Chest X-Ray
Pneumonia Detection With CNN

Michael Tillapaugh

Pneumonia Detection with CNN

According to the World Health Organization (WHO), pneumonia kills about 2 million children under 5 years old every year and is consistently estimated as the single leading cause of childhood mortality Ruden et al., 2008. It is especially an issue in developing countries where immediate treatment is needed but rapid radiological interpretation is not always available. Therefore an accurate CNN model which provides rapid results can be very beneficial in these areas where diagnosis is not as timely.

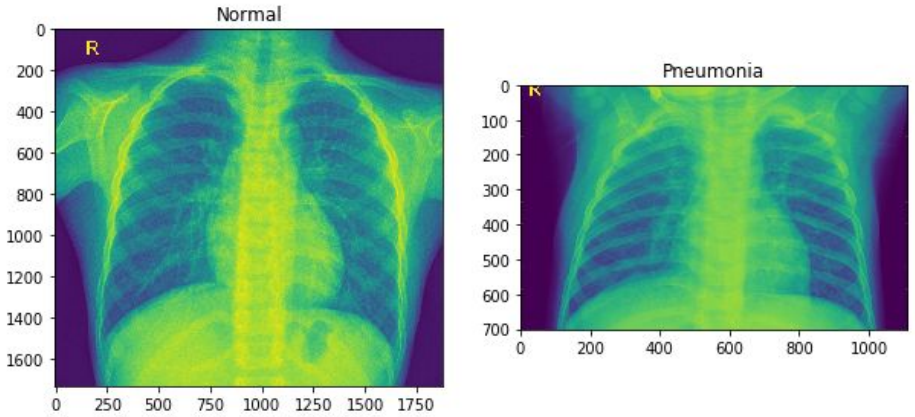
Goal: To use CNN models to classify X-Ray images as having pneumonia or not.

Info about the data set

- More information about the data set can be found here.
- Data set consists of 5,856 X-Ray Images
  - All images are labeled as “Normal” or “Pneumonia” (indicating the patient has been diagnosed with Pneumonia)
  - 1,583 labeled as “Normal”
  - 4,273 labeled as “Pneumonia”

EDA - Loading Images

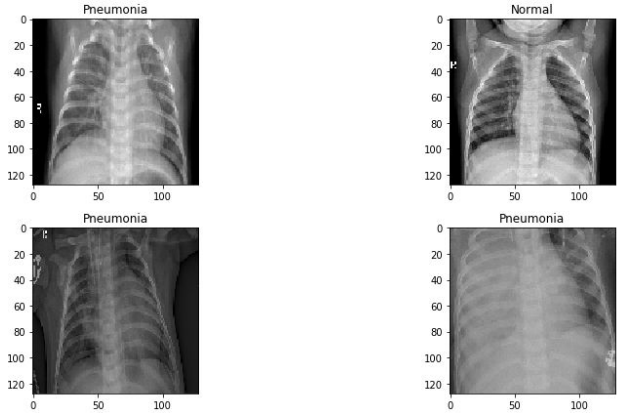
- To start here is an example of an X-Ray image labeled as ‘Normal’ and an image labeled as ‘Pneumonia’.



- Note that images are of different sizes.

Apply Labels and Reshape Images

- Next the following was performed for each image
  - Converted to np.array, resized to 128 x 128 pixels, scaled so that pixel values are between 0 and 1, and labels were applied.



- Now all images have labels and are of the same format. Above are 4 randomly sampled images.

Split Data Into Training and Testing Sets

- Next the data was split into training (80% of images) and testing sets (remaining 20% of images).
- Now data is in a format that we can feed into CNN models.

```
1 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=13)
```

```
1 X_train.shape
```

```
(4684, 128, 128, 3)
```

```
1 y_train.shape
```

```
(4684, 2)
```

