Benjamin Hanim 5/05/2024

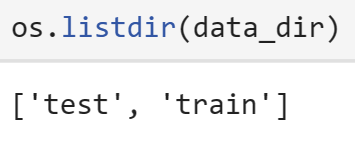
**Alzheimer's Convolutional Neural Network**

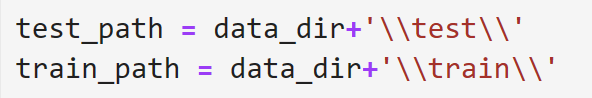
**Step 1: Set Up**

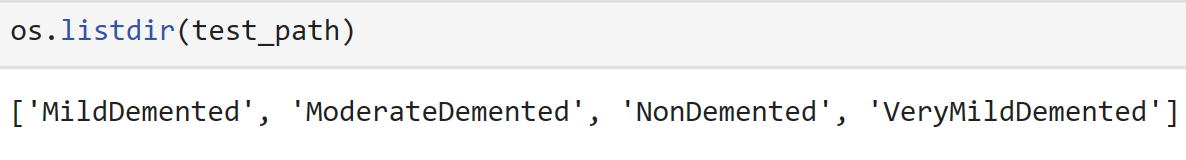
To begin my journey into convolutional neural networks, I began by finding an Alzheimer’s dataset on Kaggle ([click here](https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images/code)). I originally downloaded the zip file onto my computer and unzipped it directly into my files. I started on Jupyter Notebook, but after realizing the processing power necessary for running a multi-layered neural network, I switched over to Google Colab by uploading all of the files to Google Drive.

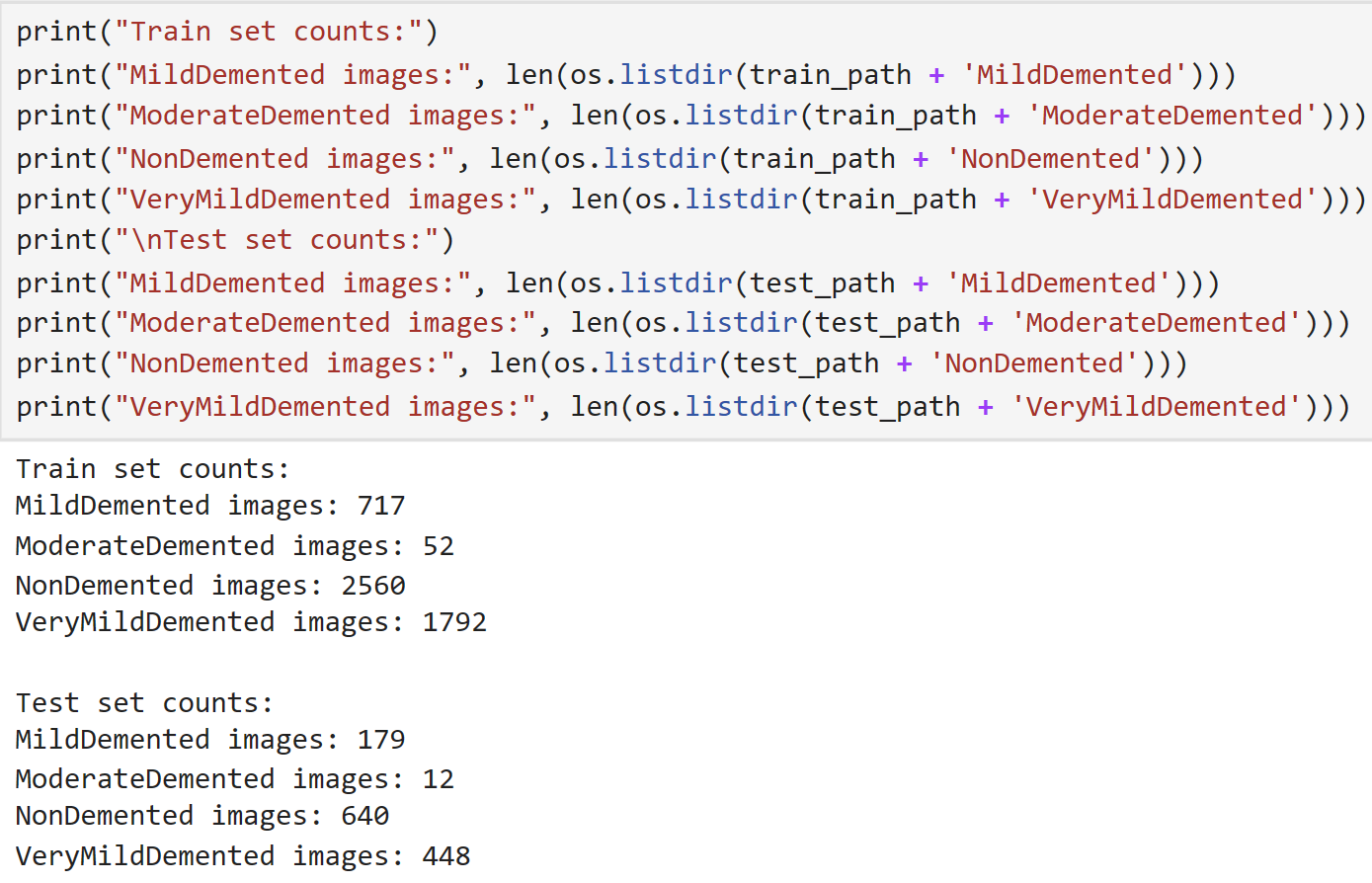
**Step 2: Data Analysis**

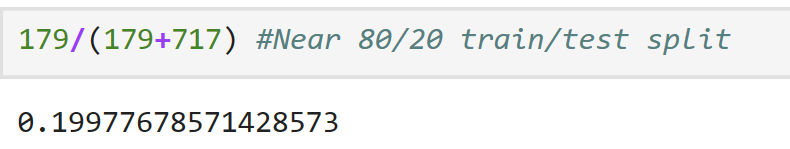
First, I needed to determine how many classes the dataset contained, how many images per class in the training and test sets, the shape of the images (to see if they needed to be reshaped), view the images themselves to see if I could notice any patterns of symmetry, and lastly see the images’ gradients to see if it had to be normalized.





