Data Mining Algorithms for Image Classification of the Foveal Avascular Zone

I) Introduction

Machine learning algorithms are being used in the medical industry to help doctors diagnose diseases, cancers, and more. This project looks at the usefulness of applying data mining and machine learning algorithms to diagnose diseases of the eye. In the eye, different diseases prompt changes to the size and shape of features in the Foveal Avascular Zone (FAZ). The project aims to analyze images of the FAZ and classify them into three categories (Diabetes, Myopia, and Normal) based upon differences inherent in the image. The goal of this research is to help ophthalmologists diagnose diseases visible from the FAZ quicker and more accurately than traditional methods.

This report consists of eight key sections:

(1) Introduction
(2) Description of the Data
(3) Description of the Algorithms Used
(4) Experimental Setup
(5) Results
(6) Summary and Conclusions
(7) References
(8) Appendices

The code used to produce this analysis can be found here: https://github.com/sacohen11/Final-Project-Group6.

II) Description of the Data

The dataset was found on Open ICPSR (Inter-university Consortium for Political and Social Research), which is a platform for sharing "social, behavioral, and health sciences research data" (Agrawal, Raman, and Lakshminarayanan). It consists of 304 retinal fundus images of the FAZ inside the eye, each with a corresponding label that classifies the eye as diabetic (107 images), myopic (109 images), or normal (88 images). Each image is originally a 6mm x 6mm picture of the retina, which has dimensions of 420 pixels x 420 pixels. An example of an image can be seen in Appendix A.

The researchers who published the data included a two datasets: one with "clear", unedited pictures and one with "marked" pictures in which the area of interest was marked in color. This project only examined the clear image dataset.

III) Description of the Algorithms Used

This project examined three classification algorithms: Support vector machine, decision tree, and k-nearest neighbor. In addition, these three methods were ensembled using a maximum voting algorithm to produce a final, ensembled model. Lastly, the team used a 5-fold cross validation technique to verify the results of each model.

a) Support Vector Machine (SVM)

A support vector machine is a data mining algorithm which uses linear algebra to draw a hyperplane. The hyperplane separates multiple classifications of labeled data. Multiple kernel types exist for SVM. This project used a linear kernel to specify the SVM, as the linear kernel was found to increase accuracy the most.

b) Decision Tree

A decision tree is a graph-based data mining algorithm which uses decision rules to make distinctions between groups in the data. Each decision rule, which represents a node in the decision tree, looks at a feature in the data and uses a cost function to determine which decision to make at each node. This project used entropy as the cost function. The maximum depth of the tree was specified to be 20 leaves to prevent overfitting.

c) K-Nearest Neighbor (KNN)

K-Nearest Neighbor is a data mining algorithm that examines the k closest points to a chosen point and uses a distance measure to compute a classification for each test point. This project chose to use the standard Euclidean distance measure as our distance metric. This project also chose a k of 17, as that k increased accuracy the most holistically.

An additional algorithm was applied to the KNN models to determine the best k (the choice of k that increased accuracy the most) for each pre-processing technique. The results of the best k analysis were reviewed by the team, but the choice of best k was not used in the final model. This decision was made so that the final model could cut down on computation time.

d) Maximum Voting Ensemble Algorithm

After these three methods were applied to each pre-processing technique used, a maximum voting algorithm was used to ensemble the predictions of each method. This ensemble method uses a function in Python called mode() to take the most common prediction for each image in the dataset. If there is a single classification category with the most predictions, that classification will be the final prediction in the ensembled model. If there is a tie, a random classification is chosen between the tied categories. The purpose behind this algorithm is to use the strengths from each of the three previous data mining algorithms to produce a boosted model.

After each of the initial three models (SVM, Decision Tree, KNN) were run, a 5-fold cross validation was performed on the datasets This procedure randomly splits up the training and testing differently five times, fits the new training data to a model, and then makes predictions on the new test data. The team took the mean accuracy from the 5-fold cross validation and compared it to the initial accuracy from the first pass of the model. Although the 5-fold cross validation accuracies were not used in the final results or ensembled model, the team used the cross-validation as a check to ensure the initial model produced was consistent with other model predictions.

