# Project

## CNN for pneumonia categorizing analysis using shap

## Modified Resnet18 rough architechture

**Initial processing** 1 convolutional layer consisting of 64 channels, 1 batch normalization, 1 relu and 1 max pool. Output is 64 channels
Layer 1:
2 basic blocks, each with  2 convolutional layers, 2 batch norms, 2 relus and 1 skip connection
Output is 64 channels
Layer 2:
2 basic blocks with same rough setup as layer 1. Output is increased to 128 channels.
Layer 3:
2 basic blocks with same rough setup as layer 1. Output is increased to 256 channels.
Layer 4:
 basic blocks with same rough setup as layer 1. Output is increased to 512 channels.
Average pooling: Flattens to a 1 x 512 vector. Essentially converting the outputs from the convolutional feature space into a “normal” 512 dimensional neural net layer.
FC: A linear layer with 2 outputs for binary classification

## Modifications from original  to project version of resnet18.

Input layer model.conv1:
Original: Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
    -The network expected 3 RGB channels
Modified: Conv2d(1, 64, kernel_size=7, stride=2, padding=3)
    -Now accepts 1 greyscale channel

No further modifications to input layer

Output layer (fc):
Original: Linear(512, 1000)
    - 1000 ImageNet classes
Modified: Linear(512, 2)
    - binary classification (pneumonia vs. healthy)
Technically fc could have been 512,1 as a binary classifier. But that would have required additional modifications to loss function into binary cross-entropy and sigmoid activation.
This was judged as extra work for little to no gain.

## Data

Pneumoniamnist xrays of lung fields were used.
The dataset is split up into train: 4708, val: 524, test: 624.
Due to unfamiliarity with pytorch code from getting_started.ipynb was copied and ended up with

28x28 lowres images.
This likely had a negative effect on model performance and consequently shap output.
On the other hand it allowed for rapid iteration and focus on core concepts rather than waiting for long training times.
The validation set was not utilized during training, as the initial tutorial code used only train/test splits. This was identified after core project requirements were completed and time constraints prohibited further exploration.

**Class balance**
The dataset contains approximately 2:1 pneumonia to healthy cases (~67% positive). While mildly imbalanced, this probably fine since the imbalance is in the direction of interest.
No class balancing techniques (oversampling, class weights) were applied, as the imbalance is relatively mild and the model achieved strong performance on both classes from the start

# Understanding transfer learning and freezing

Transfer learning is simply taking an existing architecture, weights and biases of an already trained neural network that solves a similar task. Change the input & output shapes and targets.
Run a small n epochs (3-20 depending on domain) on your new inputs.
Freezing can be summarized as selectively locking weights and biases of one or several layers in the pretrained network.

Optimal performance can be reached by itterativley unfreezing and tweaking one layer at a time working backwards. A backward-tweak!

## Best practices according to stanford cs231

Small dataset – similar to original: Freeze all convolutional blocks and train the final layer. Huge overfitting risk.
Large dataset – similar to original: Can probably work through the whole NN. Be empirical and mindful of overfitting.
Small dataset – dissimilar to original: Either just train final layer, train middle layers (maintain edge detection filters in the beginning) or both.
Big dataset – dissimilar to the original: If resources allow: Do a new nn from scratch. Save resources by using a dissimilar pretrained nn anyway. There are pretrained weights available.

It is believed that the problem for this project would fall into “small dataset” category based on the fact that total images were ~5000.
However it was hard to decide what constitutes similar or dissimilar images compared to the original. Imagenet 1k contains classes such as "German shepherd," "tiger," and "car". Certainly, there are edges and other rough shapes in such images. But they are colored and a quick visual inspection of class names does not offer any “X-ray” or “lung”. But in general it seems best to take a conservative approach to hyperparameters due to the dataset size.
Additionally, batch size was considered. But there were no sources on best practices available at a technical level appropriate for a beginner of CNNs or transfer learning.
**Inferring from cs231 logic**:
If a Small dataset and frozen layers means one only adjust a few weights and therefore increase the risk of overfitting. Then one should use a smaller learning rate to counter that. But then the Network

run the risk of getting stuck in a local minima. To avoid getting trapped in local minima with such a low learning rate, smaller batch sizes can be used to add beneficial gradient noise through stochastic gradient descent.