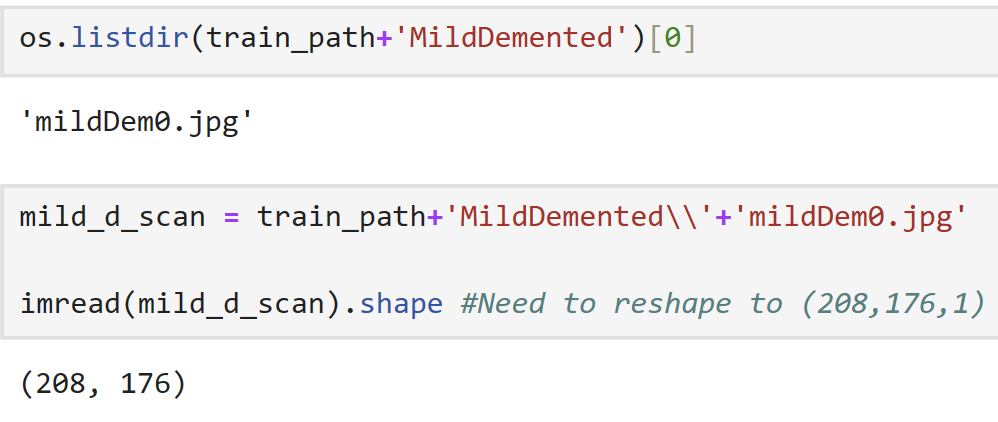






After determining the number of images in the train and test sets, I realized that moderate demented images and mild demented images would likely not be as accurate as the other images due to there being a significantly smaller sample size. I also determined that the model had a built-in train/test split of approximately 80/20. Additionally, based on the largest class, nondemented, I determined that the total amount of test images, 1279, divided by the number of nondemented test images, 640, is .5004. This means that for the model to have any real value, it must be more accurate than this value.

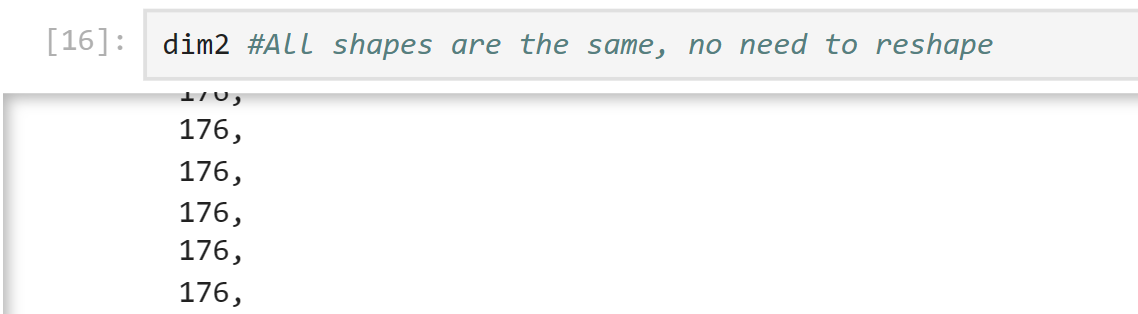
Next, I wanted to view the shapes of the images, I selected one image at random.



Since there was no specified color channel, I realized that I had to add a last element to the tuple to make the shape (208,176,1).

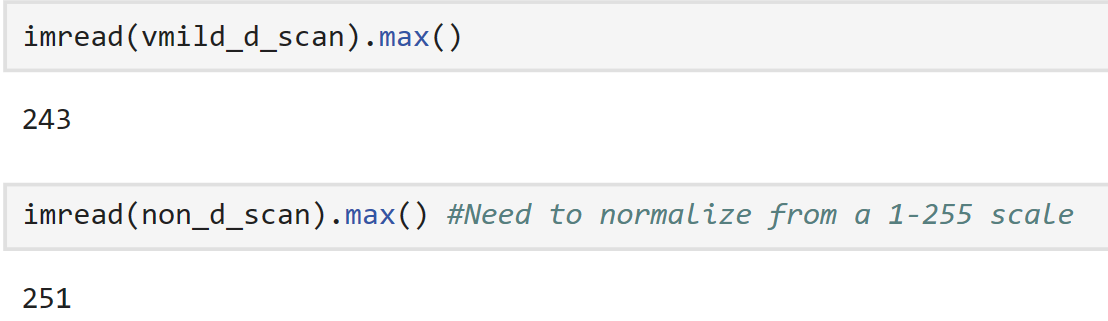
To ensure that all of the images were the same shape, I created a list of both the dimensions of the images in the whole list of non demented images.





