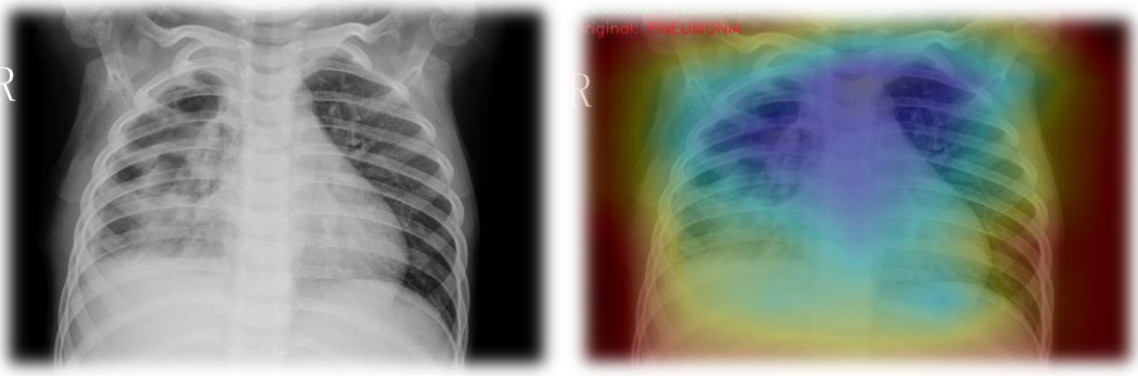
UNIVERSITY OF CENTRAL FLORIDA

CAP 5516 – PNEUMONIA CLASSIFICATION

Deep Learning-based Pneumonia Detection Using Chest X-Ray Images

In this report, we present results of applying deep learning models and techniques to detect pneumonia by using the provided pediatric chest X-rays, downloaded from kaggle. The deep learning architecture we selected is ResNet18 and the best accuracy achieved on the test set is 94.71% by leveraging ImageNet pre-training, data augmentation and hyper-parameter fine-tuning.

ALEENAH KHAN • SPRING 2024 • DEPARTMENT OF COMPUTER SCIENCE - UCF



Programming Assignment 01:
Report:

i. Implementation Details

In this section, we share the implementation details of our experiments. All the code is implemented using the PyTorch framework.

a. Network Architecture

The deep learning architecture we selected for our experiments is a very basic ResNet18 model. The detailed structure of the ResNet18 model is presented in Appendix A. The total number of training parameters were 11,177,538.

b. Hyperparameters

i. Learning Rate

We used the following values of learning rates in our experiments:
0.01, 0.001, 0.0001

ii. Batch Size

We used a batch size of 64 for all our experiments.

iii. Training Epochs

The number of training epochs varied between 15 and 40.

c. Pretraining

For Task 1.2 (a) and Task 1.2 (b), we used the ImageNet pretrained weights directly downloaded through PyTorch.

d. Data Augmentation

For Task 1.1 (b) and Task 1.2 (b), we used a series of transformations to leverage data augmentation and gain improvement in terms of model accuracy. Following table describes the transformations we used:

Transfornation Type	Description
RandomRotation	Randomly rotate the image within a range of (-20, 20) degrees
RandomHorizontalFlip	Randomly flip the image horizontally with 50% probability
RandomResizedCrop	Randomly crop the image and resize it
ColorJitter	Randomly change the brightness, contrast, saturation, and hue
RandomApply	Randomly apply affine transformations with translation

ii. Results

a. Quantitative

i. Tabular Results

First, we present the results of Task 1.1 (a) and Task 1.2 (a), side by side to make the comparison easier. In Task 1.1 (a), the ResNet18 model was trained from scratch while in Task 1.2 (a) we used the pretrained weights. We used different combinations of hyper-parameters to get the best combination possible (best highlighted in yellow; second best highlighted in green). It is to be noted that both the tasks are performed without any data augmentation. Model 2 and model 5 are the winners here.

Task 1.1 (a) - ResNet18 trained from scratch			
Without Data Augmentation			
Model_id	Epochs	lr	Test Acc.
1	15	0.01	58.01%
2	**15**	**0.001**	**84.78%**
3	20	0.0001	66.19%
4	40	0.0001	74.68%

Task 1.2 - ResNet18 trained with pre-trained weights			
Without Data Augmentation			
Model_id	Epochs	lr	Test Acc.
5	**20**	**0.01**	**82.69%**
6	25	0.001	82.53%
7	30	0.0001	80.45%
8	40	0.0001	78.89%

Next, we present the results of Task 1.1 (b) and Task 1.2 (b). These are performed with data augmentation. Model 9 and model 17 are the winners in this case.

Task 1.1 (b) - ResNet18 trained from scratch				
With Data Augmentation				
Model_id	Augmentation	Epochs	lr	Test Acc.
9	**RandomRotation**	**15**	**0.001**	**86.38%**
10	RandomRotation	20	0.001	63.46%
11	**RandomHorizontalFlip**	**15**	**0.001**	**86.38%**
12	Multiple	**15**	**0.01**	85.10%
13	Multiple	15	0.001	**80.29%**

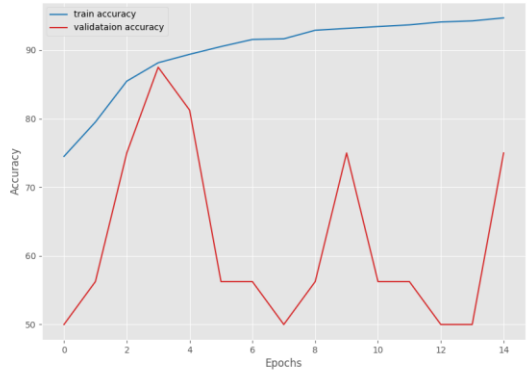
Task 1.2 (b) - ResNet18 trained with pre-trained weights				
With Data Augmentation				
Model_id	Augmentation	Epochs	lr	Test Acc.
14	RandomRotation	15	0.001	86.06%
15	RandomRotation	15	0.001	83.17%
16	RandomHorizontalFlip	15	0.001	81.25%
17	**Multiple**	**15**	**0.01**	**94.71%**
18	Multiple	15	0.001	90.87%

ii. Graphical Results

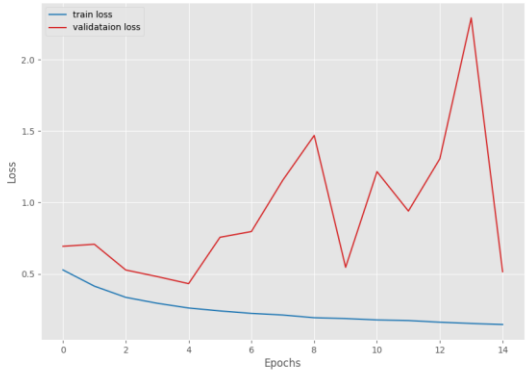
In this section, we present the loss and accuracy curves for both training and validation for the best models i.e. model 2, model 5, model 9 and model 17. The rest of the charts are presented in the attached folder at path: Pneumonia/outputs

Please note that the validation set provided is very small i.e. 8 images per class which impacts the behavior of both validation loss and validation accuracy.