Refs:
https://medium.com/we-talk-data/guide-to-freezing-layers-in-pytorch-best-practices-and-practical-examples-8e644e7a9598
https://www.geeksforgeeks.org/machine-learning/ml-introduction-to-transfer-learning/
https://cs231n.github.io/transfer-learning/
https://github.com/BVLC/caffe/wiki/Model-Zoo
https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/

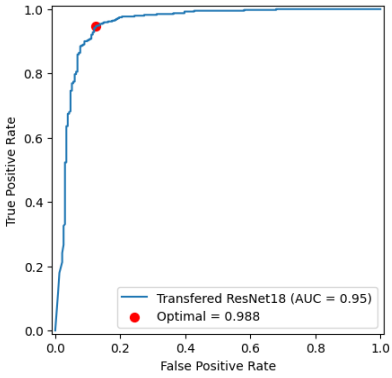
# Untuned network output and shap values

## First set of hyperparameters

Epochs=5 , lr=0.01 , momentum=0.9, batch size = 128
Learning rate, momentum and batch size were getting_started tutorial code so they were simply copied and pasted without any further justification.

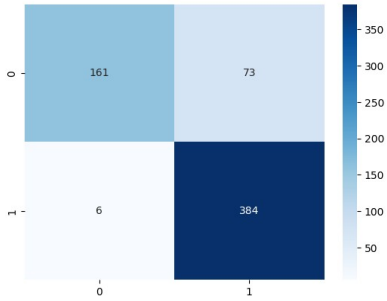
## First set of outputs

Train 0.9813 Test 0.8734 accuracy




ROC curve show good results but room for adjusting threshold if specificity is desired (Optimal = TP-FP).
This seems reasonable given the fact that taking an X-ray of a chest implies a high suspicion of pneumonia from the healthcare professional.

Confusion matrix confirms that the network is high up on the ROC curve with more false positives than false negatives




It is unclear whether this performance is considered good since searching for comparative values showed different metrics such as f1 score.
Human interpretation are fallible, X-rays are only part of the diagnosis criteria and images might contain artifacts.

Similar challenges exist in musculoskeletal imaging, where X-ray findings often with clinical symptoms (e.g., degenerative changes visible on spine X-rays show with back pain presence or severity).

Ref: https://stanfordmlgroup.github.io/projects/chexnet/

## Shap values

Shap images based on task were automatically extracted by indexing all images corresponding to each confusion matrix cell. This to avoid any mistakes in shap visualization should the network tuning make any different classifications of the original 4 images used for the first round of shap visualization.

# Gradient explainer

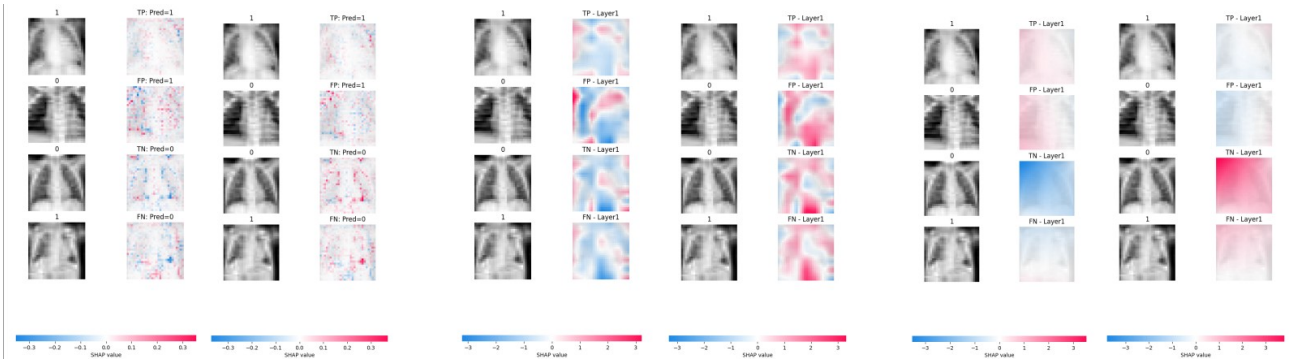
Gradient explainer was run on three different levels (model output, Layer 1, Layer 4) in order to learn more about the library and how the neural network learns.

