An Analysis on Machine Learning Techniques on the OLIVE Dataset

Damian Huerta-Ortega
Georgia Institute of Technology
damhue039@gatech.edu

Nathan Wang
Georgia Institute of Technology
nywang08@gatech.edu

***Abstract*—This document is an analysis of four different machine learning techniques applied on the OLIVE dataset for image classification.**

***Index Terms*—K-Nearest Neighbors, K-Means, Gaussian Mixture Models, Convolutional Neural Networks**

I. Introduction

In this paper, the team looks into potential ways to classify the OLIVE dataset based off severity levels. The four main techniques analyzed are K-Means, K-Nearest Neighbors (KNN), and Convolutional Neural Networks (CNN). The team analyzes the effectiveness of these four techniques and tries to rationalize why certain techniques may be better than others. The link to the GitHub can be found **here**. Presentation link: **video**

II. Methodology

A. Code Development

The team implemented all of the different algorithms using Python 3 in Jupyter Notebook files. These files were initially tested on Google Colab but then transferred over to the local computer using Visual Studio Code due to runtime disconnect and time out issues. Each algorithm is written in its separate ipynb file.

For the analysis of the images, the images were downsized to 224x224 pixel images as our default size from the initial size. This was done in consideration of the time and space needed to run the techniques. However, during our implementation and experimentation, we hit roadblocks for all clustering algorithms and KNN. Each sample was initially just a flattened version of the image with 50176 features that resulted in long training and poor accuracy results for the validation sets. We believe this likely is a result of the imbalanced dataset and poor clustering due to the similar pixel values and sensitivity to slightly rotated/blurry images. The team chose to combat these issues by representing each image as a tensor with 50 features that would best captivate information from each image. In order to translate each image into it's feature representation, the team made use of a custom Auto Encoder that takes in a vector 50176 features and attempts to learn the relationship to our 50-feature representation. The Auto Encoder implementation used an MLP approach with one fully connected layer from the input to the 50-feature sample-space and minimized the MSE as the loss function.

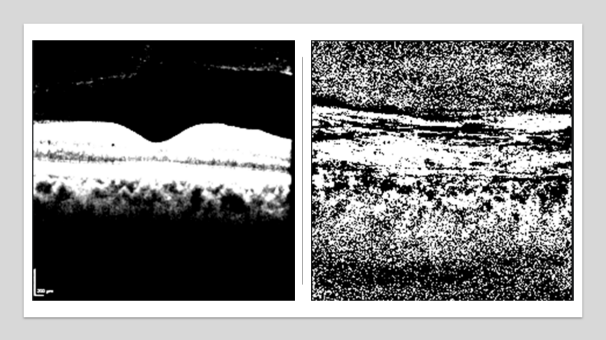

Fig. 1. normalized sample image (left) and reconstructed image as Auto encoder output (right)

As can be seen in Figure 1, the Auto Encoder is able to capture the general shape and shading of the original image with information that is captured in our 50-feature sample-space that will be used for our K-NN, K-Means, and GMM approaches. Each sample of the training and validation set was encoded into this 50-feature set and used as the samples fed to the samples mentioned above.

B. K-Means Algorithm

The first technique that the team focused on was the K-Means algorithm. K-Means focuses on trying to fit images into a certain class based on the number of centroids provided. Since the algorithm helps form clusters, the team inferred that using K-Means may be a good option for classification.

For K-Means, the team used the built in class in sklearn. The model was instantiated with 3 clusters (other amounts of clusters were tested as well) and the default initialization of the centroids. The images in the dataset were split into training and testing based off a 70-30 ratio. The encoded training dataset was then fed into the model to train it. After training the model, the centroids were then labeled accordingly to the classification of severity most associated with it. Following the identification of labels, the model then used the testing dataset to help classify the images into their closest centroid/label.

C. Gaussian Mixture Models

Since K-Means was the first classification technique used in the dataset analysis, the team decided to implement GMMs to compare the difference in effectiveness between the two

techniques. Since GMMs are typically the more effective technique due to it using soft classification compared to K-Mean's hard classification, the team hypothesized that the overall accuracy of GMM would be higher.

For the implementation of GMMs, the team used the built in class in sklearn. The model was instantiated with 3 clusters (other amounts of clusters were also tested) with a default initialization of the mean. The covariance was tested with different initialization of diag, spherical, tied, and full to analyze impact of covariance type on accuracy. The images in the dataset were once again split into training and test subsets of 70-30 ratios.

