CSC 7210: Project 2

Katherine E. Brown

Introduction

Diabetic retinopathy is one of the leading causes of blindness in the world (Fong et al. 2004; Leibig and others 2017); however, it can be treated if it is detected early. Detection of diabetic retinopathy requires visualization of the inside of the eye (Fong et al. 2004). This can be done through a manual exam where the eye is dilated or through imaging of the eye.

There has been significant work in using deep learning to detect diabetic retinopathy from retinal scans (Mateen et al. 2020; Asiri et al. 2019; Alyoubi, Shalash, and Abulkhair 2020; Leibig and others 2017). Techniques include deep convolutional neural networks, uncertainty quantification, and deep autoencoders.

Medical diagnostics provide an excellent testbed for anomaly detection techniques. In many datasets, the presence of a disease is rare, or abnormal, in comparison to "healthy" entities. My hypothesis is that anomaly detection approaches may prove beneficial in detecting diseases in heavily unbalanced datasets.

Data

The dataset used in this project is a collection of retinal scans released by EyePACS (Cuadros and Bresnick 2009) and distributed through Kaggle (Kaggle 2015). Images are resized to 100×100 pixels^2 to reduce processing time.

The dataset is further processed by altering the coloration. These retinal scans are subject to variations in colors, ranging from a yellow-orange to a deeper orange. The images were then processed using a method successfully used by (Leibig and others 2017) and (Graham 2015). The "average color" is mapped to gray with other colors shifted accordingly. The images are finally normalized such that their pixel values range between 0 and 1.

Each image is given a label between 0 and 4 (Kaggle 2015). These labels rank diabetic retinopathy from non-existent to proliferative.

Algorithms

I utilized two algorithms in this project. First, I used a deep convolutinal autoencoder. I compare this method to the DenseNet-121 neural network (Huang et al. 2017).

Autoencoders

An autoencoder is a neural network designed to compress and decompress in order to reconstruct its input (Hinton and Zemel 1994; Baldi 2012). Consider a one-dimensional example. Let n be the dimension of our input. Then, $f : \mathbb{R}^n \to \mathbb{R}^l$ encodes the original vector of length n to a vector of length l. The decoder function $g : \mathbb{R}^l \to \mathbb{R}^n$ expands this compression back to its original size. These functions are implemented as feed forward neural networks. If $x \in \mathbb{R}^n$ is input, then the output x' is given by $x' = g(f(x))$.

The process is very similar for a convolutional autoencoder for two-dimensional images (Turchenko, Chalmers, and Luczak 2017). Images are represented as tensors. Let x be an $m \times n$ image that has red, green, and blue components. Then, $x = \{R_x, G_x, B_x\}$ where R_x is an $m \times n$ matrix of the values of the red pixels in x, G_x is an $m \times n$ matrix of the values of the green pixels in x, and B_x is an $m \times n$ matrix of the values of the blue pixels in x.

The encoder function uses a combination of the convolution and pooling operators to reduce the dimension either to a compressed tensor or to a vector of real numbers. The convolution operator combines neighboring pixels together based on some kernel, and ooling layers reduce the size of its input, based on input values (O'Shea and Nash 2015). The decoder function reverses the convolution and pooling operations, allowing the image to return to its original dimension (Turchenko, Chalmers, and Luczak 2017).

DenseNet

Convolutional neural networks consist of two distinct phases: feature extraction and classification. The feature extraction stage uses convolution layers to extract the visual structures in the image. These convolution layers are supplemented by dropout layers, pooling layers, and batch normalization layers (O'Shea and Nash 2015; Srivastava and others 2014; Ioffe and Szegedy 2015). Dropout layers prevent overfitting by randomly zeroing weights between layers

Size	Activation
1024	ReLU
512	ReLU
256	ReLU
2	Softmax

Table 1: Architecture of the Classification Stage of DenseNet model used

(Srivastava and others 2014). Batch normalization speeds up training and enables more accurate neural networks by normalizing inputs into each layer (Ioffe and Szegedy 2015).

