

Image Classification Model Concatenation and Explanation

CS 770

By

Jonathan Gallegos

Instructor

Kaushik Sinha

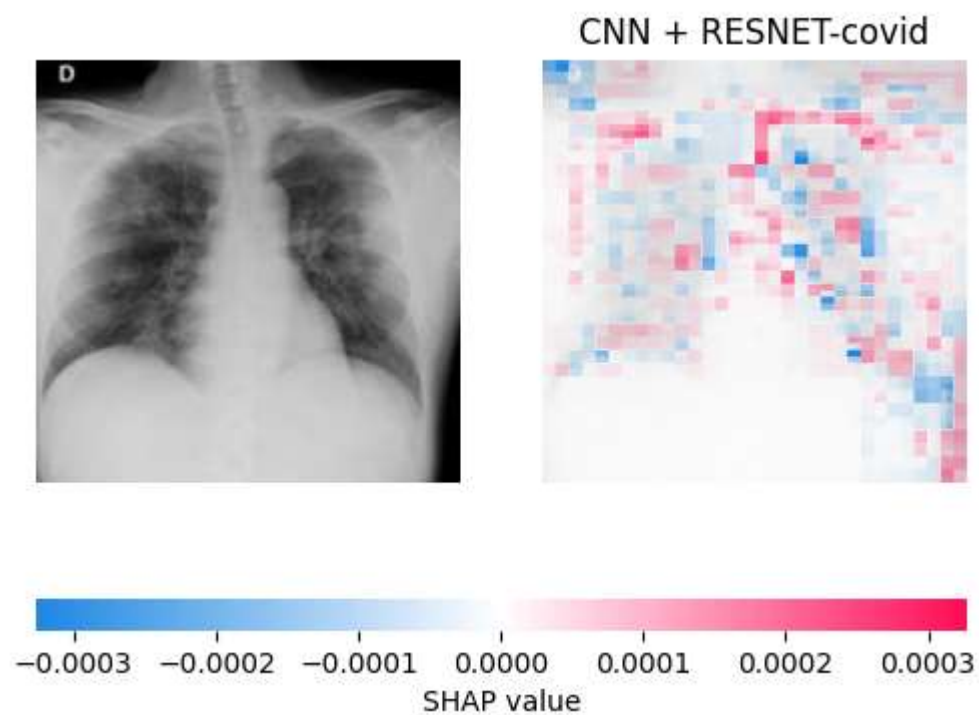


Figure 1 Sample image with associated shapley values

Problem Definition:

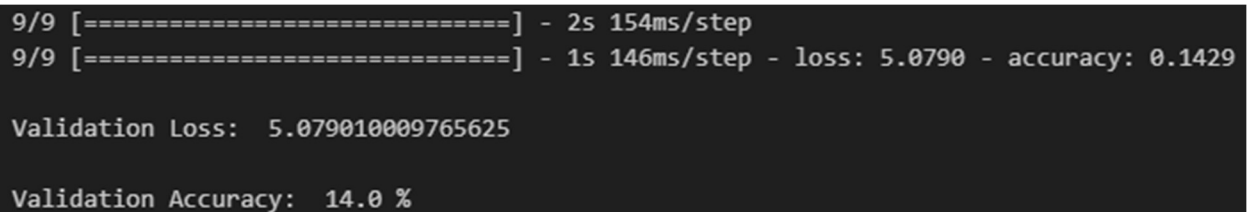
Image classification models have had major upgrades recently, as have many AI models in general. It is pretty clear that the reliability of these models have increased to the point they have become an invaluable resource in class prediction. Everything from self driving cars to shell casing analysis, models like Convolutional Neural Networks are common place today.

A natural step then with any evolving technology is how to make it smaller while still being effective. This happened with computers, taking up an entire room and now fits in the palm of your hand. The TV went from large boxes using cathode-ray tubes to flat narrow LED based systems. That ended up being the purpose of this project, to make initial attempts at studying the performance of a smaller image classification model, benchmarked against its larger counterparts. An example of value is right now the most powerful models are in the hands of corporations who can afford the massive training costs. Smaller powerful models gives enables more people to use these tools.