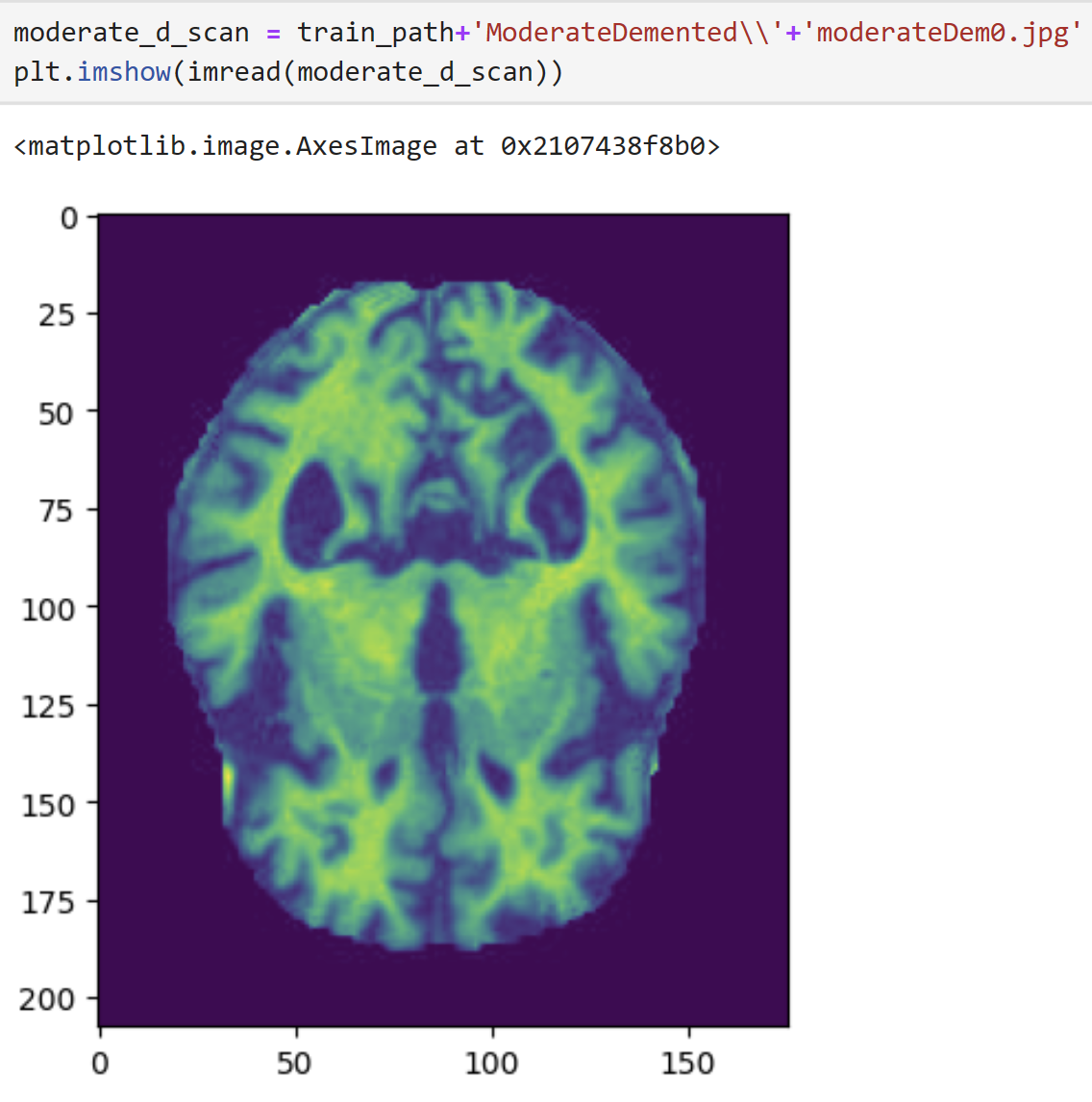
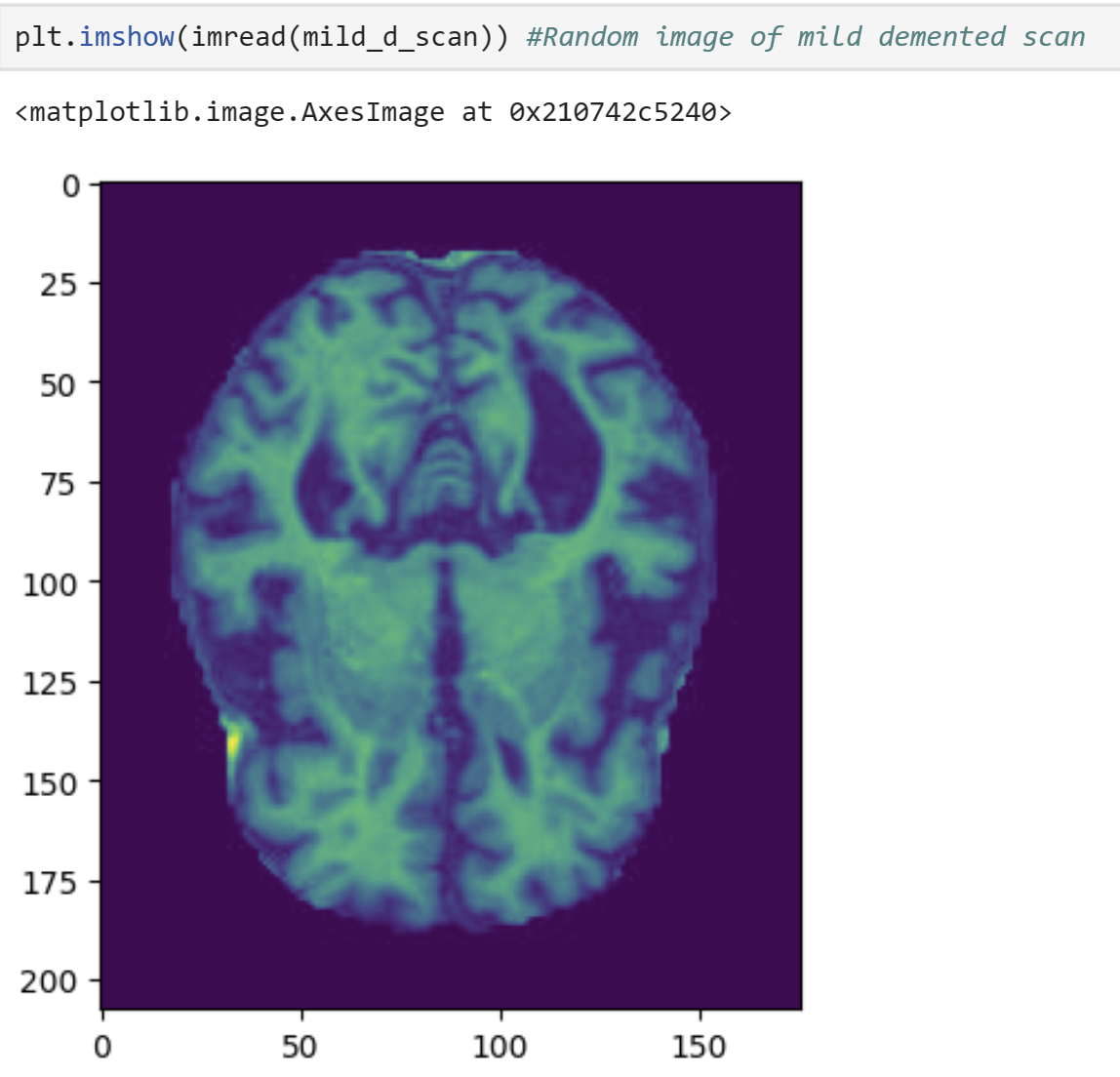
Since the entire list was 208,176, we know we can use the same operation to update the shape of every single image in the dataset.