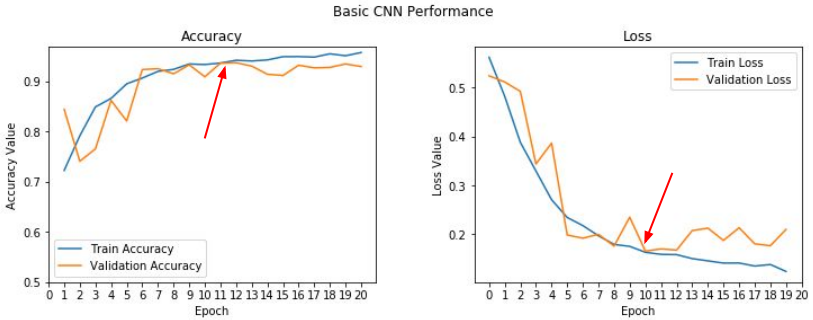
```
1 X_test.shape
```

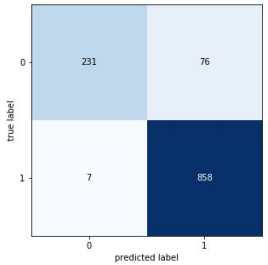
```
(1172, 128, 128, 3)
```

```
1 y_test.shape
```

```
(1172, 2)
```

Base CNN Model





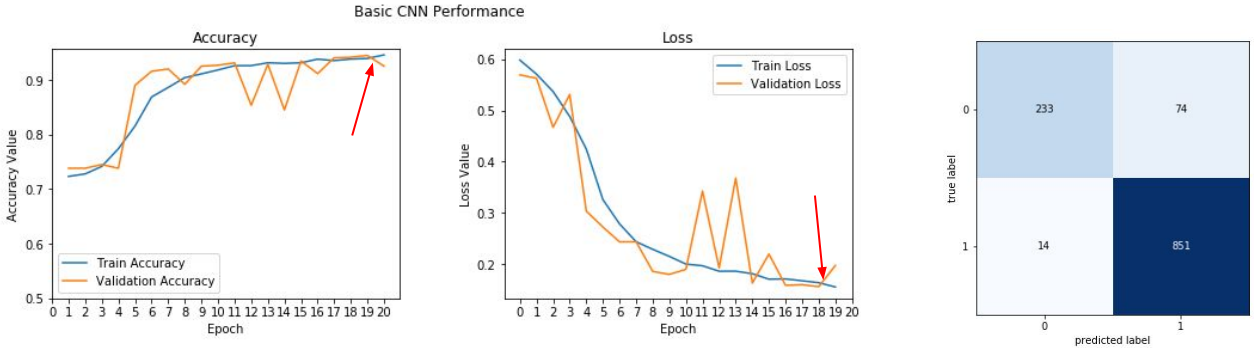

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        9280
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 512)               12845568
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 1026
=================================================================
Total params: 12,930,178
Trainable params: 12,930,178
Non-trainable params: 0
```

Summary:

- Validation Accuracy after 20 epochs: 91.8%
- Precision Score: 91.9%
- Recall Score: 99.1 %

Here the model starts to overfit a bit around 11 epochs as shown by the accuracy and loss graphs above.

Base Model with Regularization




