Comp448 Medical Image Analysis Homework-3 Report

Barış Kaplan (bkaplan18) 69054

Canberk Eker (ceker16) 60233

In order to implement the CNN classifier algorithm presented in the PDF assignment description, Python programming language was used. The favored library was PyTorch, and several classes were utilized, including ToTensor, Normalize, lr_scheduler, transforms, datasets, models, and nn. Using a split of 70% for training and 30% for validation, the training dataset was splitted into two halves. As suggested in the task description, the Google Colab environment was chosen as the coding and development platform. Initially, attempts were made to employ a local Jupyter Notebook environment, but those efforts failed due to the environment's inability to process the given data and execute the code due to a lack of GPU and CPU resources. The provided data was stored in Google Drive, and Google Colab used the "mount" method of the drive class in the Python module "google.colab" to access the data that was uploaded to Google Drive. The Google Colab GPU was the hardware accelerator selected for training the CNN model.

1-) Reading the document that was provided and that proposed the AlexNet network (Krizevky et al.) revealed that the network had been pre-trained using pictures from the ImageNet dataset. A 224 by 224 pixel image with three color channels makes up each image in the collection (RGB: 224 x 224 x 3 inputted image). However, it was discovered that the image collection for this assignment consisted of pictures with three color channels and a resolution of 256 by 256 pixels (RGB: 256 x 256 x 3 inputted image). The transforms used for the training, validation, and test sets/segments were resized in order to address the issue. For this, the Python "torchvision.transforms" library's "transforms.Resize()" function was used. The size parameter of 224 was specifically sent to this function to change each provided image's size from "256 by 256" to "224 by 224." In conclusion, the transforms were resized using "transforms.Resize(224)".

2-) The input was normalized using the "transforms.Normalize()" function from the Python "torchvision.transforms" library. The PyTorch documentation of the AlexNet Network was consulted in order to establish the mean and standard deviation parameters within this function. "mean=[0.485, 0.456, 0.406]" and "std=[0.229, 0.224, 0.225]" were the suggested values for the mean and standard deviation parameters. The following function, which was used to normalize the input, was built using these variables and the "transforms.Normalize()" method:

“transforms.Normalize(mean=[0.485,0.456,0.406],std=[0.229,0.224,0.225])”

3-) By using the "nn.Linear()" method from the Python "torch.nn" library, a linear transformation was used to replace the final layer of the AlexNet architecture. Given that there are three classes (training, validation, and test), the "nn.Linear()" function's input size parameter was set to 4096 (in_features = 4096), and the output features parameter was set to 3 (out_features = 3). The convolutional model's sixth index of the classifier was chosen in order to access and change the linear transformation. However, it was discovered that the linear transformation could also be accessed and modified by selecting either the first or fourth index. To modify the final layer, the following code block was used:

```
size_of_input = convolutional_model.classifier[6].in_features
convolutional_model.classifier[6] = nn.Linear(size_of_input, 3)
```

4-) As the loss function, we employed the **"Cross Entropy Loss"** in backpropagation.

5-) An SGD optimizer was utilized with the "optim.SGD()" function. As parameters for this optimizer, the momentum was set at 0.75 and the learning rate to 0.0015. Numerous tests were run to investigate various momentum and learning rates for the SGD optimizer. The results showed that the parameters' selected values (momentum = 0.75 and learning rate = 0.0015) enhanced both class and overall accuracy. As a result, these values were thought to be ideal. The values (momentum = 0.90 and learning rate = 0.1) stated in the SGD documentation and Krizhevsky's work were also evaluated, however the results showed poorer accuracy.

A learning rate scheduler was also used, with a step size of 4 and a gamma of 0.10. Every four epochs, this scheduler attempted to reduce the learning rate by a factor of 0.10. As a result, larger updates were used during the early training stages to obtain a more rapid and effective loss reduction. In the following stages, the goal was to add smaller updates to the learning rate in order to converge the CNN model towards an ideal solution and improve its alignment with the training data. The provided learning rate scheduler was used with the mentioned parameter values to speed up this process and somewhat reduce overfitting.

6-) A weighted sampling strategy has been used to solve the issue of class imbalance. Each class in the training, validation, and test sets has been given a weight using a function that has been developed. The total number of picture labels for each class in the corresponding sets is verified during the weight assignment procedure. A lesser weight is given to that particular class if this number is high and demonstrates an overrepresentation of that class. In contrast, if the number is low and there is underrepresentation, the class is given more weight. An inversely proportional relationship has been constructed within the weight formula in order to achieve the drop in weight when the number of picture labels for a class is higher.

Table of Accuracies

	The training portion of the training set				**The validation portion of the training set**				**Test set**			
	Class1	**Class2**	**Class3**	**Overall**	**Class1**	**Class2**	**Class3**	**Overall**	**Class1**	**Class2**	**Class3**	**Overall**
Input normalization is applied and class imbalance is addressed	0.83721	0.96825	0.95833	0.92308	0.76666	0.95455	0.89474	0.88172	0.509	0.890	0.571	0.67533
Input normalization is applied but class imbalance is not addressed	0.80952	0.93548	0.88462	0.87462	0.7557	0.90909	0.76316	0.83333	0.482	0.848	0.662	0.68485
Input normalization is not applied but class imbalance is addressed	0.78049	0.92308	0.66666	0.83077	0.71667	0.92046	0.68421	0.80645	0.500	0.876	0.584	0.68485