pneumonia-detection---convolutional-neural-network-CNN-

This notebook is intended for educational purposes, all scerios are hypothetical, and any resulting model(s) should not be used for any medical purposes.

Introduction

According to the latest publication from Meticulous Research®, the global X-ray detectors market is expected to register a CAGR of 6% during the forecast period 2022–2029 to reach $4.30 billion by 2029. The growing adoption of digital X-ray detectors, rising demand for X-ray imaging in industrial and security markets, growing geriatric population coupled with rising prevalence of chronic diseases & respiratory infections, and increasing utilization of X-ray detectors for early diagnosis & clinical applications are considered to have a positive impact on the global X-ray detectors market [TOP 10 COMPANIES IN X-RAY DETECTORS MARKET].

GE Healthcare is among the top companies operating in the global digital radiography market [Digital Radiography Market Size to Reach USD 19.82 Billion in 2028, Says Reports and Data]. GE Healthcare is also leading the way with integration of AI into their imagaing equipment and software. They currently have on the market a collection of AI algorithms embedded on X-ray systems, Critical Care Suite 2.01 (CCS), for automated measurements, case prioritization, and quality control. This application automatically analyzes images on a GE X-ray system, highlights critical information on chest X-rays, including Endotracheal Tube Positioning, Pneumothorax Triage and Notifications, Quality Care Suite2 AI algorithms that operate in parallel, and help technologists reduce image quality errors and improve efficiency [GE Healthcare].

Business Understanding

GE Healthcare would like to increase the marketablity of their CCS system by expanding their collection of AI algorithms used when automatically analyzing images. One part of this expanded collection will screen for pneumonia. This will not only help GE Healthcare stay competative as the market grows but it will also increase the functionality of their CCS system.

GE Healthcare needs a model that can successfully identify pneumonia.

Objectives

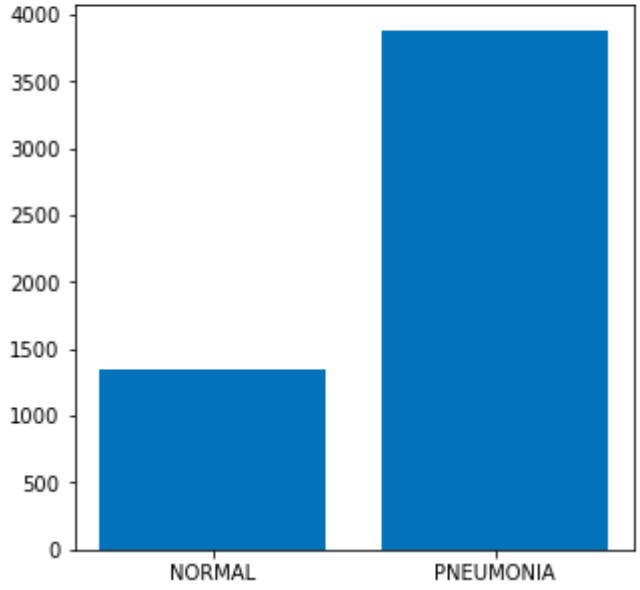
Build and test a model that can detect pneumonia using chest ex ray images, to be integrate into GE Healthcare's collection of AI algorithms embedded in the Critical Care Suite 2.01 X-ray systems.

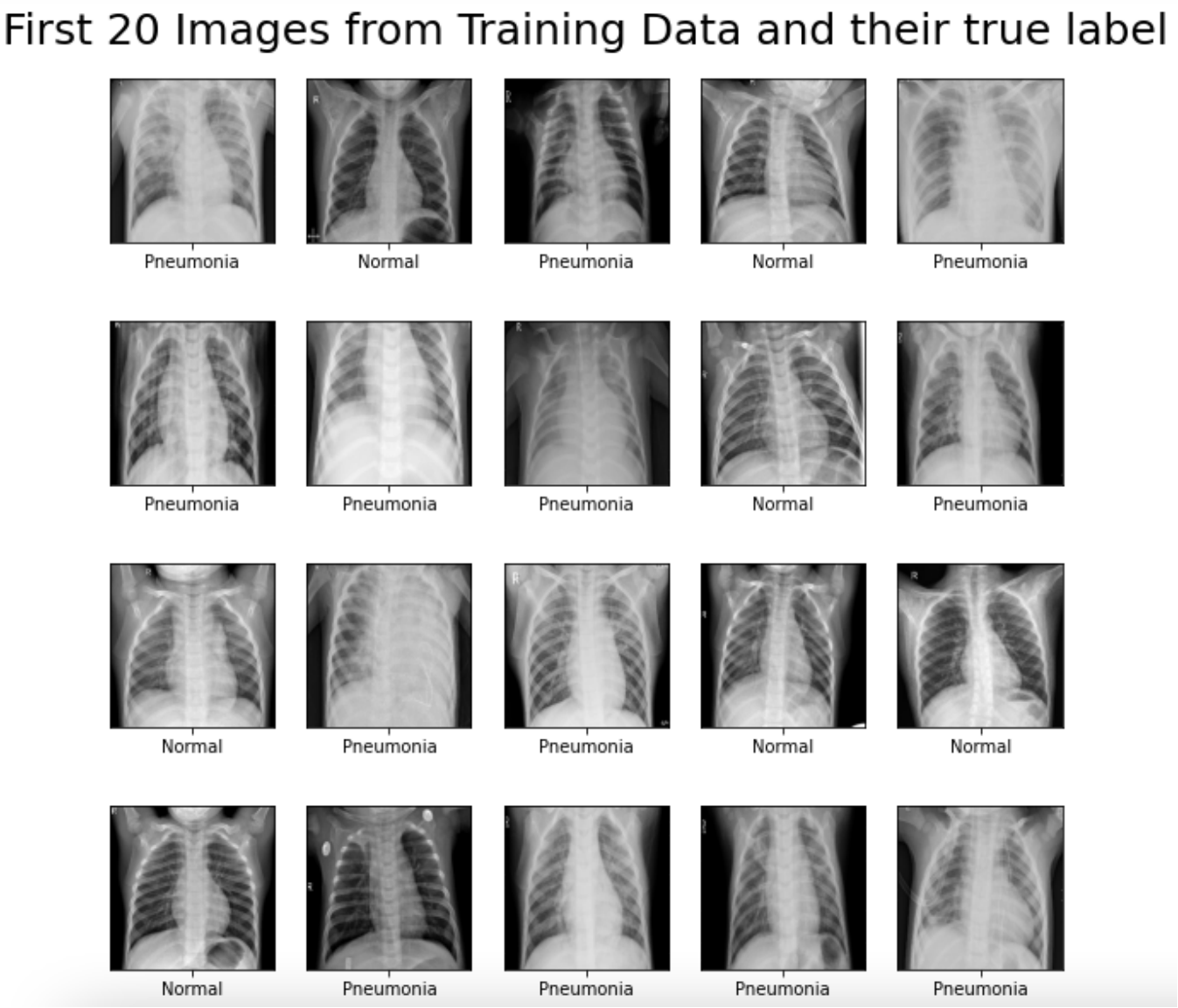
Data Understanding

The dataset comes from Kermany et al. on Mendeley Data. Images are from Guangzhou Women and Children's Medical Center and contains frontal chest X-rays of children and women. This is a large dataset with 5856 total radiographs; 1583 NORMAL and 4273 PNEUMONIA, both bacterial pneumonia and viral pneumonia.

Exploring the Data

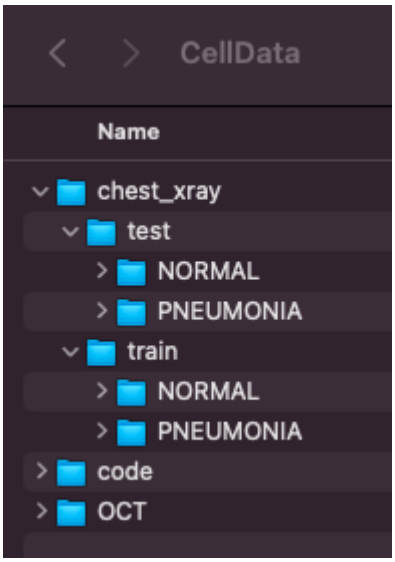
	label	count	percentage
0	NORMAL	1349	25.78%
1	PNEUMONIA	3883	74.22%





Data used for building models

After unzipping file from Mendeley Data there will be a `CellData` folder which contains a `chest_xray` folder. From there you will find the data is divided into a `train` and `test` folders and within these folders the images are seperated into `NORMAL` and `PNEUMONIA` folders.