**Model output:**
Consistent detection of the inside border of the ribs and spinal column on all 4 images.
The False Negative has one confident blob in the incorrect direction in the left lower lung field.
Meaning that the model is looking at the lung but drawing the wrong conclusions in this case.
Surprisingly big pixel spread considering decent accuracy for the first round
There also seem to be wider spread on misclassified predictions
True Negative looks very focused and confident.

**Layer 1 output:**
Layer 1 seem to be very good at extracting patients right side of the thorax. The layer output also seem very confident based on color intensity.

**Layer 4 output:**
Output dimensions are 2x2 downsampling, meaning a 14 fold interpolation had to be used.
It is probably only relevant to inspect color intensity at this level.

Note: Interpretation of Layer 4 output is not straightforward, as course material only covered single-layer CNN mathematics and provided what was likely a vast oversimplification of feature extraction in deep architectures based on the actual complexity of ResNet's architecture.



*In order: model_explainer 1, 0. Layer_1_explainer 1,0. Layer_4_explainer 1,0*

# Freezing process

Starting hyperparameters
batch size = 128/2, Lr= 0.01/3, Momentum =0.9/3, Epochs=5
Original results for reference: Train 0.9813 Test 0.8734
fc unfreeze: train 0.8734, test 0.7837
-Too conservative but smaller overtraining gap.
fc, l4 unfreeze: train 0.9751, test 0.8333
-Too aggressive. Will cut lr and momentum by half and keep freezing
fc, l4 unfreeze lr=0.01/3/2, momentum=0.9/3/2: train 0.9431 test 0.8237
-Much better on overfitting but same diff as updating the whole network

fc, l4, l3: train 0.9743, test 0.8333
-Back to overfitting will do another cut on lr/momentum but at this stage I want to go back to just freezing l4 and fc

fc, l4, l3 lr=0.01/3/2/2, momentum=0.9/3/2/2: Train 0.9480 test 0.8285

-The relative overfitting is consistent. Will backtrack to layer 4 and fc with lower lr along with decreasing batch size and increasing epochs

fc, l4 batch size=128/2/2batch  epochs=10: Train 0.9446, test 0.8301

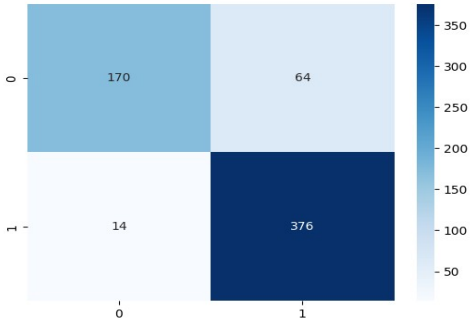
- Several iterations of fc, l4 unfreezes never coming close to original full nn output.

Smallest distance between train and test was simply fc unfrozen.
Rerunning with lower lr=0.0016 and higher epochs =15.
Documentation will be less detailed due to marginal changes in absolute values but all following the same pattern och the network getting stuck in a local minima. Train and test both bounced back and forth.
Increasing and momentum again. Still failing to exit local minima.
Resetting lr and momentum to baseline 0.01 and 0.9 with more epochs. Stuck in another local minima
Results seem to align with the small dataset leading to overfitting risk mentioned in CS231 freezing article.
But unfreezing the deeper layers didn't help enough and the network just ended up in local minimas.
No experiment was better than the default values mentioned at the start.
Reseting and doing some tuning experiments on the whole network
Batch size= 64, lr=0.05, momentum=0.045, epochs=8 ended up with 0.9964 test 0.8750.