DenseNet (Huang et al. 2017) is a series of convolutional neural network architectures that use interconnected "blocks" of convolutional layers to extract more features and improve performace. Typical convolutional neural networks feed one layer into the next, in a sequential fashion. DenseNet is composed of blocks where each layer is connected to *each* of the subsequent layers (Huang et al. 2017). This more complex architecture allows features extracted at early levels of processing to be utilized by the layers closer to the fully-connected decision network.

Experiments

In this section, experimental methodology is presented. All deep learning is implemented using PyTorch (Paszke et al. 2019).

DenseNet

First, the data was split into disjoint training, test, and validation sets. The training set was composed of 80% of the images labeled as normal or proliferative diabetic retinopathy. The testing and validation sets each were 10% of these images.

The training data was then used to fine-tune a DenseNet 121 model with a custom classification stage (Huang et al. 2017). Many image recognition tasks do not require random initialization of the weights but transfer learning can be used to accelerate training (Shin et al. 2016). The DenseNet feature extractor was pretraiend using the ImageNet dataset. The custom classification stage is given in Table 1 and trained from randomly intialized weights. This neural network trained for 15 epochs.

Autoencoders

First, the data was split into training, testing, and validation sets. The training set contained only retinal scans labeled as having no diabetic retinopathy. The testing and validation sets contained a mixture of unseen normal and images labeled as having proliferative diabetic retinopathy. Table 2 lists the architecture for the convolutional autoencoder. Note that the activation of the final layer of the autoencoder is the Sigmoid function since the images are normalized to have values between 0 and 1. The autoencoder is trained for 50 epochs.

Results

In this section, I present the results of the experiments.

Layer	Input Channels	Output Channels	Activation
Input	3	3	N/A
Convolution	3	512	ReLU
Max Pool	512	512	N/A
Convolution	512	4	ReLU
Max Pool	4	4	N/A
Convolution	4	512	ReLU
Max Pool	512	512	N/A
Convolution	512	4	Sigmoid

Table 2: Architecture of the Convolutional Autoencoder

DenseNet

Table 3 lists the results of the DenseNet-based convolutional neural network. This network performs well in this binary classification of diabetic retinopathy.

Metric	Performace
Accuracy	99.095%
Precision	94.231%
Recall	70.000%
F1-Score	80.328%
AUC	97.318%

Table 3: Average Reconstruction Error of Autoencoder on Validation Set

Autoencoder

Table 4 details the reconstruction error of the autoencoder on the normal scans and the proliferative scans from the validation set. Figure 1 shows the frequency of the errors for both types of images.

Diagnosis	Reconstruction Error
Normal	48.593
Proliferative	40.858

Table 4: Average Reconstruction Error of Autoencoder on Validation Set

Discussion

In this section, I discuss the results.

Why Did Autoencoders Fail?

In Table 3, we see the DenseNet architecture was able to successfully detect proliferative diabetic retinopathy; however, the autoencoder did not have such success. The reconstruction error on both the normal and proliferative scans range between 0 and 150 units. Figure 1 indicates significant overlap among the error for both classes of images. Thus, the "anomalous" images do not have a higher reocnstruction error, making them impossible to detect through the reconstruction error of the autoencoder.

The primary reason the autoencoders fail is a result of the structure of the data. Each retinopathy scan generally has the same structure. The raw images consist of a orange circle on a black background. The discriminating features are too small compared to all the similarities between both classes

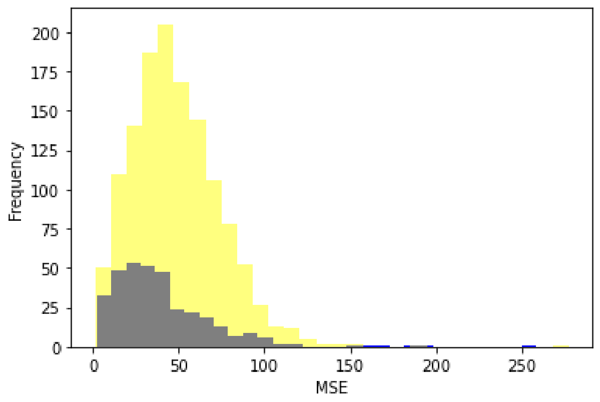

Figure 1: Plot of Mean Squared Error of the images recreated by the autoencoder. Yellow indicates error of normal images. Blue indicates error of proliferative diabetic retinopathy images

of images. Moreover, during debugging, I found that convolutional autoencoders are successful when the anomalous class and normal class have different shapes. The DenseNet architecture was able to extract these minute features and use them successfully in classification.

