

Pediatric Pneumonia Detection with CNNs

Automating Medical Diagnoses

Machine Learning II Final Project
- Kartik Mathur, Jason Witry, Jane Zeng
Group 4

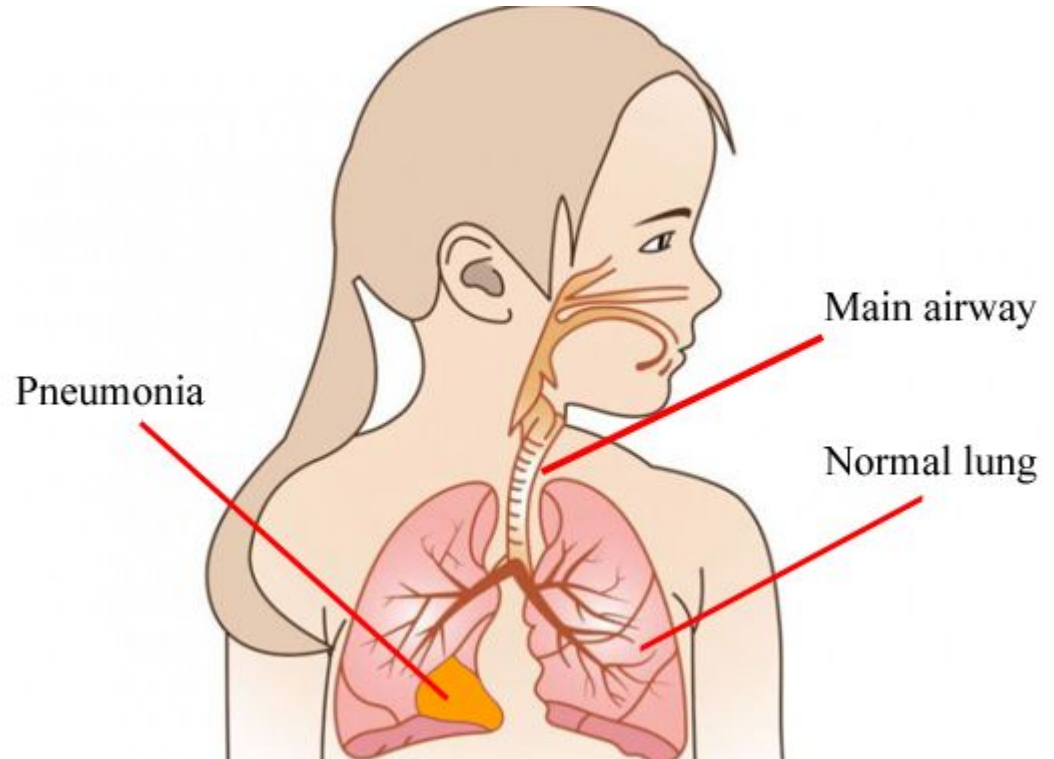


Introduction

Automating Medical Diagnoses
For Pediatric Pneumonia

Pediatric Pneumonia

- What is it ?
Infection in the lungs; air sacs fill with fluid
- What causes it?
Viruses, Bacteria etc
- Why choose this ?
Estimated to be leading cause of childhood mortality worldwide



Treatment Plans

Viral Pneumonia

- Bedrest+Fluids
- Pain Medication

Normal

- Allergies (Allergy Meds)
- Asthma (Inhaler)

