Machine Learning Application:

Diagnosing Pneumonia With Chest X-Ray Images

Eshaan Vora, Jeremy Anderson, Nima Nakhjavani

Note:

We ran our neural net in a GPU environment using NVIDIA CUDA parallel computing toolkit and Tensorflow GPU. We used data augmentation during training, extending the conv_base (vgg16) model and running it end to end on the inputs which is an extremely expensive technique, intractable on the CPU.

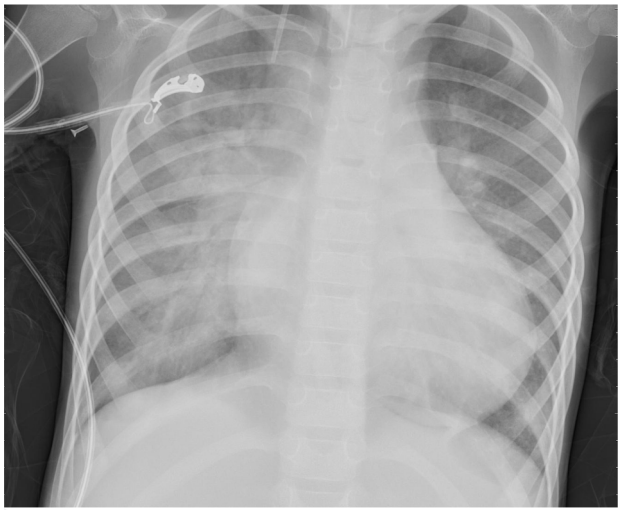
Final Model Results:

test accuracy: 0.971875
test loss: 0.0031179

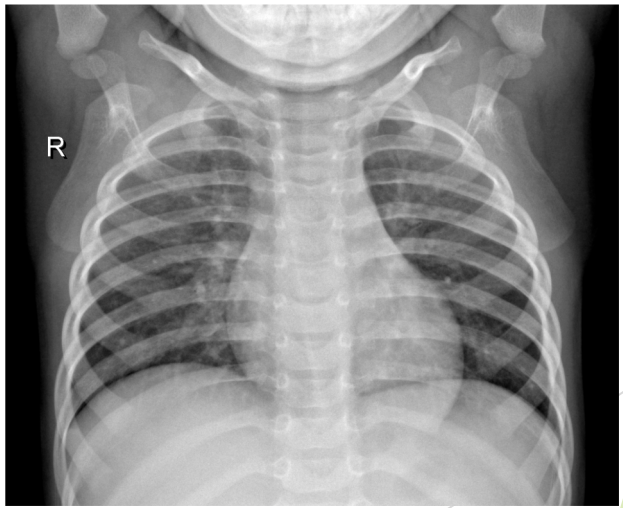
Objective:

Determine if a hospital patient has pneumonia based off of their chest x-ray images

Pneumonia



Healthy



Our Dataset

Organized into three folders

- test , train, and validation
- Sub-folders: pneumonia and normal

Consists of over 5,000 chest x-ray images

- Both healthy and pneumonia positive lungs

Chest x-rays were selected from pediatric patients 1-5 years old

All x-rays were performed as part of routine care

We created our own test train and validation subsets within the dataset

- Allowed us to train our models more accurately

Why this Relevant?

- Pneumonia is present in many severe COVID-19 cases - can be deadly in vulnerable populations such as the elderly
- Real world application
  - Could help take strain off medical professionals, providing an initial diagnostic evaluation focused on minimizing false negatives
  - Assist in more accurately predicting pneumonia cases
  - Can process and predict a large number of x-rays at a much more efficient rate
- Could be used to determine other lung conditions / other diseases in general
- Since x-rays were done as part of routine care, models can be used to spot these issues before patients even know

Our Final Model

- VGG16 - pre trained convolutional neural network model for large scale image recognition (Transfer Learning)
  - Added a convolutional base and then flattened to an output with a dimensionality of 256 - used relu activation
  - Ran dropout with a rate of 0.35
  - Flattened to final output with dimensionality of 1 - used sigmoid function because it is a binary classification (pneumonia or health)
  - Close to 17 million trainable parameters by the end
- Data augmentation allowed us to increase the amount of data available for the training model without having to physically add new data by scaling and rotating pre-existing images
- Fine tuned our model by freezing the convolutional base to get our accuracy up even higher