-A small gain and a very small increase of the overfitting gap.

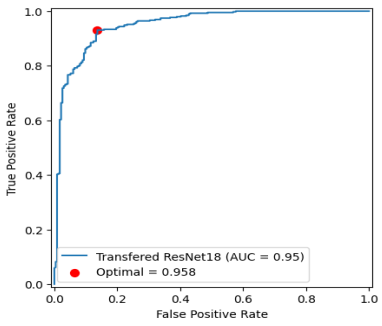
This is probably a dataset size limitation.
A possible solution would be to perturb images but time budget fr this project has already been exceeded by a large magnitude.

# Post tuning output




## Tuned hyperparameters

Batch size= 64, lr=0.05, momentum=0.045, epochs=8
train 0.9964 test 0.8750.
No statistical difference between tuned and untuned network based on chi squared test. P=1
Tuning increased accuracy but changed the sensitivity/specificity trade-off in a clinically unfavorable direction due to decreased sensitivity. Additionally there was a ~2% increase in overfitting.




This is reflected in the ROC "optimal" which is marginally lower than before (0.988 → 0.958)

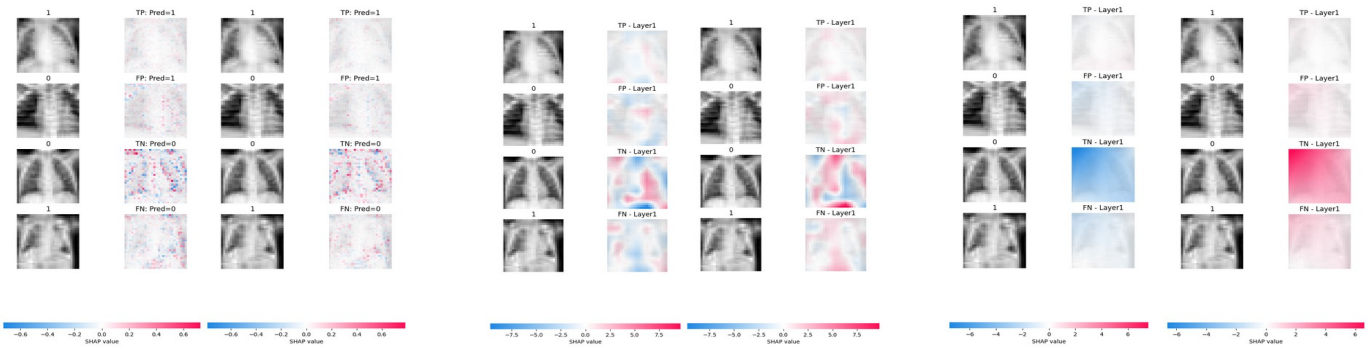
Based on clinical goals and lack of statistical difference it could be argued that original hyperparameters should be kept.
But shap values will be analyzed and compared to the old ones according to project requirements

# Gradient explainer post tuning

It was hard to quantify images compared to classifier or regression shap values but ocular inspection implies the following:
**Model output**:
Color intensity increased for True negative in both positive and negative directions, indicating confusion.
TP, FP and FN all showed decreased color intensity. In the false classification cases, this is a good thing. But in the TP case it is a worse outcome.
Overall, very mixed results.
**Layer 1 output:**
Shapes has made slight changes along with overall weaker color intensity.
This suggest tuning had an effect on early feature detection which led overall decreased certainty of the model.
**Layer 4 output:**
Due to interpolation it is even harder to quantify what happened during tuning for this layer. But the layer seems to have undergone some big changes.
FP case was nudged towards going to the correct TN classification.
But at the cost of getting less confident in FN case as there is decreased pink output in the lower right part of the lung image compared to the untuned model.

As noted earlier, interpreting exactly what the network “looks” at in layer 4 is hard due to downsampling and interpolation. Shap values are derived from a 2x2 matrix where each cell is a summation of 512 feature abstractions and then further scaled and interpolated into 28x28 to fit the image. What the shap plot tells us in the most accurate way is “upper left downsampling matrix had a number contributing correctly towards classifying to 0”.

