Deep Learning and its applications to Signal and Image Processing and Analysis

Assignment 2 - Chest X-ray Classification
361.2.1120

Linoy
Alon

September 6, 2024

1 Load and Explore the Data

1.1 Load the Data

We followed the instructions in the notebook, completed the missing code and loaded the data.

1.2 Exploratory Data Analysis

1.2.1 Data Distribution

We presented the data distribution:

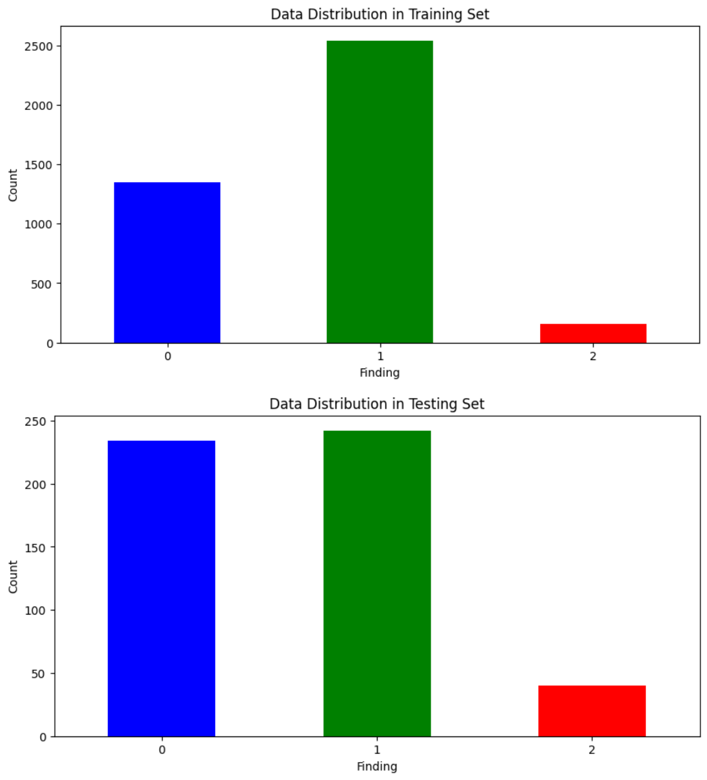


Figure 1: Data Distribution in Training set (left) and test set (right). Labels 0 - NORMAL, 1 - BACTERIA, 2 - Pneumonia/Viral/COVID-19

1.2.2 Images Exploration

We started by displaying images from each category to get a better understanding of the data we have

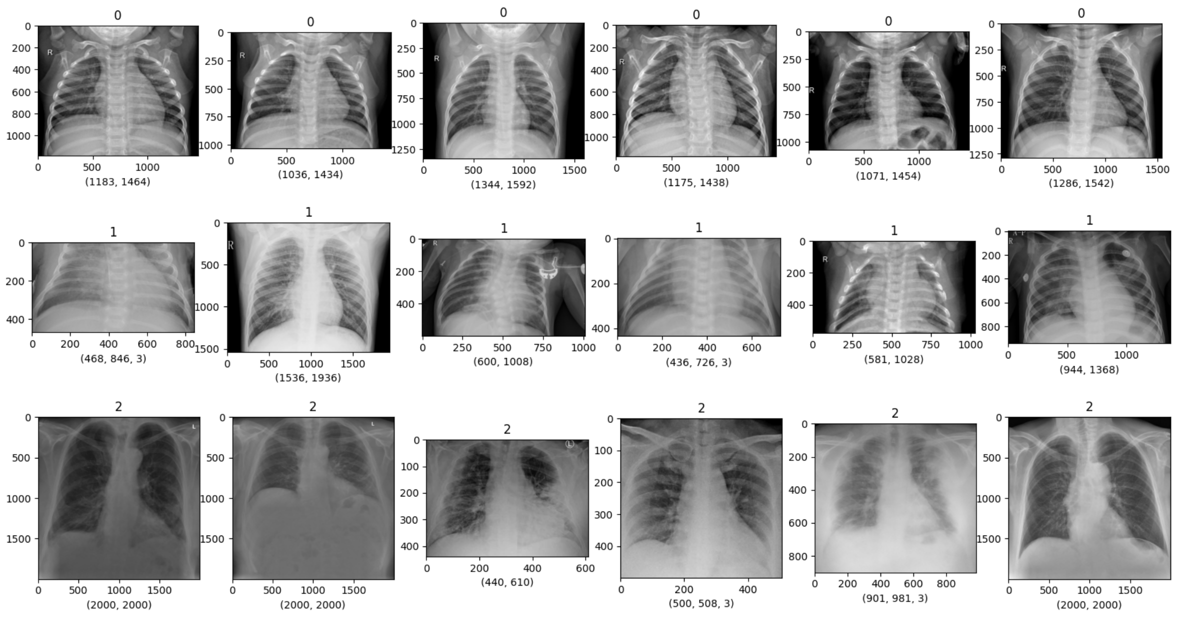


Figure 2: Displayed images from all 3 categories

1.2.3 Differences and Potential Challenges in the Data

We noticed some differences in the data, both between classes and in the same class as well. Some of then can lead to major challenges as some less then the others.

