Final Project - Milestone I

Team

- Emaan Khan - **Role**: Data-Preprocessing and Model Development
- Brandon Carrasco - **Role**: Evaluation of the Model
- Inoday Yadav - **Role**: Model Development

Context

In medicine, when a patient presents with issues within the chest, a typical procedure is to perform a chest x-ray on them. This chest x-ray is brought to radiologists, who will attempt to locate any abnormalities, analyze what they represent, and report their findings to the requesting physician.

Therein lies the problem: a radiologist must look over the chest x-ray themselves and spot any issues. While this may be a simple enough task for an experienced radiologist, the issue is that the radiologist, as a human being, is prone to many forms of human error. Long shifts with limited breaks, an inherently stressful environment, potentially tedious work, and other factors external to work; all contribute to the chance that a radiologist may make the very human error of misanalyzing a chest x-ray, which could have a major impact on the diagnosis (and indirectly, the prognosis) of the affected patient.

But how could this problem be fixed? There could be adjustments to ease the psychological burden, but if we're unable to "fix" the radiologist, then perhaps we can provide a safety net: a machine learning model — unaffected by emotions, stress, or other human limitations — that can examine a chest x-ray and provide its diagnosis.

By producing a machine learning model that can diagnose chest X-rays, we essentially provide the radiologist with an immediate second opinion on the chest X-ray. This second opinion would reinforce a radiologist's confidence in correct diagnoses, alert them to potentially misdiagnosed chest X-rays, and ultimately reduce human error's impact on chest X-ray diagnoses.

As mentioned previously, chest X-rays can be a critical component of a patient's prognosis. A misdiagnosis could lead the patient down a useless or even harmful treatment plan and waste valuable time which the patient may not have. This measure, though it's only one component in a patient's treatment, could very well tip the balance in their favor in life-or-death situations, ultimately saving human lives.

Dataset

The chest X-ray dataset has been retrieved from Kaggle (link), with the dataset being sourced on Kaggle, and created and used for a research paper (link) by Pritpal Singh. The dataset comprises 7132 samples with 4273 samples of Pneumonia, 576 samples of COVID-19, 700 samples of Tuberculosis, and 1583 samples of normal chest X-rays. The feature set will be composed of all the pixel values for each image, however, there are varying sizes of images within the entire dataset, so no exact number of features can be given. However, our preprocessing step will reduce them to a uniform size of 224 x 224, which will yield 50,176 features. Additionally, we will be using the entire dataset for training and evaluation to improve generalization, reduce overfitting and bias, and enhance the model's performance. This will allow the predicted result to be more reflective of real-world data as it exposes the model to a broader range of examples.

Proposed Solution

The chest X-ray classification problem is predictive since the goal is to correctly classify a given chest X-ray image (in .png or .jpeg format) into four distinct categories of diseases (Normal, Tuberculosis, COVID-19, or Pneumonia). For model training, we are given a dataset of chest X-ray images. The variables involved as features for this dataset will be the pixel values for each image, so there will be 50,176 features for each (planned) 224 x 224 X-ray image. Moreover, the target variable for the dataset will be the disease class label (Normal, Tuberculosis, COVID-19, or Pneumonia).

Convolutional Neural Networks (CNNs) is the ML technique we plan to use for this project, as this model is extremely effective for image classification tasks due to its ability to accurately process grid-like data, such as images. This ML model can learn complex features hierarchically, focus on local regions of pixel data, and extract complex patterns from chest X-ray images making it the best ML model for this project. To implement the machine learning pipeline using CNNs as our primary model, we will be using the following libraries/packages: NumPy, Pandas, Scikit-Learn, TensorFlow w/ Keras API, OpenCV, Matplotlib, Seaborn.

We shall preprocess our data before training through several measures. All images will be resized to uniform dimensions (i.e. 224 x 224 pixels), with their values normalized between 0 and 1, which has been demonstrated to improve model performance (regardless of the algorithm). Any categorical features (the labels associated with each image) shall be converted into one-hot encoded vectors for the purposes of multi-class classification. Afterward, we shall split the dataset into a training-validation-test split (70-15-15%) to allow for a decent balance between training and testing material. Finally, we shall apply data augmentation techniques (such as random rotations, brightness adjustment, and Gaussian blur) to our training set, artificially increasing its size and allowing our model to generalize better.

