Applied Machines Learning to the OLIVES Dataset for Diabetic Retinopathy Severity Scale Classification

Coleman Hoff
Aerospace Engineering Department
Georgia Institute of Technology
Atlanta, GA
choff7@gatech.edu

I. Introduction

Machine learning has become more common place in the medical field as models have improved over time. Image segmentation of various scans and imaging methods have been shown to assist with many medical conditions of the brain and body. Clinical diagnosis of certain eye conditions is handled by a medical professional using various biometric data along with certain eye scans. In this project four machine learning methods will be applied to volumetric images of the eye to assist in the classification of Diabetic Retinopathy severity.

II. The OLIVES Dataset

The Ophthalmic Labels for Investigating Visual Eye Semantics (OLIVES) dataset is a set of data derived from the PRIME and TREX DME clinical studies. These were two randomized clinical trials which had been run between December 2013 and April 2021 and evaluated the outcome of Diabetic Retinopathy (DR) and Diabetic Macular Edema (DME) in a set of 96 unique patients. Each visit the patient would receive an optical coherence tomography (OCT) of the eye. These OCT images are a major portion of the OLIVES dataset and are the primary focus of the analysis presented.

Each OCT scan consists of roughly 49 504 x 496-pixel images of the patient's eye. Over the course of the two studies there were a total of 698 OCT scans for a grand total of roughly 34000 images. Each image can be viewed individually or together as a volumetric scan. Each of these images is also accompanied by a Diabetic Retinopathy Severity Score (DRSS). This is a system used to grade the severity of DR.

The OLIVES dataset is incredibly imbalanced and is also a small dataset. Out of the three classes 0 makes up ~32%, 1 makes up ~48%, and finally 2 makes up ~20%. If one is not careful the model will be overfit to the 1 class and the resulting model will not make good predictions. This is dealt with in a variety of ways within the models. Additionally, the total number of volumetric scans is 698. This is then broken up into a training dataset of 495 scans and a test dataset of 203 scans. That training dataset is then further broken up into the final training dataset of 396 images and 99 validation images. This presents a difficult problem in that the classes are incredibly imbalanced, and the total amount of data is low.

To address the class imbalance and low amount of data several methods were used. The first is that the two lower classes, 0 and 1, are oversampled during training at a rate which more balances the three classes. At that point the oversampling just gets the same images contained within each dataset, so they are then augmented to create variety in each of the classes. The augmentation on the three-dimensional data is made possible by the module TorchIO, this is an extension PyTorch library whose purpose is to work with medical imaging. This makes it ideal for this dataset. With TorchIO the volumetric scans can have random affine transformations and elastic deformations applied to the images. This results in rotated and flipped versions of previous data that is now "new" to the model. This serves as a regularizing method and helps deal with the imbalanced dataset.

III. Methodologies

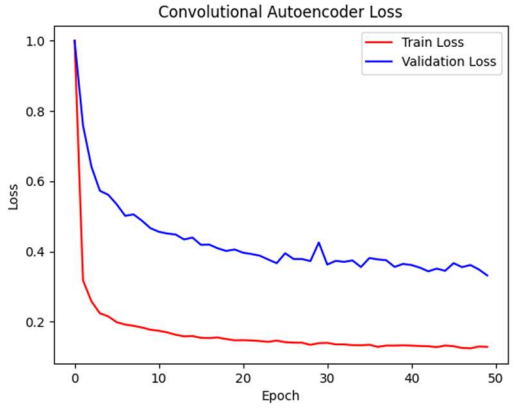
Four training architectures were used to evaluate the classification problem of the OLVES dataset. Their formulations and training results are presented.

A. Convolutional Autoencoder

The three-dimensional OCT scans once ran through the PyTorch transforms are 48 x 224 x 224 tensors. In the case of many machine learning techniques this three-dimensional image would be flattened into a 2458624 long feature vector for each image within the dataset. This poses a computational challenge to fit into memory.

One of the simplest forms of dimensionality reduction is an autoencoder, or in this case a convolutional autoencoder. A CAE initially applies an encoder to the original image. An encoder applies convolutional operations to the original image several times until it reaches what is called the latent space. The latent space is the last layer of the encoder, and it represents a compressed, lower-dimensional representation of the initial input images. There are many ways to take advantage of this latent space, but for the purpose of this project it is a much-compressed version of the initial image and is more usable within many deep learning and traditional machine learning applications with limited disk space. The second part of the CAE is the decoder. This applies deconvolution operations on the latent space until it reaches the original image space. Then a loss function is applied to the reconstructed image versus the original image, typically mean square error (MSE), and the error is then backpropagation across the network.

