of data. As we have mentioned, after we began working with the data, we realized that patients had multiple images in the set, meaning we had way less unique data than we initially thought. We saw the consequences of this when even a simple Logistic Regression model quickly overfit to the training data. With more data, our classifier could better learn general trends and avoid overfitting.

8 Efficiency vs Accuracy

Training times for each model are listed in the table in the Results section. Using these, we can consider how efficient each model is compared to its accuracy. You will notice that almost all of our models trained in less than a minute, with the major exceptions being the XGBoost models trained on the scaled data, which each took over 20 minutes to train. Additionally, both of the neural networks took about a minute to train on the Axial Scaled data.

Our most efficient model was the Coronal PCA Logistic Regression model, which took just under 3 seconds to train. This makes sense, because this model had the least amount of data, as there were fewer coronal records and the PCA data was lower dimension than the scaled data.

When looking at accuracy, for the axial models, the model with the highest test accuracy was the Axial PCA XGBoost, and for coronal it was Coronal Scaled XGBoost. When looking at the validation percentages, however, the model with the highest accuracy stays consistent for the coronal data, but for the axial data it is the Axial PCA Neural Net with Regularization. This variability in the best model speaks to the fact that the models overfit, so can be unpredictable on new data.

In the case of the Axial data, because the most accurate models were based on the PCA data, the most accurate models were also almost the fastest. The Axial PCA Neural Net with Regularization model was less than a second slower to train than the fastest Axial model, and even the Axial PCA XGBoost model took only 35 seconds to train. We would select the Axial PCA Neural Net with Regularization model as the one that best balances efficiency and accuracy, as it was the most accurate model on the validation set, the second most accurate model on the test set, and the second fastest Axial model to train.

For the Coronal data, the most accurate model, Coronal Scaled XGBoost, was also the slowest to train. In this case, the model that best balances efficiency and accuracy would be the Coronal PCA XGBoost model, as it only took 26 seconds to train and had the second highest test accuracy.

9 Conclusion

We were able to create efficient models for both Coronal and Axial images that improved over both the baseline of randomly guessing the majority class, and our logistic regression baseline. We found that in many cases, using the PCA reduced data in our models improved both accuracy and efficiency. We also were able to see significant differences in how the same classifiers performed across two different image views, and what classes each view was better at identifying. A major breakthrough in our project was the successful integration of segmentation-driven enhancements. By generating high-quality masks to isolate kidney regions, we unlocked the potential for more focused and accurate classification. This effort required overcoming persistent challenges, including stagnant model losses, noisy pseudo-labels, and the refinement of mask quality over multiple iterations. Our adaptive approach to label refinement significantly improved mask confidence, transforming an initially crude process into a reliable and repeatable method. The experience highlighted the power of persistence, iterative development, and human-in-the-loop feedback to overcome model limitations.

We believe that our biggest challenge was not having as much unique data as we thought. Because each patient had multiple scans, it's uncertain how much novel data we had, and that is likely why our models overfit so significantly. However, we were able to mitigate this some with regularization and developing a bespoke data splitting method to maximize generalizability.

References

[1] Reza Azad et al. *Medical Image Segmentation Review: The Success of U-Net*. arXiv preprint. Nov. 2022. URL: https://arxiv.org/abs/2211.14830.

[2] Alexander Berkovich. *Introduction to Morphological Image Processing: Techniques and Applications*. Nov. 2024. URL: https://akridata.ai/blog/morphological-image-processing-techniques-applications/.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[4] MN Islam et al. "Vision transformer and explainable transfer learning models for auto detection of kidney cyst, stone and tumor from CT-radiography". In: *Scientific Reports* 12.1 (July 2022), pp. 1–4.

[5] Leo Yu-Feng Liu et al. "Masked Convolutional Neural Network for Supervised Learning Problems". In: *NIH Library of Medicine* (Apr. 2020). URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC7785089.

[6] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*. arXiv preprint. Apr. 2017. URL: https://arxiv.org/abs/1506.01186.

[7] Liangchen Song et al. "Learning from Synthetic Images via Active Pseudo-Labeling". In: *ACM Digital Library* (Jan. 2020). URL: https://dl.acm.org/doi/10.1109/TIP.2020.2989100.