**Overall interpretation**
It seems the shap output indeed explains the conservative trend from changed model classification. As noted earlier, this was in a clinically unfavorable direction and not a statistically different direction.




# Discussion and conclusion

## Resnet and transfer learning

Transfer learning challenges the notion that models must be trained from scratch for each new task. The most surprising thing was how little data and training was required to get a transfer as compared to what was required for the original weights. The Resnet18 with imagenet weights was trained on 1,281,167 images according to the official page and presumably required a substantial

amount of compute and time to complete.
This makes transfer learning particularly valuable for the healthcare ML field as data is very limited due to privacy concerns and regulations.

Ref: https://image-net.org/download.php

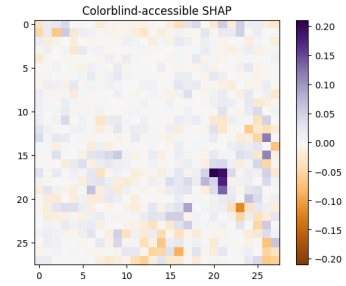
## Shap values

The SHAP values produced mixed and highly nuanced results. This is not surprising given the lack of statistical difference between pre- and post tuning. Perhaps there would have been a more enlightening contrast trying the shap values on a severly underfit network with no transfer learning. Or deliberatly setting hyperparameters extremely low and do a single epoch for transfer learning the first round of analysis.

For a neural network tuning and optimizing, doing gradient explainer with several layers seem like the best strategy. But the single image level of analysis is a weakness.
Deep explainer or partition explainer seem better from that perspective, they also seem better for a stakeholder explanation level since a practitioner could use phrases like “the computer is looking at this part of your lung” without too much anthropomorphization.

One limitation is the lack of ability to quantify model output the same way .force_plot does for regression and classification models. This challenge is exemplified by a typical scenario:
”Err I think this is a deeper red?” tabs back to pre tuned image “Maybe but there are fewer overall pixels in the lung I think. Does that sum up to more or less confidence?”

An additional consideration is color blindness. ~5% of the population is color blind and **red**/green is the most common form to my knowledge. This is a potential issue in clinical settings where either clinician or patients might be color blind and have issues interpreting the shades of grey overlaid ontop of grescale X-rays.
Shap image plotting doesn't seem to integrate with standard Python visualization libraries (matplotlib/seaborn), making it difficult to implement colorblind-friendly palettes.
The image provided is a rough idea for a colorblind version of the shap values alone. It can be displayed along with the original as a bandaid sollution for now.



*Colorblind version of deepexplainers FN output*

**A final note on interpretation.**
Course material did not cover interpretation of spatial attribution methods for images.
Understanding whether attribution patterns represent clinically meaningful features, how to compare across layers, or what constitutes "good" vs "problematic" attention patterns required self-study. However, only base cases and “perfect shap output” examples was found when searching for material on the topic at an appropriate level.
**This made definitive conclusions about model behavior difficult and largely exploratory guesswork rather than following best practices.**

# Appendix:
# Partition and Deep explainers

**Note: Writing here is of lower polish due to being outside project description but was included since writing "for the report" forced me to think more!**

## Partition explainer

Parameters for partition explainer:
batch_size=100
n_evals = 300
masker = "blur 28 28", img.shape

TN also shows more confidence
confirming gradientexplainers results
even if partition is a local explainer vs
Gns global explanation
Surprisingly both explainers are not very confident in
TP predictions even if ROC is fairly high compared to
optima

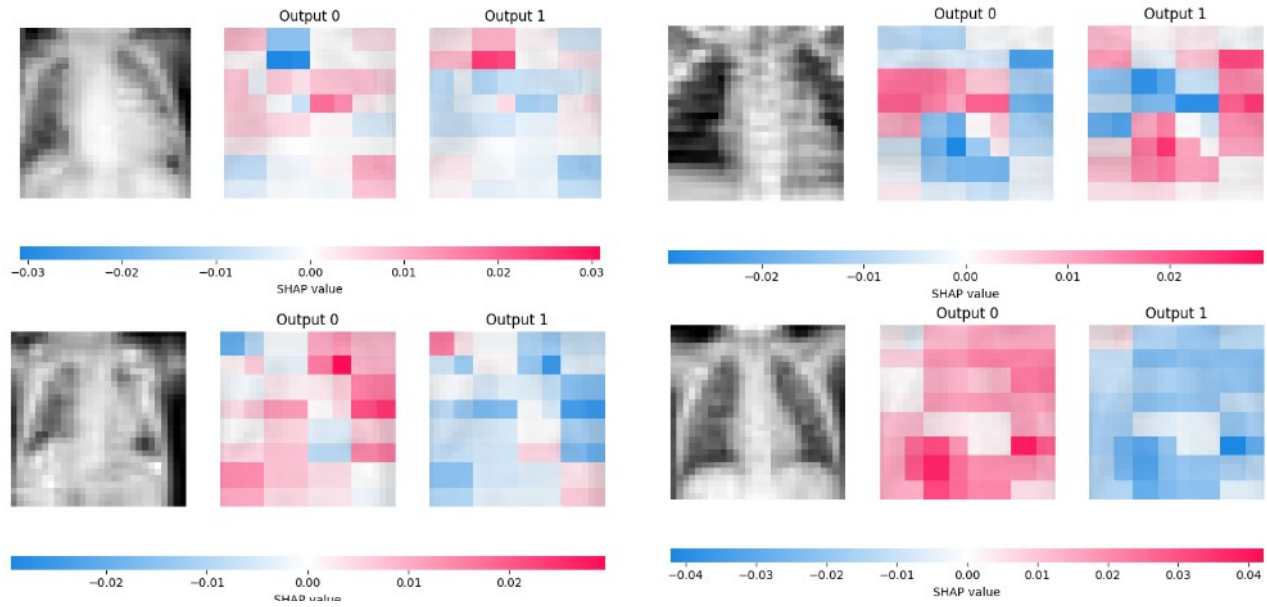



*Figure 1: Row 0 TP FP. Row 1 FN TN*

# Deep explainer

Turns out deepexplaner has had known issues with pytorch since january 2024.
It took me a bunch of digging and a ton of vibecoding before "I" got a working solution.
The main issue was that during backprop, torch does in-place modification of gradients in tensor.
We never had a lecture on tensors. So from what i read up on the topic its more or less NumPy arrays with built in gradients
I never understood the issue until I built the "Big resnet from scratch"
But it took several hours to vibe-solve and even more to understand.

**Depexplainer in a slightly less handwavy nutshell than the 10+ articles I read:**
Approximates  shapvals by measuring activation of each individual neuron
Each matrix or normal layer neuron is treated as a single feature
I think of it as more or less

For layer in model:

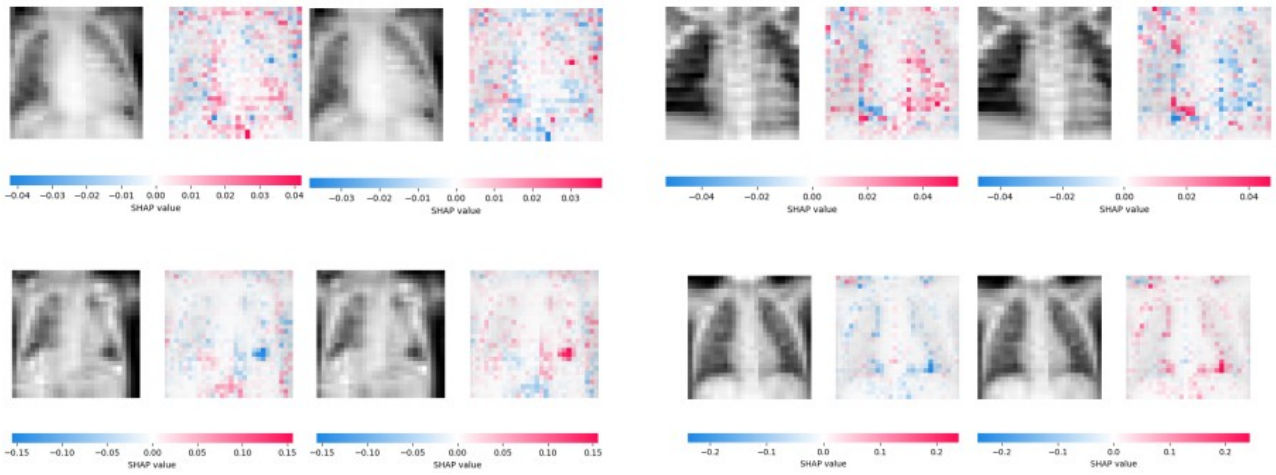
    for sublayer in layer:

        one_gradient_explainer ->down or upsample to 28x28

shapvals = avg gradient_explainers

Do this for all background images to get a baseline
Then rerun with individual predictions and plot difference

Images: pred 1, 0
TP FP
FN TN



# Post tuning

## Partition explainer

Images:

TP FP
FN TN

No change in params for partition explainer

TP: More confident

-good!

FP: Less confident

-also good!

TN: Less confident

-bad

FN: Looks like its using the LH clavicle pixels more than lungs

-real bad!

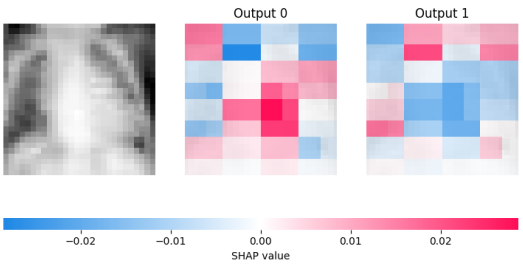
-Tuning  seemingly changed too much of the edge detection

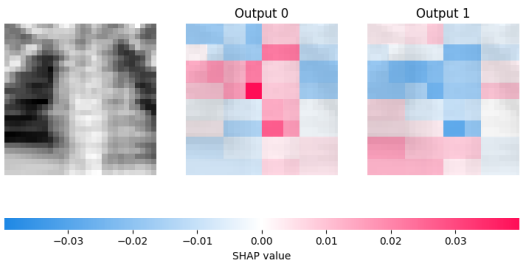
-And probably explains why the model lost sensitivity

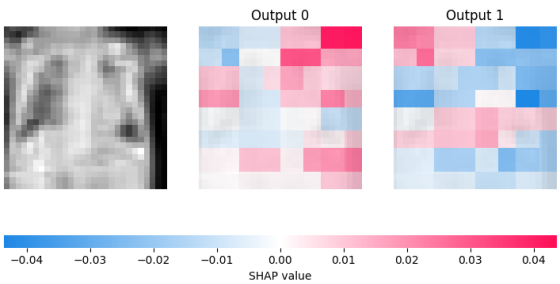
Possible solution: Gradual freezing based on these results.

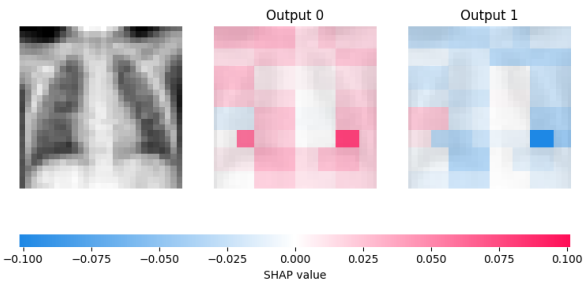
-Starting with 1 or 2 epochs on all layers and then freezing one layer every 2 epochs to avoid changing feature detection too much

-Way outside skill and time constraints



# Deep explainer

Images: pred 1, 0

TP FP

FN TN

TP: Seems overall better at following the lung outline

FP: Slightly more blue on RH lung field so moving in the correct direction

TN: Slightly less confident?

- Hard to quantify red pixels

FN: Slightly less confident

-Pushing towards the correct direction.

Overall follows partition explainer

-but visualized differently