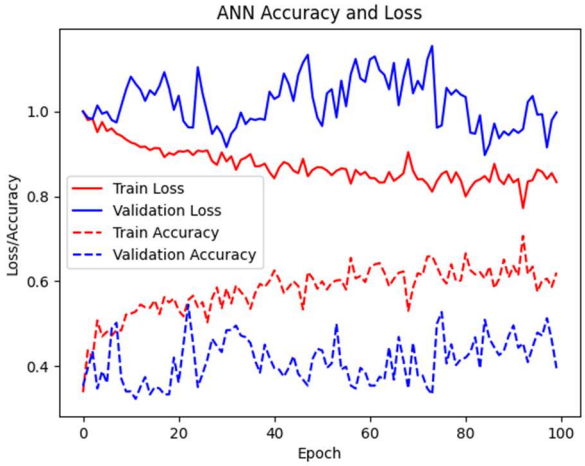
This CAE consists of four 3-D convolutional layers along with three 3-D max pooling layers. It additionally uses LeakyReLU as its activation function. The final latent space image is a 4 channel 6x28x28 image which then is flattened into an 18816-element latent space vector to be used as a lower dimensional representation of the original 2458624 length feature vector. It is less than 1% of the original size. The training of the CAE is provided below.



B. Artificial Neural Network with Convolutional Autoencoder

Artificial Neural Networks (ANN), sometimes referred to as multilayer perceptron (MLP), were one of the first forms of deep learning introduced. It is comprised of input, output, and hidden layers which themselves are comprised of individual perceptrons. Its strength is its ability to model nonlinear relationships between variables and desired outputs given an activation function within the model. Each perceptron has its own weight and bias assigned to it, so as one can imagine the more complex an ANN gets the more parameters are needed and thus the more memory is taken up.

That is why in this form of the ANN the previously trained CAE is used to generate latent space vectors of the three dimensional images to be used as the inputs to the ANN. This ANN is comprised of a single input layer of 18816 neurons, three hidden layers of 1000 neurons each, and a final output layer of 3 neurons and a sigmoid function to predict the class of the image. Between each of these layers the ReLU activation function is used. The model training over 100 epochs with the Adam optimizer with a learning rate of 1e-3, weight decay of 1e-6, and batch size of 3 is presented.

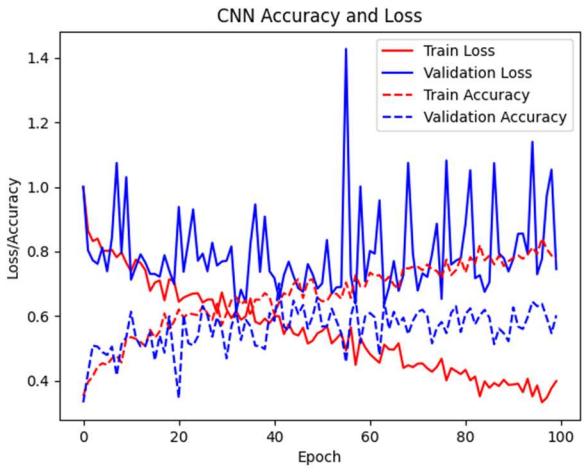


C. *Convolutional Neural Network*

A convolutional Neural Network (CNN) is a form of ANN that is specialized in processing gridded data such as images or tensors, using convolutional operations along with various other layers such as a pooling layer. CNNs are desirable to use for this kind of data because of their spatial relationships and because CNNs used shared weights thus lowering the total number of parameters needed for the entire model. That means in this case there is no need to use the previously constructed CAE.

For this problem a three-dimensional formulation of ResNet10 was used to perform the classification problem. ResNet10 is a residual neural network, a form of CNN, that uses skip connections which allows data to bypass certain blocks of the neural network. The benefit to this is that the network learns residual information by comparing the data that skipped to the output of the same or next layer which then allows for more learning. With this ResNet can capture both high- and low-level information in provided data very well, something that is ideal for a problem like this. ResNet 10 is the 10-layer formulation of ResNet. This one was selected because of how small the training dataset is. The deeper models will be prone to overfitting on the limited training data.

ResNet10 is trained using the NAdam optimizer with a learning rate of 1e-3, weight decay of 1e-6, and a batch size of 3. The batches must be small to load the large volumetric data. This isn't a problem; however, it will result in longer training time and a noisy loss plot. The training over 100 epochs is presented:



D. *Nu-Support Vector Machine with Convolutional Autoencoder*

A Nu-Support Vector Machine is a soft-margin formulation of the support vector machine. It replaces the traditional C hyper parameter with the new Nu. This Nu varies from 0 to 1 and serves as a lower bound on the fraction of support vectors and the upper bound on the fraction of margin errors.

SVMs are typically trained differently when compared to the previously presented deep learning models. Rather than gradient descent it can minimize the cost function with a direct solution assuming computer power allows for it. In this case the model is computed over a large variety of hyperparameter configurations and then 5-fold cross validation is performed on the model. Finally, the best performing model is chosen to be the final one. The hyperparameters tuned for the Nu-SVM are presented below as well as the 5-fold cross validated final values.