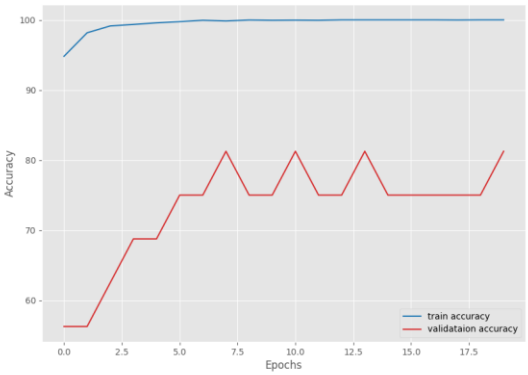
Model 02 – Accuracy



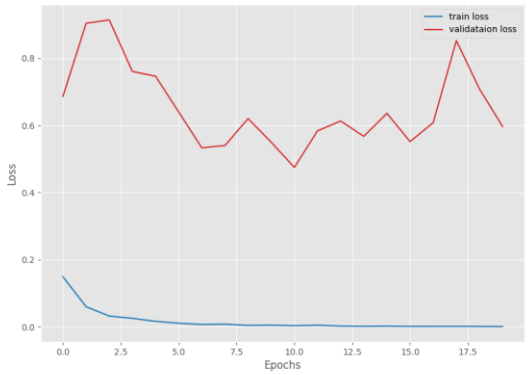
Model 02 – Loss Curve



Model 05 - Accuracy Curve

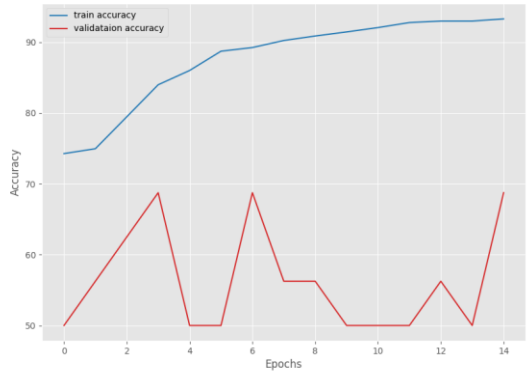


Model 05 - Loss Curve

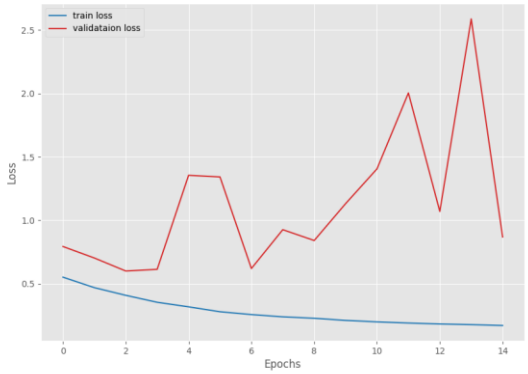


Model 09 – Accuracy Curve

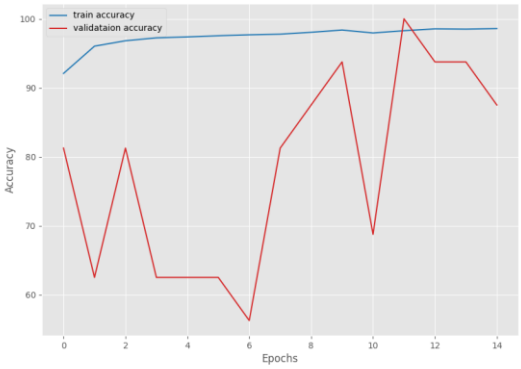
Model 09 – Loss Curve

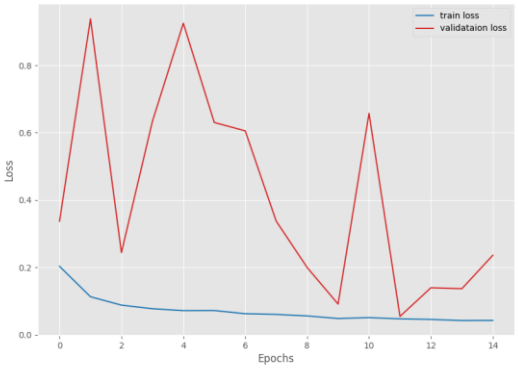


Model 17 – Accuracy Curve



Model 17 – Loss Curve



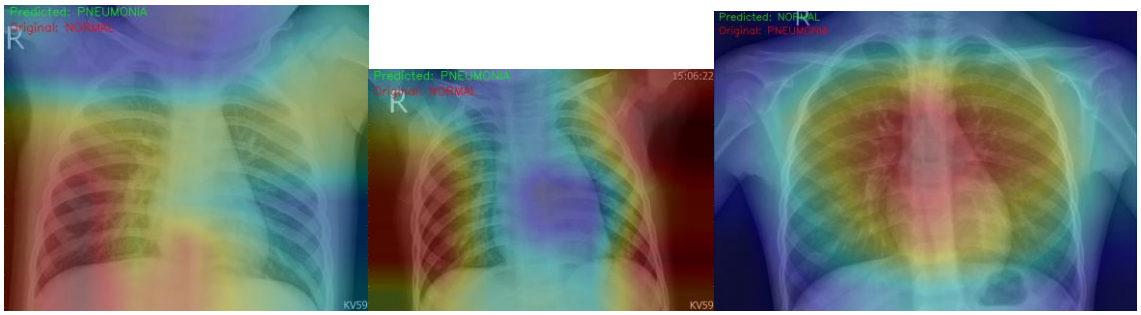


b. Qualitative Results

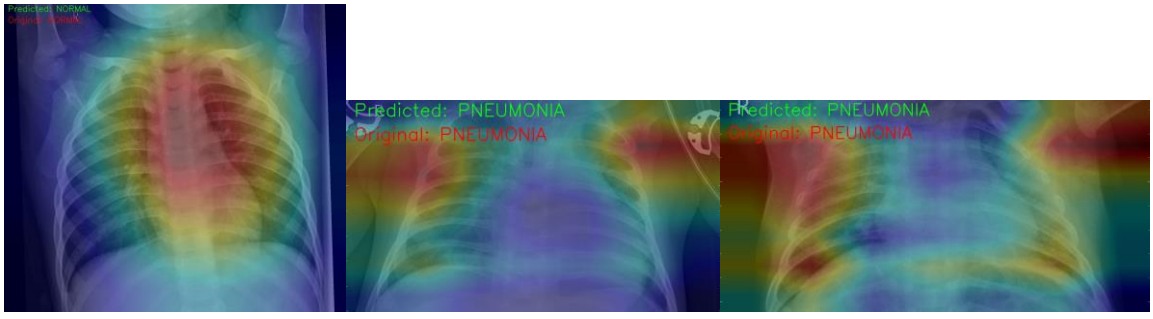
i. Visualization using CAM

In this section, we present a few examples of both failure and success cases using our overall best model, Model 17. We have provided visualization with CAM for all examples of the test set and the validation set, All other CAM images can be found at Pneumonia/CAM_visualization/cam_outputs with their corresponding inpput images at Pneumonia/CAM_visualization/cam_inputs.

1. Failure Cases



2. Success Cases



iii. Conclusion

Our best model, Model 17 is trained using ImageNet pre-trained weights and data augmentation using a combination of Random Rotation, Random Horizontal Flip, Random Resized Crop, Color Jitter, and Random Affine with a learning rate of 0.01 for only 15 epochs and achieves a 94.7% accuracy.

Appendix A

```
ResNet(
 (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (relu): ReLU(inplace=True)
 (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
 (layer1): Sequential(
  (0): BasicBlock(
   (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (1): BasicBlock(
   (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
 )
 (layer2): Sequential(
  (0): BasicBlock(
   (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
   (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (downsample): Sequential(
    (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   )
  )
  (1): BasicBlock(
   (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
 )
 (layer3): Sequential(
  (0): BasicBlock(
   (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
   (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (downsample): Sequential(
    (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   )
  )
  (1): BasicBlock(
   (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
 )
 (layer4): Sequential(
  (0): BasicBlock(
   (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
   (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (relu): ReLU(inplace=True)
   (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
   (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   (downsample): Sequential(
    (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
    )
   )
   (1): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
   )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```