For evaluation, we intend to use accuracy, precision, recall, and F1-score as our main metrics. The latter three metrics will be especially important due to the class imbalance present within the dataset — lower scores here will be more indicative of our model's performance than pure accuracy. Confusion matrices will also be useful in analyzing the results with class imbalances in mind, as we would be better able to tell if specific classes have very poor prediction accuracy. We shall use the held-out validation set for evaluation during the training process, and afterwards test its final performance on the test set. The held-out validation set will prevent us from overfitting our model to the test set while finetuning our model. Additionally, we shall graph our loss and other metrics, for visualization purposes, so that we may intuitively understand our model's performance. The graphs shall be for our training & validation sets, and the metrics & confusion matrices will apply to all sets.

Similar datasets and challenges have been explored extensively in medical imaging and machine learning fields. Research papers like *"Detection of Multiple Diseases from Chest X-Ray Images Using Various Deep Learning Approaches"* and *"Multi-Class Classification on Chest X-Ray Images Using Convolution Neural Network"* highlight the effectiveness of CNNs in diagnosing respiratory diseases. The existing solutions demonstrated the power of deep learning in medical image classification and provided us with insights into model design, training techniques, and evaluation metrics. By adapting some of these techniques we aim to improve performance and develop a robust model specifically tailored to the dataset we selected from Kaggle.

Milestone II

Data Preprocessing

Our data preprocessing begins with iterating over all 7132 data samples located within the "Chest X_Ray Dataset" (in their respective labeled directory, for example: 'Normal', 'COVID19', etc), where we use Python's CV2 library to convert the image into a 224 x 224 grayscale image. We assign respective labels to each image and convert the X-Ray images and the labels into numpy arrays. We reshape the X-ray images and normalize the values based on their max RGB value of 255. Moreover, we one-hot encode the labels and split the entire dataset with 70% training data, 15% validation data, and 15% testing data. Finally, we save the data as *.npy arrays to be accessed within testing.

Model Specifications

Hyper-parameters:
Batch Size: 32, *Epochs*: 25, *Learning Rate*: Default for Adam optimizer (0.001), *Early Stopping Patience*: 5 epochs

Model configuration, type, and family:
Model Type: The model is a Convolution Neural Network (CNN) used for multiclass classification, which is a supervised learning process.
Model Layers: The model begins with an input layer accepting grayscale images of shape (224, 224, 1). Following this are three blocks of Conv2D (kernel size (3, 3)) -> ReLU -> MaxPool2D (pool size (2, 2)), with three corresponding Conv2D filter numbers: 32, 64, 128. Next is a fully connected layer of 128 units and ReLU activation function. Following that is a dropout layer (0.5 rate). Finally, there's an output layer with 4 outputs for each of the four classes and the softmax activation function.

Training Parameters:
Optimizer: Adam, *Loss Function*: Categorical Crossentropy, *Metric*: Accuracy, *Class Weights*: Used to handle class imbalance (calculated using compute_class_weight).

Regularization Techniques:
Early Stopping: Stopped training if validation loss did not improve for 5 consecutive epochs. Restored weights to the best-performing model during training.
Dropout: Dropout rate of 0.5 applied to the fully connected layer to reduce overfitting.

Training Results:

- Training/Validation Accuracy/Loss is reported every epoch (see below)
- **Epoch 13/25** accuracy: 0.9559 - loss: 0.0952 - val_accuracy: 0.9505 - val_loss: 0.2044. Early stopping triggered at the **13th** epoch
- Final Training Accuracy: 0.9605, Final Validation Accuracy: 0.9505, Final Training Loss: 0.0805, Final Validation Loss: 0.2044

Evaluation Methodology & Results

Our preliminary evaluation strategy follows a one-two step process:

1. During Training
    a. The loss (**categorical cross-entropy**) and accuracy scores will be calculated on the training and validation sets and printed every epoch for real-time training analysis.
2. Post-training
    a. The loss and accuracy will be calculated and printed on the test set for the final model performance.
        i. Aiming for accuracy greater than or equal to *95%*
    b. Additionally, to ensure poor performance is not masked by the dataset's class imbalance, precision, recall, and F1-Score — using weighted averages to account for class imbalance — metrics are also calculated on the test set and printed.
        i. Aiming for scores greater than or equal to *90%*
    c. Finally, a confusion matrix will be calculated and printed out on the test set, and a basic visualization of it shall be displayed for analysis.

Our final performance, on the test set, is:

Metric	Score
Accuracy	95.05%
Loss	0.1728
Precision	95.04%
Recall	95.05%
F1-Score	95.01%

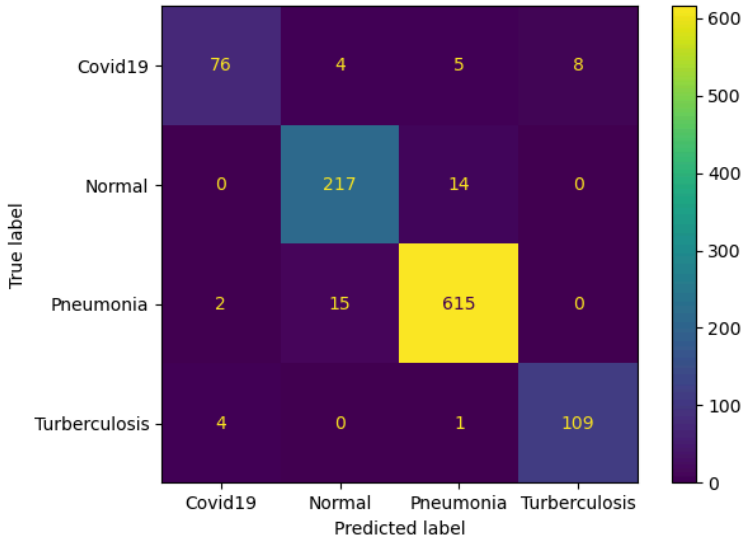


Fig 1: Basic Confusion Matrix Plot

Limitations

Our model's overall performance exceeds our specified metric scores, signaling that our current preprocessing and model strategies are working well so far. Future iterations will focus on improving performance by exploring data augmentation techniques and fine-tuning the hyperparameters of our model.

Milestone III

Data Preprocessing Improvements

Unfortunately, the data preprocessing step has remained the same. We tested various data augmentation techniques such as implementing random rotation, horizontal/vertical shift, horizontal flips, etc. The best data augmentation transformations led to a final test accuracy of 93% and a testing loss of 0.2003, which is not an improvement over the previous final accuracy and loss of 95% and 0.1728 respectively.

Model Specifications Improvements

Focusing on fine-tuning the hyperparameters of our model, we selected learning rate and batch size as the primary hyperparameters for experimenting. Smaller batches may improve generalization and faster convergence for imbalanced data, while larger ones stabilize gradient updates for faster computation. In our case, we experimented with both small (16) and larger (64) batch sizes to assess their impact on performance. Fine-tuning the learning rate can prevent overshooting the minimum and improve convergence, especially for deeper layers. Increasing the epoch size did not impact the results in the experiments due to early stopping. Despite experimenting with different batch sizes and learning rates, the results did not show improvement over the original hyperparameters of batch size 32 and a learning rate of 0.001.

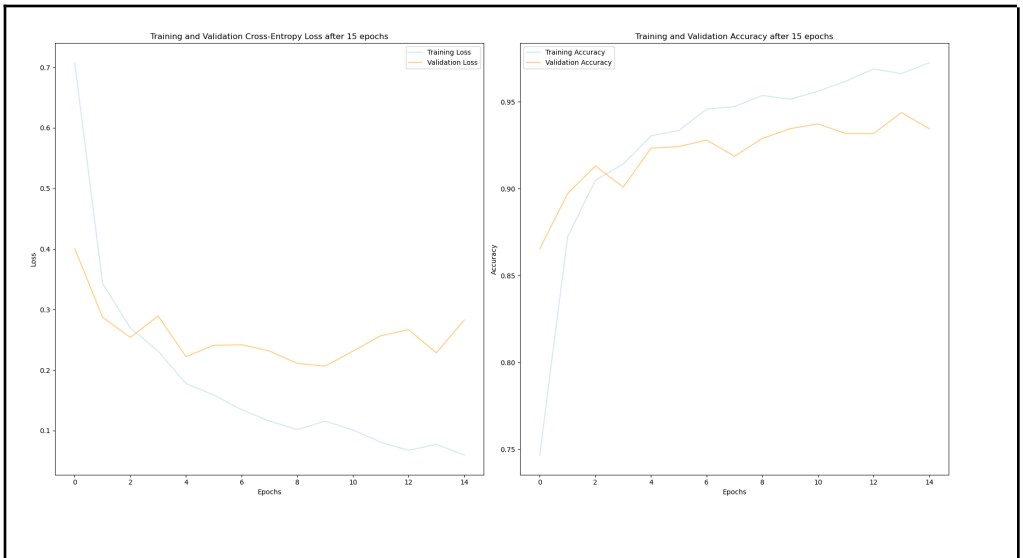
Final Evaluation & Results

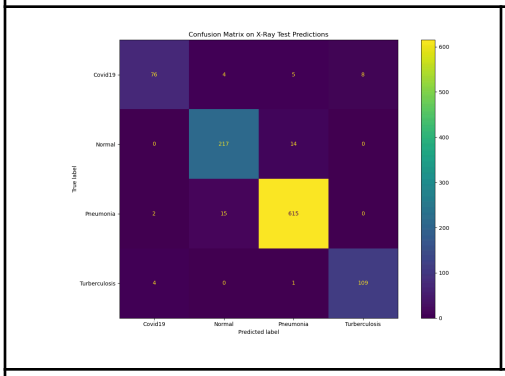
For the final evaluation of our models, our existing post-training evaluation metrics and visualizations have been extended to include: **Balanced accuracy scores**, which are balanced on classes to provide a less biased (toward most frequent class) accuracy; **ROC Curve plots**, displaying each class vs. the rest, plus a micro-averaged ROC Curve to provide an overall ROC & AUC performance; a **2D Kernel Density plot** on test data and its accompanying **hit-or-miss plots** (all brought down to 2D via t-SNE).

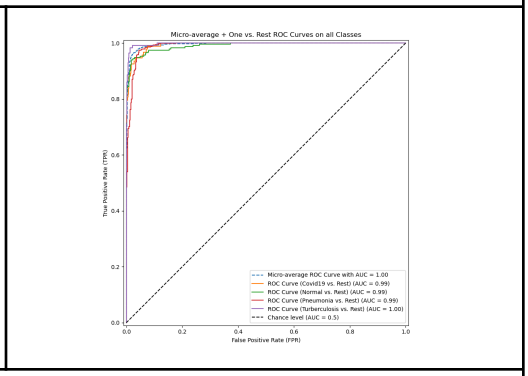
Across the four different models created — original (B=32, lr=0.001), exp. 1 (B=16, lr=0.0001), exp. 2 (B=64, lr=0.0001), exp. 3 (B=16, lr=0.001) — we have displayed their metrics results below. To save on document space, we have only displayed the plots for the best-performing model.

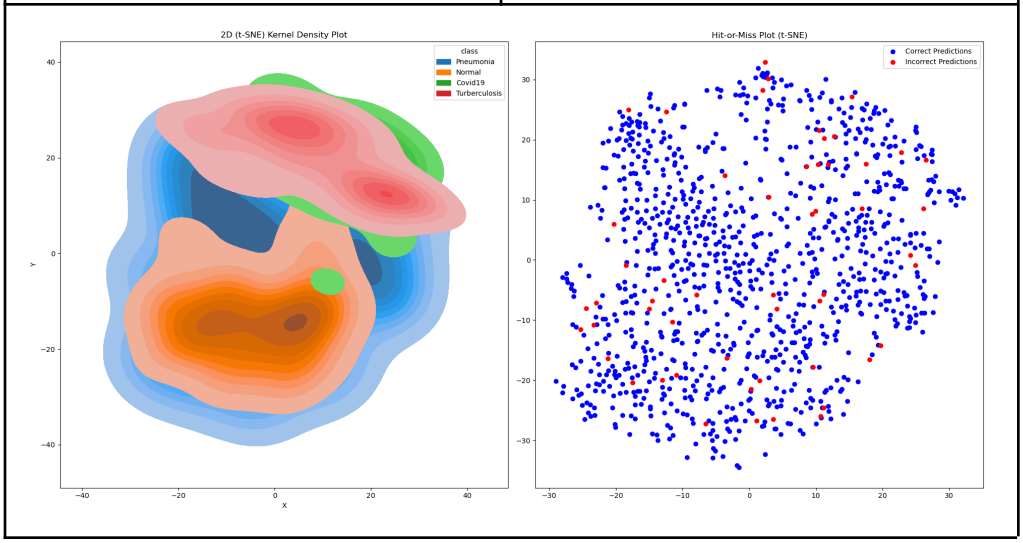
Model	Final Training Acc/Loss	Final Validation Acc/Loss	Test Acc/Loss	Balanced Accuracy	Recall	Precision	F1-Score
Original	*Acc*: **96.05%** *L*: **0.0805**	*Acc*: **95.05%** *L*: **0.2044**	*Acc*: **95.05%** *L*: **0.1728**	**92.15%**	**95.04 %**	**95.05%**	**95.01%**
Exp. 1	*Acc*: **95.41%** *L*: **0.1075**	*Acc*: **93.27%** *L*: **0.2304**	*Acc*: **94.58%** *L*: **0.1726**	**93.26%**	**94.65 %**	**94.58%**	**94.60%**
Exp. 2	*Acc*: **93.71%** *L*: **0.1425**	*Acc*: **93.27%** *L*: **0.2073**	*Acc*: **93.93%** *L*: **0.1818**	**92.11%**	**93.98 %**	**93.93%**	**93.93%**
Exp. 3	*Acc*: **95.09%** *L*: **0.1209**	*Acc*: **94.02%** *L*: **0.2208**	*Acc*: **94.21%** *L*: **0.1809**	**91.99%**	**94.22 %**	**94.21%**	**94.18%**

Exp. 2 and Exp. 3 are worse than our original model in every respect. Exp. 1, however, displays slightly better-balanced accuracy on the test set at the cost of all other metrics. It performs overall classifications better across all classes (if all classes are weighted equally in importance) than the original model. We have selected the original model as our best for its overall performance. Its evaluation visualizations are displayed below.









Our Confusion Matrices and ROC curves show excellent performance, and our training & test graphs display no issues with model training. The KDE plot informs us that Covid-19 & TB tend to share similar variations in data, and the hit-or-miss plot confirms that our model tends to misclassify instances at these areas of similarity.

Limitations

From our preprocessing and model experimentation results, it's clear that we've squeezed as much performance out of technical improvements as possible. Our KDE and hit-or-miss plot has instead shown that improvement can instead be made through sampling (and other) methods targeting our model's weakness in classifying between certain classes (such as Covid-19 & TB).

References

Looney, O. (2018, August 23). *Visualizing Multiclass Classification Results*. OWL. https://www.oranlooney.com/post/viz-tsne/

Scikit-Learn Developers. *Multiclass Receiver Operating Characteristic (ROC)*. https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for Multi-Class Classification: an Overview. *arXiv*, arXiv:2008:05756. https://doi.org/10.48550/arXiv.2008.05756

Kabir, M. M., et al. (2024). Detection of covid-19, pneumonia, and tuberculosis from radiographs using AI-driven knowledge distillation. *Heliyon*, *10*(5).https://doi.org/10.1016/j.heliyon.2024.e26801