DeepMed: Unveiling Health with Neural Networks

Dataset Overview :

The Chest X-ray (Covid-19 & Pneumonia) dataset is a collection of 6,432 chest X-ray images, developed to help create deep learning models for medical image classification and specifically to classify three important classes (COVID-19, Pneumonia, and Normal (healthy lungs)). The dataset consists of two subsets to be used for training (80%) and testing (20%), with the images from each subset being arranged into corresponding folders for the three classes. Images in the dataset are typically in .jpg or .png format, and are frontal chest X-ray images.

The chest x-ray dataset meets a critical need in our global fight against COVID-19. The current gold standard for COVID-19 diagnosis is RT-PCR but is highly limited in extremely costly, long turn-around, and currently unavailable in some under-resourced regions of the world. Chest x-rays on the other hand, can be done in minutes, are far less costly, and remain universally available, all of which make chest x-rays a good alternative tool for preliminary screening and aid in the diagnostic process.

Both COVID-19 and pneumonia present the characteristics of a lung infection, therefore they should present similar appearances in radiographic images. Since both COVID-19 and pneumonia are present in the dataset, models will learn how to differentiate between very subtle differences in categories, which will improve diagnostic accuracy. The Normal class will provide a method for the model to distinguish clearly between infected and healthy cases.

Main Statistics of Dataset:

```
print(f"Mean of the Pixel: {mean_of_image.numpy()}")
print(f"Standard Deviation of the Pixel: {std_of_image.numpy()}")
print(f"Minimum Pixel of Image: {min_pixel_of_image:.4f}")
print(f"Maximum Pixel of Image: {max_pixel_of_image:.4f}")
```

```
Mean of the Pixel: [0.4923709  0.49246326 0.4925521 ]
Standard Deviation of the Pixel: [0.22802506 0.22802255 0.22805242]
Minimum Pixel of Image: 0.0000
Maximum Pixel of Image: 1.0000
```

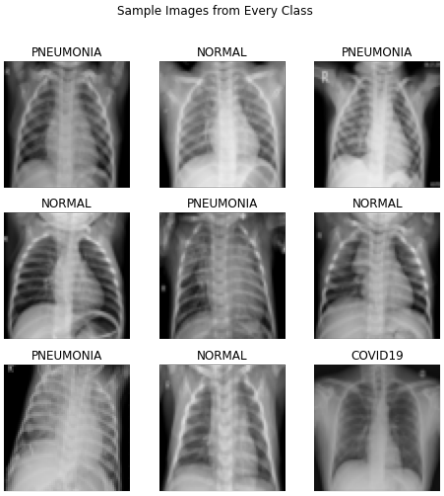
The pixel values of all of the chest X-ray images indicate that the data has been normalized to a [0, 1] range with minimum pixel values 0.0000 and maximum pixel values of 1.0000. The mean pixel values for the red, green, and blue channels ([0.4924, 0.4925, 0.4926] respectively) indicate that brightness was evenly balanced and there was no major color bias in the images. The standard deviation values for the data ([0.2280, 0.2280, 0.2281] respectively) indicate a moderate level of contrast in the images. Overall, these statistics indicate that the data are clean and ready for input into deep learning models, and will produce stable and consistent behavior during training through the neuron layers of the neural network.

Pre-processing Steps :

The preprocessing pipeline for this project involves multiple image transformations and a class balancing approach to improve model performance. All images are resized to a uniform image dimension (64×64 pixels). Data augmentation is applied to the training set through random horizontal flips and small amounts of rotation (up to 10 degrees), which helps increase the number of training samples and decrease overfitting. All training and test images are converted to tensors, and are normalized by the mean and standard deviation of the dataset, to guarantee all images have the same input distribution. To take into account the class imbalance, a weighted random sampler is used. The weighted random sampler computes sampling weights, which are inversely proportional to class frequencies, therefore affording more importance to minority classes (i.e. COVID-19). Using a weighted random sampler will ensure that each mini-batch during training will contain classes with an equal representation, improving the networks ability to generalize across all classes.

Visualizations :

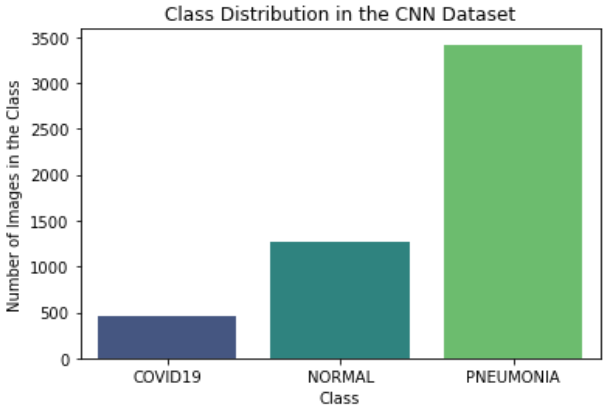
1)



The image is a composite grid of 9 chest X-ray scans presented in a 3x3 layout, presenting a current frame from a sequence of scans. Each scan illustrates the thoracic cavity of a given patient, showing primarily the lungs, heart, and ribs. X-ray scans provide a two-dimensional image of the thoracic cavity in shades of gray. The lighter the area on the image, the denser an area is (e.g., bone) and the darker the area, the less dense an area is (e.g., air in the lungs). The data have a uniform level of gray-scale contrast and resolution suggesting that they have been preprocessed and resized (to something like 512x512, or to simply normalize for use in a machine learning pipeline).