Dataset Used:

Before we discuss the methods used in the project, it is pertinent that we talk about the dataset (Tawsifur, 2021) first. The data comes from Kaggle and has four classes, Normal, Covid, Lung_Opacity and Viral Pneumonia. This kind of dataset is incredible common in the machine learning community and you can find hundreds of other projects and models for this specific classification purpose. Why I personally still chose to pursue this somewhat overused dataset is because I was more interested in studying how to manipulate a model, rather than a fresh new classification problem to solve.

Another note here is I did attempt to use a bone fracture dataset from Kaggle, but that proved to be incredibly hard to train due to the 15 different target classes involved and a very low sample size.

A terminal window with a dark background and light-colored text. It shows two lines of progress bar output, followed by validation loss and accuracy values.

```
9/9 [=====] - 2s 154ms/step
9/9 [=====] - 1s 146ms/step - loss: 5.0790 - accuracy: 0.1429

Validation Loss: 5.079010009765625

Validation Accuracy: 14.0 %
```

Figure 2 Validation poor performance results on bone fracture data

This is why, even for the Covid dataset, I decided to only train on two classes, Covid and Normal.

The dataset itself was not structured in a way conducive to how my code initiated the data for the model. I ended up manually moving subsets of the data into Train, Validation and Test folders so that my code could easily parse through and prep the data for training and inference. I could have also adjusted my code to account for the structure, but I opted instead to change the structure itself. This means however, the dataset from Kaggle will not work with my code out of the box. I will upload my adjusted dataset to Kaggle for consistency (Jonathan, 2025).

Imbalance was an issue on my first training sessions, which was solved by a suggestion from (Mohammad, 2020). Mohammad created 8 different training phases, each with the same Covid examples, and an equal number but different set of Normal examples. My approach was a little different, my data set had 2760 covid examples and 7220 normal examples. So, I copied my covid examples twice to give a total count of 8280. I have no proof this would give the same performance as Muhammad, but intuitively it appears the same with the only difference being I can now send the whole instance for training, rather than having 3 different training phases. I will demonstrate the effectiveness of this method in the results section.

Methods Applied:

To test the performance of a smaller image model we need a benchmark. In this project I implemented two image classification models, a CNN and ResNet50V2 model. From these we get performance metrics we can test against our new smaller model, as well as the framework for what the smaller model will also be made up of. This idea of concatenating two models together for x-ray classification was presented by Mohammad Rahimzadeh and Abolfazl Attar (Mohammad, 2020). In their paper, they demonstrate the usefulness of combining ResNet50V2 and Xception.