Different Image Sizes and Channel Dimensions
Images in the dataset come in various sizes and channel dimensions. Neural networks require input images to be of a consistent size and number of channels to function correctly.
Solution: Resize images to a consistent size, such as 224x224 pixels, and converted to a consistent channel format - gray scale.

Image Aspect Ratios
The images have different aspect ratios, leading to potential distortion when resizing them to a fixed size. This distortion can affect the quality of the features extracted by the CNN.
Solution: Add padding to images to maintain their aspect ratio before resizing. Alternatively, center cropping can be used to focus on the central area of the image.

Different Contrast Levels
There are varying levels of contrast across images, both between and within classes. Different contrast levels can make it harder for the model to learn consistent features.
Solution: Pixel values of images are normalized to have a consistent mean and standard deviation. Additionally, histogram equalization is applied to improve the contrast across images.

Patient Position
Patients may be positioned differently in each X-ray, introducing variability that can make it harder for the model to learn consistent features.
Solution: Data augmentation techniques such as rotation, flipping, and shifting can be used to make the model more robust to variations in patient position.

Area of Interest
Images may have different zoom levels and amounts of background, with some images being more zoomed in than others, and some containing more background information. This variability can affect the model's ability to focus on the relevant anatomical features.
Solution: We can utilize cropping the central area.

Monitoring Devices and Artifacts
The presence of monitoring devices, wires, and other artifacts in the images can introduce biases and affect model learning.
Solution: Data augmentation techniques can be used to introduce randomness, making the model more robust to such artifacts.

Class Imbalance
The dataset has an unequal number of images for different classes (e.g., fewer COVID-19 images), leading to class imbalance. This imbalance can result in biased models that favor the majority class.
Solution: We can try and balance the data using oversampling the minority class, under-sampling the majority class, adjusting the loss function to give more weight to the minority class, and using data augmentation to increase the number of samples in the minority class.

1.3 Dealing With Imbalanced Data

We created functions to create the 4 different dataset: original (data as-is), over sampled, under sampled and class weights. We will see their utilization and influence in section 3.1

2 Data Augmentation

We started by exploring some literature about COVID19 classification [2][3] to get intuition about popular augmentations that perform well on this kind of data. We then started to train our network with different kind of augmentations. We tested the following:

1. Histogram Equalization using `ImageOps.equalize`
2. Random rotation $\theta \in [-10°, 10°]$
3. Random Horizontal Flip with probability $p = 0.5$
4. Random Shifting for up to 10% to each direction
5. All images were resized to 256x256, converted to gray scale and normalized with $mean = 0.5, std = 0.5$ (Both train and test)

3 Choosing The Model

3.1 Train a classification network

We started with a simple model, small amount of data (one batch of 16 images from the dataset) and no augmentation at all. We wanted to make sure our training pipeline is working correctly. When we saw that our simple model is over-fitting the 16 image batch we started to complicate the model and used the whole data set. We started with the given data set (Unbalanced, no augmentations) and trained the model until we got what felt to us a good result. This is the model we got:

3.1.1 Our own CNN model

```python
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=3):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=5, padding=2)
        self.bn2 = nn.BatchNorm2d(128)
        self.pool2 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.fc1 = nn.Linear(128 * 32 * 32, 1024)  # Reduced size
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))
        x = self.pool2(F.relu(self.bn2(self.conv2(x))))
        x = x.view(-1, 128 * 32 * 32)  # Flatten the tensor
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

3.1.2 Parameters for optimization

We used the **ADAM optimizer** with a learning rate of **0.0001** for most of the time, **Cross Entropy Loss**, and a weight decay of **0.00005**.

3.1.3 Model Evaluation

Validation Set

We created a validation set using 20% of the training set. We made sure we are keeping the ratio between the categories the same as in the original training set. This isn't ideal because we saw at the beginning that the test set has different distribution, but when training a Machine Learning/Deep Learning model we can't assume we have the test-set distribution. Using the same validation set for all training sets isn't ideal but it helps with consistency and by using precision score instead of accuracy we can use the unbalanced validation set. Using data augmentation on the validation set as well might help us address those issue by enlarging the validation set and align it with the training set distribution in each case.

3.2 Training Process

3.2.1 Basic Evaluation

We started by training our model on the 4 different dataset (original, under-sampled, over-sampled and class weights) without any data augmentation. We wanted to see which dataset gives us the best results on the validation set and test set(We fixed the seed to eliminate the randomization factor) and then using this dataset to try and improve performance with fine-tuning, data augmentation and more.

	Validation Accuracy (%)	Test Accuracy (%)	Test Precision (%)
Original	96.54	79.26	83.76
Under Sampled	97.87	79.65	85.85
Over Sampled	98.29	84.50	86.48
Class Weights	96.66	80.04	84.87

Table 1: Results of our model on the different data sets.

We Can see that in out case the over sampled class performed the best (not by much). We believe that this is due to the fact that our data set isn't that big, especially the COVID19 class. For that reason, over sampling can help the model generalize better especially on the classes with more training data. During the training we evaluated the test's per-class accuracy and noticed that the model isn't performing that well on the NORMAL class, which really affected the precision score. This is due to the fact that this is a fairly big class so the performance on it affect the overall performance. Over sampling meant that we will train on all the sampled from this class and some duplicated as well. This increased the overall performance.

	NORMAL (%)	BACTERIA (%)	COVID-19 (%)
Original	57.69	99.17	85.00
Under Sampled	57.69	97.93	97.50
Over Sampled	**70.51**	97.52	87.50
Class Weights	60.26	98.35	85.00

Table 2: Per class accuracy

Are we over-fitting? Why? What can we do?

Yes, we are over-fitting during training. We reached 98% training accuracy on all three larger datasets (Original, class weights, and over-sampled). For the under-sampled dataset, more epochs were needed, but we quickly achieved 100% training accuracy.

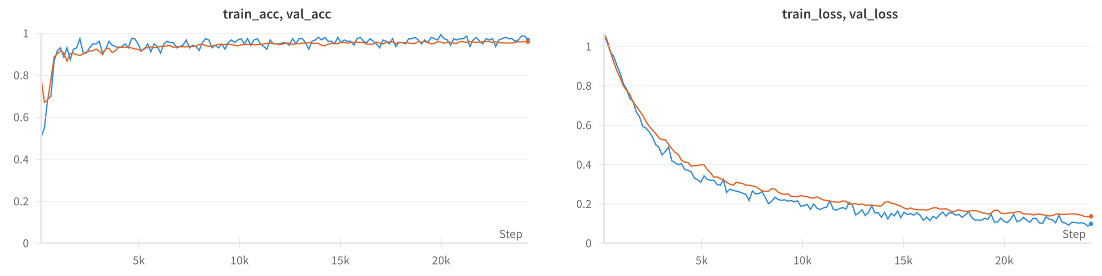


Figure 3: Training and validation loss and accuracy from training the SimpleCNN model (Validation score in Orange)