Further, DenseNet had a second advantage in that it trained on both normal retinal scans and retinal scans with proliferative diabetic retinopathy. This is similar to the experience of (Sultani, Chen, and Shah 2018). They found the video-based autoencoders produced high reconstruction error for both normal and anomalous videos, and their technique succeeded due to having data from both classes (Sultani, Chen, and Shah 2018).

What Did I Learn?

Throughout this project, I learned much about developing autoencoders and anomaly detection. First, this project gave me considerable experience developing and debugging autoencoders. In determining if autoencoders would be an effective method for detecting proliferative diabetic retinopathy, I utilized several different architecures and attempted to utilize intermediate features from a convolutional neural network in a non-convolutional autoencoder. Moreover, I gained experience using an anomaly detection workflow of modeling normal behavior and using that model to detect anomalies.

Conclusion

This work used anomaly detection techniques to possibly detect proliferative diabetic retinopathy from retinal scans. Unfortunately, supervised methods performed better. Anomaly detection and other supervised techniques may still prove useful in detecting diseases; however, the success of these techniques are can be dependent on data structure and modality.

References

Alyoubi, W. L.; Shalash, W. M.; and Abulkhair, M. F. 2020. Diabetic retinopathy detection through deep learning techniques: A review. *Informatics in Medicine Unlocked* 100377.

Asiri, N.; Hussain, M.; Al Adel, F.; and Alzaidi, N. 2019. Deep learning based computer-aided diagnosis systems for diabetic retinopathy: A survey. *Artificial intelligence in medicine* 99:101701.

Baldi, P. 2012. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, 37–49.

Cuadros, J., and Bresnick, G. 2009. Eyepacs: an adaptable telemedicine system for diabetic retinopathy screening. *Journal of diabetes science and technology* 3(3):509–516.

Fong, D. S.; Aiello, L.; Gardner, T. W.; et al. 2004. Retinopathy in diabetes. *Diabetes care* 27(suppl 1):s84–s87.

Graham, B. 2015. Kaggle diabetic retinopathy detection competition report. *University of Warwick*.

Hinton, G. E., and Zemel, R. S. 1994. Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, 3–10.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR* abs/1502.03167.

Kaggle. 2015. Diabetic retinopathy detection: Identify signs of diabetic retinopathy in eye images. `https://www.kaggle.com/c/diabetic-retinopathy-detection/`.

Leibig, C., et al. 2017. Leveraging uncertainty information from deep neural networks for disease detection. *Scientific reports* 7(1):17816.

Mateen, M.; Wen, J.; Hassan, M.; Nasrullah, N.; Sun, S.; and Hayat, S. 2020. Automatic detection of diabetic retinopathy: A review on datasets, methods and evaluation metrics. *IEEE Access* 8:48784–48811.

O'Shea, K., and Nash, R. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Paszke, A.; Gross, S.; Massa, F.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc. 8024–8035.

Shin, H.-C.; Roth, H. R.; Gao, M.; Lu, L.; Xu, Z.; Nogues, I.; Yao, J.; Mollura, D.; and Summers, R. M. 2016. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging* 35(5):1285–1298.

Srivastava, N., et al. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Sultani, W.; Chen, C.; and Shah, M. 2018. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6479–6488.

Turchenko, V.; Chalmers, E.; and Luczak, A. 2017. A deep convolutional auto-encoder with pooling - unpooling layers in caffe. *CoRR* abs/1701.04949.