Fatmanur Yaman - CMPE49T AI In Healthcare - Fall 2023

Covid-19 CT Prediction Project - Improving the Results

Complex machine learning models for understanding the hidden patterns and relationships in the data are being developed over the last decades. These models make predictions on unseen data and it is natural that the model could not find the true result for all instances. It is even impossible for the experts to predict all unseen data correctly. In this point, evaluation of the results and the suggestions for improvement are crucial for machine learning.

There are a lot of parameters in a machine learning process starting from data preprocessing to model architecture building. By changing these model components, the accuracy and the model performance can be improved.

Model Components For Improving The Results

1. **Data Augmentation:** Data quality and diversity are the crucial points for classification models. Some projects could not start due to the lack or imbalance of the data classes. It is a common problem also in medical imaging because there are some rare diseases which makes finding a positive class instance too difficult.
   In that point, data augmentation helps to increase the data size and also simulate variations in medical scans. In Covid-19 CT Prediction Project, some transformations have been applied to the image as can be seen in the code below.

   To improve the results, more unseen data can be added with different transformations.

```
def augmentation(self, data):
  ##### START OF YOUR CODE #####
  transform = tio.transforms.OneOf({
      tio.transforms.OneOf({
          tio.transforms.RandomNoise(): .25,
          tio.transforms.RandomBiasField(): .25,
          tio.transforms.RandomGhosting(): .25,
          tio.transforms.RandomSpike(): .25,
          tio.transforms.RandomAffine(degrees=10, scales=0., translation=0.): .25
      }): .8
  })
  aug_data = torch.squeeze(transform(torch.unsqueeze(data, -1)), -1)
  return aug_data
```

2. **Transfer Learning:** Transfer learning is a machine learning technique where a model trained on one task is adapted for a second related task. Instead of training a model from scratch for a specific task, transfer learning leverages the knowledge gained from solving a different but related task. It is very useful since it allows the model to transfer the knowledge that it gained from a huge dataset and apply it to the small medical image dataset.

In Covid-19 CT Prediction Project, the ResNet18 model architecture which is a pre-trained model on ImageNet has been used. The complexity of the model can be increased to improve the accuracy. The latest ResNet model which is ResNet50 can be used at that point.

```
#Create the model.
model = models.resnet50(pretrained=True).to(device)
# Freeze the convolutional layers
for param in model.parameters():
      param.requires_grad = False

# Replace the final fully connected resnet layer.
model.fc = torch.nn.Sequential(
                        nn.Linear(2048, 1), nn.Sigmoid())

model = model.to('cuda:0')
```

3. **Learning Rate Scheduling:** It is the good old learning rate :') It is one of the famous parameters that should be optimized first since it adjusts the weights by using their derivatives. But, in Covid-19 CT Prediction Project, the learning has already been scheduled because ReduceLROnPlateau has been used. It has a parameter called "patience," which is the number of epochs with no improvement in the monitored metric before the learning rate is reduced. And also it has a "factor" parameter, which denotes the factor by which the learning rate will be reduced. For example, if the factor is set to 0.1, the learning rate will be multiplied by 0.1 when triggered.

```
lr_scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=5,
                                 factor=0.1, verbose=True, min_lr=1e-9)
```

4. **Weight Initialization:** He initialization and Xavier (Glorot) initialization are techniques used to initialize the weights of neural network layers in order to facilitate the training process. Proper initialization of weights is crucial for the convergence and performance of neural networks.

5. **Regularization Techniques:** The model can overfit so the F1 score can be low due to this. In that point, dropout or other regularization techniques can be applied to prevent overfitting. Early stopping, dropout, and L2 regularization are already used in the project. ElasticNet Regularization and Batch Normalization can be used.

```
initial_config = {
    "data_dir": "/tmp/curated_data/data/",
    "image_size": 128,
    "train_batch_size": 64,
    "val_batch_size": 32,
    "test_batch_size": 1,
    "activation": "relu",
    "drop_rate": .2,
    "optimizer": "Adam",
    "learning_rate": 1e-3,
    "l2_reg": 1e-4, # Weight decay
    "nb_epoch": 50,
    "early_stopping": 15, # trigger value for early stopping
    }
```

