# COMP 448/548 – Medical Image Analysis Homework #3

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Implementation details**  Specify the platform that you used for your implementation.  Explain how you made use of the pretrained AlexNet to design your own classifier:   1. How did you make the input size compatible with the AlextNet network? 2. How did you normalize the input? 3. What parts of the AlextNet architecture did you modify? How did you modify the last layer? 4. What loss function did you use in backpropagation? 5. How did you select the parameters related to backpropagation? For example, did you use any optimizer? If so, what were the parameters of this optimizer and how did you select their values? 6. How did you address the class-imbalance problem? Additional comments, if you have any. | | | | | | | | | | | | |
|  | Training portion of the training set | | | | Validation portion of the training set | | | | Test set | | | |
| Class 1 | Class 2 | Class 3 | Overall | Class 1 | Class 2 | Class 3 | Overall | Class 1 | Class 2 | Class 3 | Overall |
| With input normalization and with addressing the class- imbalance problem | 1 | 0.938 | 0.930 | 0.976 | 0.904 | 0.636 | 0.770 | 0.868 | 0.843 | 0.760 | 0.630 | 0.854 |
| With input normalization and without addressing the class- imbalance problem | 0.881 | 0.970 | 0.695 | 0.944 | 0.900 | 0.652 | 0.769 | 0.868 | 0.843 | 0.796 | 0.680 | 0.875 |
| Without input normalization and with addressing the class- imbalance problem | 0.951 | 0.930 | 0.722 | 0.952 | 0.863 | 0.708 | 0.666 | 0.852 | 0.867 | 0.753 | 0.581 | 0.854 |

1. To ensure compatibility between the image size in the homework (256x256 pixels) and the required input size of the AlexNet network (224x224 pixels), it is necessary to resize the input images. In the provided code, this resizing is accomplished by including the transforms.Resize((224, 224)) transformation in the data\_transforms dictionary for each dataset ('train', 'valid', 'test'). By applying the transforms.Resize((224, 224)) transformation, the input images are adjusted to the desired size of 224x224 pixels. This step ensures that the input images conform to the expected dimensions of the AlexNet network.
2. The transforms.Normalize() transformation normalizes the input images by subtracting the mean values and dividing by the standard deviation values. This step helps to standardize the input data and improve the performance of the neural network. The mean and standard deviation values used for normalization are provided as arguments to the transforms.Normalize() transformation. In this case, the mean values [0.485, 0.456, 0.406] and standard deviation values [0.229, 0.224, 0.225] are specified. These values are commonly used for pre-trained models trained on the ImageNet dataset. By applying the transforms.Normalize() transformation with the specified mean and standard deviation values, the input images are normalized channel-wise. Each channel (R, G, B) is normalized independently, ensuring that the input data falls within a similar range and distribution as the data used to train the pre-trained models like AlexNet.
3. In our implementation, we made a modification to the last layer (index 6) of the AlexNet network. While we utilized the pre-existing weights for all other layers, we specifically adjusted the final layer to align with the requirements of the homework. Instead of the original output, which had more than three classes, our modified last layer produces output for three classes as suggested in the homework.
4. The cross-entropy loss function is commonly used for multi-class classification tasks, which aligns with the nature of your task that involves predicting output for three classes. This loss function is well-suited for training neural networks and computes the loss by comparing the predicted probabilities to the true labels. By utilizing the cross-entropy loss function during backpropagation, you aimed to optimize the network's parameters to minimize the loss and improve the accuracy of the classification task.
5. In the code, the backpropagation process is facilitated by using the SGD optimizer and a learning rate scheduler. The SGD optimizer is initialized with a learning rate of 0.001 and a momentum of 0.9. A learning rate of 0.001 determines the step size for updating the model's parameters during backpropagation, while a momentum of 0.9 helps accelerate convergence by considering a weighted average of past updates. The specific parameters of the optimizer ensure a balance between learning rate and momentum to optimize the model effectively. Additionally, a learning rate scheduler is employed with a step size of 7 and a gamma value of 0.1. This scheduler reduces the learning rate by a factor of 0.1 every 7 epochs, gradually fine-tuning the model over time. The chosen parameters for the learning rate scheduler aid in achieving better convergence and performance during the training process. Overall, the selection of these specific parameter values involves fine-tuning based on empirical experimentation and the characteristics of the dataset and model architecture.
6. To address the class-imbalance problem, we implemented a solution that involved weighted sampling during the data loading process. By assigning weights to the samples based on the class distribution, we ensured that the model receives a balanced representation of the different classes during training. This was achieved by utilizing the `WeightedRandomSampler` with the computed weights, allowing for proportional sampling of the dataset. By giving more importance to the minority class samples, we aimed to mitigate the impact of class imbalance and improve the model's ability to learn from all classes, potentially leading to better performance on imbalanced datasets.