My downsampled train set will consist of 500 randomly selected images from this `train` folder, 250 images `NOMAL` and 250 `PNEUMONIA`. I will also create a validation set of images not used in my downsampled train set of 25 `NORMAL` and 25 `PNEUMONIA`.

MODEL ARCHITECTURE

OPTIMIZERS:

SGD: "...recent studies show that Adam often leads to worse generalization performance than SGD for training deep neural networks on image classification tasks" Adam vs. SGD: Closing the generalization gap on image classification.

ACTIVATION FUNCTION:

For hidden layers: ReLU, This is the standard activation function for hidden layers. There is no data to suggest deviating from this norm would be beneficial to this image classification task.

For output neron: Sigmoid, because this is a binary classification we can use sigmoid function to give probability of a given image being pneumonia. These probability predictions range from 0-1 making it easy to convert to percentages if so desired.

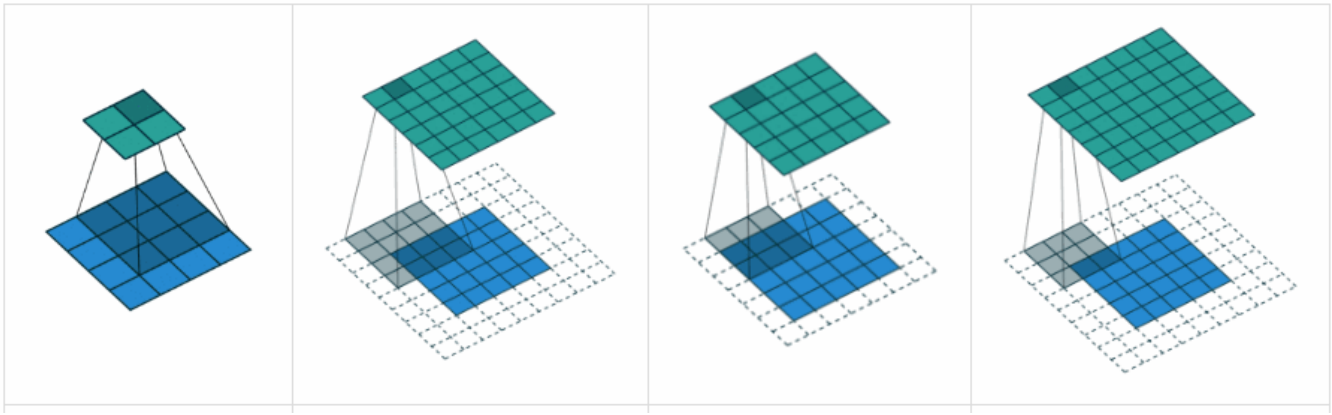
PADDING:

For convolutional layers I will set padding equal to 'same'. This means there is one layer padding with blank pixels and the resulting pixels is the same size as the input image. It ensures that the filter is applied to all the elements of the input. Conversely, when padding is set to equal 'valid' there can be a loss of information, generally, elements on the right and the bottom of the image tend to be ignored. See below graphic from this article:

padding='valid' is the first figure. The filter window stays inside the image.

padding='same' is the third figure. The output is the same size.

Visualization credits: vdumoulin@GitHub



LOSS:

binary_crossentropy: This is a binary classification problem

METRICS:

accuracy: I am using accuracy because I want a reliable model that does a good job at accurately labeling cases. However, false negatives are a much bigger issue that false positives so in addition to accuracy I will be focusing on pneumonia recall score from the classification report as a secondary metric for guaging model performance. And more specifically, I will be focusing on the difference between training recall scores and testing recall scores, they should be as similar as possible to show that the final model will reliably perform at the same level on new unseen data/cases as it did with the training data.

MAXIMUM POOLING

Calculate the maximum value for each patch of the feature map. The result of using a pooling layer and creating down sampled or pooled feature maps is a summarized version of the features detected in the input A Gentle Introduction to Pooling Layers for Convolutional Neural Networks.

L2 KERNAL REGULARIZATION

l2 regularization uses a lambda coefficient of .005.

Performing L2 regularization encourages the weight values towards zero (but not exactly zero). Smaller weights reduce the impact of the hidden neurons. In that case, those hidden neurons become neglectable and the overall complexity of the neural network gets reduced, less complex models typically avoid modeling noise in the data, and therefore, there is no overfitting.

Choosing the right lambda coeffient value:

- If value is too high you run the risk of underfitting your data. Your model won't learn enough about the training data to make useful predictions.
- If value is too low you run the risk of overfitting your data. Your model will learn too much about the particularities of the training data, and won't be able to generalize to new data.

DROPOUT

During dropout, some neurons get deactivated with a random probability P to reduce model complexity resulting in less overfitting. I am using P=.03

Models

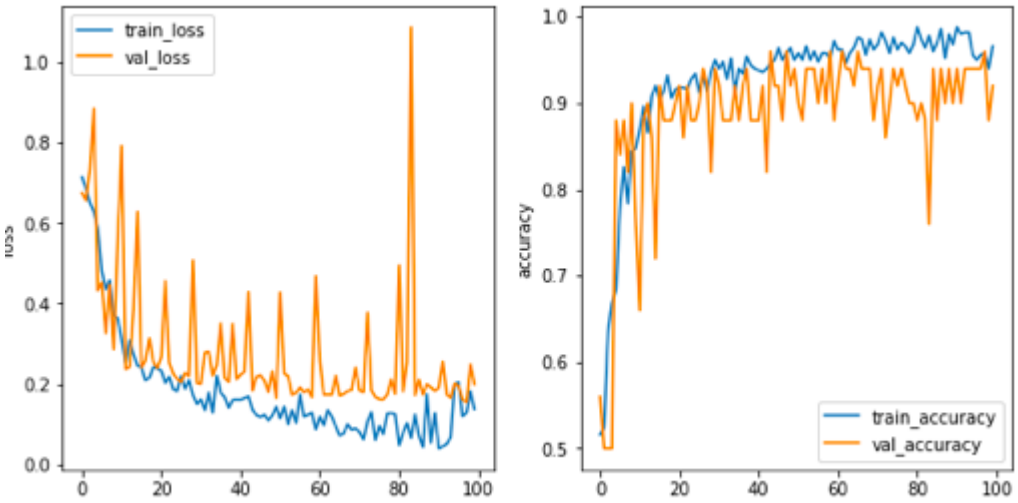
base_model

Baseline fully connected model without convolutional layers:

base_model.summary()