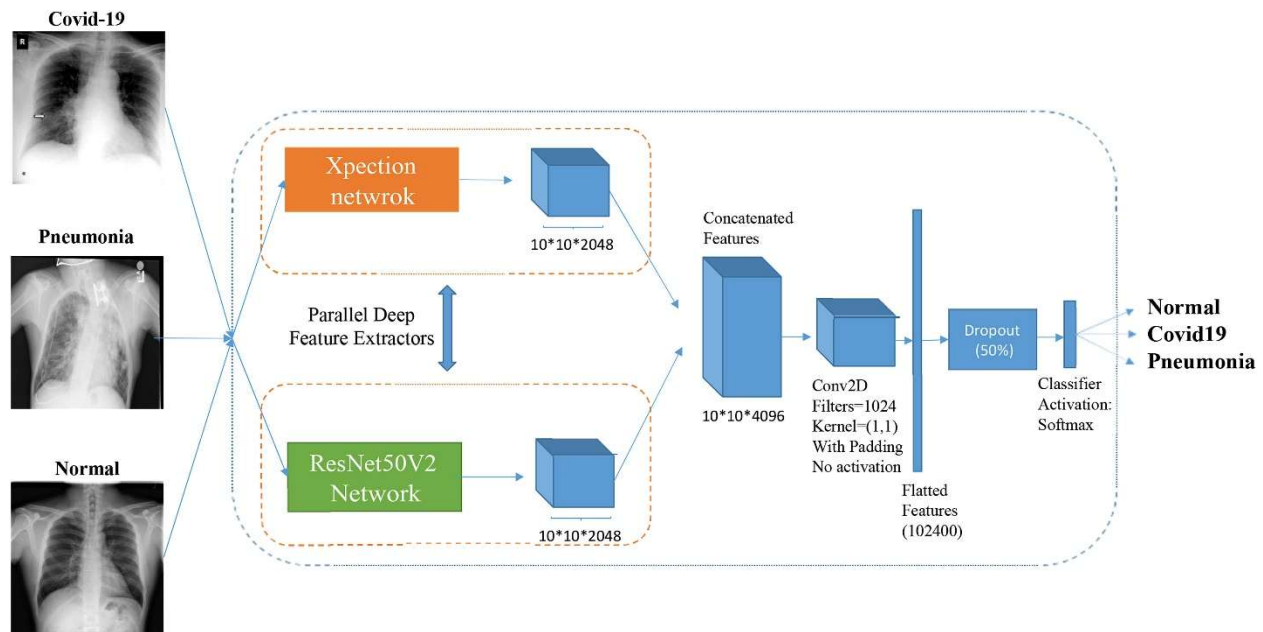


Figure 3 Previous work using a concatenated model

Since I am relatively new to building these models, instead of using the Xception model, I used a simple CNN model, built by Luís Fernando Torres (Luis, 2023). I owe a lot of thanks to Luis for his article and foundational code that made this project possible. The CNN used here is exactly the same as what Luis built, and performed unexpectedly well for being essentially an out of the box solution.

In the beginning, my intent was to combine the CNN and ResNet50V2 at the last full dense layer (2048 shape) and observe the improvement in accuracy. In other words, initially I was not going to test the performance of smaller models. However, the ResNet50V2 performed so well that it seemed doing this would not serve any value. This is where the idea of benchmarking a smaller model came about.

The ResNet50V2 model comes directly from Keras in python. I did adjust the output to be similar to that of the CNN, but the majority was initiated as the prebuilt package. My final combined model was the ResNet50V2 spliced at the conv3_block4_2_relu keras layer and the CNN spliced at the conv2d_8 (Conv2D) keras layer. From here both had a dense 2048 layer added and then concatenated into a 4096 dense layer. Here is the model summary output.

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 256, 256, 3)]	0	[]
sequential_1 (Sequential)	(None, 32, 32, 2048)	391104	['input_3[0][0]']
model (Functional)	(None, 16, 16, 2048)	1651712	['input_3[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	['sequential_1[0][0]']
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0	['model[0][0]']
concatenate (Concatenate)	(None, 4096)	0	['global_average_pooling2d[0][0]', 'global_average_pooling2d_1[0][0]']
dense_2 (Dense)	(None, 2048)	8390656	['concatenate[0][0]']
dropout_3 (Dropout)	(None, 2048)	0	['dense_2[0][0]']
dense_3 (Dense)	(None, 2)	4098	['dropout_3[0][0]']
=====			
Total params: 10437570 (39.82 MB)			
Trainable params: 10429570 (39.79 MB)			
Non-trainable params: 8000 (31.25 KB)			

Figure 4 Combined model summary

The CNN had 69,244,610 (264.15 MB) parameters, the ResNet50V2 had 292,006,402 (1.09 GB) and the combined model had 10,437,570 (39.82 MB) total parameters. So in the sense of information storage, our new combined model is 15% the size of the CNN and 3.6% the size of the ResNet50V2.

All three models were trained on the exact same dataset, this of course was fundamental. Once the models were trained on the Train and Validation sets, I used an unseen Test set to capture the classification report summary. I observed some discrepancies when attempting to get the accuracy from batches I sent into the model, so the results are from single inference steps and the True/False Positives/Negatives were recorded.

Another small but important caveat is that because of my initial labelling, Covid examples are represented with a 0 and Normal are 1's. This is bad practice and should have been changed, however it was also much simpler to continue with what I had and be consistent.

Results:

The first model trained was the CNN and, as stated above, this came directly from Luis' code, and enabled me to build my own combined model. Before I show the best version of the CNN, here is a training instance with the unbalanced set.

		precision	recall	f1-score	support
Covid	0	0.72	0.74	0.73	526
Normal	1	0.93	0.92	0.93	1994
accuracy				0.88	2520
macro avg		0.82	0.83	0.83	2520
weighted avg		0.89	0.88	0.89	2520

Figure 5 Unbalanced data resulted in very high positive recall

And here is the balanced dataset.

		precision	recall	f1-score	support
	0	0.79	0.94	0.86	526
	1	0.98	0.93	0.96	1994
accuracy				0.94	2520
macro avg		0.89	0.94	0.91	2520
weighted avg		0.94	0.94	0.94	2520

Figure 6 Balanced Dataset also balanced out the recall

Here we can see that all metrics across the board increased and benefited from seeing more covid cases. The reason why balancing is so important is first, we can see it helps the model learn the data better. And secondly, especially in medical classification, it is very dangerous to have a model that only captures 74% of actual covid cases. If the purpose here was to actually deliver a useable model, I would intentionally attempt to adjust the Covid class recall to be as high as possible.

Next is the ResNet50V2 which as stated above performed exceptionally well.

	precision	recall	f1-score	support
0	0.98	0.98	0.98	526
1	0.99	1.00	1.00	1994
accuracy			0.99	2520
macro avg	0.99	0.99	0.99	2520
weighted avg	0.99	0.99	0.99	2520

Figure 5 ResNet50V2 classification report

Besides the obvious problem of covid recall being the lowest, ResNet50V2 showed an incredible capacity for learning these types of images. There may have been some data leakage which we will explore later in the paper, but the results still greatly impressed me.

Finally we have the results from the combined model.

	precision	recall	f1-score	support
0	0.61	0.92	0.74	526
1	0.98	0.85	0.91	1994
accuracy			0.86	2520
macro avg	0.79	0.88	0.82	2520
weighted avg	0.90	0.86	0.87	2520

Figure 6 ResNet50V2 and CNN concatenation classification report

Discussion:

It is unfortunately obvious the combined model had a lower performance than the first two. However, given the significant reduction in model size, it may have performed as expected. A major drawback from any conclusions we would like to make is the lack of more test models. It would be greatly beneficial (and possibly the only way to gather true meaning from the project) to adjust what layers are used in concatenation from both models. Would a little more or less from either parent model make a significant impact? What does an 8% and 13% drop in accuracy mean when the model itself is 15% and 3.6% smaller respectively? These are questions we cannot answer since we cannot assume the change in performance follows a linear path.

Still, this shows that there is capacity for the development of smaller models, and possibly by continuing to redesign them we can learn more about how to make these larger models better as well.

Another metric I kept an eye on was training time. Even though the combined model was smaller, it had a training time (1254sec/epoch) between the CNN (838sec/epoch) and the ResNet50v2 (2074sec/epoch). This was a disappointing find, as it most likely means something in my code was not optimized correctly. I am sure this could be adjusted for future use, but it is worth noting that a smaller model does not necessarily warrant less computational cost, especially when coded by a beginner.

To fully take advantage of work accomplished in th, I also implemented the use of shapley values to explain model features, in this case, to explain the pixels each model found useful for making predictions.

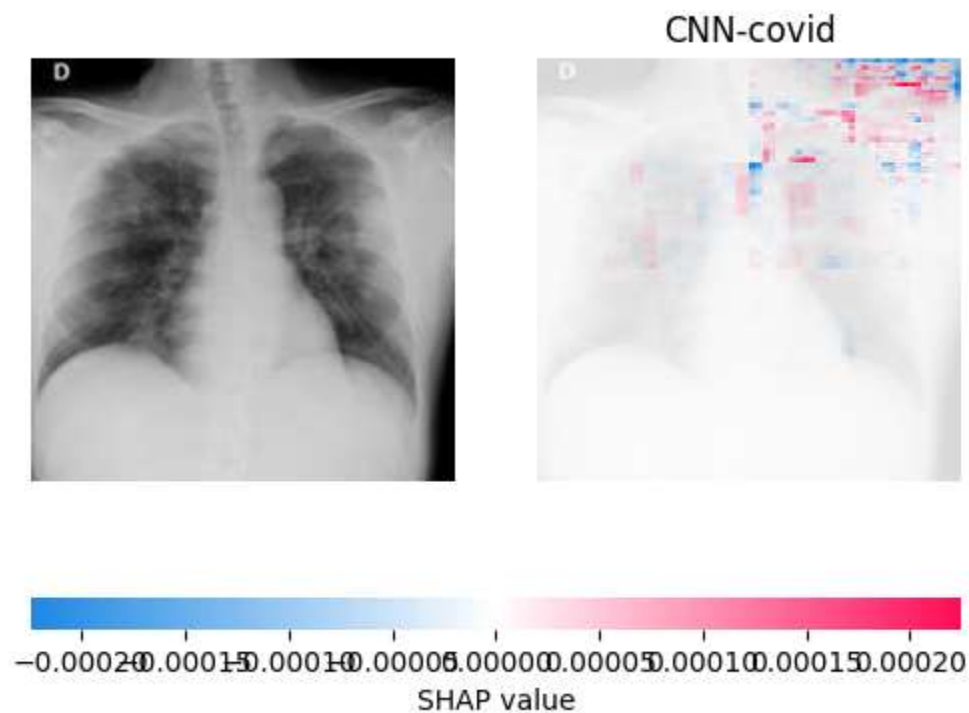


Figure 7 CNN Shapley Values Example

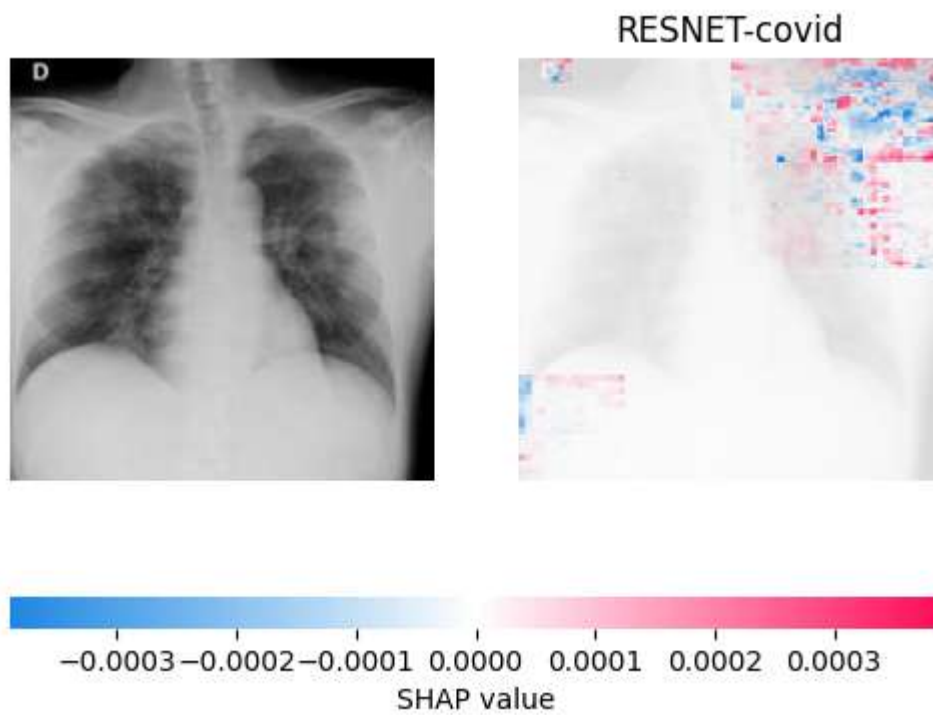


Figure 8 ResNet50V2 Shapley Values Example

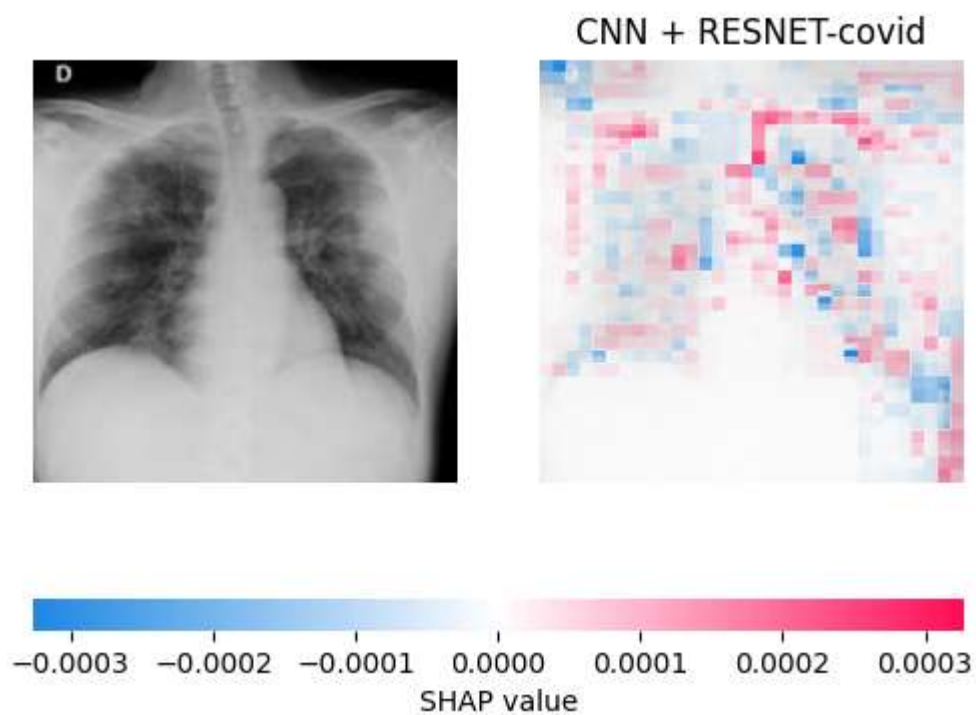


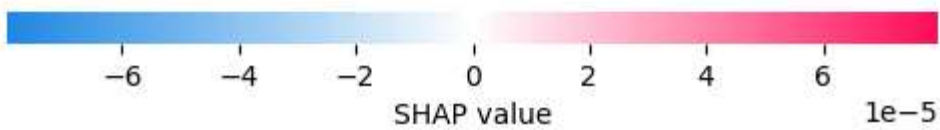
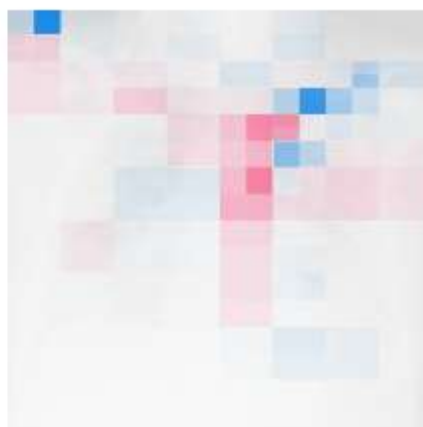
Figure 9 Combined Model Shapley Values Example

These examples are limited, particularly because to get this level of granularity, the code ran for nearly four hours. This is the nature of shapley values since it is iterating over all the combinations of features and creating probabilities for each combination. The Shap package of course has some optimization in the background, but it still increases factorially the more “n” features I present it with.