IV) Experimental Setup

To begin pre-processing, the image was cropped to remove a text banner located at the bottom of each image. Then, the image was resized using a resize function created by the team. Each image was also put into grayscale. This way, each image was a consistent size and contained only the view of the FAZ.

To preprocess our data, the project used seven different techniques: original image, crop/zoom on the original image, edge detection, feature creation, feature creation on the cropped/zoomed image, applying a binary inverse threshold on the original image, and applying a binary inverse threshold on the cropped image. Each of these techniques were used to create a different training/test set that was applied to each of the three above models (SVM, Decision Tree, KNN). The seven methods are described below. Each method has an appropriate function in the combo.py file on the team's Github. Examples of each pre-processing technique can be seen in Appendix A.

a) Original Image

For this method, the original image was not altered. The Python function used is called no_transformation().

b) Crop/Zoom on Original Image

For this method, the original image was cropped to zoom in on the black dot in the center of the FAZ zone. This dot is the focus point for classifying the type of disease. The Python function used is called no_transformation().

c) Edge Detection

For this method, the original image was modified using a Canny edge detection algorithm from the Python computer vision package OpenCV. The resulting image shows only the major edges present in the original image. The Python function used is called edge_detection().

d) Feature Creation

For this method, the original image was put through the a KAZE algorithm in OpenCV. This algorithm finds the key points and other descriptors in the image and returns a vector consisting of these key features. This is the only method used that does not return an image. The Python function used is called feature_creation().

e) Feature Creation on Cropped/Zoomed Image

This method is the same as described above, except it was performed on the cropped/zoomed image. The Python function used is called feature_creation().

f) Binary Inverse Threshold

For this method, and threshold function was applied on the original grayscale image. The threshold function (binary inverse) set the value of each pixel to either black (0) or white (255) based on a threshold value. The threshold value chosen was 30, as this was seen to improve accuracy the greatest. Then, the pixel values were reversed on the image, so black became white and white became black. This allowed the team to isolate the inner dot and turn the color to white. The Python function used is called threshold().

g) Binary Inverse Threshold on Cropped/Zoomed Image

This method is the same as described above, except it was performed on the cropped/zoomed image The Python function used is called threshold().

Next, the data for each pre-processing method was split into a training set and a testing set. An 80%-20% split was chosen to split the data. After the set was split, data augmentation was applied to the minority class.

h) Data Augmentation

Data augmentation was implemented on the minority class. Data augmentation is a technique that increases the number of images in the minority class by applying transformations (such as rotation, flipping, etc.) to the original image and adding the transformed image to the training set. For each pre-processing technique aside from feature creation on the original and cropped/zoomed image, data augmentation added 1 image to the Diabetic classification and the 17 images to the Normal classification so that each classification would have the same number of images in the training set (87 images).

Data augmentation was done using the augment() function created by the team in Python. The function chooses a random number between 0 and 4 and, based upon the random number, decides to add a transformed image to the minority classes. The transformations in the function are flipping the image, adding noise to the image,

rotating the image 90 degrees, shifting the image 100 pixels in both directions using a warp function, and rotating the image 180 degrees.

After data augmentation, a Standard Scalar was applied to the image to ensure that each the scale of each image was standardized. The 21 training and testing sets (created from 7 pre-processing techniques applied to 3 models) were subsequently ready for the prediction models.

To judge the results and success of each model, the team decided to use accuracy as its measure. Accuracy is defined as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Number\ of\ total\ predictions}\ X\ 100$$

The team also viewed confusion matrices to identify for which classifications were the models predicting better or worse.

V) Results

The accuracies for each model and pre-processing technique are shown in the table below. D.A. indicates the accuracy after data augmentation was applied to the training and testing sets. Those accuracies highlighted in blue were accuracies over 60% while the accuracies highlighted in red are accuracies under 40%.