6. **Hyperparameter Tuning:** This is another way of improving the model performance since it tries different values for the hyperparameters. But, in Covid-19 CT Prediction Project, the hyperparameter tuning has already been used for the activation function, optimizer, dropping rate, weight decay, and the learning rate parameters. Thus, there is no improvement point here for the project.

```
import random
parameter_dict = {
    ##### START OF YOUR CODE #####
    "activation": {"values": ["relu", "leaky_relu", "gelu"]},
    "optimizer": {"values": ["Adam", "sgd"]},
    "drop_rate": {"distribution": "uniform", "min": 0.0, "max": 0.9},
    "weight_decay": {"min": 0.0,"max": 0.1},
    "learning_rate": {"distribution": "uniform","min": 0.0001,"max": 0.1},
    ##### END OF YOUR CODE #####
}
```

7. **Train-Test-Split Rates:** The data is being splitted into train, validation, and test datasets. These classes should have enough instances so that the model can both learn and test appropriately. In the Covid-19 CT Prediction Project, the train-val-test split is 0.3-0.2-0.5. The train size can be increased so that the model can see more inputs.

8. **Model Architecture Adjustments:** Deeper or shallower architectures can be used to increase the model complexity. VGG16 and VGG19, GoogLeNet, or EfficientNet model architectures can be followed as a guide as it has done for ResNet18.

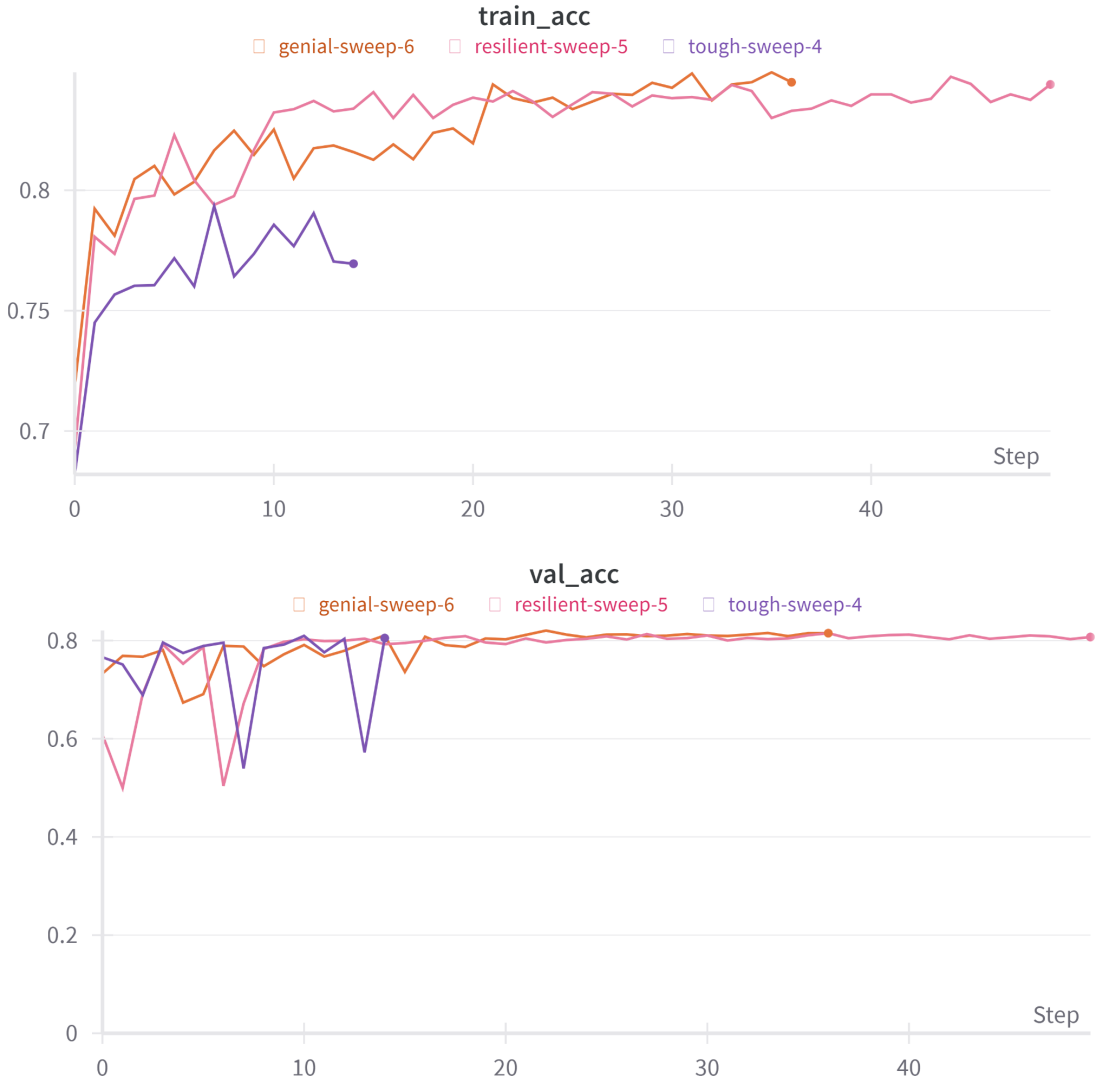
9. **Class Imbalance Handling:** If the dataset has imbalanced classes, techniques such as class weighting or oversampling the minority class to prevent the model from being biased toward the majority class should be considered. It is crucial while working on the medical datasets since it is difficult to find instances for the positive cases in rare diseases.