D. K-Nearest Neighbors

The first non-clustering algorithm we implemented was using the K-NN algorithms with various values of K to gauge the accuracy with a larger amount of neighbors considered. As in the previous clustering algorithms, each sample is a downsized image that is flatted into a tensor in order to have feasible memory usage. The set of all training samples are fed to the model for storage and each validation sample takes into account the class of the k nearest neighbors to take a majority vote and assign a class.

The K-NN algorithm is a very simple approach for classification but can perform rather well if the data is clearly separable. Unfortunately for our case, our gray-scale images have very similar data points for each pixel which likely would impact the accuracy. This is another clear motive for utilizing our 50-feature encoded samples produced by our Auto Encoder. Each sample is sent to the classifier and we end up with an array of the predicted class that can be used to check our accuracy.

E. Convolutional Neural Networks

CNNs were the final approach taken to classifying images in the OLIVE dataset. In our other attempts, we had to adjust each sample due to the high dimensions of each image. However, our CNN is able to downsize the sample with the use of convolutional and max pool layers. This means we were able to input the samples with a 224x224 resolution. Furthermore, this classifier was our most thorough approach with different approaches/optimizations including a custom CNN to using pre-existing models such as Resnet [NEED REF].

Our custom CNN model was loosely based on Alexnet with 21 layers that takes in images of size 224x224 and produces a tensor with the probabilities of a sample belonging to each class. This approach was our preferred out of all 4 due to the optimization ability in changing complexity, dropout, and loss functions. We started our experimentation with Cross-Entropy as our loss functions but made use of others such as MSE and Focal Loss.

III. Results

A. K-Means

The team ran the K-Means algorithm utilizing images of pixel sizes of 65, 128, and 224 and received the best testing accuracy of 13.62% with image sizes of 128 which is near guessing standards. During the training and prediction of testing samples, the team became aware of issues using flattened images due to memory usage, similar pixel values, training times, etc. Reducing the feature space seemed to be the best approach and the decision was made to encode each sample using an Auto encoder trained on the training samples as described in section II. With the Auto Encoder implementation, K-Means is able to achieve a testing accuracy of 43.87%. This result provided credibility that encoded samples would improve the performance of classification, which led to the use of the encoded-samples for future classifiers.

After obtaining these results, the team tried to try and understand why the model was still only able to achieve an accuracy that is not as high as expected. The biggest reason that the team found was due to the extreme similarity between all of the images regardless of severity level. All of the images provided had similar formats with only minor changes in the image in locations of the eye damage. This means that a good chunk of the images would have overlapping clusters. As K-Means uses hard classification, having an image dataset with overlapping clusters would make the model guess one of the classifications from the overlap.

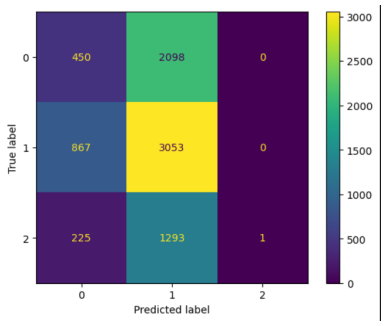

Fig. 2. Confusion Matrix for K-Means achieving a testing accuracy of 43.87%.

As shown in Figure 2, the confusion matrix of K-Means shows how our model performs in predicting classes compared to the true classes. This was the first instance we discovered there was a big class-imbalance with nearly half of the images belonging to class 1. Therefore, K-Means is biased to choosing class 1 and likely interprets most of the training data as one big cluster pertaining to true class 1. Because of this, K-Means is able to predict 77.88% of true class 1 test samples as class 1. The team attempts to resolve this issue in our CNN approach, but fails to accommodate the imbalance with all other classifiers. This would account for low accuracy across the board since the data restricts how much the models can learn from the less represented classes (0 and 2). As a result, models may be biased toward predicting class 1 for most samples and have a difficult time predicting class 0 and especially 2 as reflected in Figure 2. This hinders the use of K-Means, as it is not able to really identify a cluster for class 2, the highest severity class, and predicts only one sample as class 2.

B. Gaussian Mixture Models

After various experiments running K-Means with little change in accuracy/loss, the team hoped that using GMMs could generate better results. The initial classification attempt utilized flattened images of resolution (32x32) which was the highest resolution we were able to train with all samples. As a result, the non-encoded approach achieved a training accuracy of 33.77% and 44.71% respectively.

As done with KMeans, the team made use of a built in sklearn class named GaussianMixture with our encoded 50-feature samples and with regular image samples. Unfortunately, the higher compute required for GMMs restricted the image size we could provide to the model to 32x32 when not utilizing the trained Auto Encoder.

When utilizing our encoded samples, the GMM approach was able to achieve a training and testing accuracy of 32.9% and 50.6% respectively. The GMM classification has an accuracy that reflects guessing for training but does a much better job with classifying the testing set. Figure 3 displays the confusion matrix for the testing set using the 'tied' covariance type and 'kmeans' as the initialized centroids. These results are superior than the training and testing accuracy of 33.77% and 44.71% respectively achieved with the flattened images. (224x224).

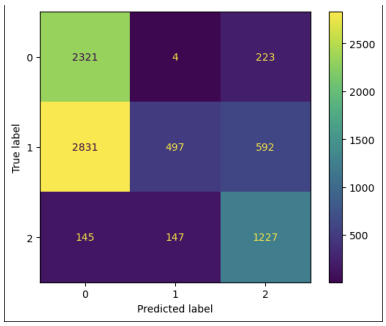

Fig. 3. Confusion Matrix for GMM test set with 50.6% classification accuracy.

After analyzing the results, the team identified that the classifier does a great job at classifying samples in class 0 when their true label is 0, but suffers nearly all of its miss-classifications predicting samples with true labels 1 as 0. The model is able to correctly classify samples with true labels 0, 1, and 2 with accuracies 91.09%, 12.67%, and 80.77% respectively. The team believes that there is such a big discrepancy in accuracies due to the inbalanced distribution of data points in each class and the similarity in feature values. Essentially, our GMM model is unable to make a clear distinction between samples in true class 0 and 1 which would translate to a GMM cluster (predicted class 0) that accounts for true labels 0 and 1. In addition, the model likely suffers from the different perspectives of each image when making a decision on classification. Finally, another issue could be the amount of images provided along with the quality of images. After analyzing the dataset, the team noticed that there was not a lot of images provided. This means that the model would have to learn extremely quickly. If the images were of lower quality, the model learning process would be even further delayed.

C. K-Nearest Neighbors

The final approach that uses encoded samples is the K-NN classifier which takes a look at the K nearest neighbors and performs a majority vote with those samples. The team believed we could achieve similar accuracies to the clustering classifiers with this much simpler approach with the encoded values as 50-features is much easier to separate than a flattened image. K-NN performed best with a k value of 4 and a testing accuracy of 44.71%. K-NN was also applied to the flattened images as samples and achieved a testing accuracy of 38.4%.

TABLE I
TESTING ACCURACIES FOR K-NN WITH DIFFERENT K VALUE

K-Value	Testing Accuracy
k = 3	43.86%
k = 4	44.71%
k = 5	44.42 %
k = 10	43.82%
k = 50	41.75%
k = 100	41.19

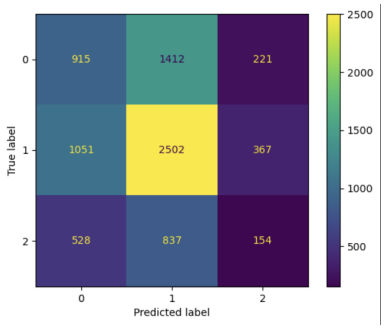

Fig. 4. Confusion Matrix for K-NN (K=4) test set with 44.71% classification accuracy.

D. Convolutional Neural Networks

Instead of incorporating an Auto Encoder to reduce our feature-space, the last approach was to make use of CNNs. The initial thought was to pick a pre-existing CNN architecture available in sklearn.models and train it on our dataset. The team evaluated various options, but decided to test the ResNet architecture with the Adam optimizer and use of Focal Loss ($\gamma = 5$). The goal of using Focal loss was to dynamically scale the cross entropy loss to down-weight the contribution of easy to classify examples and focus on the samples that are hard to classify. The team believed this would lead to some increase in accuracy due to the class-imbalance of the training sample (about half of the samples being class 1. Ultimately, ResNet is able to achieve an training and testing accuracy of 75.38% and 44.94% respectively. This gap in accuracies implied the potential for overfitting, so the team decided to implement a custom CNN as shown in Section II.

The custom CNN, which the team calls O-Net, takes in tensors of size (1,224,224) and produces tensors of size (1,3) that with the probabilities of a sample belonging to each class. O-Net is trained in batches of size 32 over 20 epochs that resulted in 51.06% testing accuracy as our best result over various experiments. Each experiment took a look at loss functions, gamma values for focal loss, epoch numbers, learning rates, optimizers, dropout, and different CNN architectures (number of layers, dropout, etc). Table 1 displays the accuracies of our 3 best models

TABLE II
TRANING AND VALIDATION ACCURACIES FOR O-NET WITH VARIOUS PARAMETERS/ARCHITECTURES

Parameters	Training Accuracy	Validation Accuracy
Model 1	48.22%	77.71%
Model 2	46.78%	81.54%
Model 3	51.06%	95.90 %

Model 1: 23 Layers, .7 dropout before every fully connected layer, Adam Optimizer, Learning rate of 1e-4, and Cross Entropy Loss
Model 2: 23 Layers, .5 dropout before every fully connected layer, Adam Optimizer, Learning rate of 1e-4, and Cross Entropy Loss
Model 3: 21 Layers, .2 dropout for last 2 fully connected layers, Adam Optimizer, Learning rate = 1e-4, $\gamma = 5$, and Focal loss to emphasize hard to classify samples in update.

Figure 5 displays the loss and accuracy for the training and testing data set during the training process for Model 3, the best-performing model. As expected, the training loss generally decreases as the training progresses but we see unstable accuracy results for the training set with a large discrepancy between training and testing accuracy/loss. This is likely a by-product of over-fitting but is a better result than some of our other models that experienced greater discrepancies and worse accuracies. Those results were the motive of introducing and increasing the dropout in the classifier portion of our model.

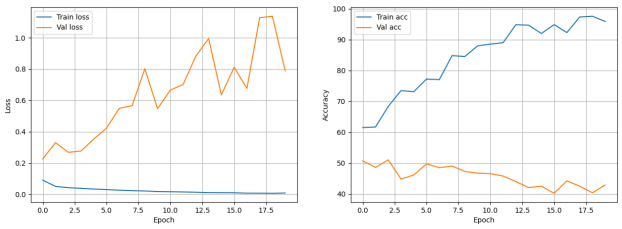

Fig. 5. Loss and Accuracy on Training and Validation datasets for Model 3

Furthermore, Figure 6 is the confusion matrix for the testing samples. As we can see, our CNN performs best at predicting class 0 and achieves the highest accuracy of all models.

IV. CONCLUSION

After the team completed all experiments across all classifiers, the best result was from CNNs with a testing accuracy

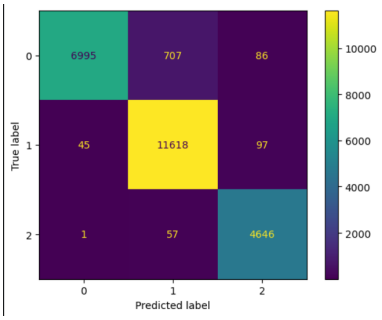

Fig. 6. Confusion Matrix for testing dataset produced by Model 3 predictions

of 51.06%. In terms of percentages, this is still not an ideal result and it likely has to due with the nature of the dataset. The images provided for each patient have a great class imbalance and most of the images have little correlation between the image and the class. In other words, many images per patient have little to no medical insight to the state of the eye-health in terms of the illness in question. Therefore, these samples could have an impact on our accuracy in terms of noise.

Regardless, the team made the decision to incorporate an Auto Encoder which resulted in better accuracies for each classifier that made use of it compared to the raw samples. Furthermore, our GMM model displayed very high testing accuracies for class 0 and 2 which is of great use for those classes and could greatly help filtering out samples in a medical environment. Specifically, it is important that we are able to predict class 2 with accuracy 60.1% which is the most severe of all the classes and poses the greatest risk. Figure 3 emphasizes that nearly all true label class 2 and 0 testing samples are predicted as class 2. Similarly, Figure 2 displays that nearly all true label class 1 samples are predicted as class 1. In other words GMM performs best for class 0 and 2 and K-Means performs best for class 1 out of all classifiers. Each classifier has it's own tradeoff, but perhaps the most useful utilization of these classifiers for real-world use is the combination of GMMs and K-Means.

V. FUTURE WORK

Despite our results, there is much room for improvement in areas such as feature reduction when using non-CNN classifiers. Exploring Convolutional Auto Encoders, PCA, or other techniques could be useful. This is critical to ensure feasibility while still capturing the most information possible. Another area of improvement is the data-set itself. Creating a more class-balanced data set would greatly help models have better information to learn from. Finally, testing different architectures for CNN could yield better results with an improved data set.