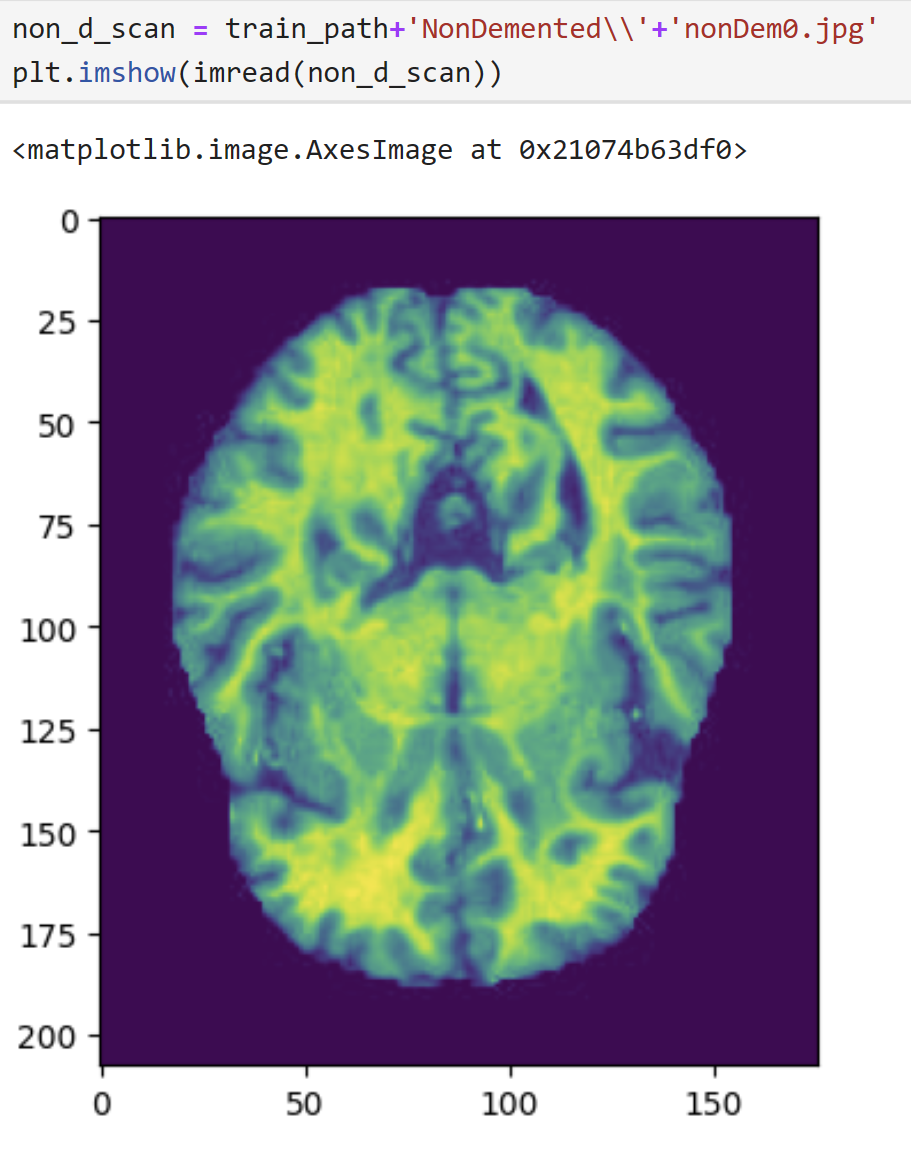
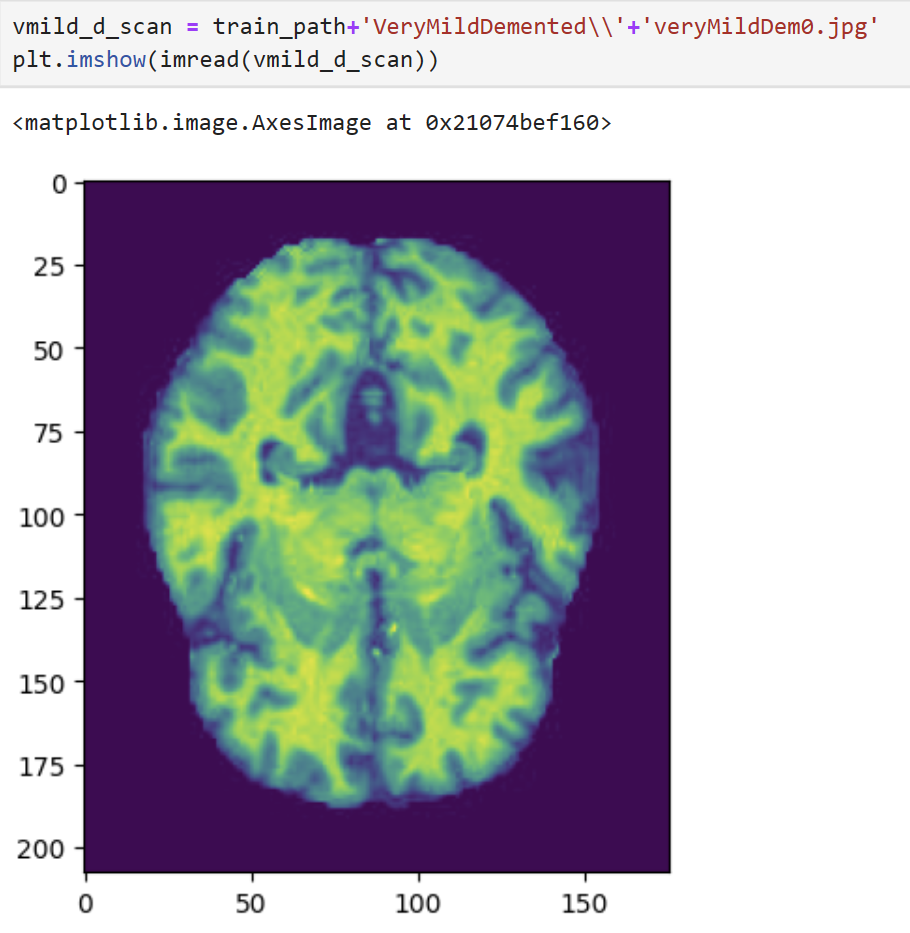
Next, I wanted to see the pixel values in the dataset to determine if they required normalization.



Since the greatest pixel values were near 255, we need to divide all of these values by 255.

Lastly, I plotted one image of every classification to see if I could pick up on any patterns:

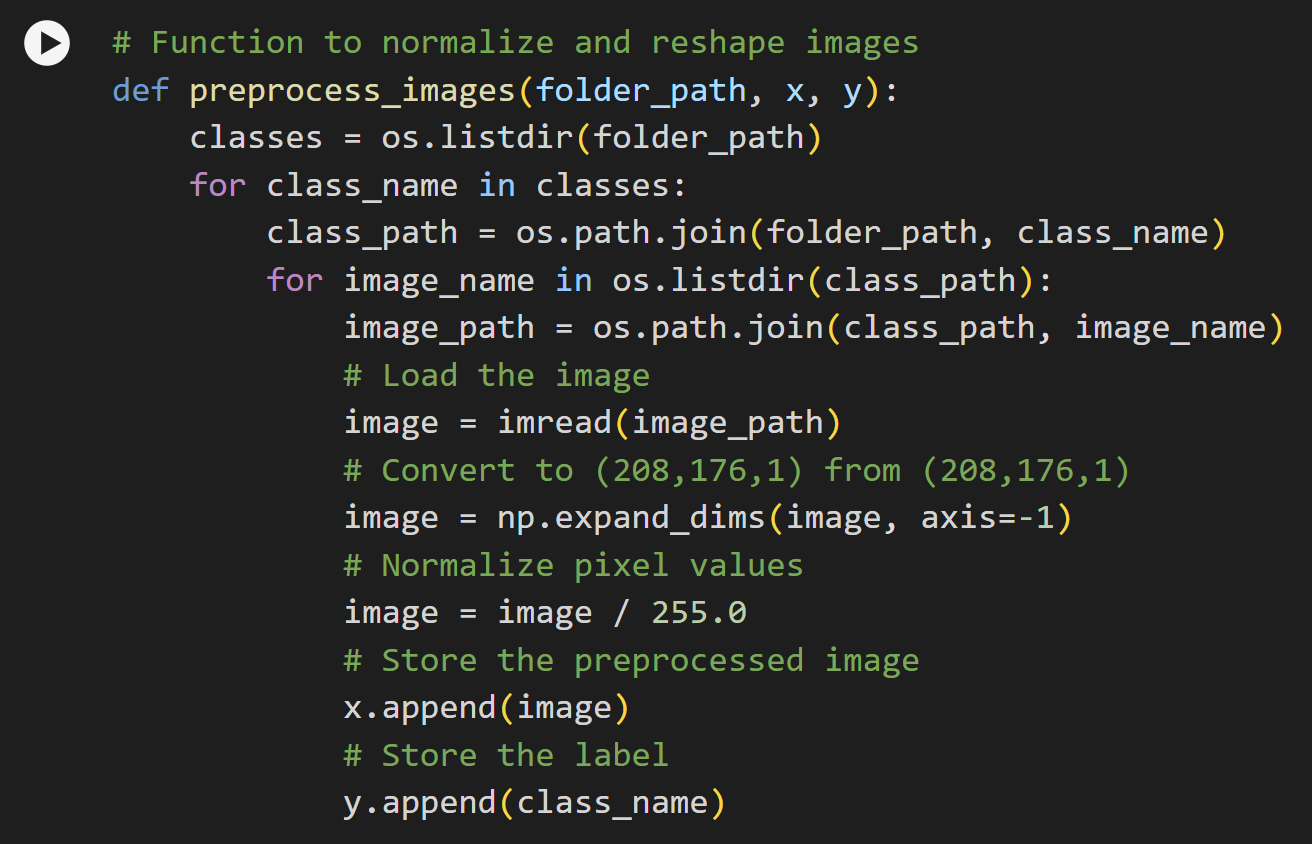


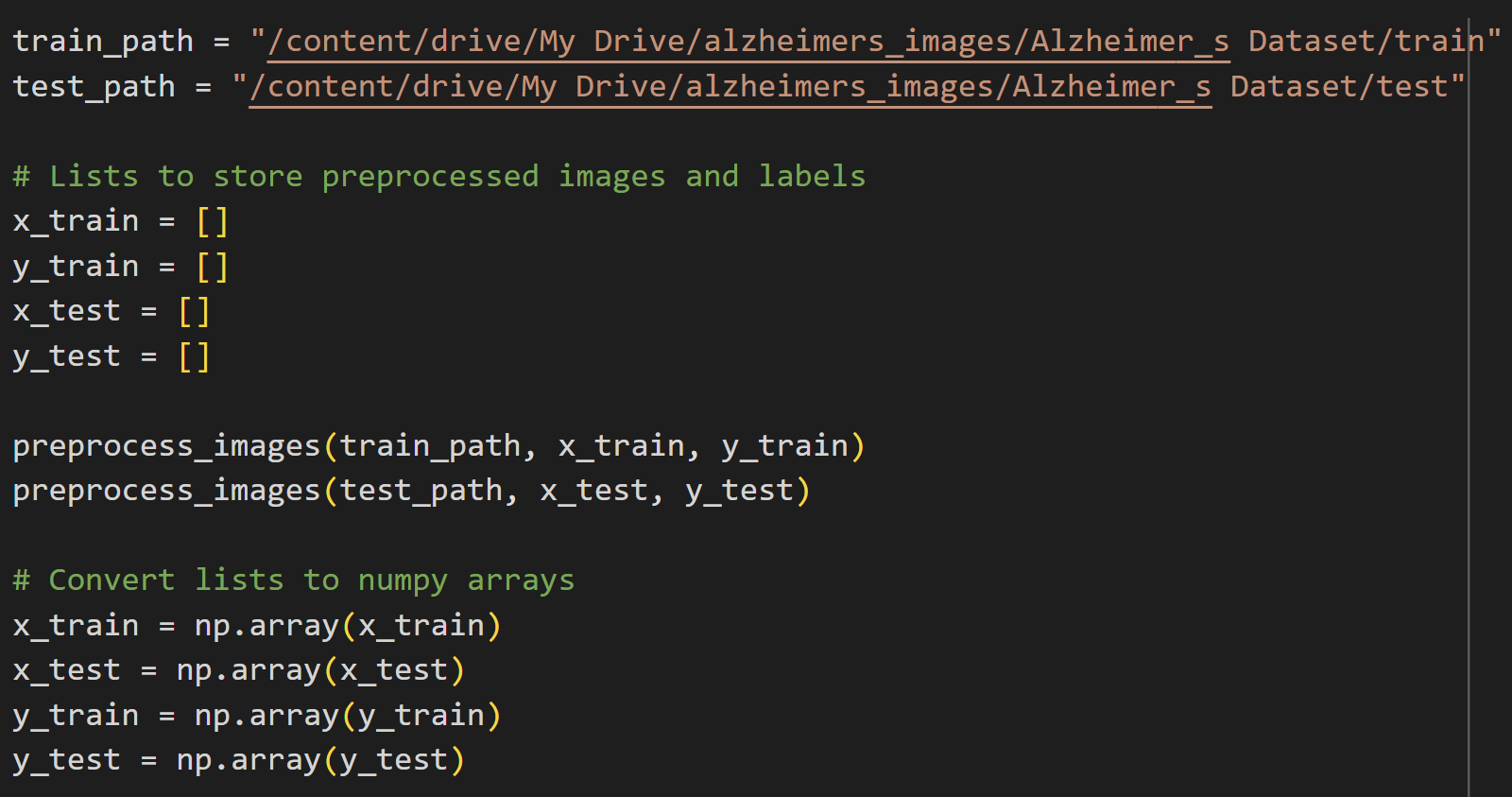
 

Through these images, the only things that I could pick up on was that there was less brightness the more significant the Alzheimer’s became. This means that reshaping the data is likely not very necessary.