Improvement Trials

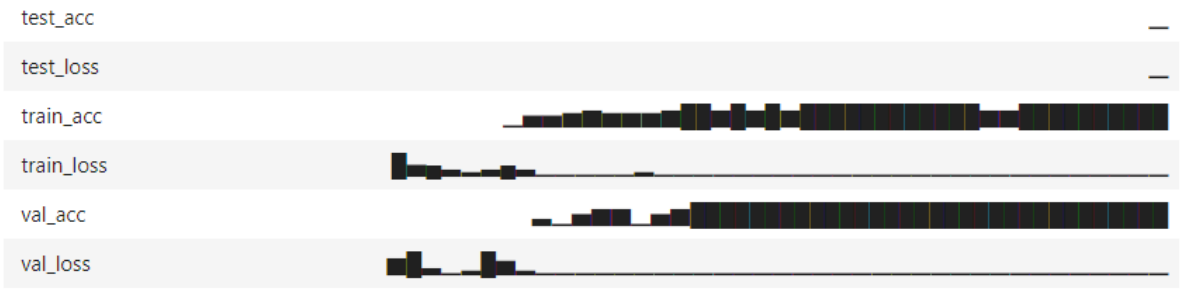
1. Transfer Learning: ResNet50

The power of transfer learning has been used with ResNet50. The time complexity is reduced since the model only learns the last layer. A linear and a sigmoid layer are added. Since it is a classification problem, the last layer should be a sigmoid layer. The improved results are in the below:

I had a chance to try the model with 3 different hyperparameter sets. The test accuracy is improved up to %85. Also, the charts for the training and validation accuracies are given in the below for 3 different hyperparameter sets. These should be as high as %95 accuracy, but my model did not give such high numbers. It can be because of the hyperparameters, or there is a small mistake somewhere.



Run history:



Run summary:

test_acc	0.84982
test_loss	0.35654
train_acc	0.84396
train_loss	0.36808
val_acc	0.80724
val_loss	0.43526

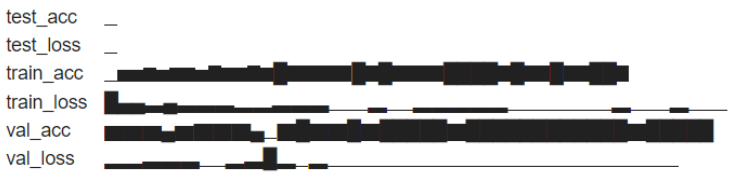
2. Regularization Techniques: Elastic Net

Elastic Net is a regularization technique used in machine learning and statistical modeling. It combines both L1 (Lasso) and L2 (Ridge) regularization penalties in the loss function during the training of a model. The objective function of Elastic Net is a linear combination of the L1 and L2 regularization terms.

In the training phase, the code above was added to the loss function to regularize the cost. As a result, the accuracy is improved.

```
#Calculate the loss and accuracy
l1_reg = torch.nn.L1Loss()(outputs,labels)
l2_reg = torch.nn.MSELoss()(outputs,labels)
alpha = 0.0001
beta = 0.001
loss = criterion(outputs,labels) + alpha * l1_reg + beta * l2_reg
```

Run history:



Run summary:

test_acc	0.85201
test_loss	0.34889
train_acc	0.83847
train_loss	0.37367
val_acc	0.80043
val_loss	0.44868

3. **Train-Val-Test Size Change**

The train-val-test is being changed as 0.5-0.15-0.35 so that there will be more input to the model for learning. It might not create huge differences since the data is being shuffled before it is given to the model. The accuracy is improved as in other suggestions.

```
# Set seed to get the same result (I specifically chose this s
np.random.seed(58)
val_split_size = .15
test_split_size = .35
```

```
New best model saved.Epoch 15With accuracy 0.8040128986026513
```