Student Name: Karahan Sarıtaş
Student ID: 2018400174

Special Topic in CmpE: Artificial Intelligence in Healthcare (CmpE 58P) Assignment 4

Introduction

Results

In the present iteration of our model, I have attained a training accuracy of approximately 99% and a validation accuracy of around 77%. It is noteworthy that the validation accuracy showed fluctuations, ranging from 80% to 65% during the initial stages of the training, before finally stabilizing at 77%. Upon testing the model on 10 separate occasions, the average accuracy on the test set was determined to be **82.997%**.

Improvement

After analyzing the initial outcomes and charts generated during the training process, it becomes apparent that the model suffers from overfitting. While the training accuracy ultimately reaches an exceedingly high level, the validation accuracy plateaus at a certain point, accompanied by an increase in validation loss over time. Overfitting arises when the model fails to generalize and becomes overly focused on fitting the training dataset. To address this issue, we can consider the following potential solutions:

- **Expanding the Dataset**: One approach to enhance the model's validation accuracy is by increasing the size of the training dataset with additional images. This would expose the model to a wider range of variations, enabling it to develop a more generalized solution.
- **Data Augmentation**: We can try different data augmentation techniques to change the sample data slightly every time the model processes it. This way, we can make the model more robust to the variance in the dataset.
- **Reducing the model complexity**: Simplifying the model can aid in avoiding the incorporation of noise in the data.
- **Regularization techniques and early stopping**: Different regularization techniques such as L_1 and L_2 regularizations or dropout layers can be used to prevent the model from overfitting the training dataset. Additionally, early stopping can be used to pause the training phase before the machine learning model learns the noise in the data.
- **Transfer Learning**: Alternatively, we can use a model pre-trained on a larger dataset to solve our current problem. Using transfer learning is a common practice when dealing with predictive modeling tasks that involve image data as input.

In the following chapters, I adopted three different solution strategies to improve the test accuracy of our model. Firstly, I utilized extra data augmentation techniques and also used the images without any augmentation. As a result, the size of the dataset increased by a factor of three. Secondly, I applied transfer learning using ResNet-50 architecture, trained on imagenet-1k dataset, which spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. My motivation was to use a model that was trained on a larger dataset, therefore more robust against overfitting issues.

1) Data Augmentation and Expansion

In our initial method, I applied data augmentation techniques to each image retrieved from the dataset with a 0.8 probability. This resulted in the replacement of original images with augmented versions, without increasing the overall number of images. To counter this issue, I need to expand the dataset by obtaining or generating additional data. Including the original versions of the transformed images during training can also help increase the dataset size and potentially improve validation accuracy.

To create additional data for training, I applied extra augmentation techniques such as RandomMotion, RandomBlur and RandomGamma using TorchIO [1]. In short, `RandomMotion` adds a random MRI motion artifact. `RandomBlur` blurs an image using a random-sized Gaussian filter. `RandomGamma` randomly change the contrast of an image by raising its values to the power γ.

```python
def augmentation(self, data):
    transforms_dict1 = {
      tio.RandomNoise(): .25,
      tio.RandomBiasField(): .25,
      tio.RandomGhosting(): .25,
      tio.RandomSpike(): .25,
    }
    transforms_dict2 = {
      tio.RandomMotion(): .25,
      tio.RandomBlur(): .25,
      tio.RandomGamma(): .25,
      tio.RandomAffine(degrees=10, scales=0., translation=0.): .25
    }

    if(self.w == 0): transform = tio.OneOf(p = 1.0, transforms  = transforms_dict1)
    else: transform = tio.OneOf(p = 1.0, transforms = transforms_dict2)
    data = transform(data.unsqueeze(-1)).squeeze(-1)
    return data
```

Also I updated the `__getitem__` method to apply augmentation if `self.augment` is set to `True`. By adding this extra parameter, I am able to use the training dataset without any augmentation as well.

```python
def __getitem__(self, idx):
    ...
    im = self.augmentation(im) if self.partition == 'train' and self.augment else im
    ...
```