Model Images

```
Epoch 86/100
33/33 [==============================] - 44s 1s/step - loss: 0.0478 - acc: 0.9788 - val_loss: 0.0222 -
val_acc: 0.9531
Epoch 87/100
33/33 [==============================] - 46s 1s/step - loss: 0.0768 - acc: 0.9712 - val_loss: 1.0973 -
val_acc: 0.9125
Epoch 88/100
33/33 [==============================] - 45s 1s/step - loss: 0.0545 - acc: 0.9773 - val_loss: 0.1436 -
val_acc: 0.9258
Epoch 89/100
33/33 [==============================] - 43s 1s/step - loss: 0.0795 - acc: 0.9712 - val_loss: 0.1811 -
val_acc: 0.9375
Epoch 90/100
33/33 [==============================] - 43s 1s/step - loss: 0.0674 - acc: 0.9697 - val_loss: 0.1999 -
val_acc: 0.9194
Epoch 91/100
33/33 [==============================] - 43s 1s/step - loss: 0.0717 - acc: 0.9742 - val_loss: 0.1186 -
val_acc: 0.9312
Epoch 92/100
33/33 [==============================] - 43s 1s/step - loss: 0.0467 - acc: 0.9833 - val_loss: 0.3639 -
val_acc: 0.9355
Epoch 93/100
33/33 [==============================] - 44s 1s/step - loss: 0.0662 - acc: 0.9712 - val_loss: 0.0296 -
val_acc: 0.9344
Epoch 94/100
33/33 [==============================] - 44s 1s/step - loss: 0.0503 - acc: 0.9833 - val_loss: 1.0079 -
val_acc: 0.9000
Epoch 95/100
33/33 [==============================] - 44s 1s/step - loss: 0.0568 - acc: 0.9773 - val_loss: 0.5223 -
val_acc: 0.9094
Epoch 96/100
33/33 [==============================] - 45s 1s/step - loss: 0.0847 - acc: 0.9652 - val_loss: 0.0737 -
val_acc: 0.9290
Epoch 97/100
33/33 [==============================] - 44s 1s/step - loss: 0.0610 - acc: 0.9833 - val_loss: 0.0732 -
val_acc: 0.9250
Epoch 98/100
33/33 [==============================] - 45s 1s/step - loss: 0.0656 - acc: 0.9742 - val_loss: 0.4918 -
val_acc: 0.9419
Epoch 99/100
33/33 [==============================] - 45s 1s/step - loss: 0.0492 - acc: 0.9803 - val_loss: 0.2409 -
val_acc: 0.8875
Epoch 100/100
33/33 [==============================] - 44s 1s/step - loss: 0.0590 - acc: 0.9727 - val_loss: 0.2595 -
val_acc: 0.9226
```

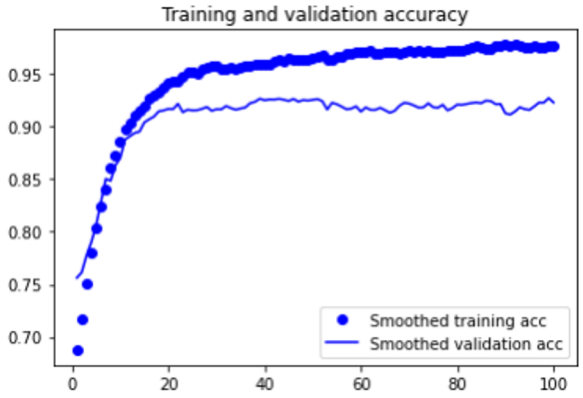
Convolutional Base:

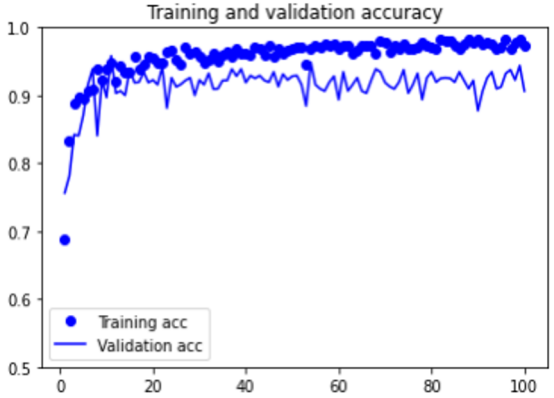
```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
```

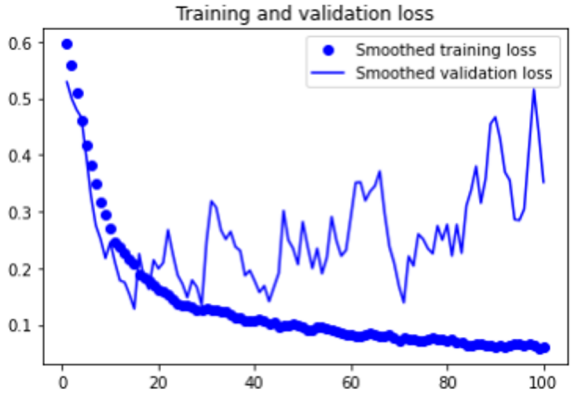
Model Summary:

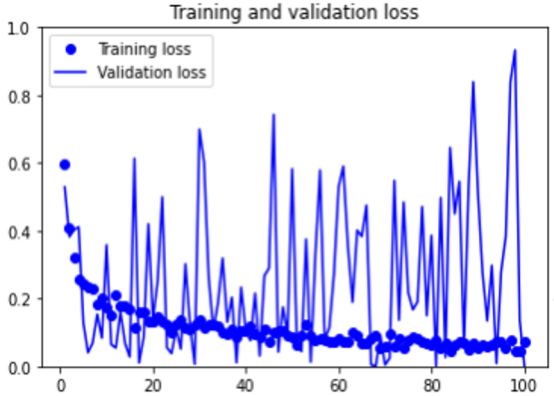
```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_5 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               2097408
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

Model Results










```
test acc: 0.971875011920929
test loss: 0.0031179096549749374
```

Future Steps

- Reevaluate neural network architecture and readjust to improve accuracy, loss, and/or runtime
  - Even on a GPU, this algorithm is very expensive, so goal is to experiment with the architecture improve runtime without compromising model performance
  - Ex: Trying various dropout rates and learning rates
- Determining the precision and recall of the algorithm
- Implement K-Fold Cross Validation
- Testing model with different image datasets

Dataset Link

https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia