Project

CNN for pneumonia categorizing analysis using shap

CNNs & Pytorch background

- No practical lectures on deep nns above 3-5 layers
- One short lab on pytorch in earlier course
- We will have “Neural networks” course in 1 year
- Course lectures: Basecase 1 layer CNN
  - No mention of channels, transfer learning, freezing or best practices
- Resnet18 : Nested convs, skip connections, batchnorms, pooling

So I built one of my own to understand our task

Sorry that was the wrong image

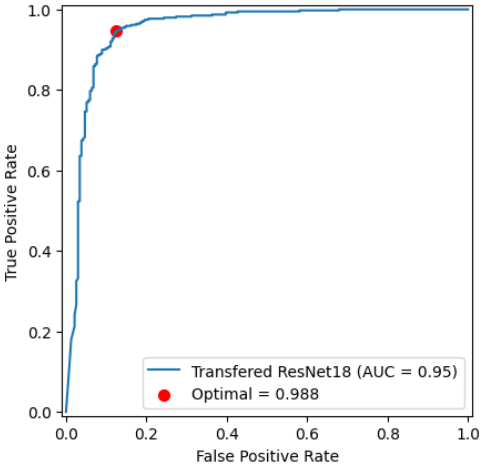
```
class hbblock(nn.Module):
    def __init__(self,in_channels, out_channels, identity_downsample=None, stride=1):
        #connections in, connections out, identity_downsample=we keep dimensions across skip connection stide=1 for convolution steps
        super(hbblock, self).__init__()
        self.expansion=4 #bottleneck block thing. Something about decreasing calculation size and then return to "normal" after a given block
        self.conv1= nn.Conv2d(in_channels, out_channels, kernel_size=1,stride=1, padding=0) #1st convolutional layer, doing normal jumps
        #kernel size= size of conv filter, 1 in this case is for cheap channel reduction
        self.bn1 = nn.BatchNorm2d(out_channels) #z scale output keeping output dimensions
        self.conv2= nn.Conv2d(out_channels, out_channels, kernel_size=3,stride=stride, padding=1) #double out channels to maintain size
        #2nd conv adds some padding and takes in stride argument in case end user wants to tune
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.conv3 = nn.Conv2d(out_channels, out_channels*self.expansion, kernel_size=1, stride=1, padding=0) #output will 4x the size after the bottlene
        self.bn3 = nn.BatchNorm2d(out_channels*self.expansion)
        self.relu = nn.ReLU() #why did we learn of 10+ activation functions in last course when 90% is relu?
        self.identity_downsample = identity_downsample #changes x from skip connection to match block output
```

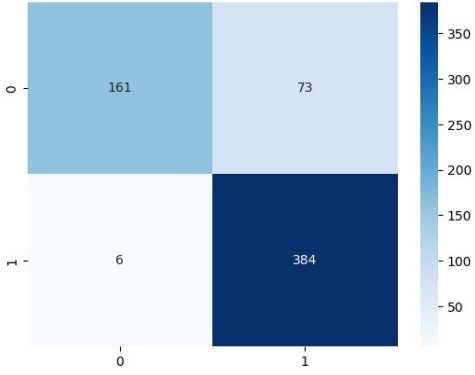
```
    x+= identity #AND THIS IS WHERE SHAP DEEPEXPLAINER BROKE!
    #It wants x = identity + x since that is not an in place modification but im not sure why that breaks shap
    # reason for in place modification is so a new (BIG) tensor isnt created before the old one is deleted.
    # So shap-ing a big nn will be extra intense
    #but I think you can do in place while training and re define forward when its time for shap
    x=self.relu(x)
    return x
```

```
        self.in_channels=out_channels*4 #this will create a *4*4 (256) effect due to self.expansion in hbblock class
        # NOTE at this point I think one can visualize number of channels for a given a CNN with bottleneck layers sortof like a multimodal violin plot

        #building more bottleneck blocks
        for i in range(num_residual_blocks-1): #-1 since one block is already created above
            layers.append(block(self.in_channels, out_channels, identity_downsample=None, stride=1))
            #start of block: inchannels= 256, out=64, end of block: out=64*4=56
        return nn.Sequential(*layers) #unpack the list for pytorch

hbresnet50(img_channels=1, num_classes=2): #guide says img_channels=3 RGB and 1k classes but ill just start with ch=1,classes=2 for pneumoniamnist
return(hbResNet(hbblock, [3,4,6,3],img_channels, num_classes))

#The level of abstraction is actually insane here. Very cool
hbresnet101(img_channels=1, num_classes=2):
return(hbResNet(hbblock,[3,4,23,3],img_channels, num_classes))
```

First untuned run and CM images

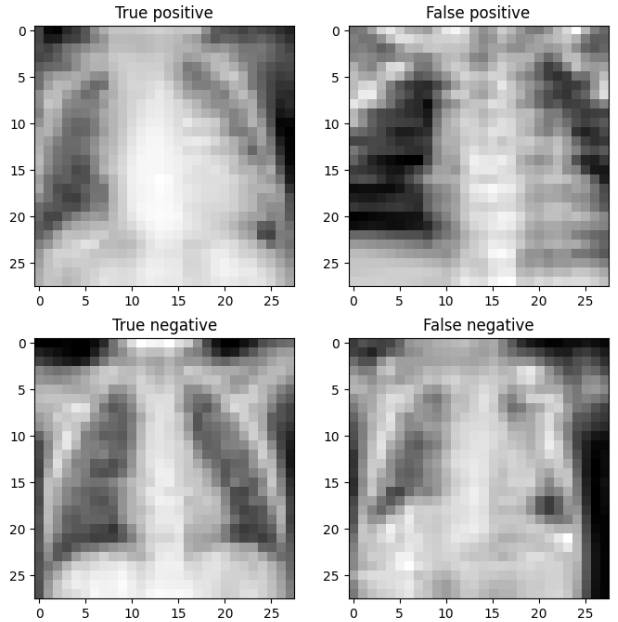
- NUM_EPOCHS=5
  -Small since its pretrained
- lr=0.01
- Momentum=0.9
- BATCH_SIZE = 128
- Train/Test progress shows room for more epochs (assuming no frozen layers):
  - Epoch 1: Train 0.8736 Test 0.7324
  - Epoch 4: Train 0.9539 Test 0.8061
  - **Epoch 5: Train 0.9813 Test 0.8734**
- ROC curve show good results but room for adjusting threshold if we want more specificity (Optimal = TP-FP). This seems reasonable given the fact that a chest Xray implies high suspicion of pneumonia.
- CM confirms that we are indeed high up on the ROC curve.
  -More false positives than false negatives.





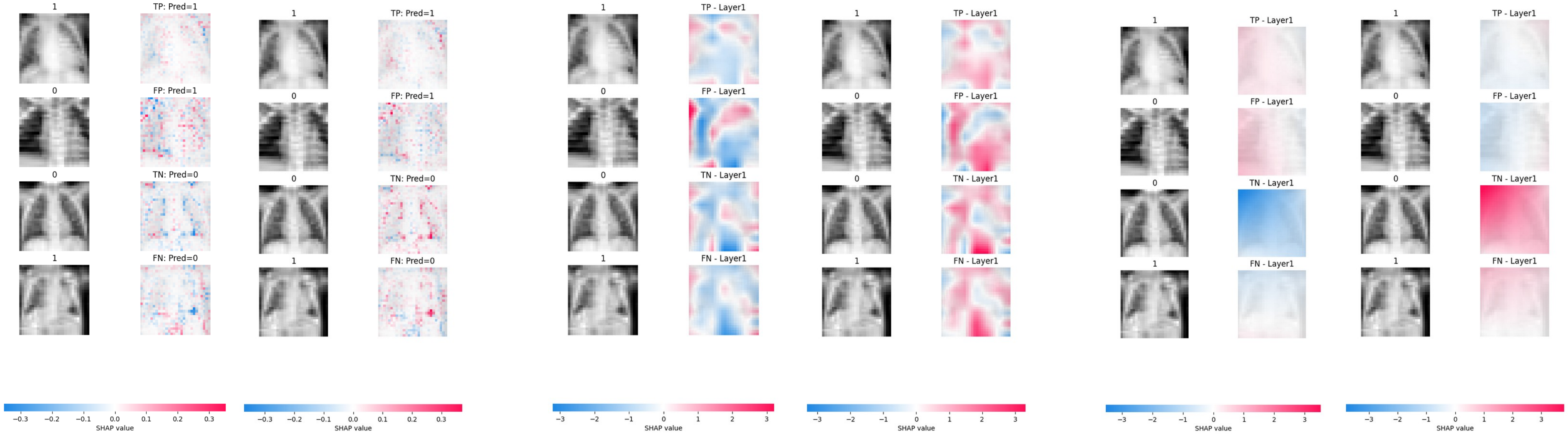
Shap first run

- Automatic “CM” index and image extraction lists
- Meaning we might get different images for shap analysis if the tuned NN classifies the images here.
  -Tuned shap image plots might differ wildly because of this
- TN looks very happy not having Pneumonia!
  -Or possibly doing a snatch



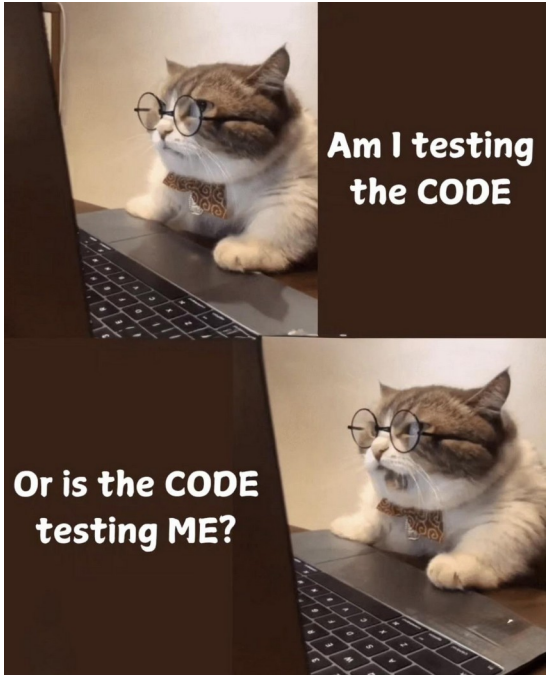
Gradient explainer

- In order: model_explainer 1, 0. Layer_1_explainer 1,0. Layer_4_explainer 1,0
- **Model pixels**. Detects the shapes around the lungs.
    - -The false negative has one wrongly confident blob in the LH bottom of the image.
    - - Big pixel spread considering decent accuracy for the first round
    - - There also seem to be more spread on missclassified predictions
    - - TN seem very confident across the whole NN
- **Layer 1**. You can see the patients right thorax! Probably fine to freeze this
- **Layer 4**. Highly interpolated due to 2x2 downsampling! Only color intensity relevant?



Bonus!

- In the sense that “I” sort of solved these 2 before project description changed from "Partition and deepexplainer" to "Gradient explainer"
- It is also impossible to discuss image classification with less than 1 cat picture (https://arxiv.org/abs/2012.01768)



Partition explainer

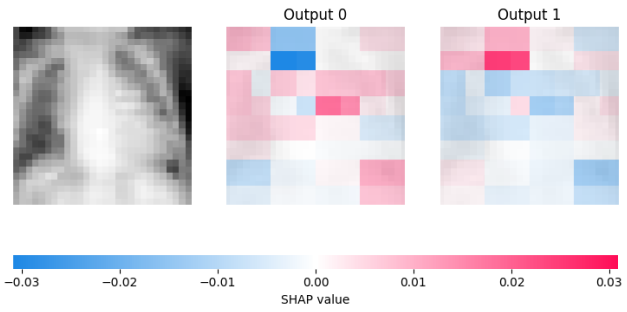
- Grabs one image. Blurs a given pixel or superpixel, see what model does.
  Repeat until we get "marginal (super)pixel contribution" == Img shap values.
  Does not need a "shap training set" the way one needs for a normal regression model for example
  Only does local image analysis

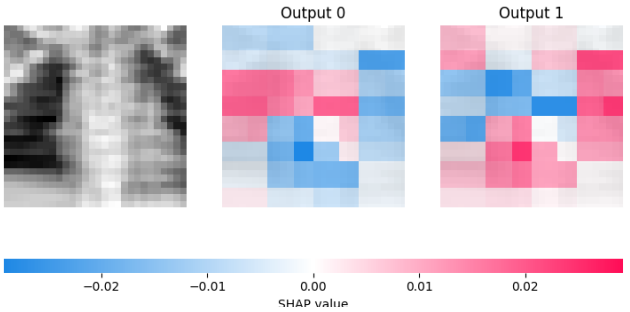
- Params:
  batch_size=100
  n_evals = 300
  masker = “blur 28 28”, img.shape

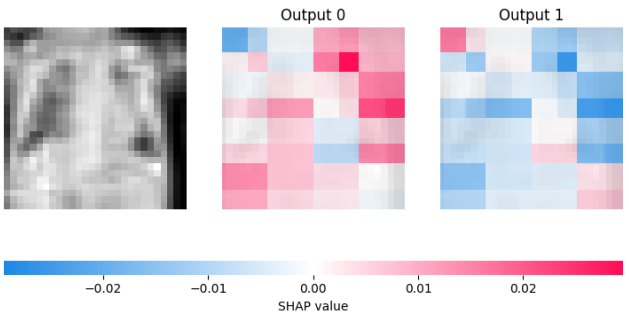
- Images:
  TP FP
  FN TN

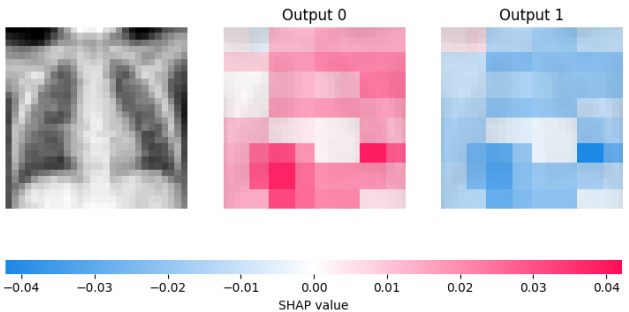
- TN also shows more confidence confirming gradientexplainers results even if partition is a local explainer vs Gns global explanation

- Surprisingly both explainers are not very confident in TP predictions even if ROC is fairly high compared to optima







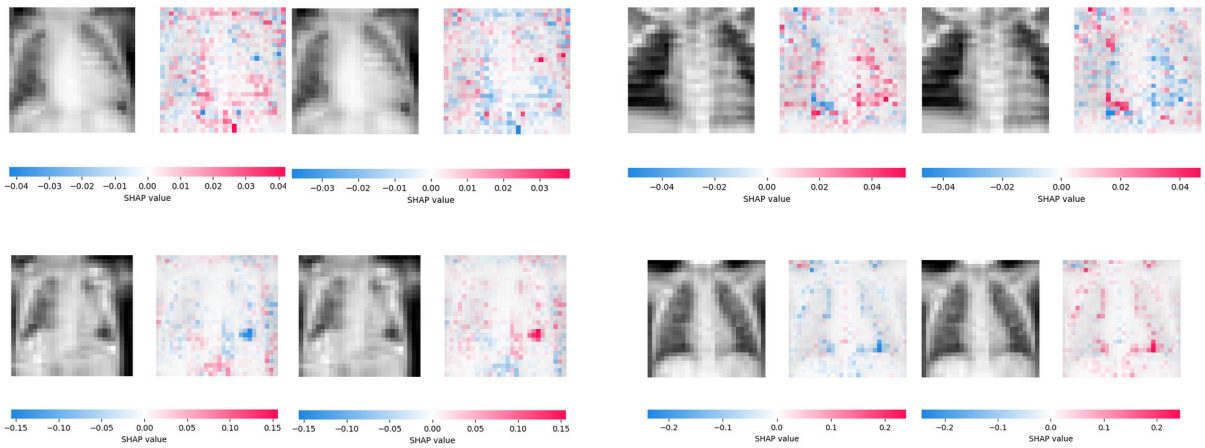


Deepexplainer

- 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. Along with in place modification of relu.
  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

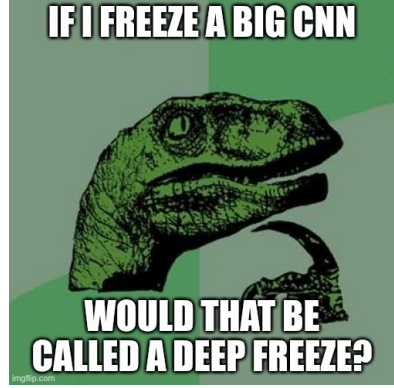
- Images: pred 1, 0
  TP FP
  FN TN



The method name is misleading

- Deepexplainer uses neurons and conv/pool indices to **approximate shap vals**
- It works by comparing each neuron's activation for your image to its activation for a "neutral" baseline image.
  The difference is then calculated as weight * (change from neutral) back through the network's connections to the input pixels.
- Inference: Deepexplainer should get worse as nn gets deeper
- This is almost as bad as skip connections!
  -Functionally they boost the gradient by 1

Freezing and tuning



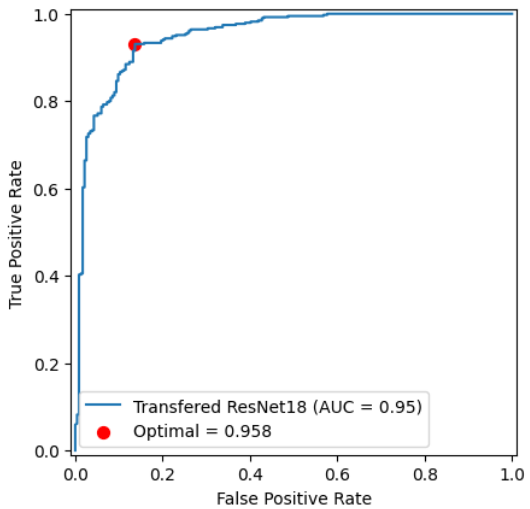
- No prior explanation on this topic so this is mostly inference and guesswork.
  Main ref is stanford cs231 article “4 scenarios”
- ***Small dataset – similar to original: Freeze all conv and train the final layer. Huge overfitting risk.**
- Large dataset – similar to original: Can probably work through the whole NN. Be empirical and minful of overfitting.
- ***Small dataset – dissimilar to original: Either just train final layer, train middle layers (maintain edge detection filters in the beginning) or both**.
  -This falls in line with gradient explainers layer 4 plot. Its only “certain” on TN. The other shaps are not confident.
- Big dataset – dissimilar to the original: If you have money and time: Do a new nn from scratch. Save money by using a dissimilar pretrained nn anyway.
- LR and momentum should generally be smaller than baseline since weights might be too perturbed otherwise.
  -Analogy in my head: We are running the final few epochs and lr_decay=True
- Couldn’t find anything on batch size that is on a level for a 2nd year undergrad to understand.
  Inferences:
  -Small dataset and frozen layers means we only adjust a few weights and increase the risk of overfitting.
  - Use smaller learning rate to counter overfitting. Increased risk of stuck in local minima.
  - Counter local minima by using smaller batches for stochastic GD
- Adjusting epochs last and go right to the overfitting edge
- Will mostly look at acc scores while tuning

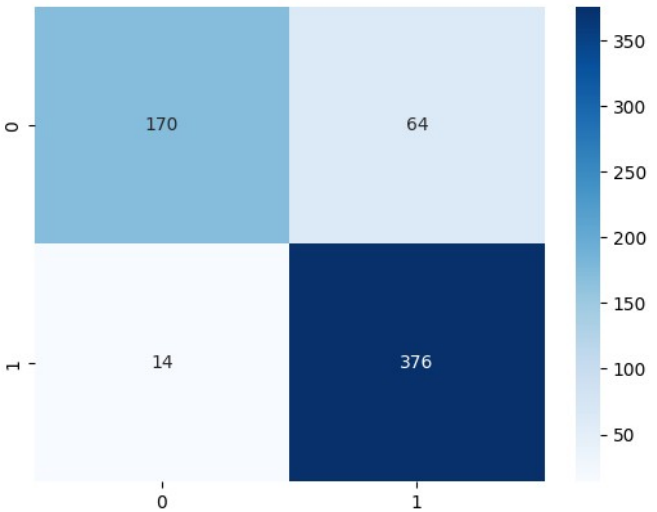
Freezing diary

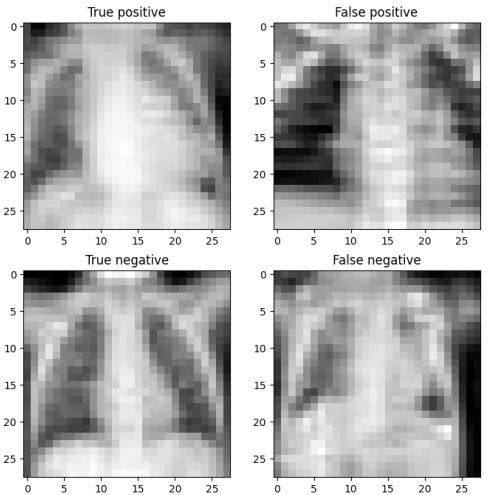
- Starting hyperparameters: Batch_size = 128/2, Lr= 0.01/3, Momentum =0.9/3, Epochs=5
- Original results for reference: Train 0.9813 Test 0.8734
- fc unfreeze: train 0.8734, test 0.7837
  -Too conservative but smaller overtraining gap.
- fc, l4 unfreeze: train 0.9751, test 0.8333
  -Too aggressive Will 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 and always getting stuck in local minima.
  - Train and test both bounced back and forth no matter what tuning combination used.
- Results seem to align with the small dataset leading to overfitting risk mentioned in CS231 freezing article.
- No experiment was better than the default values mentioned at the start.
- 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 has already been exceeded by a large magnitude.**

Changes of outputs from tuning?

- Batch size= 64, lr=0.05, momentum=0.045, epochs=8 ended up with train 0.9964 test 0.8750.
- No change in CM index images
  -TN is still happy!
- No statistical difference! Chi² p-value=1!
- Tuning increased accuracy but changed the sensitivity/specificity tradeoff in a clinically unfavorable direction due to decreased sensitivity
- This is reflected in the ROC “optimal” which is marginally lower than before (0.988→0.958)







Gradient explainer

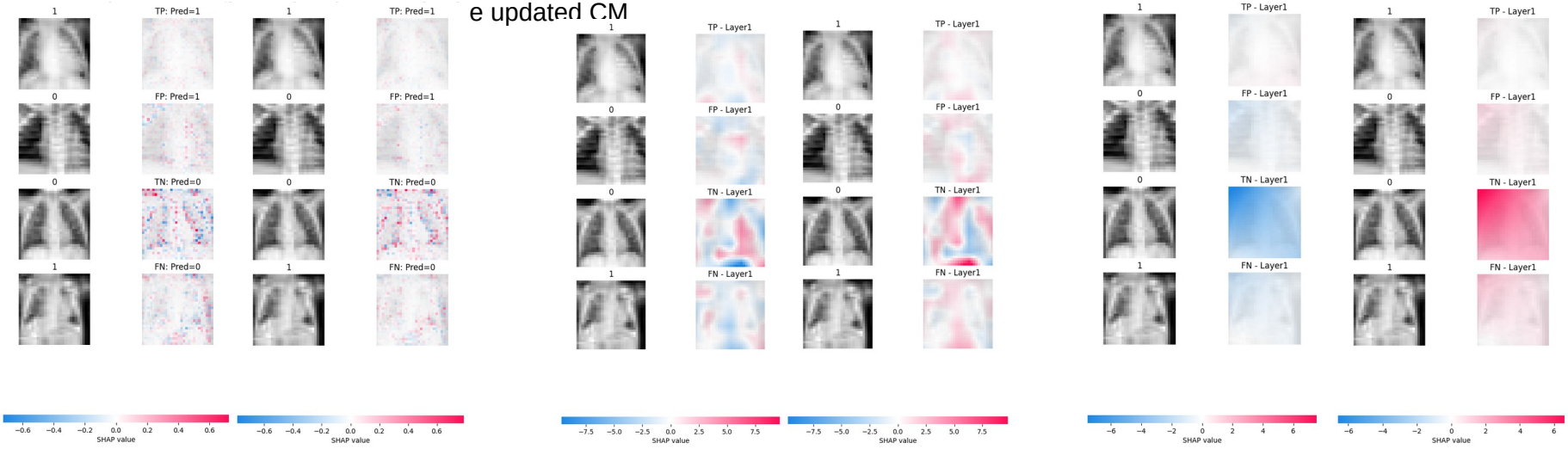
- Hard to quantify images compared to clf or regression shap values but eyeballing implies

- **Model** confusion!
  -Confidence increased for True negative in both directions, indicating confusion.
  -The other CM colors are weaker, showing a decrease in confidence

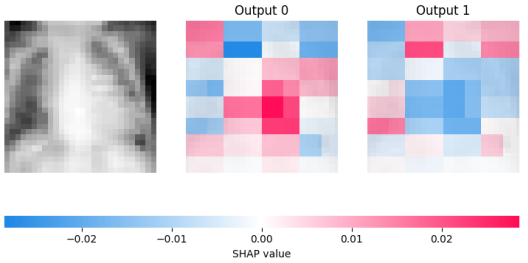
- **Layer 1** has shifted shapes slightly
  - But look overall less confident. Local minima explanation!?

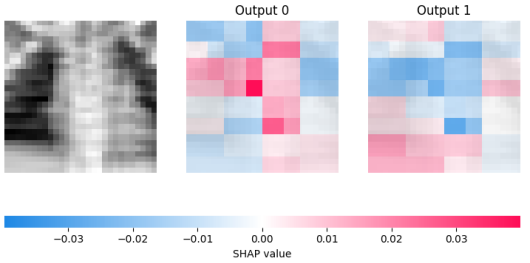
- **Layer 4** though has done some major updating!
  -Pushing FP image towards going to the correct TN classification.
  -But at the cost of getting less confident in False negative (lost some pink in the bottom)

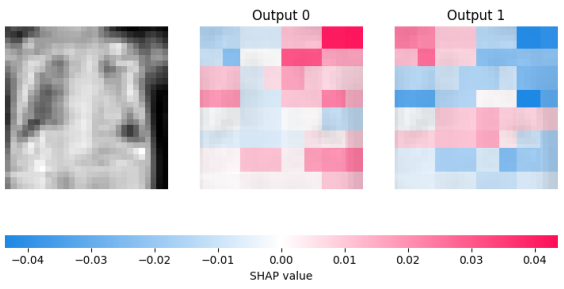
e updated CM

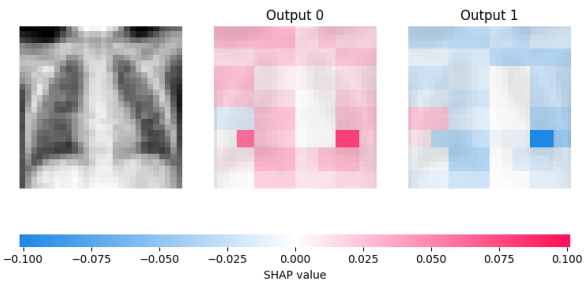


Partition explainer





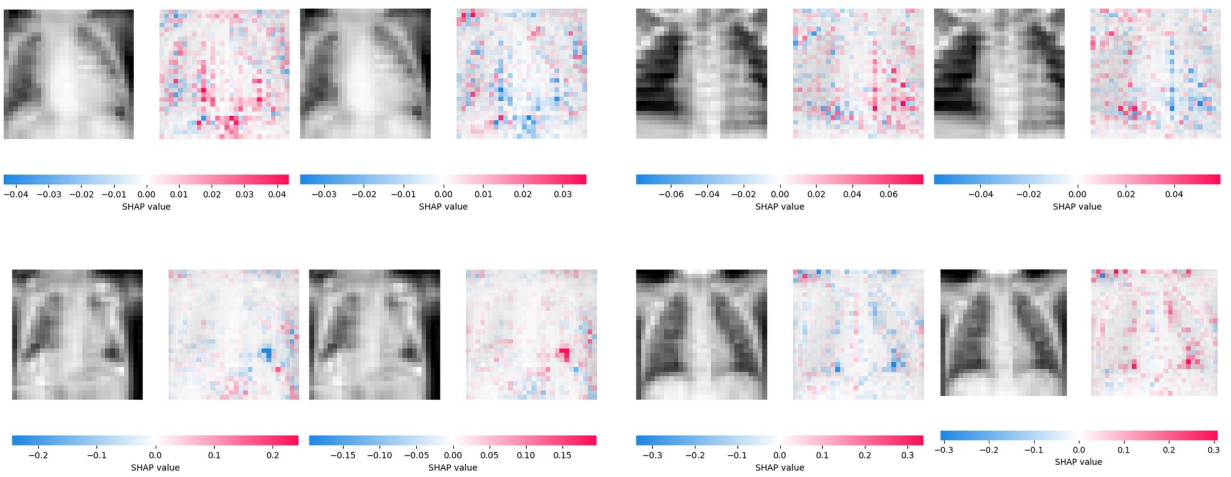




- Images:
  TP FP
  FN TN
- No change in params for partition explainer
- **TP**: More confident
  -good!
- **FP**: Less confident
  -also good!
- **TN**: Less confident
  -bad
- **FN**: Looks like its using the LH clavicle pixels more than lungs
  -real bad!
  -Tuning seemingly changed too much of the 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

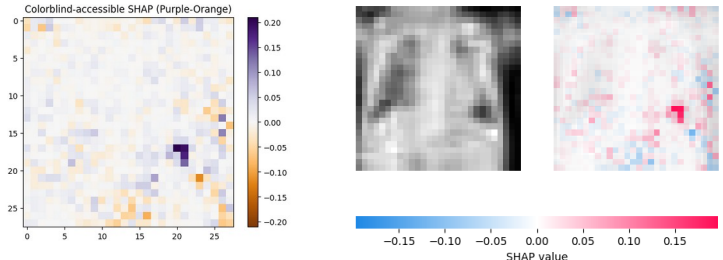
Deepexplainer

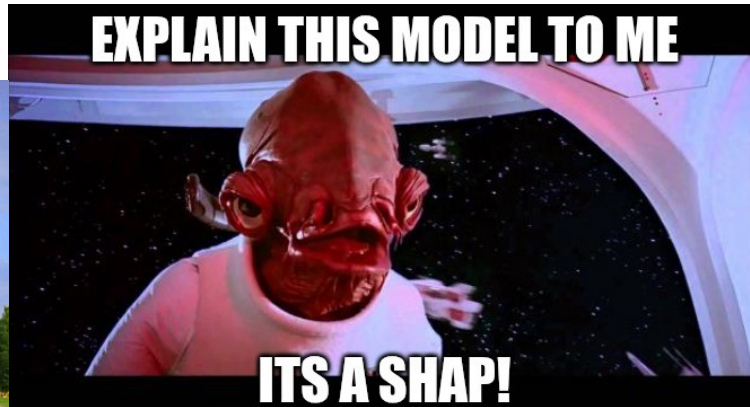
- Images: pred 1, 0
  TP FP
  FN TN
- **TP**: Seems overall better at following the lung outline
- **FP**: Slightly more blue on RH lung field so moving in the correct direction
- **TN**: Slightly less confident?
  - Hard to quantify red pixels
- **FN**: Slightly less confident
  -Pushing towards the correct direction.
- Overall follows partition explainer
  -but visualized differently



Final thoughts on shapvals

- Overall mixed and tiny, nuanced results.
  - -Not surprising given lack of significant difference pre and post tuning
  - -Stumbled on too good starting hyperparams!
- Doing gradient explainer with several layers seem best for NN freeze tuning
  - -Loads of work though
- Really dislike not having quantification the same way .force_plot does
  - -”Err I think this is a deeper red?”
- Also: Color blindness?!





Challenges working with Shap

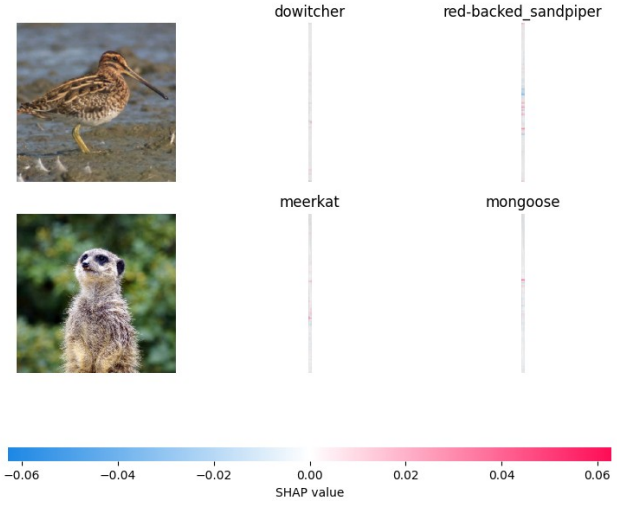
- <1/2 of image classification sample code is on keras.
  Very different from torch requirements since it more or less outputs correct shapes by default from what I can deduce.

- Warning messages in both “getting_started” and shap sample notebooks indicate badly maintained example code:
  -UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  -UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future.

- “Resnet50 partitionexplainer”
  -No warning signs on example code for runtime or memory usage
  -Lack of code comments indicates users know what they are doing
  -Need ~32gb gpu minimum
  -Or risk runtime disconnecting on google collab
  -OOM errors produce TensorFlow stacktraces (remember, we haven’t learned about tensors yet)

```
# here we explain two images using 500 evaluations of the underlying model to estimate the SHAP values
shap_values_fine = explainer_blur(X[1:3], max_evals=5000, batch_size=50, outputs=shap.Explanation.argsort.flip[:4])
```
18.3s

- “Explain an Intermediate Layer of VGG16 on ImageNet (PyTorch)”
  -Output is...Not optimal for learning the library



- Deepexplainer error even asked me to post the code on github:

AssertionError: The SHAP explanations do not sum up to the model's output! This is either because of a rounding error or because an operator in your computation graph was not fully supported. If the sum difference of %f is significant compared to the scale of your model outputs, please post as a github issue, with a reproducible example so we can debug it. Used framework: pytorch - Max. diff: 0.7187470377143654 - Tolerance: 0.01