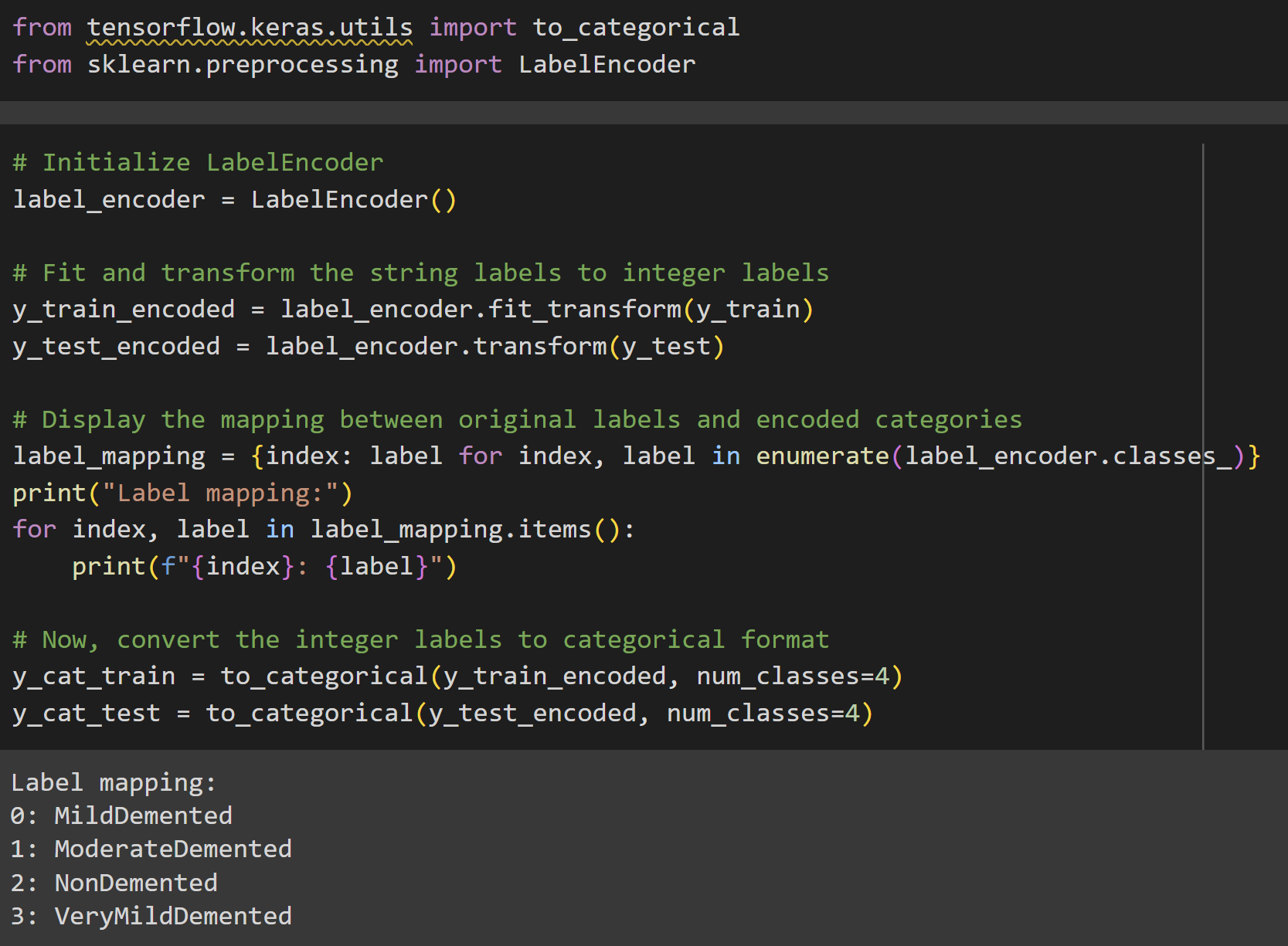
**Step 3: Data Preprocessing**

First, I created a function to read in the folder path, and arrays which stored the x values of the images for training and test data in array format, and the y values of the images for training and test data which were the correlating classifications of the images. This function transforms the image to fit the machine-learning model by adding one color channel for a grayscale image. It then normalizes the pixel values of the arrays. It stores the image pixel values to x, and the class name to y. After passing the training data and test data through this function, I convert all of these lists of data to NumPy arrays for compatibility with machine learning algorithms.





Next, I needed to convert all of the y values to categorical by first mapping the category values to integers(which are displayed below), and then converting those integers to categorical format for compatibility with the neural network.

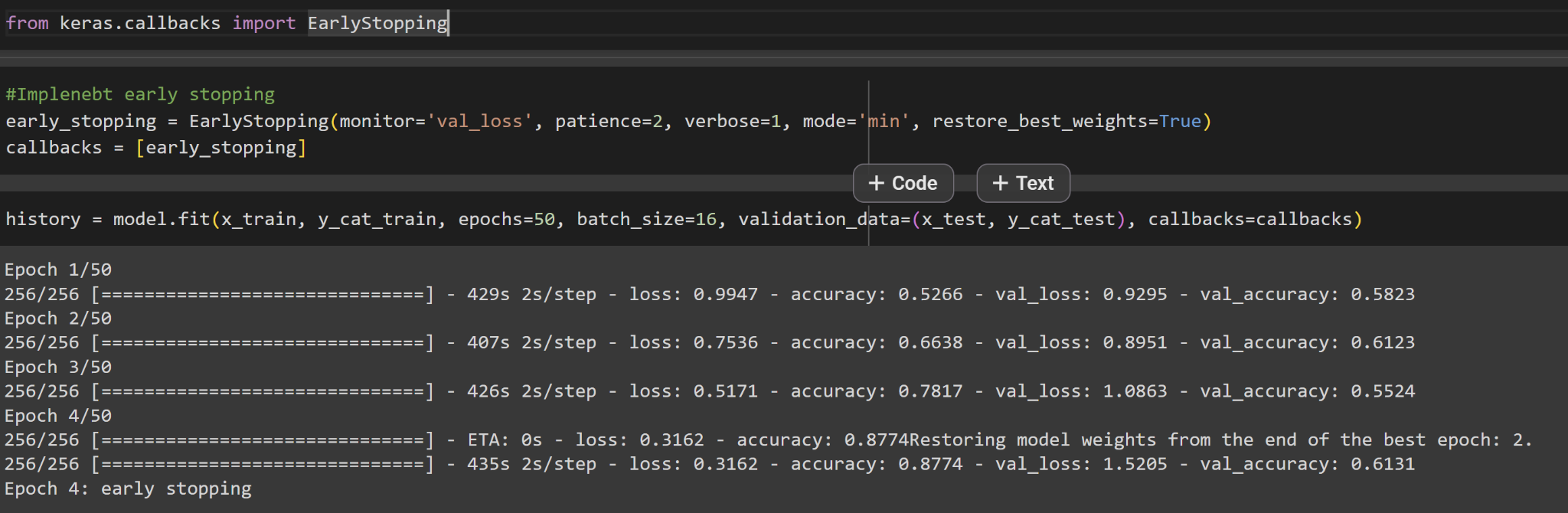


**Step 4: Training the Model**

I then set up the convolutional neural network architecture by importing Sequential, Conv2D, MaxPooling2D, Flatten, Dense, and Dropout. I created many layers including 3 convolutional layers, 3 max-pooling layers, a flattening layer, two dense layers, including the output layer, and a dropout layer. I ensured the output layer had 4 neurons, for the 4 categories. The model used categorical cross-entropy to measure loss and used the Adam optimizer.