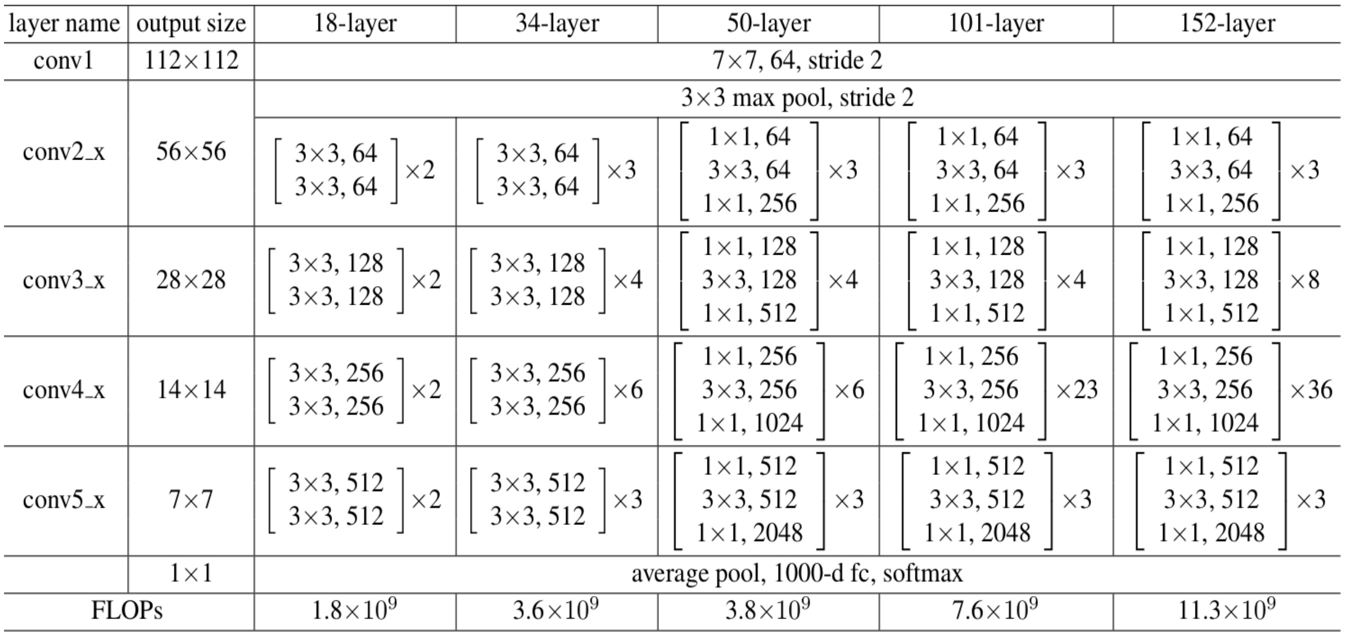
According to the input parameters, the model may apply a transformation from the first set of transformations, or the second set of transformations. According to the value of the `self.augment`, the model may not apply any transformation as well.

I proceed with creating two additional datasets and concatenating them using `ConcatDataset`:

```python
data_dict = {x: CustomDataset(initial_config['image_size'], initial_config['data_dir'], x)
                                                        for x in ["train", "val", "test"]}
# no augmentation
train2 = CustomDataset(initial_config['image_size'],
            initial_config['data_dir'],
            "train",
            augment = False)
# augmentation with second set of parameters
train3 = CustomDataset(initial_config['image_size'],
            initial_config['data_dir'],
            "train",
            augment = True,
            w = 1)
data_dict['train'] = torch.utils.data.ConcatDataset([data_dict['train'], train2, train3])
```

As a consequence, I have increased the size of our training dataset by a factor of three. Our model achieved a training accuracy of approximately 99%, while the validation accuracy exhibited minor outliers and ranged from 80% to 85%. On average, our model achieved an accuracy of **86.936%** on the test dataset across 10 runs. These results suggest that augmenting the training dataset or adding more data can effectively address the issue of overfitting and lead to improved validation accuracy in the long term. Although such an increase in accuracy might suggest that even better results can be achieved through additional data and augmentation, it also results in increased training time.