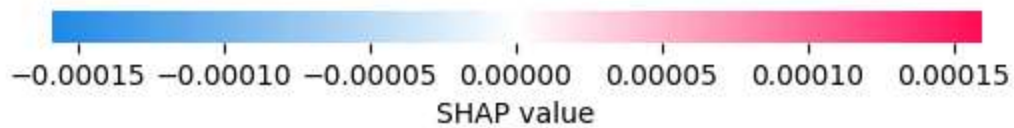
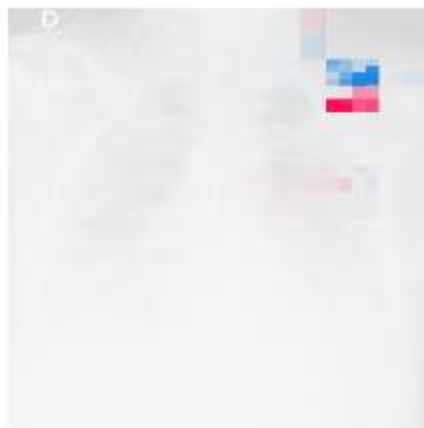
A helpful piece of information that was obtained was it appears the CNN and ResNet50V2 both observed the same useful pixels. This was true for many other instances of the shapley value explanations. Another interesting find was the ResNet50V2 consistently put value in the letters that appeared in the images. We can see an example of this from figure 8 above in the upper left corner (D). This may be an indication of data leakage, but this was an inconsistent find as there are many other examples of this not happening.