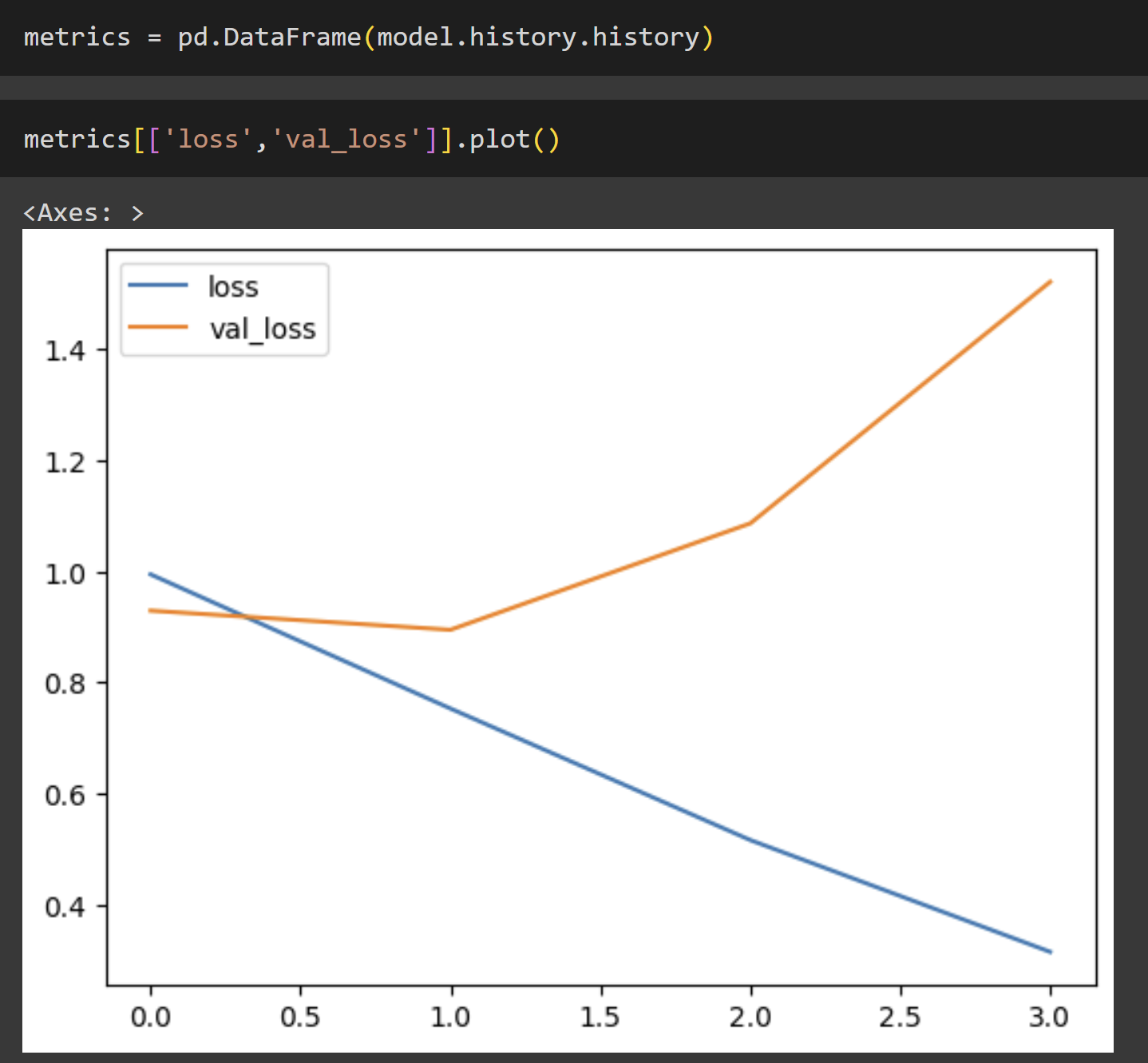
****

Lastly, I implemented early stopping, and fit the model to the training data, and validation data.

****

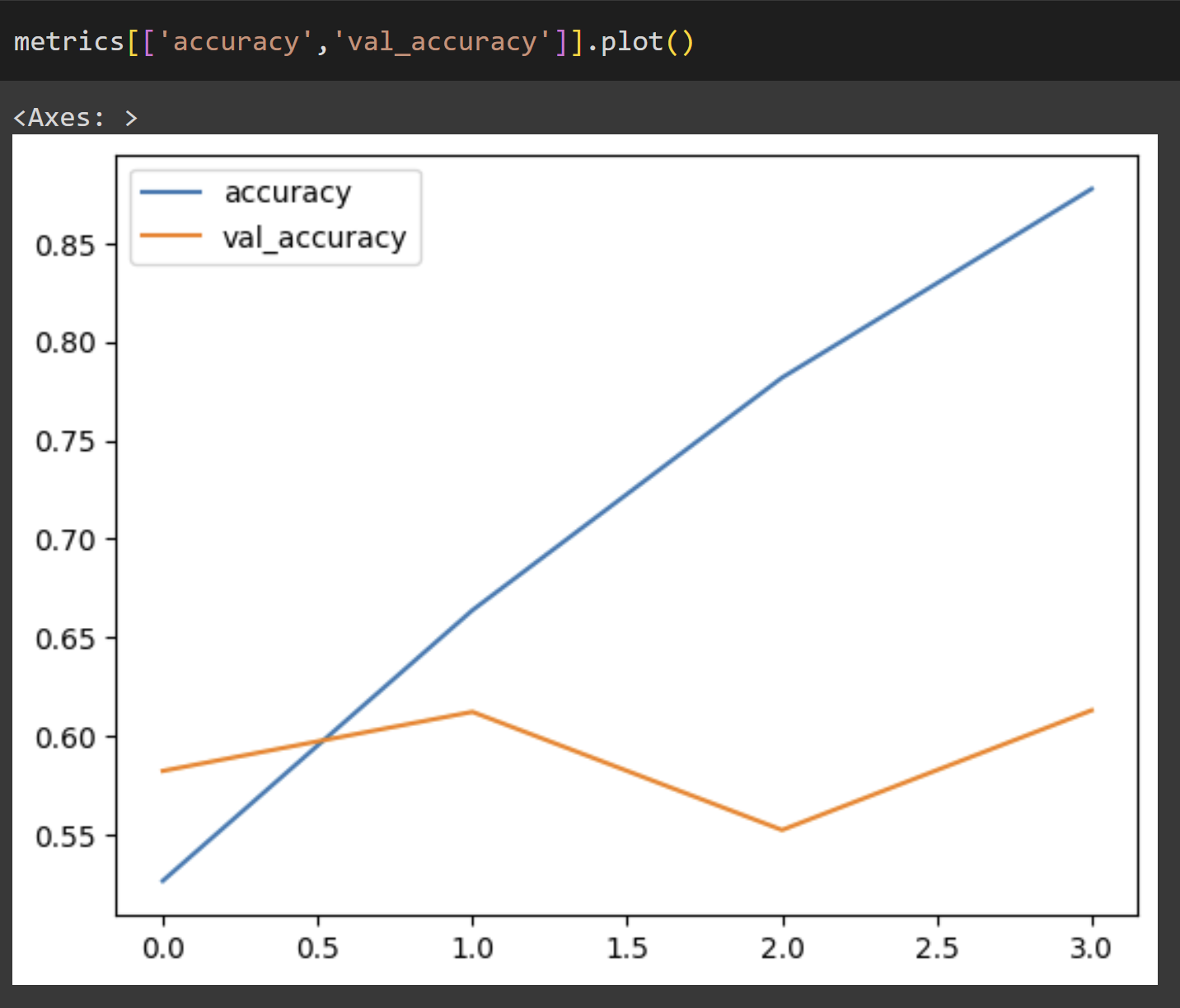
**Step 5: Analyzing the Results**

First, let’s view the training and validation loss.