Preprocessing	SVM Accuracy		KNN Accuracy		Tree Accuracy	
	Original	D.A.	Original	D.A.	Original	D.A.
No Transformation	63.9%	63.9%	44.3%	42.6%	34.4%	52.5%
Image Zoom	72.1%	67.2%	47.5%	47.5%	52.5%	47.5%
Value Threshold	60.7%	55.7%	44.3%	34.4%	47.5%	54.1%
Value Threshold & Crop	67.2%	65.6%	32.8%	32.8%	54.1%	36.1%

Edge Detection	54.1%	52.5%	37.7%	39.3%	32.8%	32.8%
Feature Selection	45.9%	X	42.6%	X	41%	X
Feature Selection & Crop	47.5%	X	39.3%	X	31.1%	X

Table 1: Model Accuracies

Across the board, the accuracies of the models are on the low side. A few observations stuck out to the team after reviewing the accuracies of each model. First, the SVM models fared significantly better than the decision tree and KNN models on the whole. The reason for this is uncertain. The team hypothesizes that, since the images are almost identical on the surface, algorithms such as KNN that rely on distance away from close points would fair poorly on very similar data. Yet, this is just a theory.

The second key observation the team made was that the data augmentation step, which is intended to boost the classification of the minority classes, had mixed results in terms of its effects on accuracy. For example, looking at the decision tree models, for the no transformation pre-processing technique, adding data augmentation increased accuracy by 18.1%. Yet, for the image zoom pre-processing technique, adding data augmentation decreased accuracy by 5%. Looking at the effects of data augmentation across all three models and all seven pre-processing techniques, it is hard to discern a general effect of data augmentation.

Next, the team dived deeper into the accuracy scores by examining the confusion matrices produced by each pre-processing technique and model. The table below shows each of the 21 confusion matrices. The darker the shade of blue, the more predictions were made for each category. The first column shows the predictions for the Diabetic classification, the second column shows the predictions for the Myopic classification, and the third column shows the predictions for the Normal classification. All the confusion matrices besides the feature creation matrices have data augmentation applied to them.