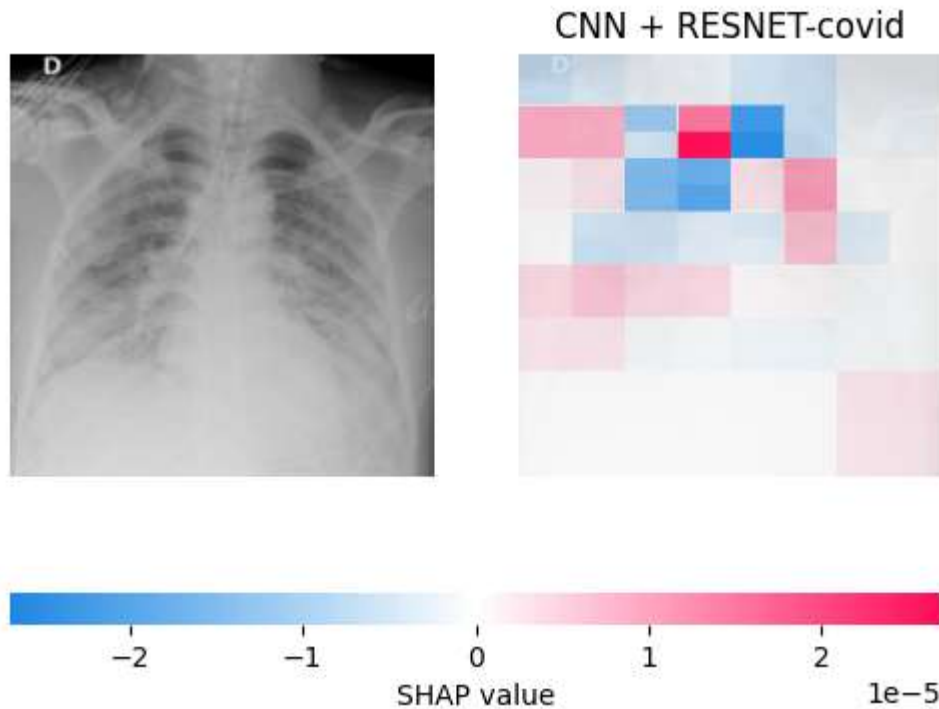


CNN-covid



RESNET-covid





Since this was caught at the very end while running explanation as well, this data leakage was not accounted for. A solution would be to crop each image sufficiently to cut out any text. This would ensure that what the models are looking at is purely the subject x-ray imaging and no other external factors. I am also not well versed in radiology and do not know what these symbols and small text are used for, which could also make a difference with this interpretation.

Altogether, I greatly benefitted from this project. I learned how to better build AI models using Keras, and how models interact with each other and different datasets. I hope to continue working on bettering my own understanding of these practices, and in the future provide meaningful and insightful research into these powerful AI designs.

References:

Gallegos, J. (2025). Image Classification Project CS 770 (Version 0.0.1) [Computer software]. https://github.com/JonPGallegos/image_classification_covid

Jonathan Gallegos. Dataset for CS 770.

<https://www.kaggle.com/datasets/jongallegos/dataset-for-cs-770/data>, 2025, Kaggle.

Luís Fernando Torres. Convolutional Neural Network From Scratch.

<https://www.kaggle.com/code/lusfernandotorres/convolutional-neural-network-from-scratch>, 2023, Kaggle.

Mohammad Rahimzadeh, Abolfazl Attar, A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2, Informatics in Medicine Unlocked, Volume 19, 2020, 100360, ISSN 2352-9148, <https://doi.org/10.1016/j.imu.2020.100360>.

Tawsifur Rahman (Owner), Dr. Muhammad Chowdhury (Editor), Amith Khandakar (Editor). COVID-19 Radiography Database.

<https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database>, 2021. Kaggle