Bacterial Pneumonia

- Bedrest+Fluids
- Pain Medication

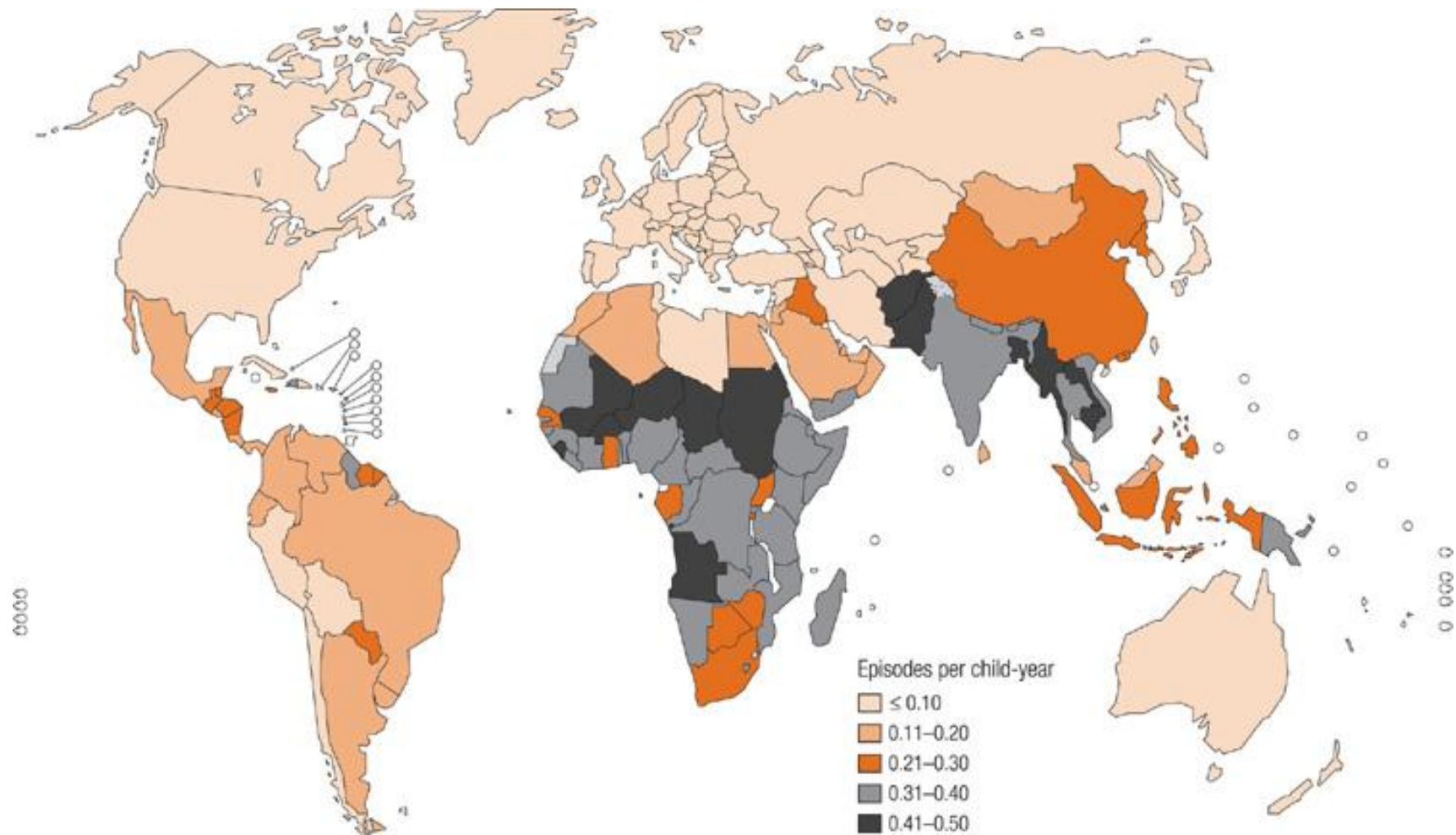
Interchangeable?



- Antiviral (Most cases resolve on their own however)

- Antibiotics IMMEDIATELY
(Cases in children DO NOT often resolve on their own)

NO



About the dataset

Categories

- 5216 Training Images
- 16 Validation Images
- 624 Test Images

Problem Statement

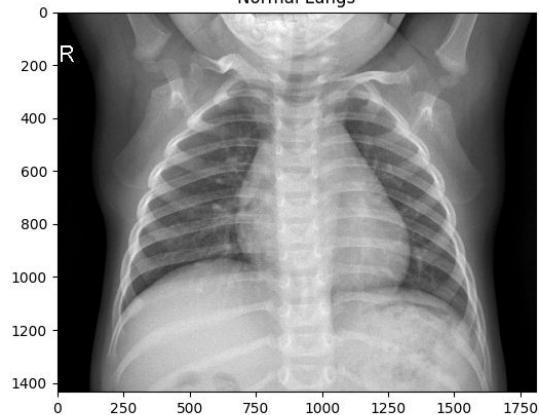
- Normal
- Pneumonia

Modified Problem Statement

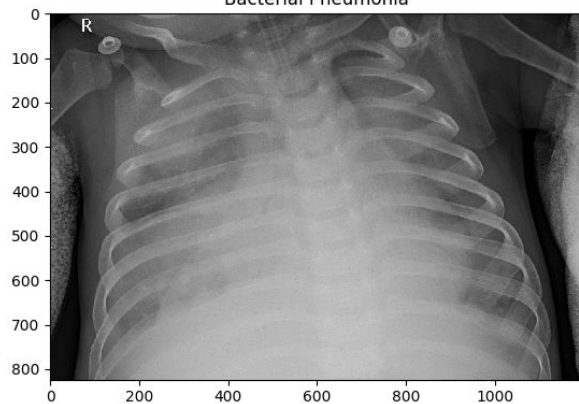
- Normal
- Viral Pneumonia
- Bacterial Pneumonia

Sample Images

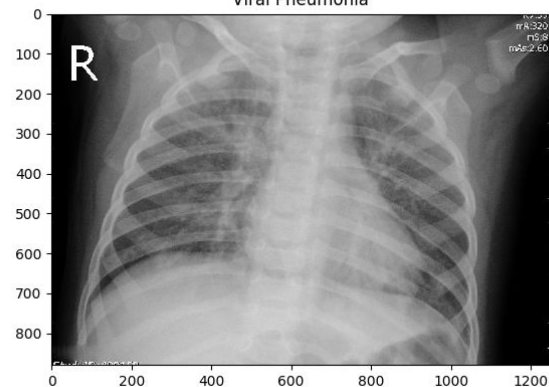
Normal Lungs



Bacterial Pneumonia



Viral Pneumonia



Pre -Processing

Original Dataset Breakdown

Folder	Class	Number
train	Normal	1341
	Bacterial Pneumonia	2530
	Viral Pneumonia	1345
validation	Normal	8
	Bacterial Pneumonia	8
	Viral Pneumonia	0