2) Transfer Learning



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

In my second approach, I used a different architecture with transfer learning. My motivation was to use another model pre-trained on a huge dataset - as a result, less prone to overfitting. To do so, I preferred using ResNet-50 architecture [2], trained on imagenet-1k dataset. I made some modifications to my code to make it suitable for the model.

```python
from torchvision.models import resnet50, ResNet50_Weights
model = resnet50(weights=ResNet50_Weights.DEFAULT)
for param in model.parameters():
  param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)
model = model.to(device)

# Decay LR by a factor of 0.1 every 7 epochs
data_dict = {x: CustomDataset(initial_config['image_size'], initial_config['data_dir'], x) for
x in ["train", "val", "test"]}
dataloader_dict = {x: DataLoader(data_dict[x], initial_config[x + '_batch_size']) for
x in ["train", "val", "test"]}

loss = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=initial_config['learning_rate'],
weight_decay=initial_config['l2_reg'])
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, min_lr = 1e-9)
```

I froze the first layer to use the model as a fixed feature extractor and add a final layer for binary classification. I also slightly updated the training and validation functions to change the output format.

This time, our model achieved a training accuracy of approximately 85% at the end of the iterations, while the validation accuracy eventually reached around 81%. Unlike previous runs, the validation loss did not show an increasing trend this time. On average, our model achieved an accuracy of **85.6873%** on the test dataset across 10 runs. These results indicate that using a model pre-trained on a larger dataset may assist in addressing the overfitting issue. Additionally, employing a more complex model could also enhance the training accuracy. Furthermore, it is worth noting that in this case, the variation in the test results was significantly smaller compared to the previous models.

3) Dropout Layers

In order to incorporate regularization into the existing architecture, dropout layers were employed at the conclusion of each convolutional block. These layers serve to mitigate the excessive co-adaptation of units to specific inputs, as outlined in the seminal work by Srivastava et al. [3].

> The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much.

Throughout the experiment, the drop rate for the dropout layer is kept at 0.2. Validation accuracy fluctuated between 70% and 80%. In contrast to the original model, the graph representing the validation loss exhibits a more gradual slope, suggesting a potential resolution of the overfitting issue to a certain degree. On average, our model achieved an accuracy of **85.6873%** on the test dataset across 10 runs.

Conclusion

As a result, we get the following average, max, and min test accuracy results out of 10 runs for each case. Data augmentation and expansion showed that we can deal with the overfitting problem by introducing different types of data augmentation methods and expanding the training dataset. Transfer learning with

ResNet-50 showed that using a more complicated architecture trained on a vast amount of images can help us improve the performance of the model as well. One can use better architectures to get even better performances.

Table 1: Binary Classification Accuracy Results

	Accuracy			
Approach	Maximum test accuracy out of 10 runs	Minimum test accuracy out of 10 runs	Average test accuracy out of 10 runs	Standard deviation of results
Original approach	85.898	79.626	82.997	2.202
Data Augmentation and Expansion	**89.382**	**86.021**	**86.936**	1.030
Transfer Learning with ResNet-50	86.513	84.969	85.687	**0.514**
Dropout Layers	87.046	83.739	84.671	1.074

Reports can be found in the following links:

1. Data Augmentation and Expansion: `https://api.wandb.ai/links/agape/qn36cqbf`
2. Transfer Learning with ResNet-50: `https://api.wandb.ai/links/agape/wk82unly`
3. Dropout Layers: `https://api.wandb.ai/links/agape/m8c45v73`

References

[1] Fernando Pérez-García, Rachel Sparks, and Sébastien Ourselin. Torchio: a python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning. *Computer Methods and Programs in Biomedicine*, page 106236, 2021.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[3] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.