Nu-SVM Hyperparameters		
Parameter	**Search Space**	**Final**
Nu	0.1 to 9.0	0.33
Kernel	linear, rbf	rbf
Gamma	auto, scale	scale
Class Weight	balanced	balanced

As discussed earlier. The main hyperparameter of the Nu-SVM is Nu itself. This controls how many points may lie on the support vectors. Next is the kernel used by the SVM. In this case it is either a linear kernel or a radial basis function (rbf) kernel. The rbf kernel allows the SVM to model non-linear relationships present within the data. These non-linear relationships may prove to be key to classifying volumetric images. The gamma hyperparameter is the kernel coefficient for rbf kernel. There auto and scale methods are two methods for selecting this value for the rbf.

E. *K-Nearest Neighbors with Convolutional Autoencoder*

K-Nearest neighbors' classifier is a supervised machine learning algorithm for various classification tasks. It classifies data points based off the chosen distance metric and k-nearest neighbors of a new data point.

There are many hyperparameters available to tune for the algorithm. Some of them are the number of neighbors used to classify, the weights of the samples, the distance metric, and the underlying algorithm. A table of the

various k nearest neighbors hyperparameters along with the 5-fold cross validated final parameters are given:

K-Nearest Neighbor Hyperparameters		
Parameter	Search Space	Final
n neighbors	3 to 30	15
weights	distance	distance
p	1, 2, 3	2
algorithm	auto'	auto
leaf size	10 to 50	18

IV. Results

Model Total Metrics				
Model	Balaced Accuracy	Total Recall	Total Precision	F1 Score
ANN	0.33	0.44	0.28	0.335
CNN	0.32	0.51	0.36	0.377
Nu-SVM	0.43	0.51	0.503	0.506476
K-Nearest	0.35	0.43	0.4	0.414458

ANN Class Based Precision and Recall		
Classes	Precision	Recall
Class 0	0	0
Class 1	0.47	0.85
Class 2	0.28	0.16

CNN Class Based Precision and Recall		
Classes	Precision	Recall
Class 0	0.21	0.15
Class 1	0.53	0.71
Class 2	0.19	0.1

Nu-SVM Class Based Precision and Recall		
Classes	Precision	Recall
Class 0	0.42	0.6
Class 1	0.62	0.62
Class 2	0.33	0.11

KNN Class Based Precision and Recall		
Classes	Precision	Recall
Class 0	0.35	0.33
Class 1	0.53	0.64
Class 2	0.18	0.1

V. Discussion

The results presented above paint a difficult picture for the models. Because of the small dataset and highly imbalanced class distribution the models all struggle on at least one of the classes. They typically have higher precision, recall, and class accuracy for class 1. It then follows that typically class 0 is better than class 2. This exactly follows the class distribution present in the original dataset and shows that more still needs to be done to alleviate this issue. In the worst case the model solely predicts class 1 giving the appearance of good accuracy for that class, however as a whole model is not good. It shows how difficult a problem it is since the statistics for training and validation data were good across each of the models. Some reaching 80% balanced accuracy.

While the models do have issues, there is use in them. For example, the Nu-SVM has a precision and recall greater than 0.5 for class 0 and for class 1. This shows that the model could be used for discriminating between the two. Additionally the F-Score of the model is above 0.5 indicating that the model is usable and has some value.

VI. Conclusion

The OLIVES dataset is a challenging classification problem. It presents an imbalanced class distribution and a very low number of samples to train on. This was attempted to be dealt with through oversampling of the smaller imbalanced classes and through image augmentation using TorchIO.

The resulting accuracies, precision, and recall indicate that this is a difficult problem to solve, however the models still prove to have some use. Each one can make worthwhile predictions on at least class 1 and class 0. Class 2 has proven to be a challenge because of the low amount of data. With more training time, fine tuning of the models, and compute power it is probable that the problems presented here can be alleviated to the point that good model results.

Appendix

A. Github Link

https://github.com/choff24/OLIVES_Classification_ECE8803

B. Presentation Link

https://github.com/choff24/OLIVES_Classification_ECE8803/blob/main/ColemanHoff_ECE8803_FinalProject_Slides_Audio.pptx

C. Paper Link

https://github.com/choff24/OLIVES_Classification_ECE8803/blob/main/ColemanHoff_ECE8803_FinalProject_Paper.pdf

References

Géron Aurélien. *Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, 2020.

Hastie, Trevor, et al. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2017.

"Learn: Machine Learning in Python - Scikit-Learn 0.16.1 Documentation." *Scikit*, https://scikit-learn.org/.

Prabhushankar, Mohit, et al. "Olives Dataset: Ophthalmic Labels for Investigating Visual Eye Semantics." *ArXiv.org*, 22 Sept. 2022, https://arxiv.org/abs/2209.11195.

"Pytorch." *PyTorch*, https://pytorch.org/.

Xmuyzz. "Xmuyzz/3D-CNN-PyTorch: Pytorch Implementation for 3D CNN Models for Medical Image Data (1 Channel Gray Scale Images)." *GitHub*, https://github.com/xmuyzz/3D-CNN-PyTorch.