Resizing & Balancing

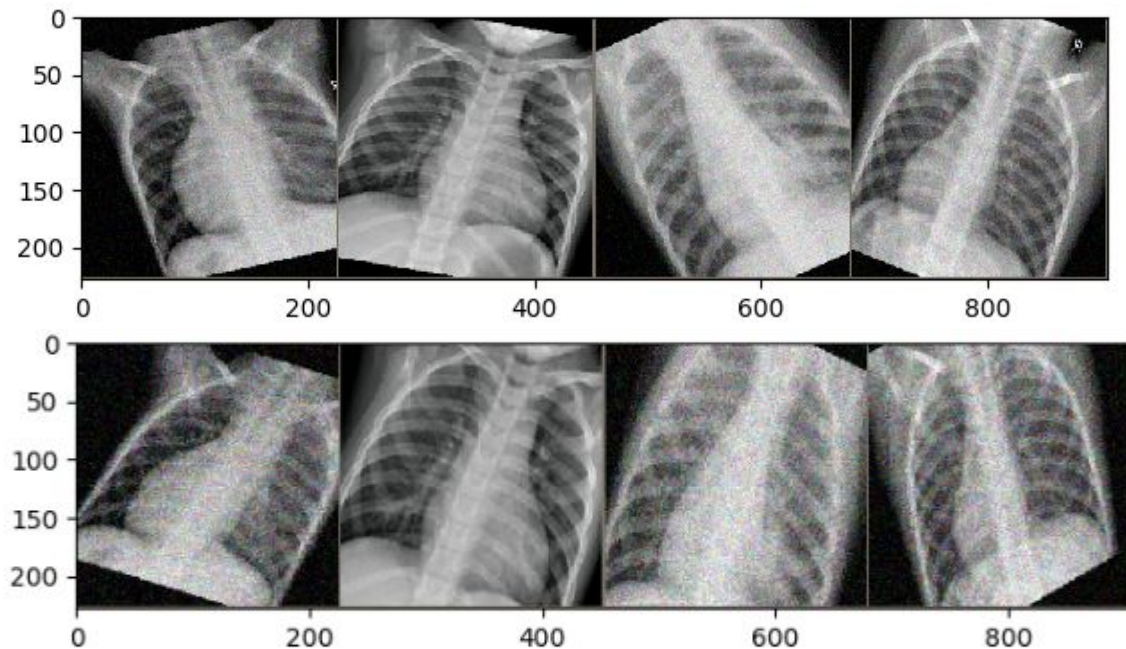
- Size of Images:
 - 28,200,**256**,350,800
- Transformations on Training Dataset
 - Resize
 - CenterCrop
 - Scaling
 - RandomHorizontalFlip
 - ToTensor
 - Normalization
 - RandomAffine
 - Gaussian Noise



After Data Balancing

Folder	Class	Number
train	Normal	1075
	Bacterial Pneumonia	2264
	Viral Pneumonia	1071
validation	Normal	277
	Bacterial Pneumonia	277
	Viral Pneumonia	277

Basic Transformations & Augmentations





Network and Training Algorithm



From the Scratch Algorithms & Basic Pre-Trained Networks

Basic CNN

MODEL

- Number of Layers(Conv2d()): 2-16
- **Activation Function:** relu(), LeakyRelu(),
- Batch Normalization: **Yes/No**
- Pooling Layers: MaxPool2d(k=2)
- Output Layer: Linear(out_features = 3)
- **Optimizer:** Adam, SGD, ASGD, AdaDelta
- **Performance Index:** CrossEntropyLoss()

Results:

Val Loss: 2.24 - 1.16

Val Accuracy: 33% - 69.24%

Hyper Parameters:

- Batch Size = 32,64,128,200
- Learning Rate = 5e-3
- Decay = 0.001(Optimizer)
- EPOCHS = 30
- dropout = 0.2, 0.3

Densenet - Pre-Trained Networks

- Densenet121
 - 4 Blocks -
 - Layers/Block = (6,12,24,16)
 - ~ 45 min
 - Faster than Basic CNN

- **Results:**

Val Loss: 1.37- 1.12

Val Accuracy: 39% - 71.24%

DenseNet(

(features): Sequential(

(conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)

(norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu0): ReLU(inplace=True)

(pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)

(denseblock1): _DenseBlock(

(denselayer1): _DenseLayer(

(norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu1): ReLU(inplace=True)

(conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)

(norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu2): ReLU(inplace=True)

(conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

)

Resnet - Pre-Trained Networks

- Resnet18
 - 4 Such Layers
 - 2 Basic Blocks / Layer
 - Faster than Basic CNN
 - ~ 38 min
- Resnet34
 - 4 such layers
 - Blocks/Layer = (3,4,6,3)
 - ~ 32 min
 - Similar Results to R18

- **Results:**

Val Loss: 0.97- 0.83

Val Accuracy: 48% - 76.24%

Overfitting:

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)

)