```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 126, 126, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 61, 61, 64)        9280
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 128)       147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               2359808
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 1026
=================================================================
Total params: 2,854,658
Trainable params: 2,854,658
Non-trainable params: 0
```

Summary:

- Validation Accuracy after 20 epochs: 92.5 %
- Precision Score: 92.0 %
- Recall Score: 98.4 %

This second model improves precision score and validation accuracy. By looking at the accuracy plots above you can see that even after 20 epochs there's little to no over fitting to the training data.

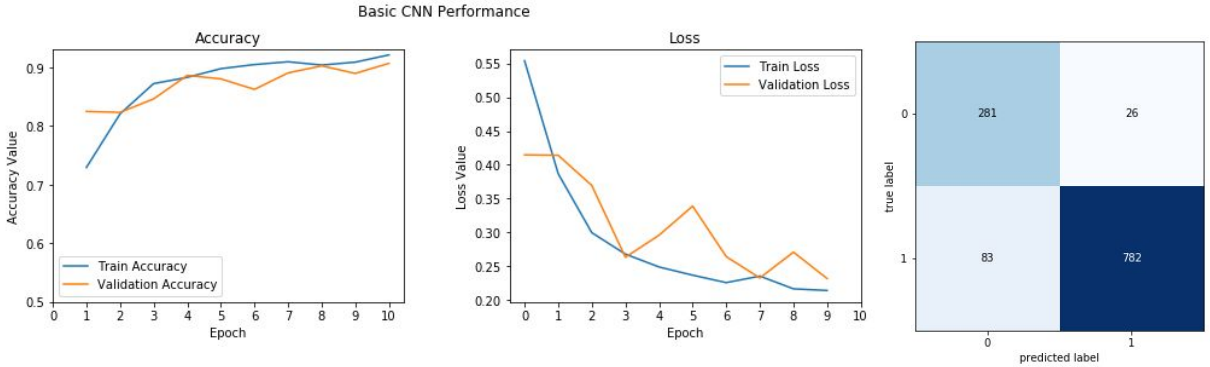
Base Model with Image Augmentation

Summary:

- Validation Accuracy after 20 epochs: 86.1%
- Precision Score: 98.1 %
- Recall Score: 82.8 %

While Precision score is still high, the validation accuracy and recall score are lower than the previous two models. This shows that the first two models performed well and had enough data to build a good model. A lot of the usefulness of image augmentation comes from expanding a limited dataset into more images, but it appears that was not needed here.

Transfer Learning with VGG-16



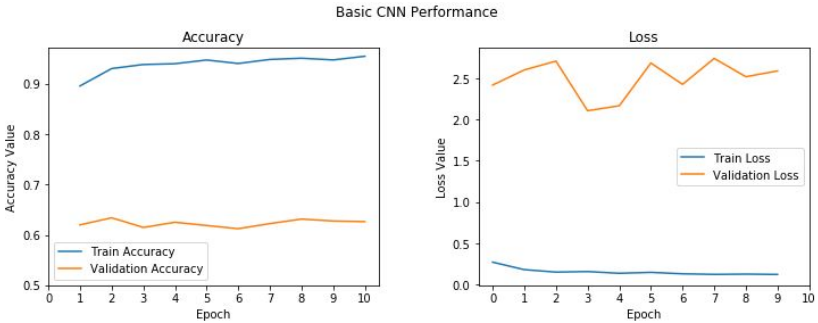

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 512)               14714688
_________________________________________________________________
flatten_6 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_20 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 2)                 1026
=================================================================
Total params: 15,241,026
Trainable params: 526,338
Non-trainable params: 14,714,688
```

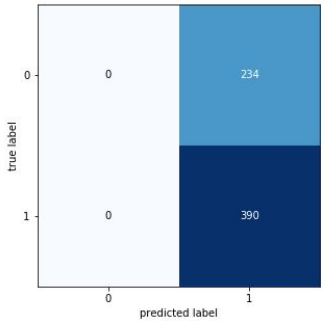
Summary:

- Validation Accuracy after 20 epochs: 90.7%
- Precision Score: 96.8 %
- Recall Score: 90.4 %

While the results here are decent it did not perform as well as the first two models of this notebook. This may be attributed to the fact that VGG-16 benefits from having more complicated images to train on and that it uses so many classifiers in its predictions. Therefore, as a simple Transfer Learning exercise it provides decent results, but for this use case it is not the best performing.

Transfer Learning with MobileNetV2






```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_128 (Model) (None, 4, 4, 1280)        2257984
_________________________________________________________________
flatten (Flatten)            (None, 20480)             0
_________________________________________________________________
dense (Dense)                (None, 256)               5243136
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 7,567,426
Trainable params: 5,309,442
Non-trainable params: 2,257,984
```

Summary:

- Validation Accuracy after 20 epochs: 62.5%
- Precision Score: 62.5 %
- Recall Score: 100 %

This example shows some shortcomings of the MobileNet model. Like VGG-16 it benefits more from more complex images as it was trained using many classifiers. MobileNet may not be a suitable option for this dataset. One big highlight from this example is the drastic overfitting we see between the training set (95% accuracy) and the testing set (64% accuracy).

Summary

Overall the above models shows some examples of how to set up various CNN models in Tensorflow. Some key take-aways are as follows:

- Transfer Learning can provide good results by using a pre-trained CNN model and adding a few addition dense layers to train for a new classifier or example.
- Image Augmentation can help expand a small data set by randomly creating augmented images in the training set. In this particular case it did not improve model results which tells us that our dataset had enough training data to begin with.
- Dropout can help overfitting by removing some of the data that gets retrained after each epoch.

Overall the best performing model was using increasing neurons in our first layers and connecting with 3 dense layers at the end. Adding in a dropout of 30% for each dense layer helped improve the results.

Final Results:

- Validation Accuracy after 20 epochs: 92.5 %
- Precision Score: 92.0 %
- Recall Score: 98.4 %

Thank you!