Why are we over-fitting? We think that the simple answer is that the model is large enough and the dataset is small. In [4], it was shown that a neural network with $2n + d$ parameters is capable of perfectly fitting any dataset of n samples of dimension d. In our case, $n \approx 7500$ (for the over-sampled dataset) and $d = 256 \times 256 = 65,536$. This means that $2n + d \approx 80,000$, while our 'SimpleCNN' network has 134,953,603 parameters.

Later on, we realized that this is a huge amount of parameters! most of them from the linear layers. We think that this is also something that caused the over-fitting and hurt the generalization but it was too late for us to change and run training and evaluation on another model. At the end of the document we will present some results on what we think is a better suited model for this case.

While our network's performance on the over-sampled class was the best, the results require further examination. In Table 1, we see that the validation scores are very high, whereas the test scores are good but not as high. Ideally, we would expect similar results for both validation and test sets, assuming all samples are drawn from the same distribution and both sets contain images the model hasn't seen before.

However, the difference suggests otherwise. We assume this might be because the validation data distribution was chosen to match the training data, while the test data has a different distribution. This difference could introduce bias, which the model might have learned during training. Although this explanation provides some insight, we believe there might be additional factors contributing to this difference that we haven't identified yet.

What can we do?

To reduce overfitting, several strategies can be employed, generally falling into three categories: increasing the amount of training data, reducing the complexity of the model, and employing regularization techniques. Here are the main approaches:

Increasing the Amount of Training Data

- **Data Augmentation**: Apply transformations such as rotations, translations, flips, scaling, and color jittering to artificially increase the diversity of your training dataset. This helps the model generalize better by seeing more varied examples.

Reducing Model Complexity

- **Simplifying the Model**: Reduce the number of layers or the number of units in each layer to make the model simpler. A simpler model is less likely to over-fit the training data.

Regularization Techniques

- **Dropout**: During training, randomly set a fraction of the input units to 0 at each update to prevent the model from relying too heavily on any one feature. This can be implemented in neural networks using dropout layers.

- **L2 Regularization (Weight Decay)**: Add a penalty term to the loss function to penalize large weights. This discourages the model from fitting the training data too closely.

Hyperparameter Tuning

- **Hyperparameter Optimization**: Use techniques like grid search or random search to find the best hyperparameters for your model, such as learning rate, batch size, and regularization parameters. This can also include learning rate scheduling to reduce the learning rate as training progresses.

Can the classes be distinguished?

We will look into two examples of different datasets (original and over-sampled). To check if we can distinguish between the classes in the dataset we plotted the 2D t-SNE. To plot this we used the feature layer which we choose to be the last linear layer before the classification layer.

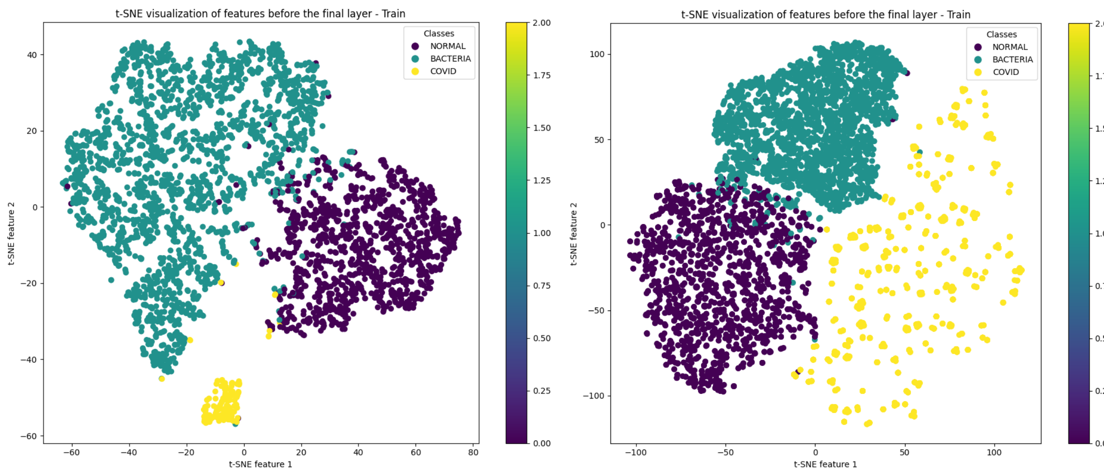


Figure 4: t-SNE of the training data from the last linear layer of our model. (over-sampled on the right)

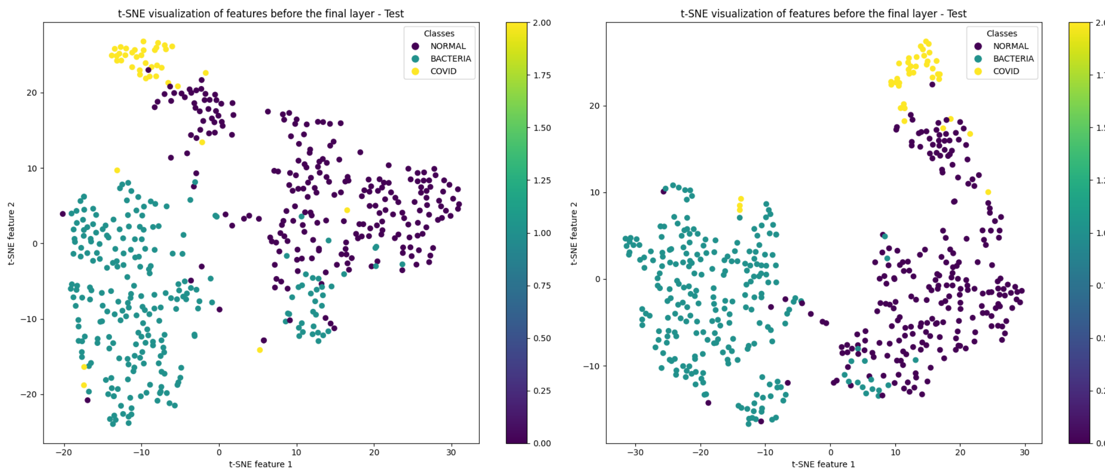


Figure 5: t-SNE of the test data from the last linear layer of our model (over-sampled on the right)

We can see on Figure 4 that on the training data the model was able to separate the classes very well (Which fits the over-fitting we witnessed during training) but on the test set it was harder for the model to separate the classes as depicted in Figure 5. We can also see here our claim of lower accuracy rate on the 'NORMAL' class, in the test t-SNE the separation of this class is much harder.

Another way we can see it is by watching the confusion matrix for the same cases

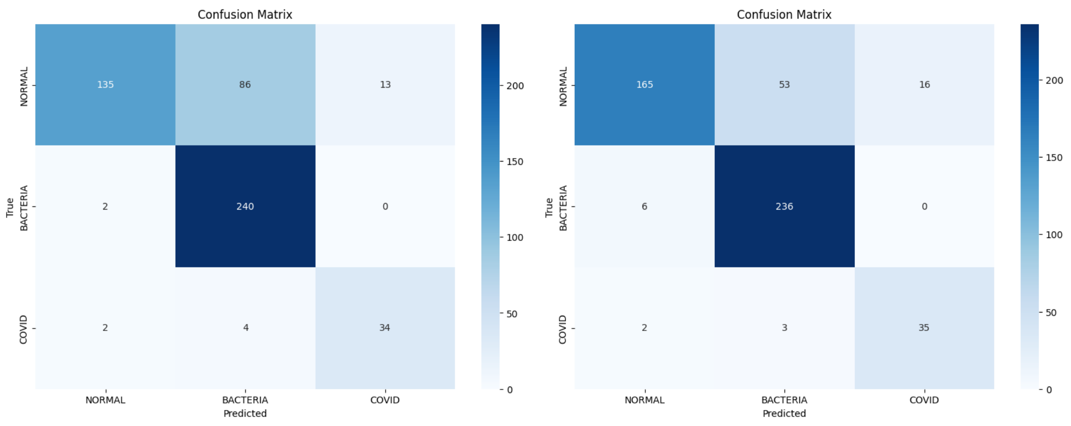


Figure 6: Confusion matrices on the test set for over-sampled and original datasets (over-sampled on the right)

What are the data limits? Can we filter/process it better?

We addressed some of the data issues in the beginning of the assignment but we do have some more ideas how to handle it better. **Investing a lot of time in going through the data**, class by class and try to understand if there are re-appearing things or anything that can affect the model's learning but probably shouldn't. For example, we saw that there are cases where there is some nodes/devices/wires connected to the person's body. From a quick look they are more common in the NORMAL/COVID19 clients. Maybe it is because they are already treated for the disease. We think that this can add some bias to the model which isn't completely true.

Plotting the images resolution distribution and see if it contains any interesting information. This is connected to the question about **Is it possible that we are classifying the X-Ray machine and not the case?** Different machines meaning different images ratio, maybe different background? maybe some additional tool/wires/components that are part of the machine and has no affect on the disease itself?

For example, we can see in Figure 7 a set of random images from the COVID-19 data that contains arrows drawn by humans. If this is something that appears only in the COVID-19 images the model can use this to classify images with arrows to this class because it is very unusual.

It is hard to say if it is possible that we are classifying the X-Ray machine because we don't have any prior knowledge on how many different machines there are and what is the differences between them. I think that maybe in the bigger model, if we have a lot of samples from the same machine with the same class we might be classify the machine. If we assume that each machine gives us the same distribution of each class this will just add to the overall biased we mentioned before.

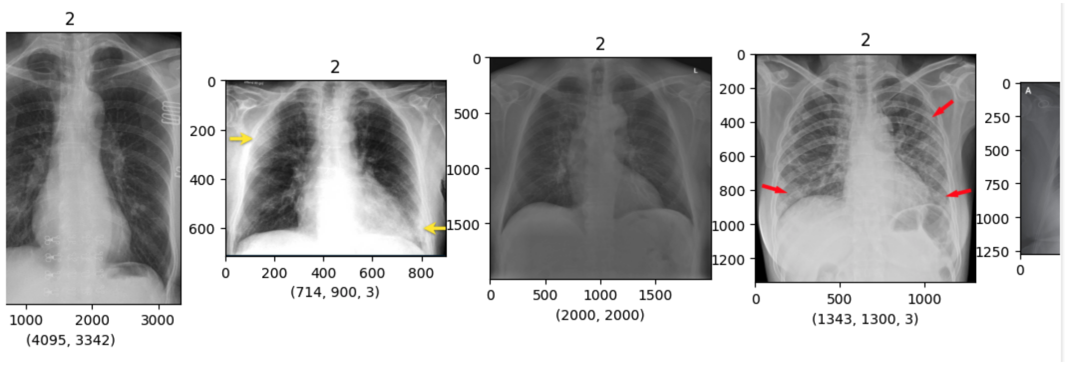


Figure 7: Random samples from the COVID-19 data

3.3 Transfer Learning

3.3.1 ResNet50 Model

ResNet50, or Residual Network 50, is a deep convolutional neural network architecture introduced in the 2015 paper "Deep Residual Learning for Image Recognition" [1]. It is known for its use of residual connections, which help mitigate the vanishing gradient problem in very deep networks.

Key Features

- **Residual Blocks**: The core idea is the residual block, which allows the network to learn residual functions with reference to the layer inputs. This helps prevent the degradation problem and allows for the training of much deeper networks.
- **Depth**: ResNet50 consists of 50 layers, including convolutional layers, batch normalization layers, and fully connected layers. It is organized into four stages of residual blocks with increasing filter sizes.
- **Architecture**:
  - Initial convolutional layer and max pooling layer.
  - Four stages of residual blocks with 3, 4, 6, and 3 blocks respectively.
  - Global average pooling layer.
  - Fully connected layer with 1000 outputs for classification tasks.

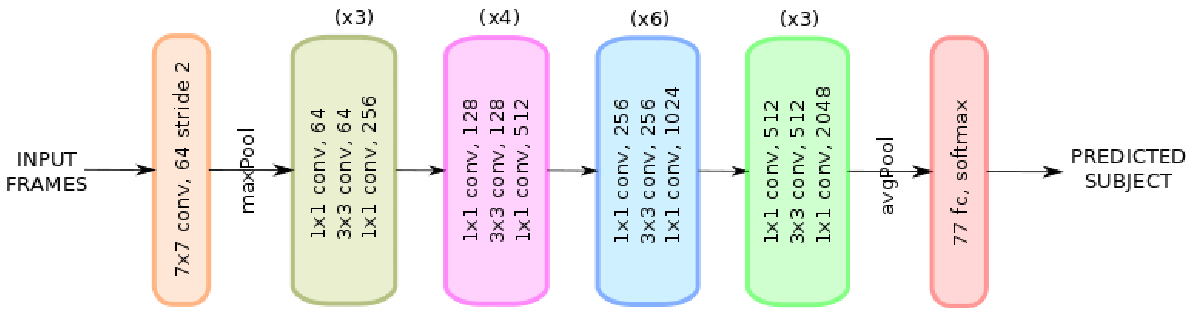


Figure 8: Resnet50 Architecture

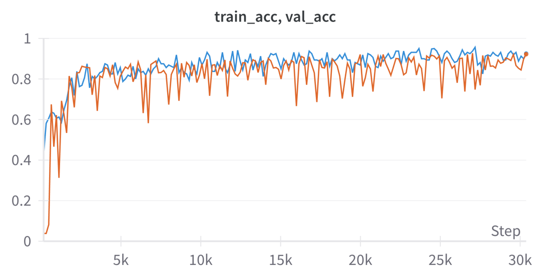
We followed the instruction and loaded the pre-trained weights from torch, froze all the layers but the last 3 and added 3 linear layers of our own:

```python
class ModifiedResNet50(nn.Module):
    def __init__(self, num_classes, pretrained=False):
        super(ModifiedResNet50, self).__init__()
        self.resnet = ResNet50(pretrained=pretrained)
        # Changing the first layer to accept 1 channel for grayscale images
        self.resnet.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2),
        ↪ padding=(3, 3), bias=False)

        # Freeze all layers
        for param in self.resnet.parameters():
            param.requires_grad = False

        # Unfreeze the last bottleneck block in layer4 and the fully connected layer
        ↪ (fc)
        for param in self.resnet.layer4[2].parameters():
            param.requires_grad = True
        for param in self.resnet.fc.parameters():
            param.requires_grad = True

        # Replace the final fully connected layer
        num_ftrs = self.resnet.fc.in_features
        self.resnet.fc = nn.Sequential(
            nn.Linear(num_ftrs, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        return self.resnet(x)
```

We then trained the model on the same dataset that performed the best with our model (The over-sampled dataset) for different number of epochs and learning rates. It took this model more time to converge and the results we achieved using it was worst then our model (Training, Validation and testings results). We assume that this has something to do with our fine-tuning parameters. Maybe re-train only the last 3 layers wasn't enough. We were expected to achieve better results with the pre-trained resnet50 then out model.

	NORMAL (%)	BACTERIA (%)	COVID-19 (%)	Overall Accuracy	precision
SimpleCNN	70.51	97.52	87.50	84.5	86.48
ResNet50	79.06	96.28	87.50	87.79	88.6

Table 3: Comparing our SimpleCNN model evaluation to the fine-tuned Resnet50 model



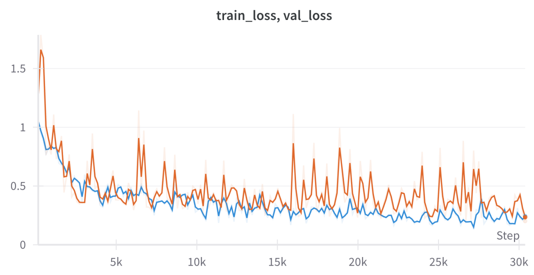


Figure 9: Training and validation loss and accuracy from fine-tuning the ResNet50 model (Validation score in Orange)

Compared to our model, we noticed that the resnet50 isn't over-fitting as much. We assume that this is because it was designed very well to generalize as much as possible without over-fitting.

We did see that the fine-tuning process wasn't as stable as we expected (large variance in the accuracy/loss scores). We did tried to use lower learning rate but then the model wasn't converging to the same results. We do believe that with some kind of learning rate scheduler and other training method and techniques we would be able to achieve better performance but it requires a lot of time to train and evaluate.

4 Test Time Augmentation (TTA)

In this part we were asked to perform random augmentations to the test set images such that each image is passed through the network several times but with different random augmentation. By averaging the prediction the final prediction is determined. We performed our evaluation by comparing the over sampling model from section 3 with two TTA models: TTA 1 and TTA 2. In each model (TTA 1 and TTA 2) the prediction is the average between three predictions: model prediction without TTA, model prediction on test set obtained by augmentation 1 and model prediction on test set obtained by augmentation 2. The difference between the models are the augmentations applied on the test set. For clear explanation the below flow chart is attached.

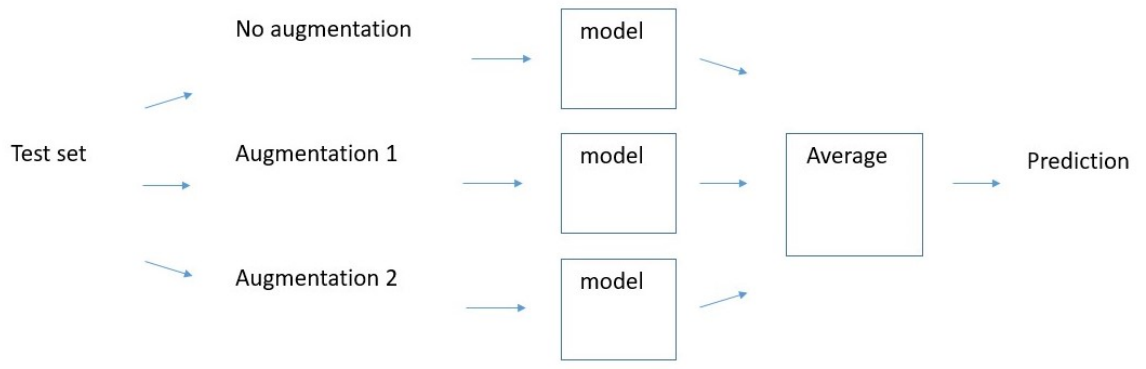


Figure 10: Flow Chart of single TTA model

TTA 1 - first model

The augmentations we chose for Augmentation 1 are:

- **Random Rotation**: Apply a rotation matrix with a randomly chosen angle within the specified range.
- **Random Affine**: An affine transformation matrix is constructed using randomly chosen parameters for translation, rotation, scaling, and shearing.

The augmentations we chose for Augmentation 2 are:

- **Histogram equalization**: A method used to improve the contrast of an image. It works by redistributing the pixel intensity values so that the histogram (which plots the number of pixels for each intensity level) is approximately flat. This process enhances the global contrast of the image, making details in both dark and bright areas more visible.
- **Random horizontal flip**: image is flipped horizontally (left-to-right) with a certain probability. This is particularly useful for tasks where the orientation of objects is not fixed, such as in natural images.

TTA 2 - second model

The augmentations we chose for Augmentation 1 are:

- **Random Rotation**
- **Random horizontal flip**

The augmentations we chose for augmentation 2 are:

- **Histogram equalization**
- **Random Rotation**

The reason we chose these augmentations is that these augmentations tackle issues rises from this specific data set as explained in section 1.2.3 (”Differences and Potential Challenges in the Data”). We assumed that using the above augmentations will enhance robustness of the model and by that deal with the issues of the data set.

4.0.1 Model Evaluation

Validation Set

In this part the validation set is the same as in section 3, as it does not matter to TTA since TTA is applied for the test set.

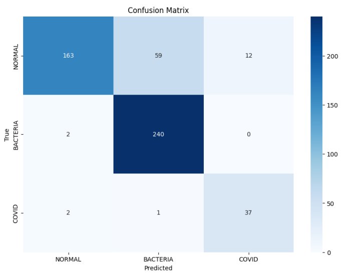
4.0.2 Results Evaluation

To evaluate the results we loaded the weights of the model we used in the previous section. We then performed the TTA augmentations and got the results in Table 4.

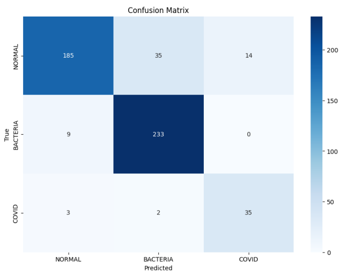
	NORMAL (%)	BACTERIA (%)	COVID-19 (%)	Overall Accuracy	precision
No TTA	70.51	97.52	87.50	84.5	86.48
TTA 1	79.06	96.28	87.50	87.79	88.6
TTA 2	83.76	95.87	87.50	89.73	90.24

Table 4: Per class accuracy - TTA

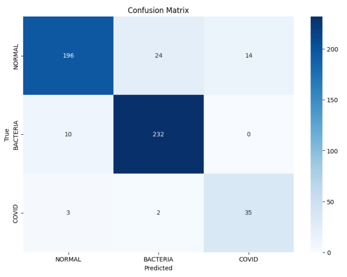
Confusion Matrix



(a) Confusion matrix without TTA



(b) Confusion matrix with model TTA 1



(c) Confusion matrix with model TTA 2

Figure 11: Comparison of confusion matrices with and without TTA

In overall, We can say that the use of TTA improved the performances, considering the two different models which uses different augmentations. This can be seen clearly by the overall accuracy and precision. Moreover, the use of TTA increased the model's ability to generalize, as the main diagonal of the confusion matrix is higher for the TTA cases we can see it more clearly In the per class accuracy improvement - The results are more uniform across classes.

4.0.3 Why using Test Time Augmentation improves the results of the classifier?

Enhances Robustness: TTA exposes the model to a wider range of variations and scenarios during inference, similar to those seen during training with data augmentation. This exposure helps the model become more robust and perform better on unseen data.

Averages Out Noise: Different augmentations can introduce variations that help smooth out the noise in the data. When the model predictions for multiple augmented versions of an image are averaged or combined, the effect of noise is reduced, leading to more accurate predictions.

Improves Confidence: By considering predictions from multiple augmented versions of an image, the model can better estimate its confidence in the final prediction. This ensemble-like approach tends to yield more reliable and confident results.

Boosts Performance in Edge Cases: Augmentations can help the model handle edge cases or less common variations in the data, leading to improved performance on challenging or ambiguous samples.

5 Additional Results

As promised on section 3.2.1, We tried to find a better model for our SimpleCNN model after realizing our model has more then 130,000,00 parameters while ResNet50 has around 25,000,000. We created a model with more convolution layers and less (and smaller) linear layers:

```python
class test_SimpleCNN(nn.Module):
    def __init__(self, num_classes=3):
        super(test_SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=5, padding=2)
        self.bn2 = nn.BatchNorm2d(128)
        self.pool2 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool3 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(512)
        self.pool4 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.fc1 = nn.Linear(512 * 8 * 8, 256)  # Adjust the input size according to the final pooling layer's output
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))
        x = self.pool2(F.relu(self.bn2(self.conv2(x))))
        x = self.pool3(F.relu(self.bn3(self.conv3(x))))
        x = self.pool4(F.relu(self.bn4(self.conv4(x))))
        x = x.view(-1, 512 * 8 * 8)  # Flatten the tensor
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

This model has 10,075,011 parameters. We trained it for 4 epochs on the over-sampled dataset as well. The training process does look more stable and less over-fitted. We reached lower training and validation accuracy scores and higher loss which were closer to the test set results.

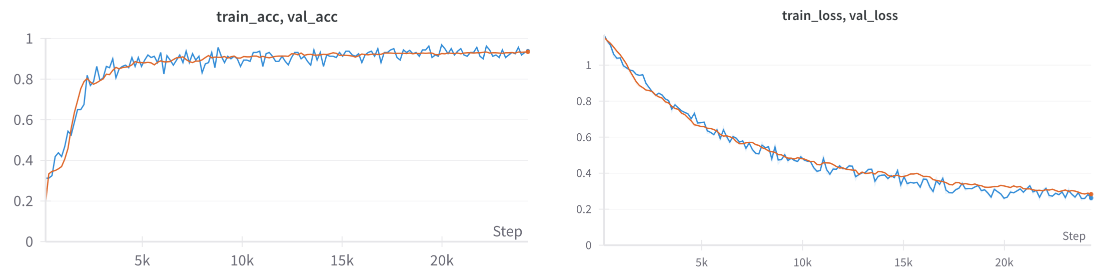


Figure 12: Training and validation loss and accuracy from training the test SimpleCNN model (Validation score in Orange)

Compare to the our original SimpleCNN model we achieve the results in Table 5.

	NORMAL (%)	BACTERIA (%)	COVID-19 (%)	Overall Accuracy	precision
SimpleCNN (134M)	70.51	97.52	87.50	84.5	86.48
SimpleCNN (10M)	75.64.06	90.08	92.50	83.72	85.41

Table 5: Evaluation of the SimpleCNN(134M parameters) vs the SimpleCNN(10M parameters)

We can see that we got very close results in term of overall accuracy and precision with much smaller model. We believe that with some more time to train and finding the best parameters we can achieve better scores.

References

[1] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[2] Daniel Schaudt et al. “Augmentation strategies for an imbalanced learning problem on a novel COVID-19 severity dataset”. In: *Scientific Reports* 13.1 (Oct. 2023), p. 18299. DOI: 10.1038/s41598-023-45532-2.

[3] Weijun Tan and Hongwei Guo. “Data Augmentation and CNN Classification For Automatic COVID-19 Diagnosis From CT-Scan Images On Small Dataset”. In: (2021), pp. 1455–1460. DOI: 10.1109/ICMLA52953.2021.00234.

[4] Chiyuan Zhang et al. “Understanding deep learning requires rethinking generalization. arXiv 2016”. In: *arXiv preprint arXiv:1611.03530* (2019).