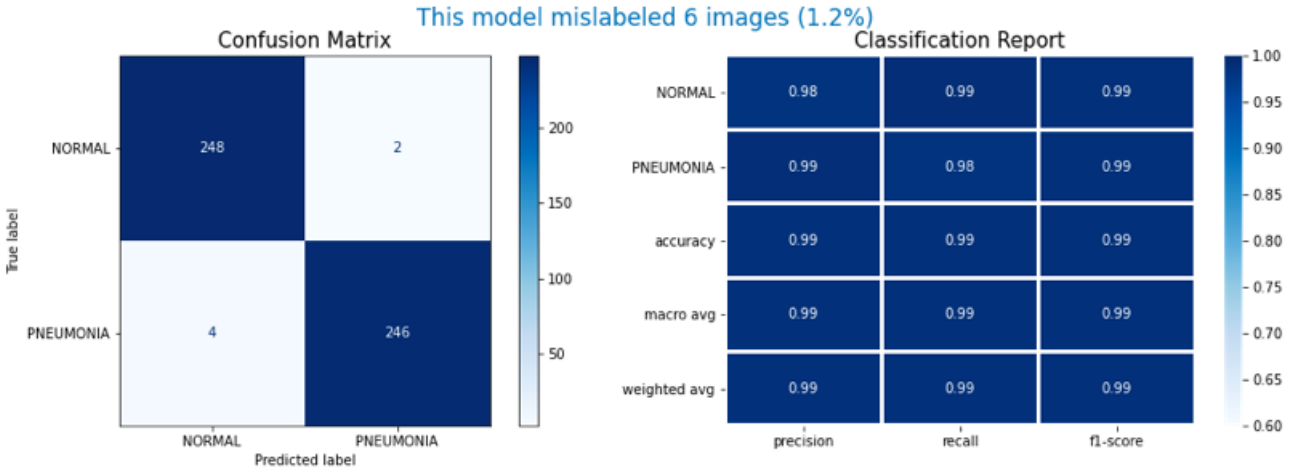
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               6553700
_________________________________________________________________
dense_1 (Dense)              (None, 68)                6868
_________________________________________________________________
dense_2 (Dense)              (None, 30)                2070
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 31
=================================================================
Total params: 6,562,669
Trainable params: 6,562,669
Non-trainable params: 0
_________________________________________________________________
```

`base_model` Loss and Accuracy across epochs

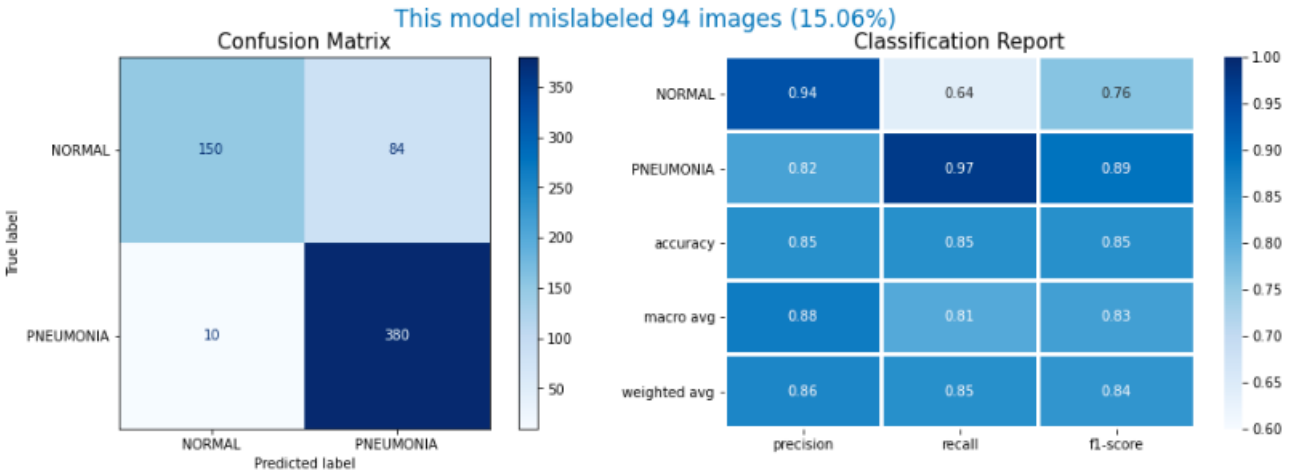


`base_model` Confusion Matrices and Classification Reports

`base_model` Train



`base_model` Test



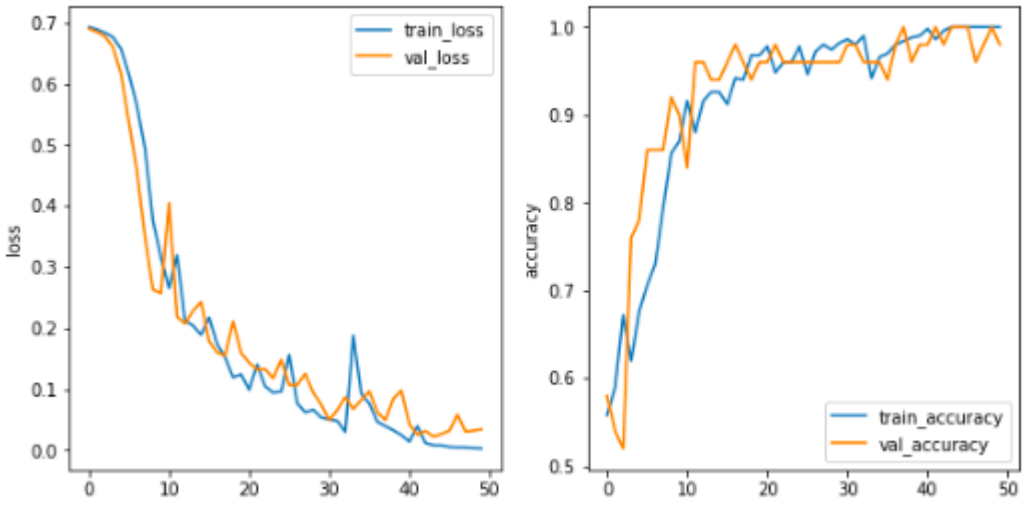
base_cnn

Add convolutional layers to `base_model`

base_cnn.summary()

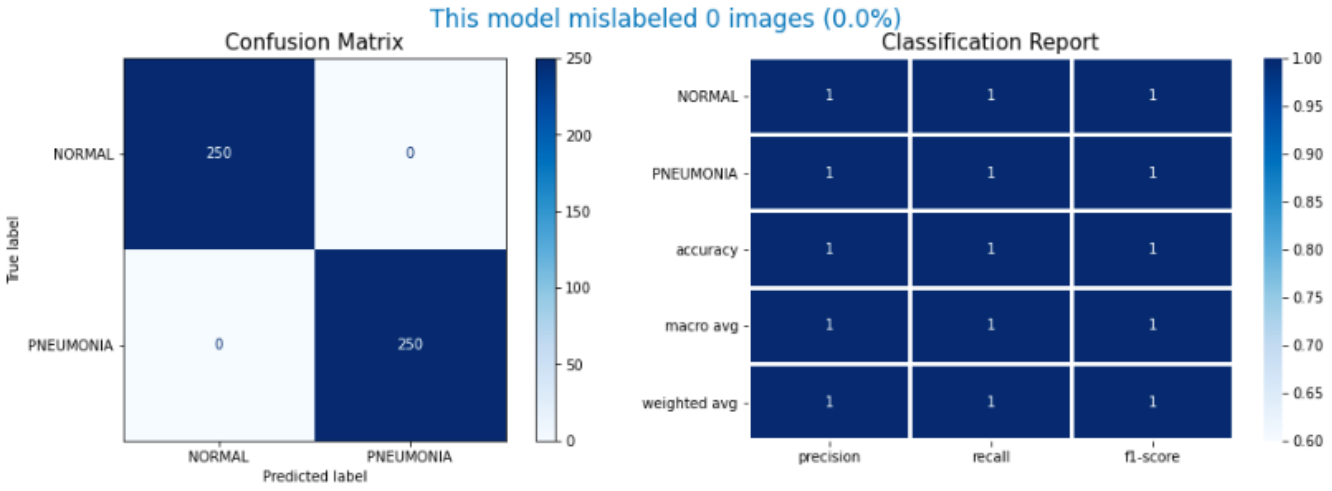
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 100)     1000
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 100)     0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 80)      72080
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 80)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 70)        50470
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 70)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        40384
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               1638500
_________________________________________________________________
dense_5 (Dense)              (None, 68)                6868
_________________________________________________________________
dense_6 (Dense)              (None, 30)                2070
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 31
=================================================================
Total params: 1,811,403
Trainable params: 1,811,403
Non-trainable params: 0
_________________________________________________________________
```