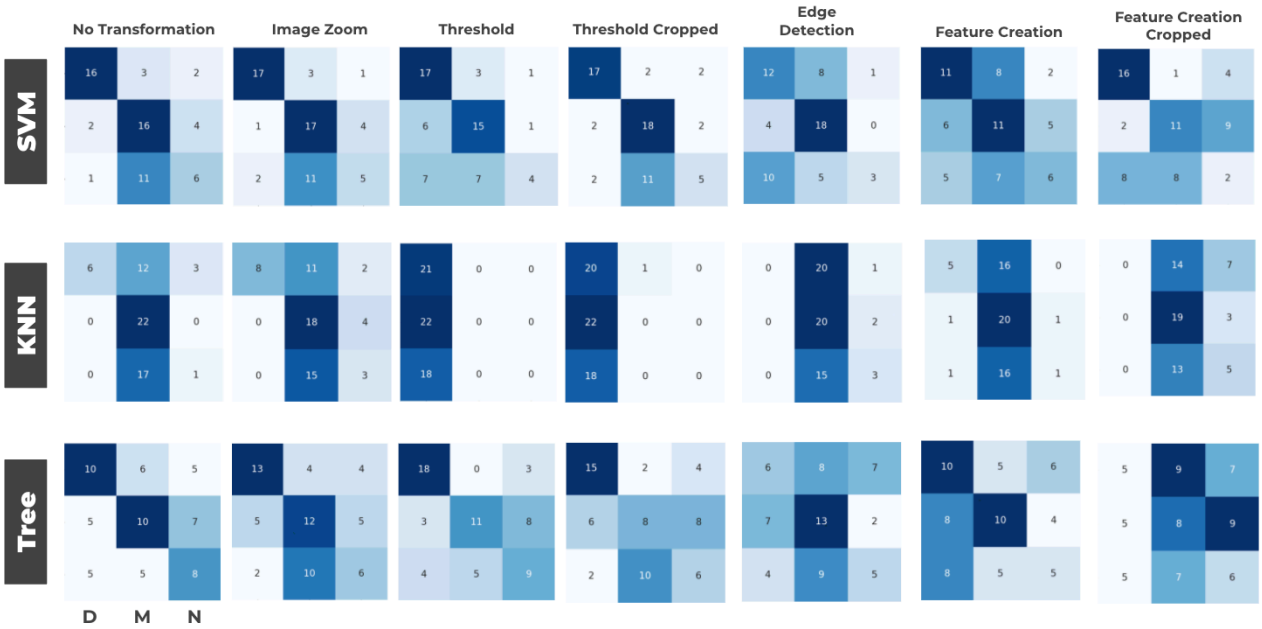


Table 2: Confusion Matrices

The team noticed that the normal classifications were frequently underrepresented in the predicted category. In the same vein, the Myopic category was most often predicted correctly. For example, looking at the SVM models, for each pre-processing technique, at least a majority of the Diabetic and Myopic classifications were True Positives, meaning that their predictions and labels were both Diabetic or Myopic. Yet, in the SVM models, the Normal classification never even had a plurality of the predictions be True Positives. The other two models are a bit more sporadic in their confusion matrix results, but the general trend is that the Normal classification was often mis-classified as Myopic.

The team has several hypotheses for these results. First, the Normal classification was the minority class in terms of image representation. 88 normal images were in the dataset compared to 107 Diabetic images and 109 Myopic images. The purpose of the data augmentation was to correct this imbalance, yet as we observed, the data augmentation step did not always increase accuracy. Therefore, the Normal images may be underrepresented in the dataset and are subsequently underrepresented in the model. The other hypothesis the team had is that the Normal images and Myopic images appear very similar and as a result of the underrepresentation of the Normal images in the data, the Normal images often get classified as Myopic. When viewing the Diabetic image, one can make out with their eyes the difference between those images and the Normal images. The black dot at the center of the FAZ is not as defined in the Diabetic images as compared to the Normal images. Yet, these distinctions do not exist between the Myopic and Normal images, at least not to the human eye. Therefore, due to the underrepresentation of the Normal images and the inherent similarities between the Myopic and Normal images, the algorithms are mixing up the Normal images as Myopic.

Another interesting observation in the confusion matrices is that the KNN models tend to have large numbers of the same predictions. For example, both threshold techniques have a large percentage of Diabetic classifications, while the other five pre-processing techniques have a

large percentage of Myopic classifications. This observation hits on the team's earlier hypothesis that the KNN algorithm performed poorly because the images were all very similar and distance between points was a poor measure for this type of analysis.

Finally, a max voting algorithm was applied to ensemble all 21 models together. The resulting accuracy and confusion matrix is shown below.

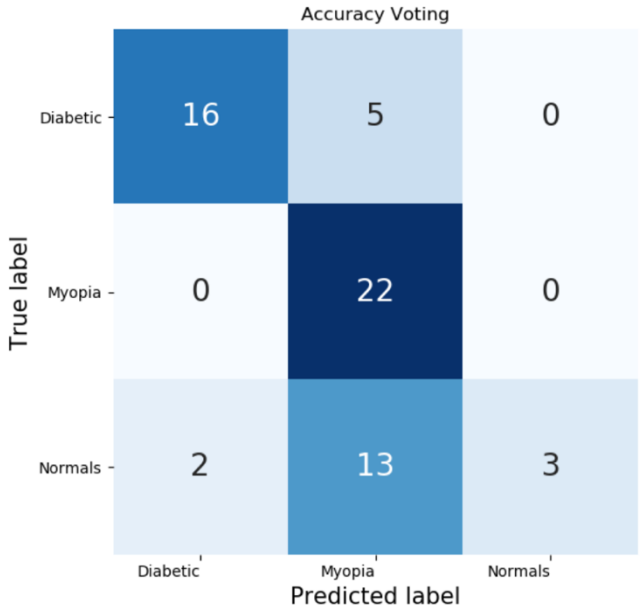


Table 3: Max Voting Ensemble Model

This ensembled confusion matrix corresponds to an accuracy of 67.2%. This ensembled accuracy score is higher than 92% of the 36 original models and is only less than one of the original models (the zoomed/cropped original image without data augmentation). These results show the power of ensembling various models together to get the strengths from each model. Still, the confusion matrix in Table 3 shows a majority of the Normal images being classified as Myopic. The trait of predicting Normal as Myopic is a weakness in the model.