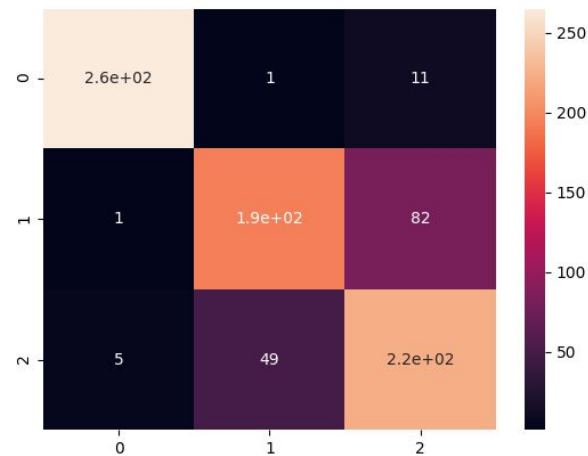
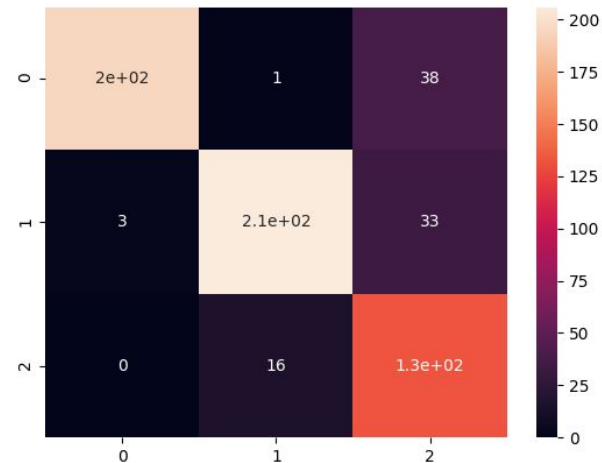
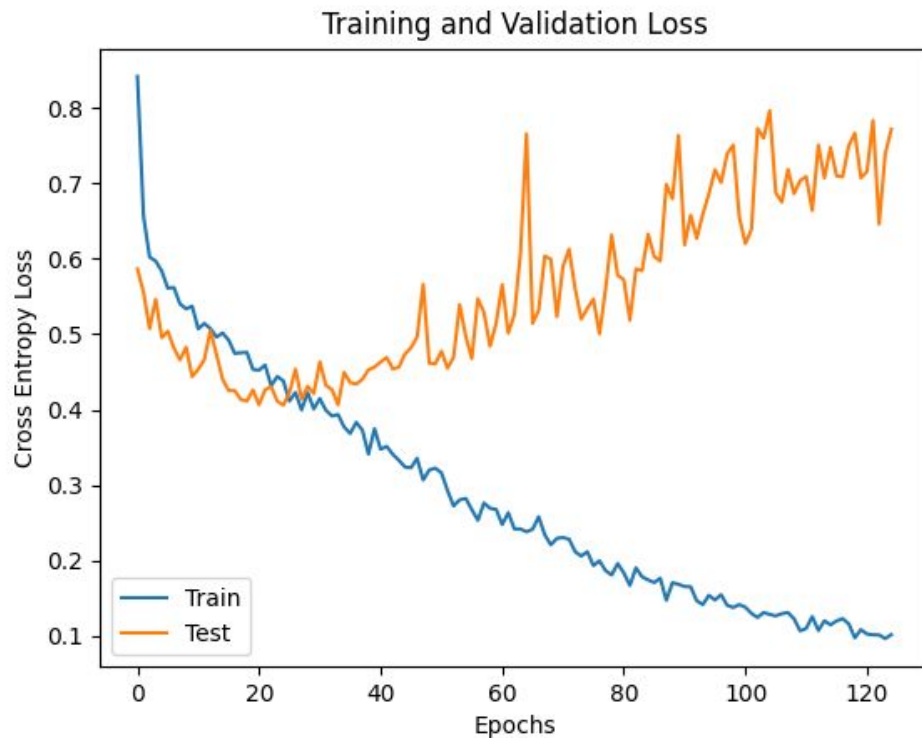
ResNet

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				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	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

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

ResNet18 with Classifier

ResNet Only



ResNet Only

Validation:

Precision	0.826
Recall	0.821
F1 Score	0.821
Cohen's Kappa	0.731

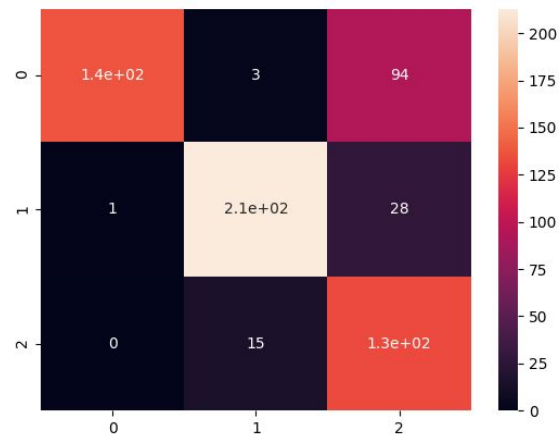
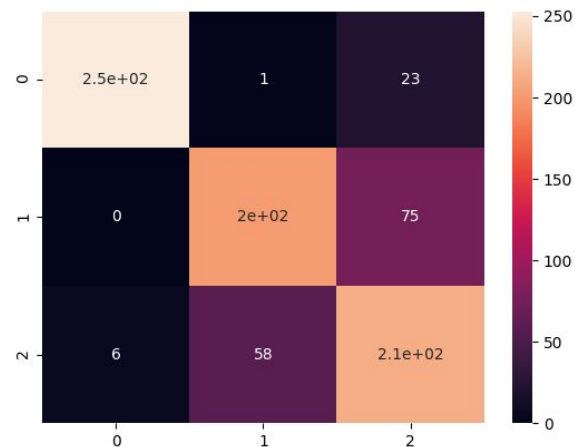
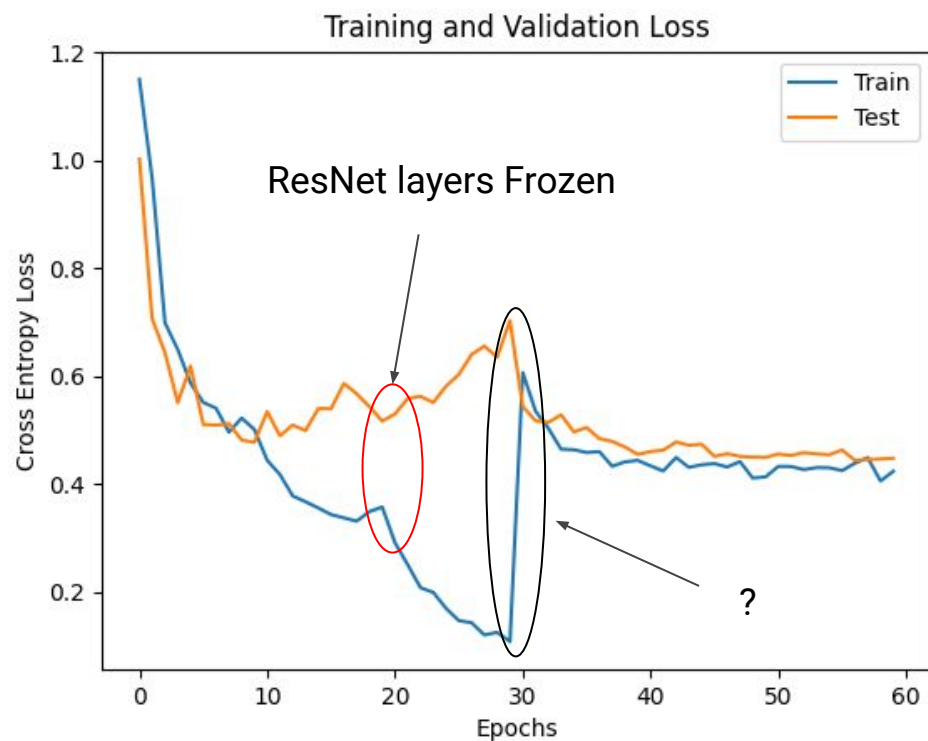
Test:

Precision	0.882
Recall	0.854
F1 Score	0.861
Cohen's Kappa	0.781

Fully Connected Classifier

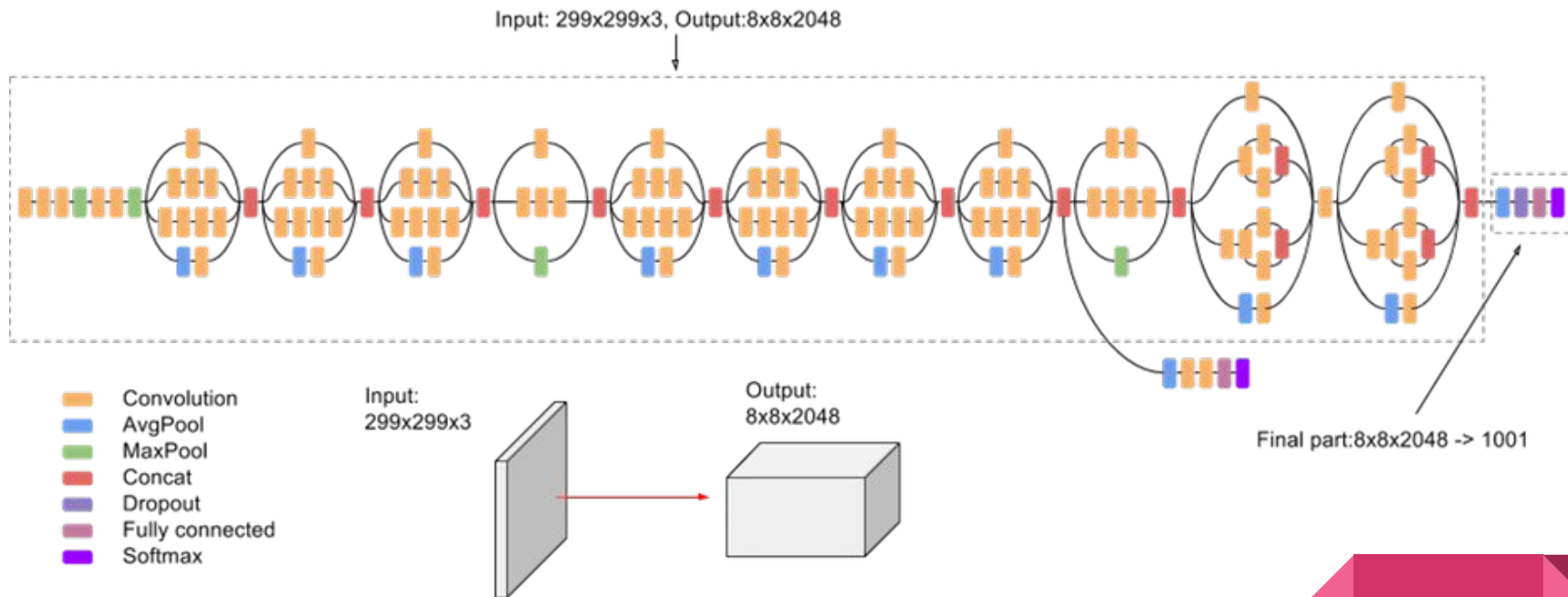
```
(1): LeakyReLU(negative_slope=0.01)
(2): Dropout(p=0.5, inplace=False)
(3): BatchNorm1d(4096, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(4): Linear(in_features=4096, out_features=4096, bias=True)
(5): LeakyReLU(negative_slope=0.01)
(6): Dropout(p=0.5, inplace=False)
(7): BatchNorm1d(4096, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(8): Linear(in_features=4096, out_features=2048, bias=True)
(9): LeakyReLU(negative_slope=0.01)
(10): Dropout(p=0.5, inplace=False)
(11): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(12): Linear(in_features=2048, out_features=2048, bias=True)
(13): LeakyReLU(negative_slope=0.01)
(14): Dropout(p=0.5, inplace=False)
(15): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(16): Linear(in_features=2048, out_features=1024, bias=True)
(17): LeakyReLU(negative_slope=0.01)
(18): Dropout(p=0.5, inplace=False)
(19): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(20): Linear(in_features=1024, out_features=3, bias=True)
```