base_cnn Loss and Accuracy across epochs

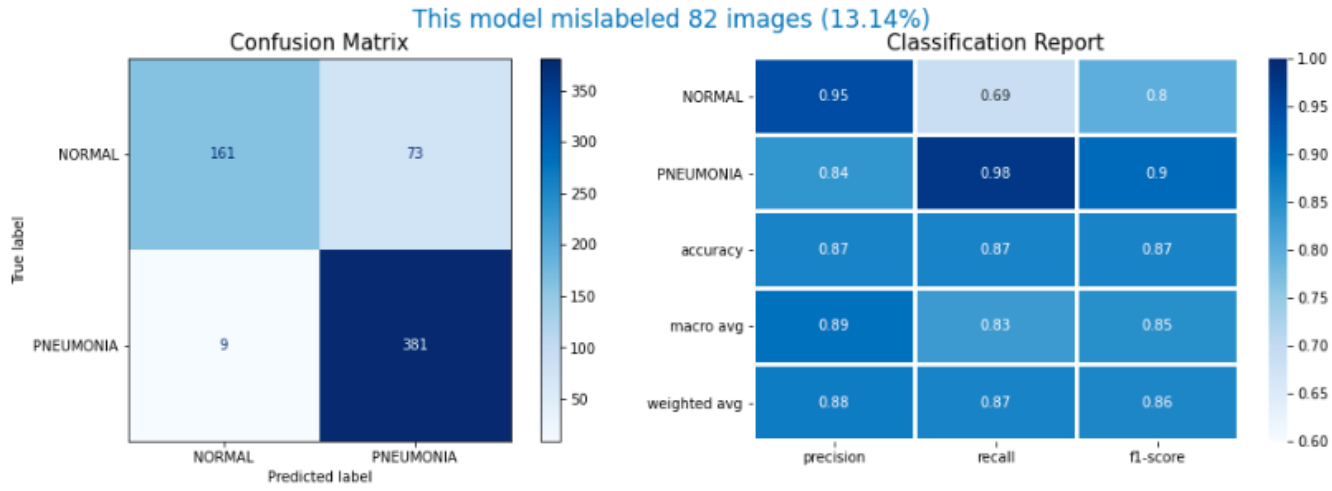


`base_cnn` Confusion Matrices and Classification Reports

`base_cnn` Train



`base_cnn` Test



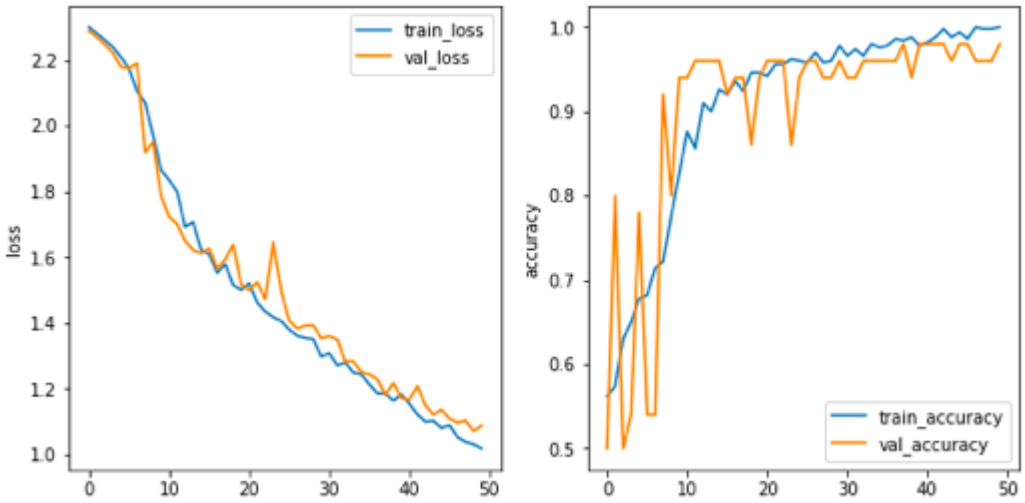
`reg_cnn`

Add l2 kernel regularization to `base_cnn` model

`reg_cnn.summary()`

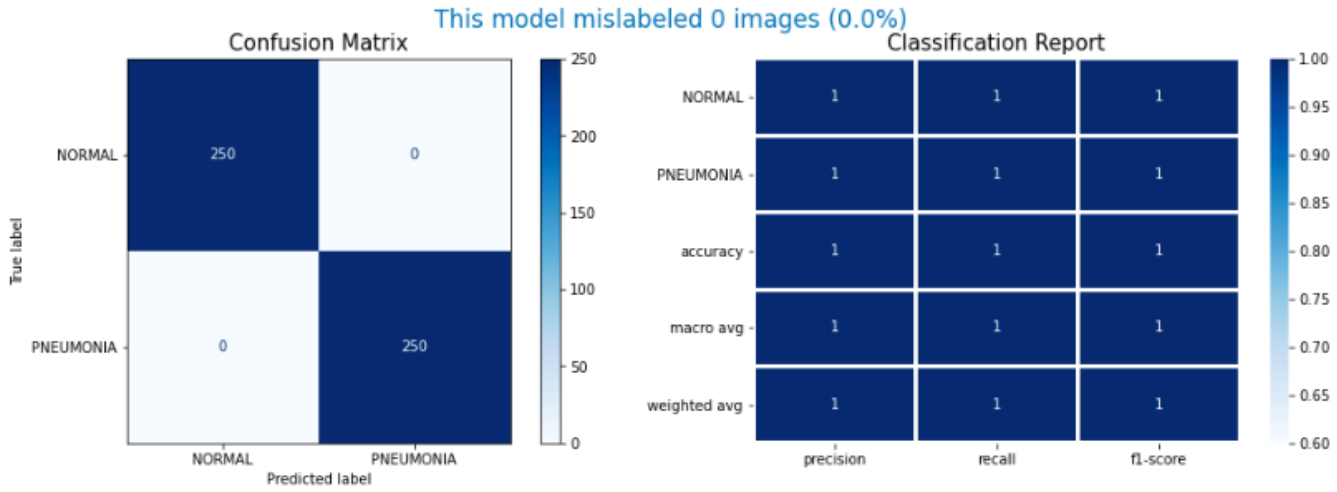
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 256, 256, 100)     1000
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 128, 128, 100)     0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 128, 128, 80)      72080
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 64, 80)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 70)        50470
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 32, 32, 70)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 64)        40384
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_8 (Dense)              (None, 100)               1638500
_________________________________________________________________
dense_9 (Dense)              (None, 68)                6868
_________________________________________________________________
dense_10 (Dense)             (None, 30)                2070
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 31
=================================================================
Total params: 1,811,403
Trainable params: 1,811,403
Non-trainable params: 0
_________________________________________________________________
```

`reg_cnn` Loss and Accuracy across epochs

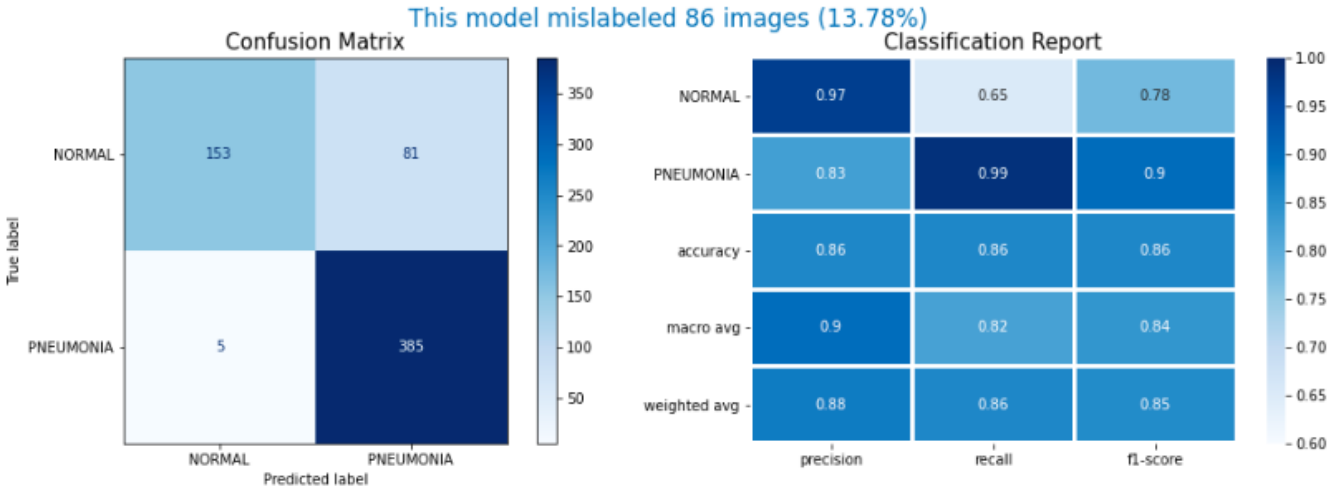


`reg_cnn` Confusion Matrices and Classification Reports

`reg_cnn` Train



`reg_cnn` Test



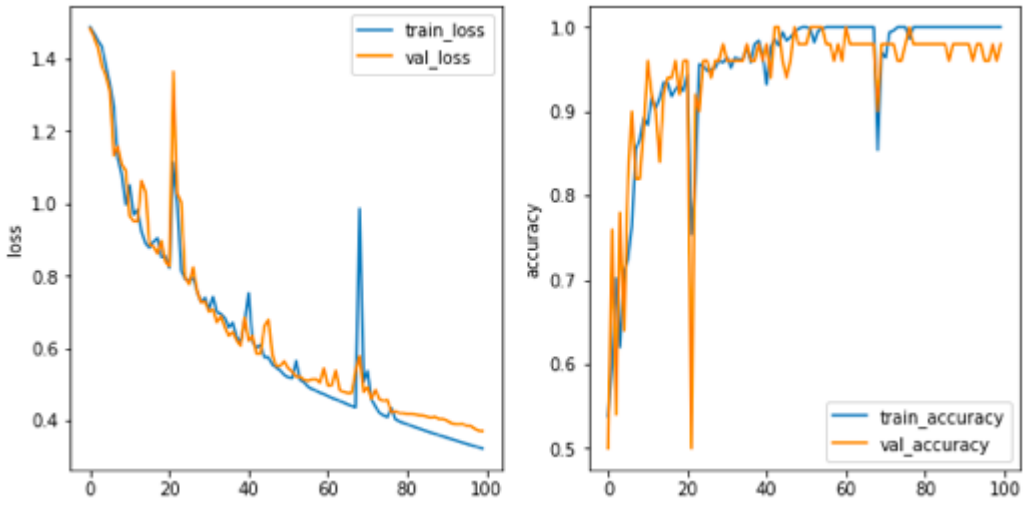
`reduced_nodes`

Reduce number of nodes in each layer by half

`reduced_nodes.summary()`

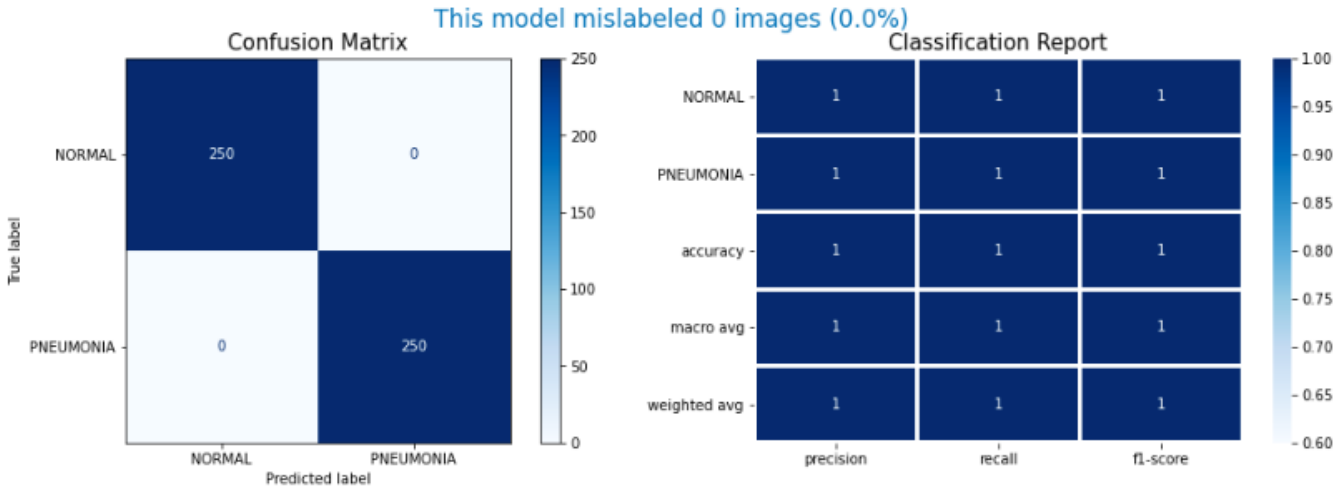
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 256, 256, 50)      500
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 128, 128, 50)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 128, 128, 40)      18040
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 64, 64, 40)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 64, 35)        12635
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 32, 32, 35)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 32, 32)        10112
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_12 (Dense)             (None, 50)                409650
_________________________________________________________________
dense_13 (Dense)             (None, 34)                1734
_________________________________________________________________
dense_14 (Dense)             (None, 15)                525
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 16
=================================================================
Total params: 453,212
Trainable params: 453,212
Non-trainable params: 0
_________________________________________________________________
```