VI) Summary and Conclusions

In summary, the final ensembled algorithm performed at a 67.2% accuracy. In addition, one can see from Table 1 that the SVM algorithm produced the results with the greatest accuracies for most of the pre-processing techniques. To boost the accuracies of these models, the team looks to collect more images of the FAZ to add to the dataset. Adding more original images will allow the models to learn from a larger subset of data and learn more about the characteristics of each classification of image.

For future research, the team would like to explore the effects of other classification algorithms on the dataset and continue to expand the number of pre-processing techniques used. The goal of this research would be to create a more accurate ensembled model.

VII) References

Research References

Agarwal, A., J, J. B., Raman, R., & Lakshminarayanan, V. (2020, February 10). The Foveal Avascular Zone Image Database (FAZID). Retrieved April 28, 2020, from https://www.openicpsr.org/openicpsr/project/117543/version/V1/view

Allibhai, E. (2018, October 2). Building a k-Nearest-Neighbors (k-NN) Model with Scikit-learn. Retrieved from https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a

OpenCV-Python Tutorials. (n.d.). Retrieved April 28, 2020, from https://docs.opencv.org/trunk/d6/d00/tutorial_py_root.html

Singh, A. (2020, March 22). A Comprehensive Guide to Ensemble Learning (with Python codes). Retrieved April 28, 2020, from https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

Support Vector Machines (SVM) in Scikit-learn. (n.d.). Retrieved April 28, 2020, from https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python

Code References

Bhatnagar, T. B. T. (2018). Resize an image without distortion OpenCV. Retrieved April 28, 2020, from https://stackoverflow.com/questions/44650888/resize-an-image-without-distortion-opencv

Fellman, N. F. N., & Fellman, N. F. N. (2010). How can I get the name of an object in Python? Retrieved from https://stackoverflow.com/questions/1538342/how-can-i-get-the-name-of-an-object-in-python

Jafari, A. (2020, February 25). Github. Retrieved April 28, 2020, from https://github.com/amir-jafari/Data-Mining

Nikishaev, A. (2019, November 30). Feature extraction and similar image search with OpenCV for newbies. Retrieved April 28, 2020, from https://medium.com/machine-learning-world/feature-extraction-and-similar-image-search-with-opencv-for-newbies-3c59796bf774

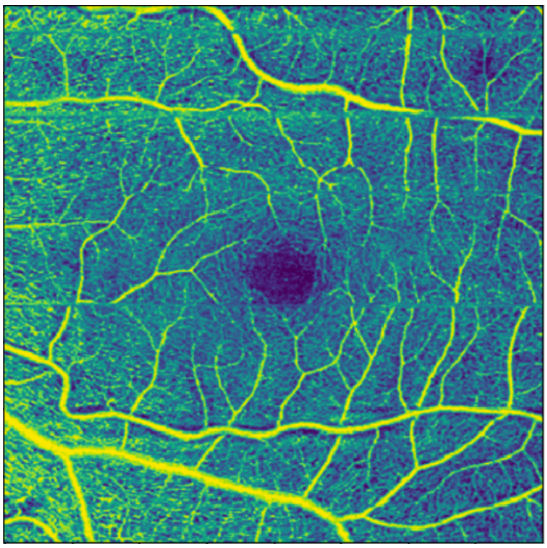
Python cv2.getRotationMatrix2D() Examples. (n.d.). Retrieved April 28, 2020, from https://www.programcreek.com/python/example/89459/cv2.getRotationMatrix2D

SanchitSanchit 2. (2014). How to add noise (Gaussian/salt and pepper etc) to image in Python with OpenCV. Retrieved from https://stackoverflow.com/questions/22937589/how-to-add-noise-gaussian-salt-and-pepper-etc-to-image-in-python-with-opencv