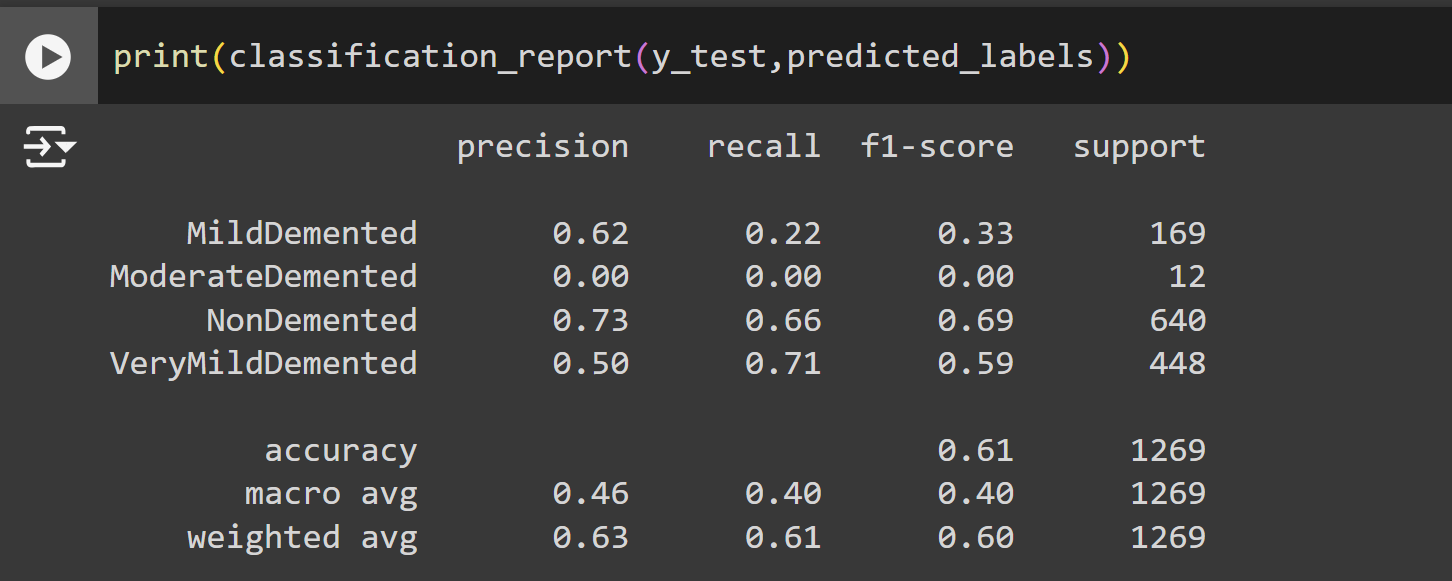
We can see that the model overfitted to the training data since the training loss decreased over time, while the validation loss increased over time. This is likely due to a lack of data in some of the categories, which leads to the same few images being overfitted. The main way that I attempted to combat this was through early stopping and restoring the best weights.

Now, let’s view the training accuracy and validation accuracy.



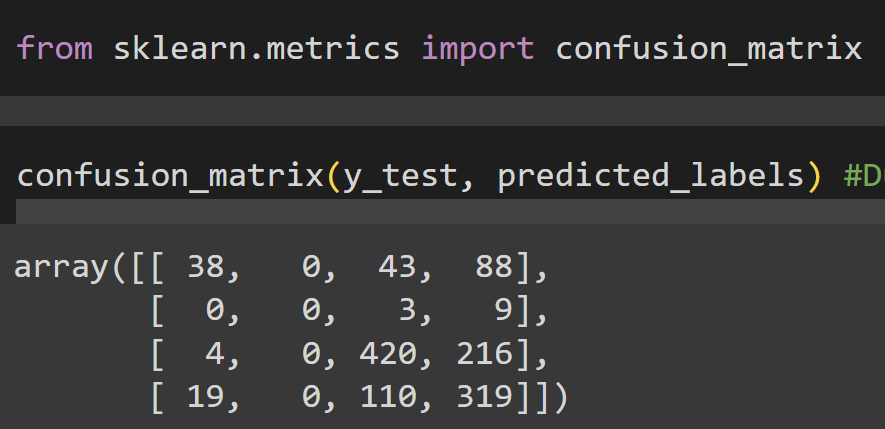
We can again see the effects of overfitting the model due to the significantly higher accuracy of training data as compared to validation accuracy. I have not viewed all of the images in the training data, but I think this may be due to the validation images and training images not having many similarities. This assumption is not only based on my training but on the other Kaggle codes that I viewed, where the greatest accuracy any TensorFlow user achieved was .56.

I then created a classification report to see what classes were most accurate, and the overall accuracy of the test data.



We can see that as compared to the .56 accuracy of other users; my model was able to achieve an accuracy of .61. Additionally, it is worth noting that the MildDemented and ModerateDemented classes had very low f1 scores due to a scarcity of data.

Lastly, I created a confusion matrix to analyze where the model’s 39% of inaccurate results came from.



We see that the majority of predictions for all of these groups fall into NonDemented and VeryMildDemented which is not surprising since there was a lot more data for these two groups, leading to overfitting. It is interesting, however, that these two groups were still more distinguishable from each other than random guesses. 420/640 NonDemented images were correctly identified, and 319/448 VeryMildDemented images were identified. We can also see that only 38/169 MildDemented images were correctly identified, and 0 out of 12 ModerateDemented images were identified. This is likely due to the model getting negative feedback whenever ModerateDemented was the predicted class since it is only accurate 12/1269 times (because of the scarcity of the data), resulting in 0 total predictions of ModerateDemented images.

Additionally, MildDemented images were frequently mistaken for VeryMildDemented images, and much less frequently mistaken for NonDemented images. VeryMildDemented images were also quite frequently mistaken for MildDemented images. NonDemented images were rarely mistaken for MildDemented, and were never mistaken for ModerateDemented images, however, were occasionally mistaken for VeryMildDemented images, meaning that if the model found a NonDemented image, it was very accurate in determining that there was an extremely low level of Alzheimer's, either NonDemented or VeryMildDemented, in the patient(636/640 times).