`reduced_nodes` Loss and Accuracy across epochs

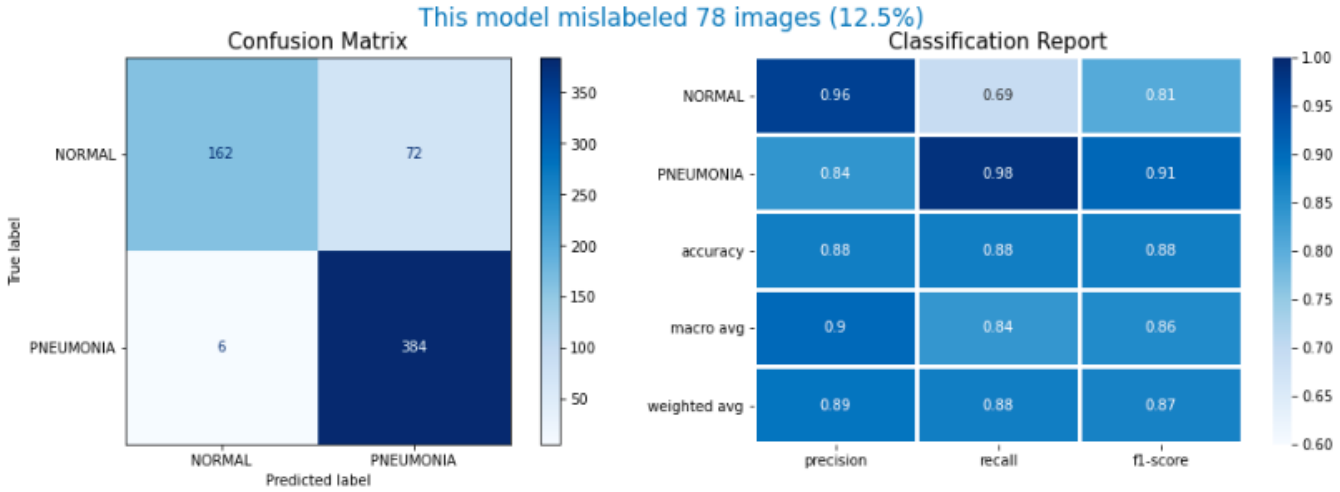


`reduced_nodes` Confusion Matrices and Classification Reports

`reduced_nodes` Train



`reduced_nodes` Test



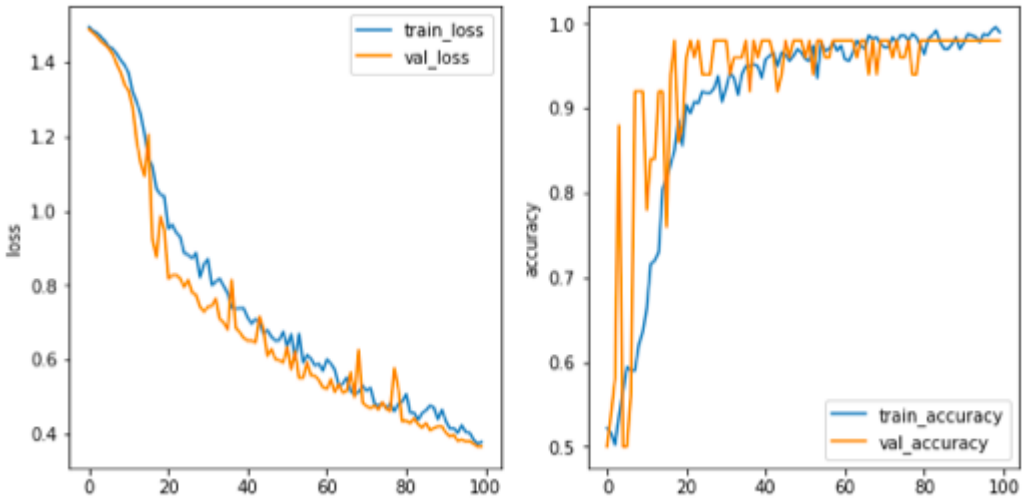
dropout

Add dropout to `reduced_nodes` model

`dropout.summary()`

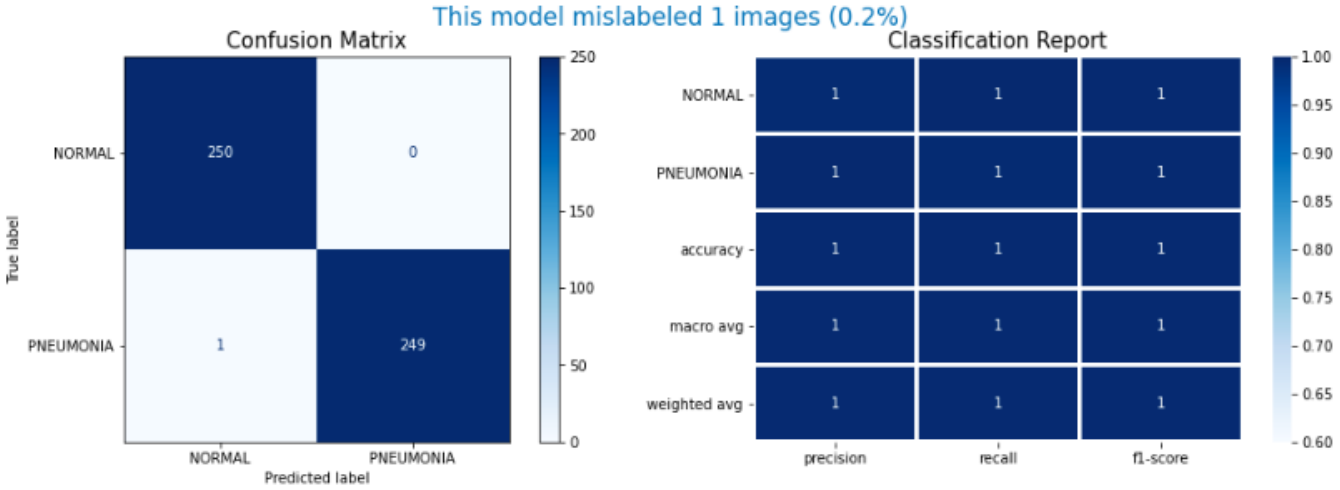
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 256, 256, 50)      500
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 128, 128, 50)      0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 128, 128, 40)      18040
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 64, 64, 40)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 64, 64, 35)        12635
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 32, 32, 35)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 32, 32, 32)        10112
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 8192)              0
_________________________________________________________________
dropout (Dropout)            (None, 8192)              0
_________________________________________________________________
dense_16 (Dense)             (None, 50)                409650
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_17 (Dense)             (None, 34)                1734
_________________________________________________________________
dropout_2 (Dropout)          (None, 34)                0
_________________________________________________________________
dense_18 (Dense)             (None, 15)                525
_________________________________________________________________
dropout_3 (Dropout)          (None, 15)                0
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 16
=================================================================
Total params: 453,212
Trainable params: 453,212
Non-trainable params: 0
_________________________________________________________________
```

`dropout` Loss and Accuracy across epochs

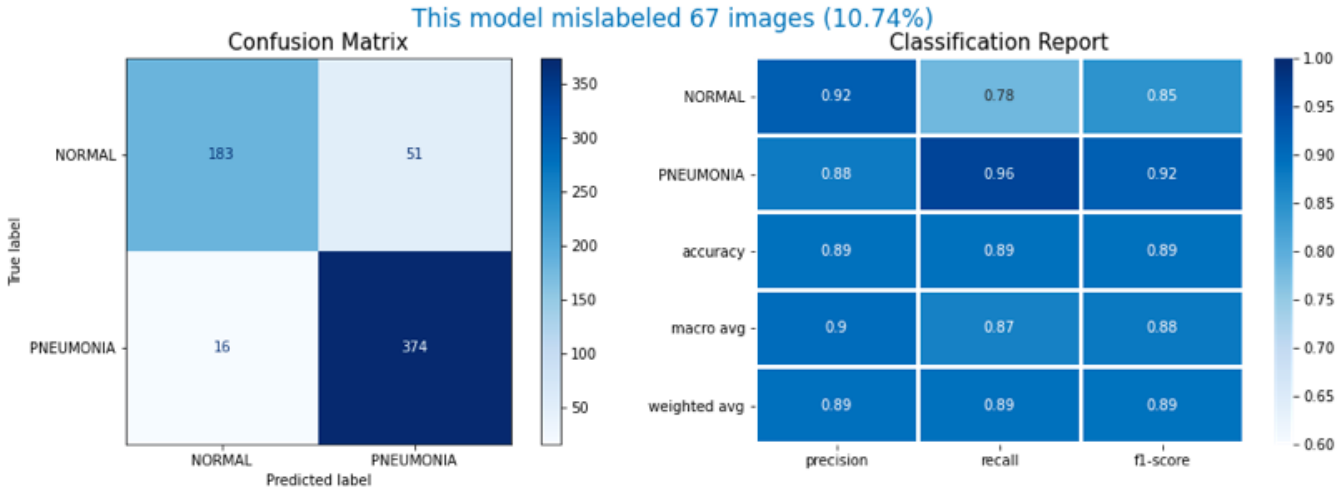


`dropout` Confusion Matrices and Classification Reports

`dropout` Train



`dropout` Test



Compare the models

`compare` df:

	pneumonia_recall	misclassified	accuracy
base_train	0.98	0.010	0.99
base_test	0.97	0.150	0.85
cnn_train	1.00	0.000	1.00
cnn_test	0.98	0.131	0.87