ResNet w/ FCC

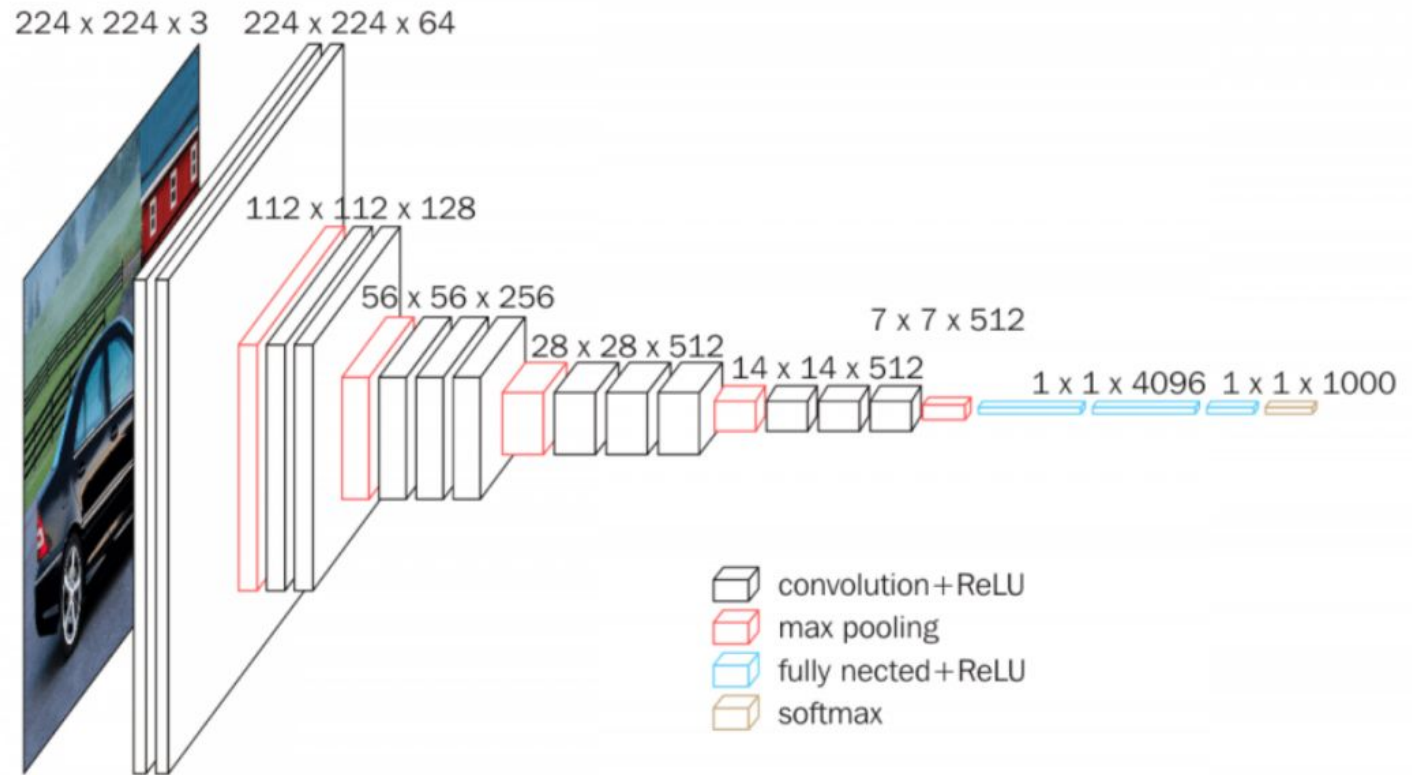


Resnet, VGG16 & Inception V3 Ensembling

Inception v3



VGG16



VGG16 & Inception V3

❑ transforms.ColorJitter

```
model1 = torchvision.models.inception_v3(aux_logits=False)
for param in model1.parameters():
    param.requires_grad = False
model1.fc = nn.Linear(model1.fc.in_features, 3)
model1 = model1.cuda()
```

```
model2 = torchvision.models.vgg16()
for param in model2.parameters():
    param.requires_grad = False
model2.classifier = nn.Sequential(nn.Linear(model2.classifier[0].in_features, 4096),
                                  nn.ReLU(),
                                  nn.Dropout(0.5),
                                  nn.Linear(4096, 4096),
                                  nn.ReLU(),
                                  nn.Dropout(0.5),
                                  nn.Linear(4096, 3))
model2 = model2.cuda()
```

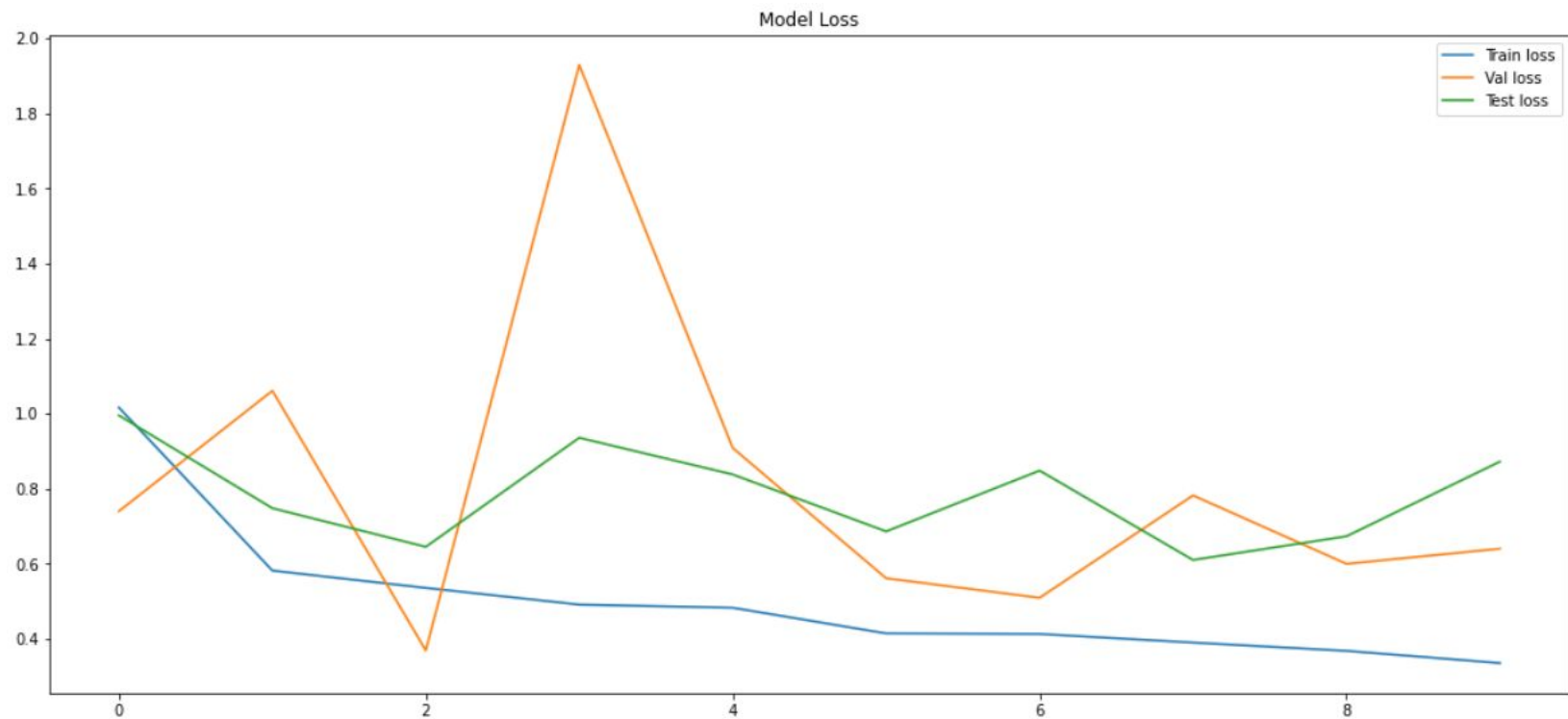
```
train batch [1303/1304]: loss 0.168365538120269786
val batch [3/4]: loss 0.5017175078392029
test batch [155/156]: loss 0.820534169673919785
Train loss: 0.4435696732252836, Train accuracy: 0.8002300613496933
Val loss: 1.0874995440244675, Val accuracy: 0.5
Test loss: 1.0613460964881456, Test accuracy: 0.7243589743589743
```

Variations

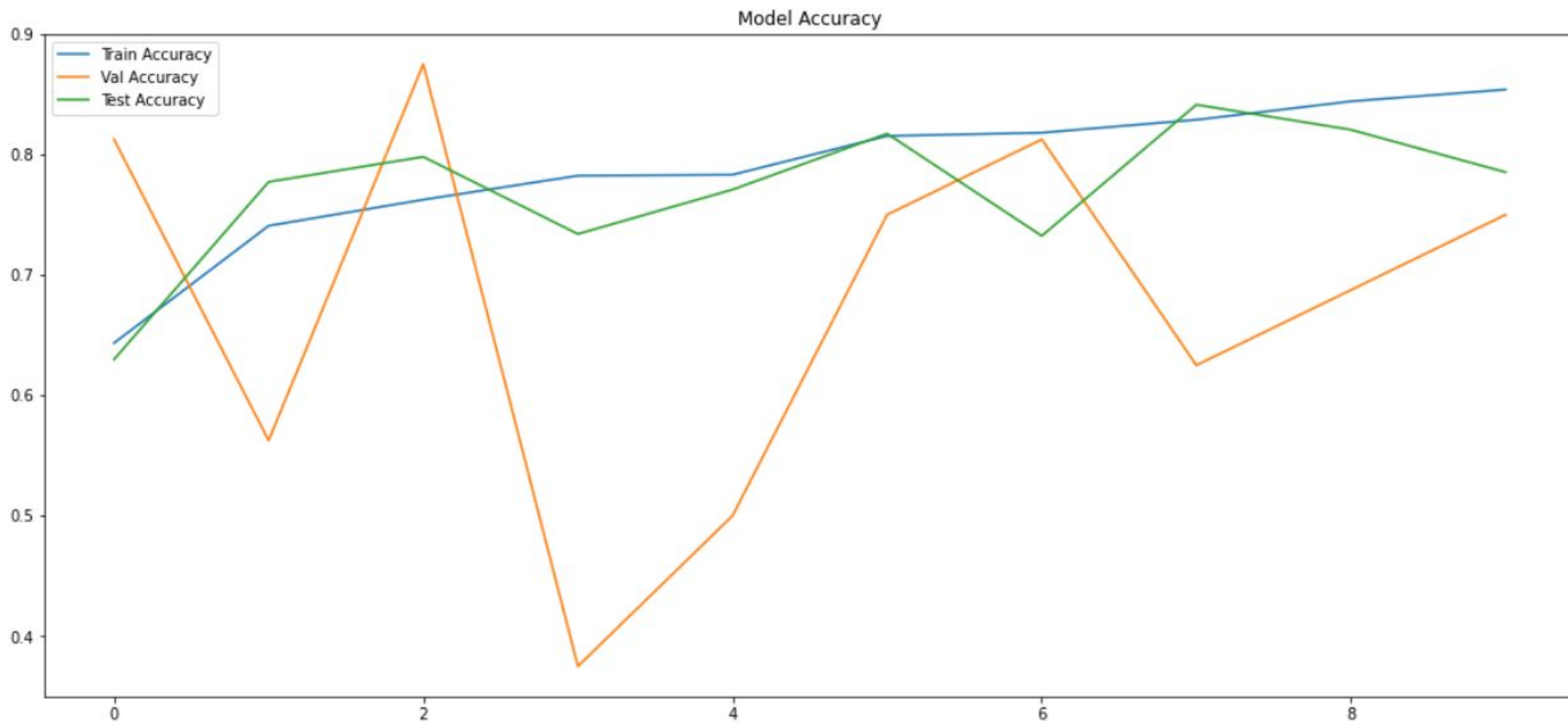
- ★ vgg16_bn() - VGG16 with Batch Normalization
- ★ Resize
 - 256, 350
- ★ Optimizers
 - SDG
 - **Adam**
- ★ Epoch
 - 10 - 50
 - **20**



Result



Result





Summary and conclusions

Summary & Observations

Best Model :

The stand alone **ResNet 18 model** with only one layer classifier performed the best of our models.

- **Accuracy = 0.851**
- **Precision = 0.882**
- **Recall = 0.854**
- **F1 Score = 0.861**
- **Cohen's Kappa = 0.781**

Observations:

- Optimizer : SGD
- Balanced Dataset is better
- Complex Problem - Complex Solution
- Image size : 256
- Dropout in FC6 may be too high, or too few layers
- Pre-trained Networks better than Basis CNN

Future Improvements

- ❖ Two models
 - Normal vs Pneumonia
 - Bacteria vs Virus
- ❖ Modify the ColorJitter
 - brightness, contrast, saturation and hue of an image
- ❖ Batch Size and Epochs
- ❖ Collect More Data



Thank You!

