Detecting COVID-19 from Medical X-Ray Images

by Rayan Wali

Cornell University

December 2021

Motivation Behind Study

Problem Statement

When a medical image scan is taken by radiologists, they often have to decipher the image by classifying it as a single case.

The Problem

Given an X-ray medical image scan of a patient, decide whether the patient has COVID-19 or not.

AI as an Asset in Medical Imaging

- Reduces manual workload.
- Improves efficiency and results in precise decisions.
- Enables speedy delivery of results to patients.

Source: The Computer Society

Problem Statement (cont...)

We can encode this problem as a binary classification task, with the two classes being "Positive" and "Negative".

Class #1: Positive

represents positive COVID-19 cases

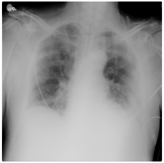
Class #2: Negative

represents negative COVID-19 cases

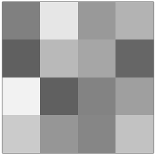
Images

Pixel Intensities

Each pixel of an image holds an intensity, and this property is used by the CNN to recognize the image as a whole. For an 256×256 dimensional image, there are in total $65,536$ pixels. This pixel intensity array is then passed into the first layer of the CNN.



(a) Sample X-Ray Image



(b) Sample Grayscale Image

20	120	50	80
10	85	76	15
225	10	22	78
100	48	24	90

(c) Pixel Matrix for **(b)**

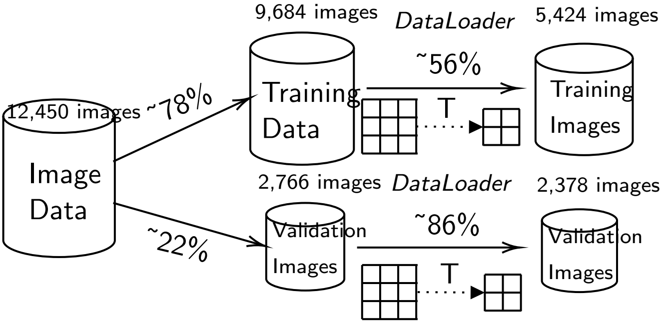
Tools & Technologies

- **Languages:** Python (Deep Learning Model), HTML (Webpage Structure), CSS (Webpage Styling).
- **Libraries/Packages:** Keras, TensorFlow (Data Preprocessing, Deep Learning Model Training + Testing), SkLearn (Data Preprocessing + Scientific Evaluation), Flask (Interactive Webpage).

Outline

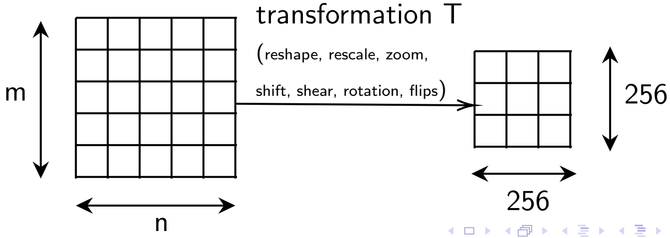
1. **Preprocess** Input Data
2. **Develop** Model Architecture
3. **Train** the Model
4. **Evaluate** the Model on Training and Validation Data
5. **Deploy** Model for Web Application Use

Phase I: Preprocessing Data

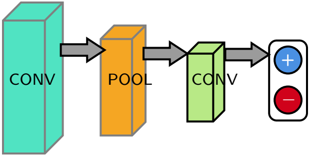


$$\frac{5424}{5424 + 2378} = \mathbf{0.70}$$

$$\frac{2378}{5424 + 2378} = \mathbf{0.30}$$



VGG-16: A 16-layer CNN, with a mix of convolutional and pooling layers.




```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img_input (InputLayer)      [(None, 256, 256, 3)]     0

 vgg16 (Functional)          (None, None, None, 512)   14714688

 flatten (Flatten)           (None, 32768)             0

 fc1 (Dense)                 (None, 1024)              33555456

 fc2 (Dense)                 (None, 512)               524800

 preds (Dense)               (None, 1)                 513

=================================================================
Total params: 48,795,457
Trainable params: 48,795,457
Non-trainable params: 0
_________________________________________________________________
```

Phase III: Training the Model

To train the model on the data, I used **stochastic gradient descent** (SGD). Stochastic gradient descent is an optimization technique that minimizes the following expression:

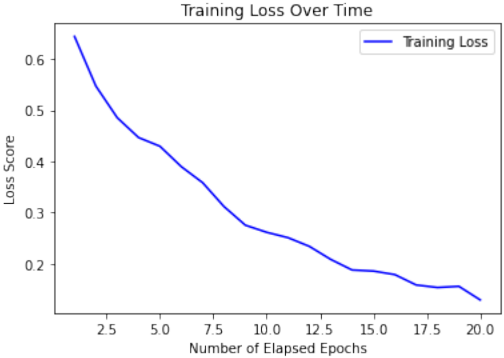
$$\min\ J(\theta) = \min\ \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(h_\theta(x_i), y_i\right) \tag{1}$$

Hyperparameters

- I trained my model for 20 epochs and achieved an 95% training accuracy in the end of the training process.
- For the gradient descent backpropagation phase, I used the binary cross-entropy loss function.
- I found that $\alpha = 0.01$ was the optimal learning rate out of the ones I experimented with.

Phase III: Visualizing the Training Process

In the following line plot, we observe (1) a continual decrease in the binary cross-entropy loss over time as the number of epochs that have elapsed grows. We also observe (2) that the rate of decline in the loss in the earlier epochs is greater than that in the later epochs.



To evaluate my trained model, I considered three metrics outlined as follows:

(1) **Accuracy Metric:** The number of correct instance classifications divided by the number of total instances classified.

(2) **Loss Score:** Accounts for the confidence in the model's prediction in comparison to the true label, and not only on the difference between the true and predicted labels (accuracy metric).

(3) **F1 Score Metric:**

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}},$$

$$\text{where}\ \ \text{Precision} = \frac{TP}{TP + FP} \text{ and } \text{Recall} = \frac{TP}{TP + FN}.$$

Phase IV: Evaluating the Model (Accuracy)

[$\alpha = 0.01$, SGD Optimizer]

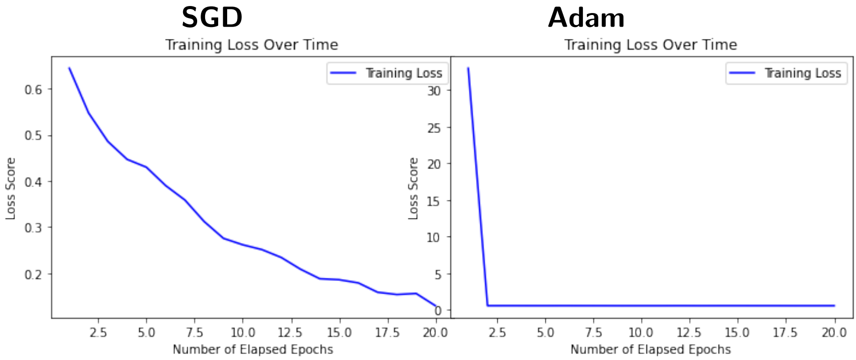
No. Epochs	Avg. Training Accuracy	Avg. Validation Accuracy
5	82%	81%
10	88%	86%
20	94%	95%

Phase IV: Evaluating the Model (F1 Score)

[$\alpha = 0.01$, SGD Optimizer]

No. Epochs	Training F1 Score	Validation F1 Score
5	0.8929	0.8734
10	0.9052	0.8805
20	0.9187	0.9125

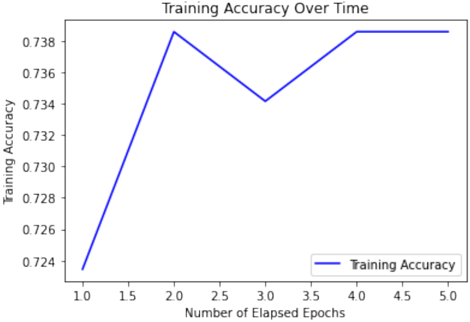
Phase IV: Other Experiments (Optimizer) [$\alpha = 0.01$, 20 Epochs]



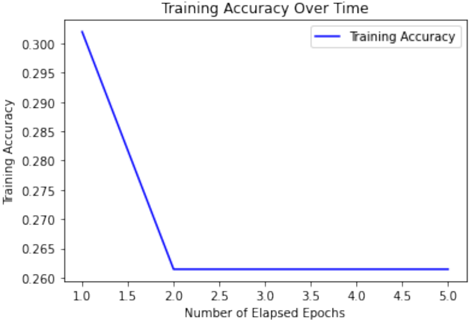
Phase IV: Other Experiments (Learning Rate α)

[SGD Optimizer, 5 Epochs]

$\alpha = 0.01$

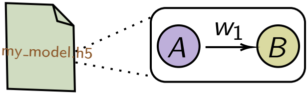


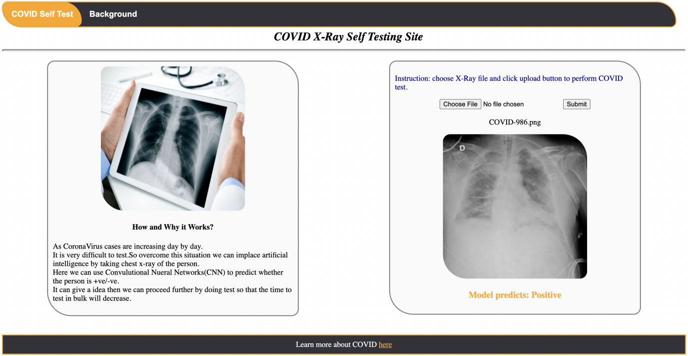
$\alpha = 0.1$



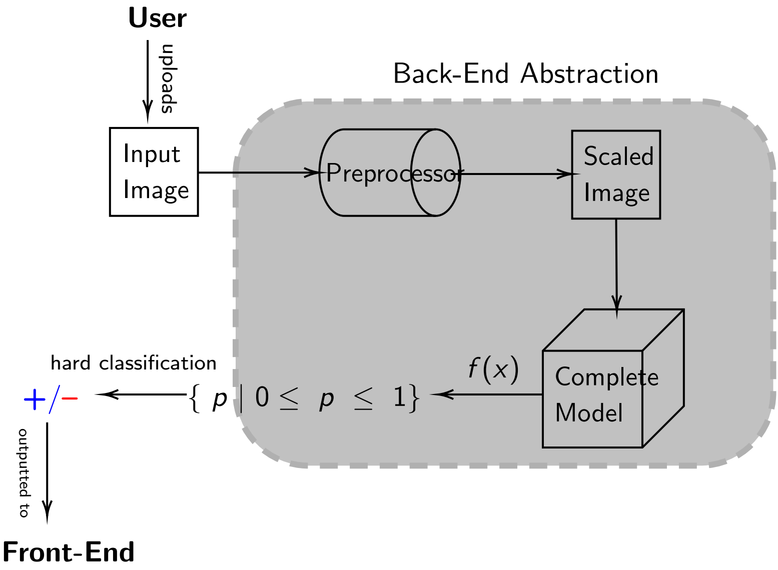
I built a web-application with Flask and HTML intended for use by radiologists who can input a patient's chest X-ray image and retrieve an output on whether the patient is COVID-19 positive or negative.

Model Compression (`.h5` file):





End-to-End Picture



Potential Improvements & Optimizations

1. Perform hyperparameter optimization with random search or Bayesian optimization to obtain the optimal number of epochs and the optimizer type resulting in a lower overall loss.
2. Add more data augmentation to the training data in the preprocessing phase (to increase model generalizability).
3. Re-perform the training process on a different model architecture, e.g., on Microsoft's ResNet architecture.
4. Make the system interactive, i.e., the input of radiologists should be able to influence the output prediction.