reg_train	1.00	0.000	1.00
reg_test	0.99	0.139	0.86
reduced_train	1.00	0.000	1.00
reduced_test	0.98	0.125	0.88
drop_train	1.00	0.002	1.00
drop_test	0.96	0.107	0.89

`metrics_df` :

	train_loss	train_acc	test_loss	test_acc	loss_diff	acc_diff
base_model	0.040549	0.988	0.598990	0.849359	0.558442	0.138641
base_cnn	0.001954	1.000	0.899908	0.868590	0.897954	0.131410
reg_cnn	1.012247	1.000	1.839380	0.862179	0.827133	0.137821
reduced_nodes	0.321644	1.000	1.059986	0.875000	0.738342	0.125000
dropout	0.348353	0.998	0.967050	0.892628	0.618697	0.105372

My goal for adding dropout was to close the gap between the training and test loss difference and train and test accuracy difference. The `metrics_df` shows that this `dropout` model did infact reduce that gap from the previous model iteration. However, the `compare` df shows that while the accuracy improved with this `dropout` model, the pneumonia recall score for test data dropped. You can also explore this in greater detail by comparing the confusion matrices and classification reports of the train data from the last two models.

At this point I need to move forward with selecting a final model. I can chose the higher accuracy `dropout` model, or the higher recall score of `reduced_nodes` model.

Because the intent of this model is to flag cases of concern, I will go with the higher recall score model, `reduced_nodes`, so that fewer cases are given a false negative label of NORMAL. This may result in increased number of cases prioritized for immediate review from radiologists however, I will capture as many true pneumonia cases as possible which in the end will result in expidited treatment for these patients.

Final Model

For the above mentioned reasons I have chosen my final model to be the `reduced_nodes` model. For the `final_model` I downsampled the pneumonia class to match that of the normal class. This will bring the total number of training images to 2698 with an even split of both classes. I then selected 25 from each class for validation data.

`final_model` Architecture

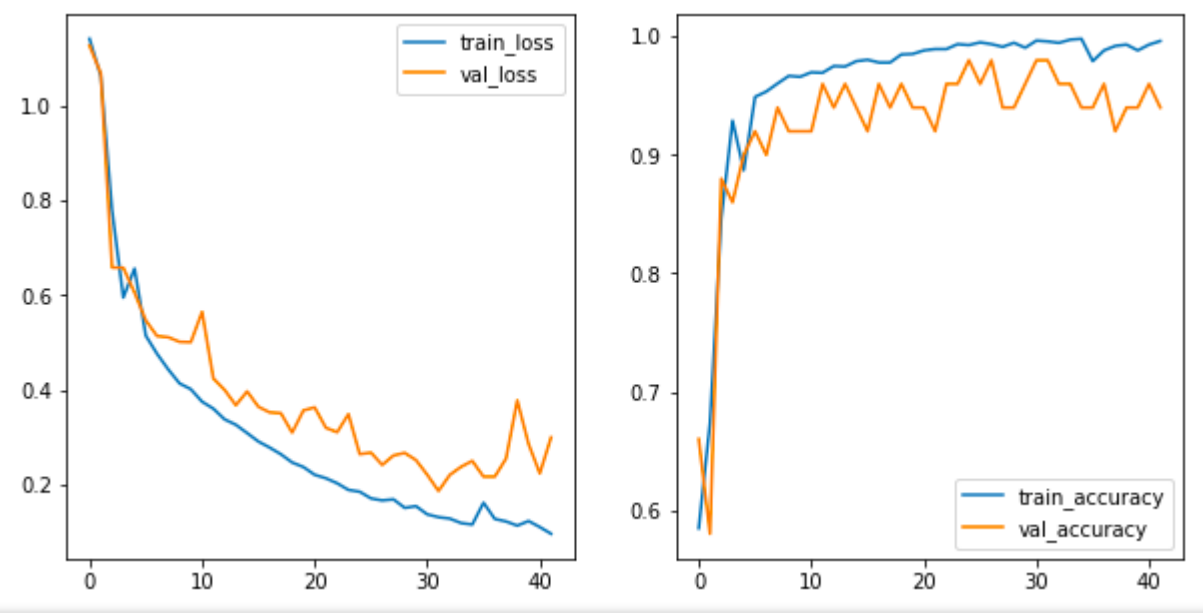
```
final_model = models.Sequential()
final_model.add(layers.Conv2D(50, (3, 3), padding='same', activation='relu', input_shape=(256, 256, 1)))
final_model.add(layers.MaxPooling2D((2, 2)))

