Jameson Schryer

MSDS 686

12/08/2022

**Alzheimer’s MRI Classification**

The purpose of this deep learning project is to work with a dataset of Alzheimer’s effected brain scans to train a custom Convolutional Neural Network (CNN) to predict the severity of the illness. This project encompasses many techniques to increase accuracy: Max Pooling, Batch Normalizaton, Dropout, and more. This project also aims to make use of plots to visually display the image data and accuracy of the model’s predictions.

The dataset is made up of 6400 images. These are greyscale images of brains ranging from the following classificaitons: MildDemented, ModerateDemented, NonDemented, VeryMildDemented. These images are at 176x208 resolution. This dataset is pre-split into training and test data. The graphs below depict the distribution of classes in both the training and testing data:

Chart, bar chart

Description automatically generatedChart, bar chart

Description automatically generated

This project was carried out on Google Colab, a web based Python IDE. This interface is set up with all the necessary packages for deep learning. It also provides access to GPU, which allows for neural networks to run quickly. To begin this project, a new Colab notebook was created, and a host of python packages were imported. The packages imported were chosen for data manipulation, visualization, data flow, and for building the architecture of the CNN. Next, a personal Kaggle token was uploaded to allow for dataset to be pulled from the source. A new folder was then created, and the dataset was downloaded there. Next, a basic exploratory data analysis was carried out to visually display the counts of images in each class in both the training and testing data (as seen above).

After confirming that both datasets had similar distributions of classes, a new folder was created for the validation data. The validation dataset was manually created by extracting a random 20% of the training data and moving it to the new validation folder. Next, a normalization process was created through defining a Keras imagedatagenerator that rescaled the images to 1/255. Next, these imagedatagenerators were used on data flowing from a Keras flow\_from\_directory aimed at all three folders of data. Their target size was set to the image size, the batch size was set to 16 to remain a power of 2 to take advantage of the GPU, and the class\_mode was set to “categorical” as this was a multiclassification problem. The results of these generators were printed, showing that we had 4098 images in the train data, 1023 in the validation data, and 1279 in the test data.

Next, a loop statement was written to plot ten example images from the train generator, along with their labels. These images were shown to be similarly oriented greyscale images. Due to the uniformity of the images, it was determined that only light data augmentation should be carried out. The function of the CNN is to detect patterns of brain structure maladies in the images. If the images were to be stretched or modified to a large extent there could be risk for false positives or negatives. With this determination made, a new image generator was only made to rescale and flip horizontally. This new image generator was applied to the training and validation data. The Test data was not augmented.

Next, the CNN model was defined. This model was built in the form of a function, where a variable named “model” is set to models.Sequential(). For every subsequent line, layers are added to “model”. The first layer is a convolution layer, with 64 filters, a kernel size of 3x3, activation set to relu, and the input shape set to the size of the images. A max pooling layer follows this to create a down sampled feature map.

The next block in the model is made up of two back to back convolutional layers, a max pool layer, and a drop out layer. The two convolutional layers are made up of 64 filters, and activation set to relu. The max pool layer is set to default settings, and the dropout layer is set to 0.3. This block is repeated two more times.

The next block begins with a dense layer. This layer contains 2048 units, and has a relu activation. Next, a batch normalization layer is added for a standardization of the output of the dense layer. Next, a dropout layer set to 0.7 is added to randomly reduce the connectivity of nodes to prevent overfitting. Next, another dense layer with 128 units is added with a relu activation. Next, another batch normalization and dropout layer were added. The dropout layer was set to 0.5. These were set to further normalize the data and to reduce the risk of overfitting. The following block of code was simply a repeat of the convolutional layer x 2/ max pool layer/ drop out layer combination that was used previously in this model.

The next block begins with three dense layers with relu activation. The units were respectively set to 64, 128, 64. The output of these was transformed into a one dimensional vector via a flatten layer. This was fed into another dense layer set to 32 units and activation relu, which was followed by a dropout layer of 0.5. The output was fed into a final dense layer with four units, representing each possible classification. The activation for this final layer was softmax, as this was a multiclassification problem. Next the model was compiled, with optimizer set to adam, loss to CategoricalCrossentropy, and metrics to auc.

The model function was then set to the variable name “model”, and it was fit to the training data with 100 epochs, validation dataset was called, callbacks were set to early stopping with the monitor as auc, patience as 3, and the restoration of best weights set to true. This model returned an AUC score of .84, with a loss of .91.

AUC (Area Under Curve) refers to the space under a Receiver Operator Characteristic Curve. This metric essentially explains how well a model does in distinguishing between classes. It is a much more wholistic approach to evaluating a model rather than just pure accuracy alone. The AUC of .84 is not bad, but it could be better. The loss of 1.3 is indicative of a high volume of error, which could explain the less than perfect AUC.

There are a few factors that could have effected the model’s performance. Obviously, the architecture of the CNN could be modified to increase performance. This could consist of modifying the epochs, batch sizes, order of layers, adding layers, removing layers, etc. I think there is a larger issue with an unbalanced dataset. As I plotted at the beginning of the notebook, there is a severe imbalance between classes. The “Moderate Demented” class was severely lower than the “Non Demented” class. This seems to be a route cause of error. In the future, this could be remedied by targeted up sampling via augmentation.