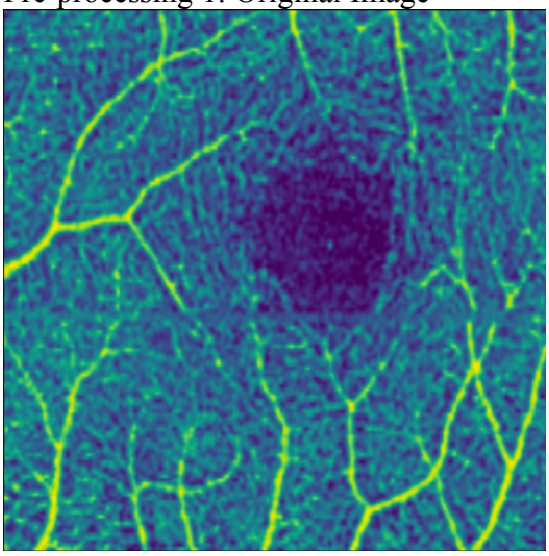
Translation of image. (n.d.). Retrieved April 28, 2020, from http://wiki.lofarolabs.com/index.php/Translation_of_image

VIII) Appendix A

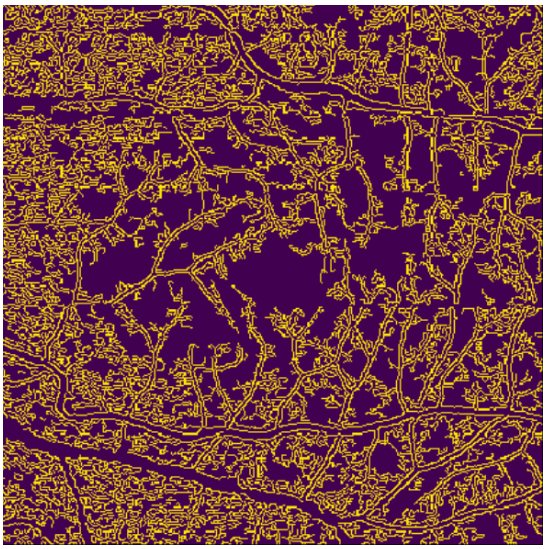
Examples Images with Different Pre-Processing Techniques Applied



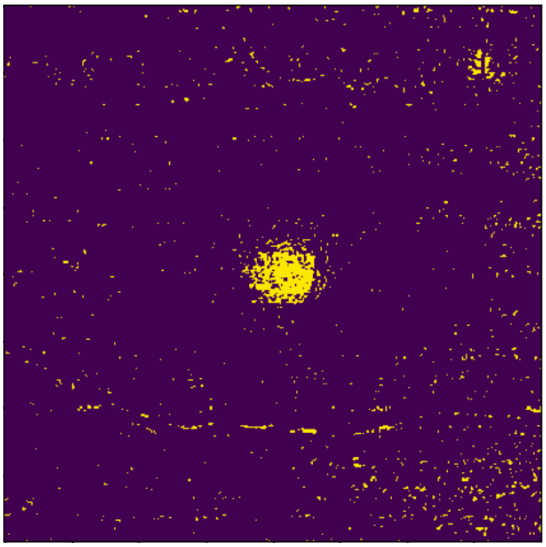
Pre-processing 1: Original Image



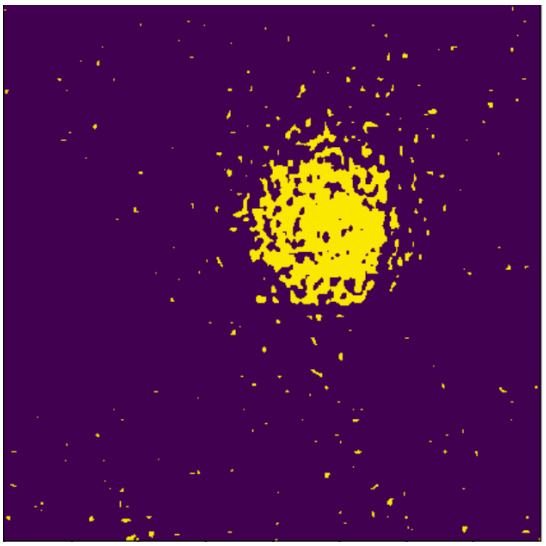
Pre-Processing 2: Crop/Zoom on Original Image



Pre-Processing 3: Edge Detection



Pre-Processing 6: Binary Inverse Threshold



Pre-Processing 7: Binary Inverse Threshold on Cropped Image