final_model.add(layers.Conv2D(40, (3, 3), padding='same', activation='relu'))
final_model.add(layers.MaxPooling2D((2, 2)))

final_model.add(layers.Conv2D(35, (3, 3), padding='same', activation='relu'))
final_model.add(layers.MaxPooling2D((2, 2)))

final_model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
final_model.add(layers.MaxPooling2D((2, 2)))

final_model.add(layers.Flatten())

final_model.add(layers.Dense(32, kernel_regularizer=regularizers.l2(.005), activation='relu'))
final_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(.005),activation='relu'))
final_model.add(layers.Dense(5, kernel_regularizer=regularizers.l2(.005),activation='relu'))
final_model.add(layers.Dense(1, activation='sigmoid'))

final_model.compile(optimizer='SGD',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

final_model.summary()
```

`final_model` Summary

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 256, 256, 50)      500
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 128, 128, 50)      0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 128, 128, 40)      18040
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 64, 64, 40)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 64, 64, 35)        12635
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 32, 32, 35)        0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 32, 32, 32)        10112
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
flatten_6 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_28 (Dense)             (None, 32)                262176
_________________________________________________________________
dense_29 (Dense)             (None, 16)                528
_________________________________________________________________
dense_30 (Dense)             (None, 5)                 85
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 6
=================================================================
Total params: 304,082
Trainable params: 304,082
Non-trainable params: 0
_________________________________________________________________
```

`final_model` Loss and Accuracy across epochs

Plot



NOTES: The early stopping was triggered after 42 epochs. It took approximately 180 seconds per epoch on this machine. It is clear from the above plots that this model is still slightly over trained (take note of the scale of y axis). Next Steps will discuss options to address this.

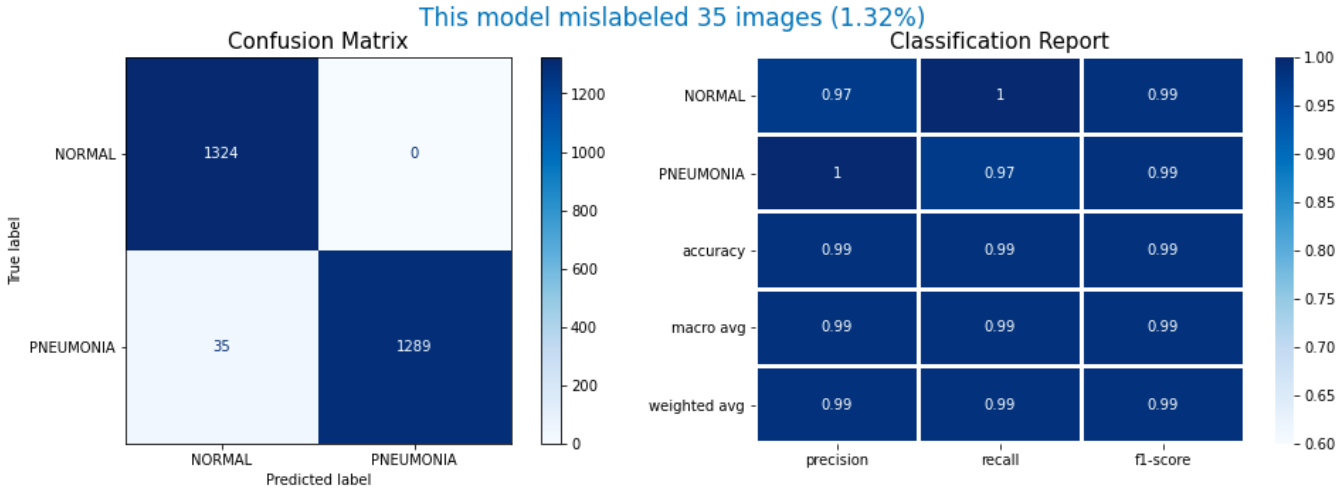
Compare with `compare` df

	pneumonia_recall	misclassified	accuracy
base_train	0.98	0.010	0.99
base_test	0.97	0.150	0.85
cnn_train	1.00	0.000	1.00
cnn_test	0.98	0.131	0.87
reg_train	1.00	0.000	1.00
reg_test	0.99	0.139	0.86
reduced_train	1.00	0.000	1.00
reduced_test	0.98	0.125	0.88
drop_train	1.00	0.002	1.00
drop_test	0.96	0.107	0.89
final_train	0.97	0.013	0.99
final_test	0.97	0.088	0.91

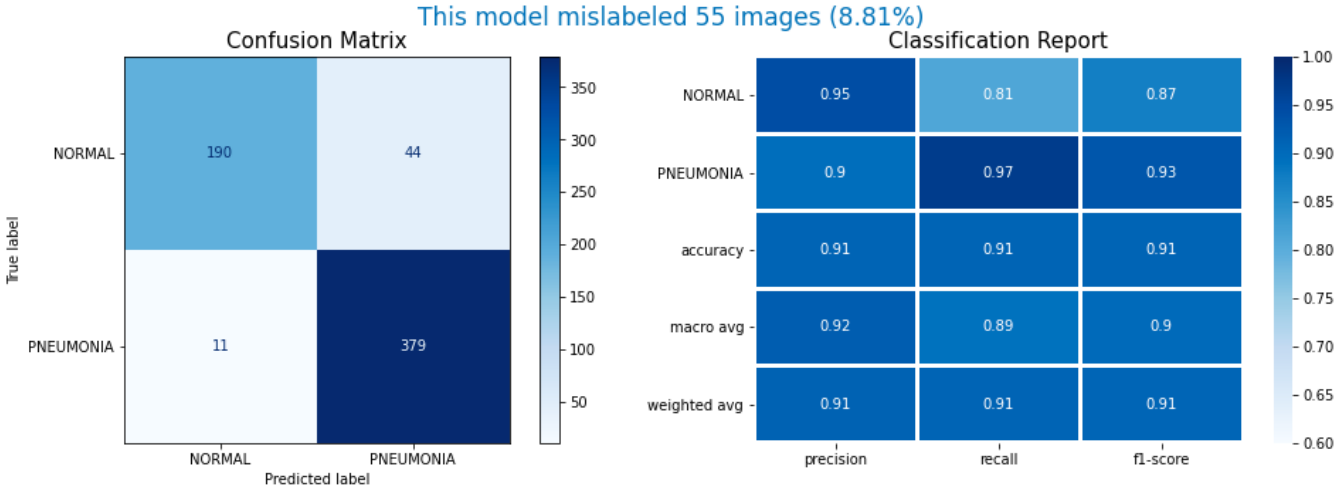
NOTES: There is still a very high accuracy with both train and test data, ~98.7% and ~91.2% respectively. Additionally, the differences between training and test, loss and accuracy are the lowest in this final model. Remember, this is the `reduce_nodes` model and the only difference is the amount of data that has been used to train the model.

`final_model` Confusion Matrices and Classification Reports

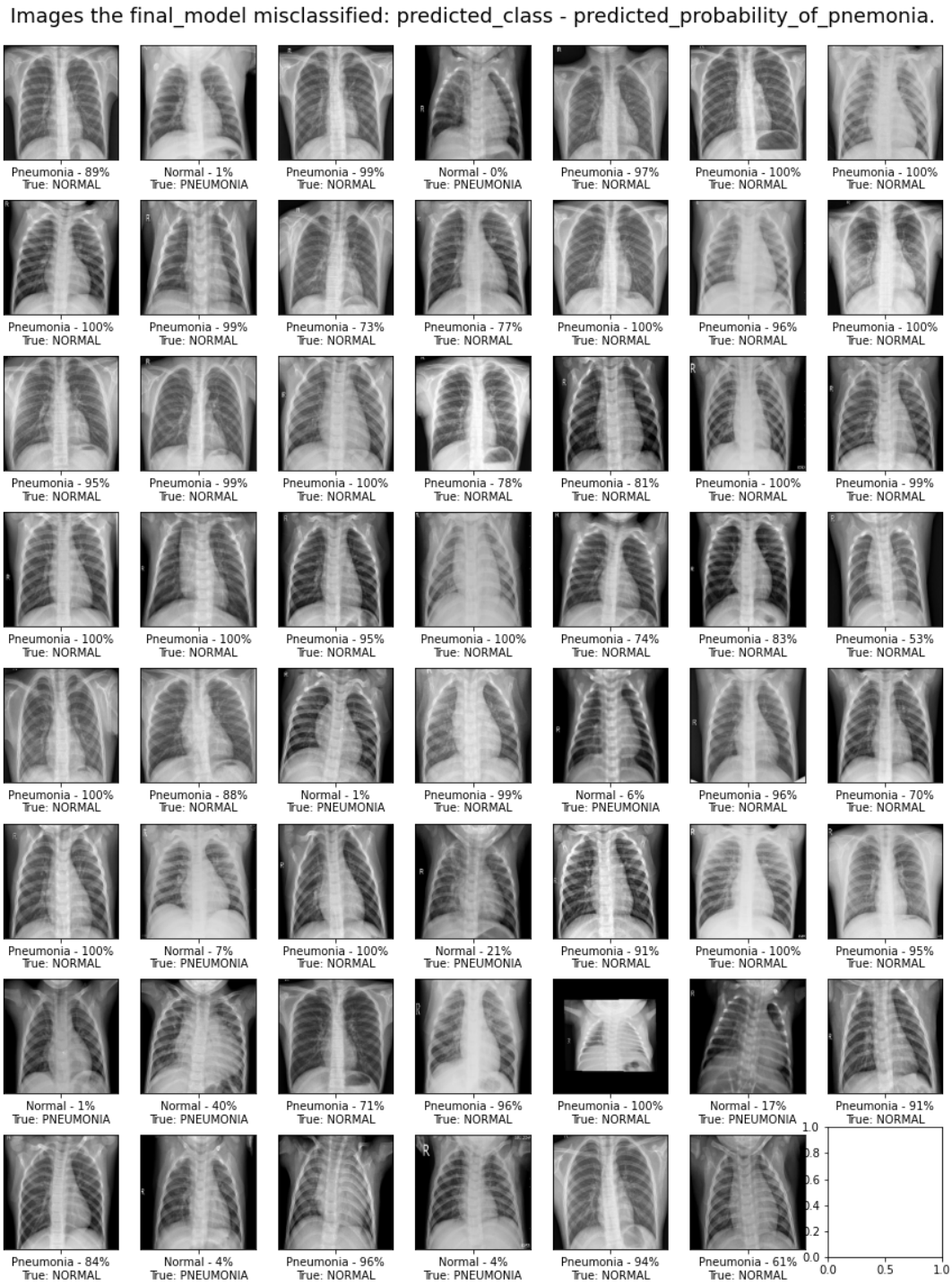
`final_model` Train



`final_model` Test



View misclassified images from testing data



Final evaluation

`metrics_df`

	train_loss	train_acc	test_loss	test_acc	loss_diff	acc_diff
base_model	0.040549	0.988000	0.598990	0.849359	0.558442	0.138641
base_cnn	0.001954	1.000000	0.899908	0.868590	0.897954	0.131410
reg_cnn	1.012247	1.000000	1.839380	0.862179	0.827133	0.137821
reduced_nodes	0.321644	1.000000	1.059986	0.875000	0.738342	0.125000
dropout	0.348353	0.998000	0.967050	0.892628	0.618697	0.105372
final_model	0.113301	0.986782	0.449658	0.911859	0.336356	0.074924

NOTES: From the `compare` df you can see the testing data accuracy scored highest with this model. You can also see in both training and test data pneumonia recall score is the same. This is important because we want to avoid false negatives slipping by without immediate attention called to them. While it is not the highest of all the models it is important they match because we want a reliable model that is outputing similar results on unseen data as it does on testing data. As noted above, from the `metrics_df` you can see there is still a very high accuracy with both train and test data, ~98.7% and ~91.2% respectively. Additionally, the differences between training and test, loss and accuracy are the lowest in this final model. Remember, this is the `reduce_nodes` model and the only difference is the amount of data that has been used to train the model. This demonstartes the potential for this model to improve with more available data to learn with.

Next Steps

While this model has the lowest difference between training and test, loss and accuracy there is still a slight difference. In future iteration, increasing the lambda coefficient in the l2 regularization may further decrease these differences. Additional methods may also be to reduce the kernel nodes even further, or decrease the dropout p value.

As we have seen, the more data available to train the model, the better it performs. One option to increase data, while maintaining equal distribution of classes, is instead of down sampling the pneumonia class to match that of the normal class, we can up sample the normal class using data augmentation to match the number of pneumonia cases. This will result in 2534 more "NORMAL" images and increase the total number of training images from 2698 to 7766.

Recommendations

My recommendation is that GE Healthcare have built into the CCS machines software the ability to not only flag cases of concern for immediate review, but also have the radiologist either confirm or reject the models predictions. This collaborative approach gives further opportunities for the model to learn and improve performance, and also provides a direct actionable way to improve clinical outcomes and elevate patient experience. This could even potentially mean a CCS machine could learn the proclivities of the particular office or medical practice that it is located in.

An additional recommendation would be to push software updates to existing machines, providing ongoing improvements of predictions. This would be particularly useful for small rural locations who don't have a high volume of patients. If they can get updates, and improve, due to the images being collected from clinics in more densely populated urban locations then you are giving every doctor's office around the globe a reason to want this technology for their own practice no matter how big or small they are.

These recommendations will make these CCS units more marketable to a larger range of businesses. The more CCS machines learning from new cases, the better these AI predictions can get. And the better the predictions, the more justification to bring this technology into every medical practice.

Thank You

Let's work together,

- Email: cassigroesbeck@emailplace.com
- GitHub: @AgathaZareth
- LinkedIn: Cassarra Groesbeck