The composite grid of X-ray scans is likely just a small sample from the dataset. Each image has a consistent orientation of the patient and the degree of exposure (see the footer for detailed X-ray annotation). This suggests that not only was the data processed but it was normalizable especially for input into a convolutional neural network (CNN) for classification tasks (e.g., determining if the case is COVID-19, pneumonia, or normal). Using a grid of images allows for a friendly visual inspection of the overall image quality and diversity of classification (labeling).

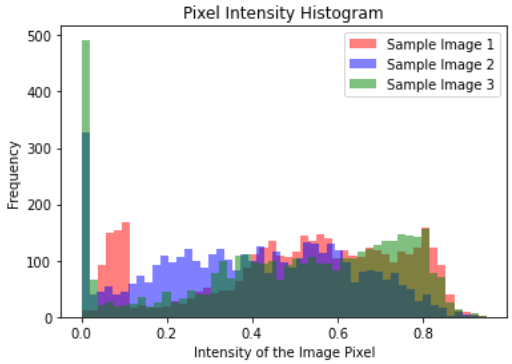
2)



The bar chart shows the class distribution of the chest X-ray dataset, which has three classes - COVID-19, Pneumonia, and Normal. From the chart, it is clear that the dataset is imbalanced, with the COVID-19 class has

the least samples, then Pneumonia, and with the Normal class having the most images. The imbalance makes it difficult to train the model fairly with respect to the class distributions as a model may favor the majority class and therefore have skewed predictions from both COVID-19 and Pneumonia samples. In practice, approaches such as weighted loss functions, oversampling, and/or data augmentation are typically used so that the model is able to learn to generalize across all classes equally with respect to each. The bar chart also provides a quick visual description of the distribution of classes within the dataset and shows the class imbalance that must be considered in medical imaging tasks.

3)



The histogram depicts the respective pixel intensity distributions for the three sample chest X-ray images. The red represents Sample Image 1, the blue represents Sample Image 2, and the green represents Sample Image 3. The x-axis represents normalized pixel values from 0 through 1 and the y-axis corresponds to the frequency of those pixel values within the specified sample image.

In reviewing the plot, we find that in most instances, a greater assembly of pixels tended towards the lower pixel intensities on the left of the histogram that are characteristic of most medical X-rays where large parts of the display are dark (air-filled) spaces representing the lungs. Variation in mid and high intensity values captures structural aspects of the human body, such as bones and soft tissue. The nearly overlapping (or shared) region of the three sample images indicates that similar pixel intensities are present across images, however the distinct peaks indicate differences in pixel brightness or contrast among the image samples.

Models :

Model 1 : VGG Architecture

The model described here is a custom implementation of the VGG16 architecture for classifying chest X-ray images into a user-defined number of classes. The model has two main components, feature extractor and classifier. The feature extractor builds the model using a deep stack of convolutional layers grouped into five blocks, with a max pooling layer following after each block to increasingly decrease the spatial resolution in a controlled manner. The convolutional layer in each block never down-sampled spatial resolution. All convolutional layers have a 3×3 kernel with padding used to maintain spatial resolution, apply ReLU activations after convolutions to add non-linearity to the architecture, and consistently increase for feature extraction from 64 to 512 filters for learning features at increasing levels of complexity.

Following feature extraction, the model turns the output into a flattened representation, followed by a fully connected classifier with two ReLU activations and 4096 units each, dropout (0.5) no regularization, and finally maps it to the user-defined number of target classes. The architecture described here provides excellent potential for use with most image classification tasks. The architecture mimics the original VGG16 design, with necessary adaptations to actually indequately work with resized 64×64 inputs, making very efficient use of computational resources for a medical imaging applications.

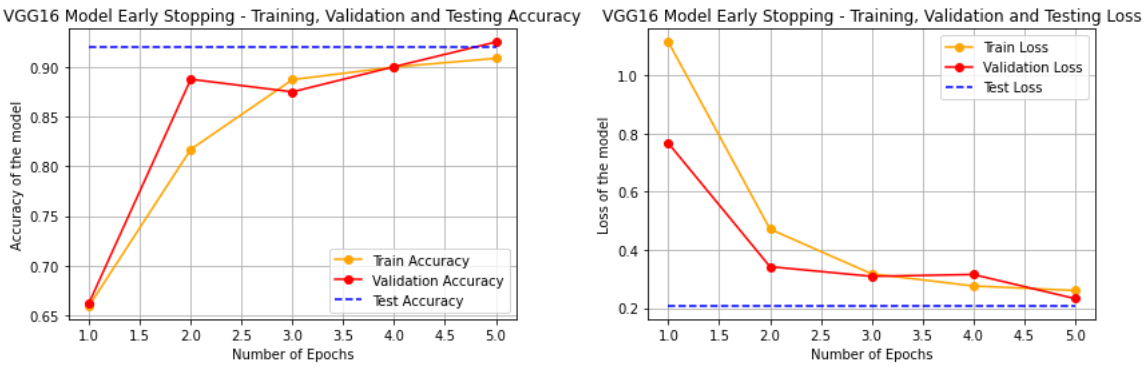
Training :

The VGG16 model was trained for a total of 5 epochs. Each epoch led to substantial improvements in both the training and validation aspects. The first epoch produced a training accuracy of 64.87% and a validation accuracy of 69.92% - indications that the model was learning good features, but not that the model was able to generalize those features well yet. Over the epochs, training accuracy improved steadily, and training loss decreased from 0.8657 to 0.3371, while validation loss decreased from 0.7539 to 0.2812. By the 5th epoch, the results yielded a training accuracy of 87.52%, and a validation accuracy of 90.52% - very good results indicating that the model generalized well. Both training and validation loss consistently improved and reduced, and the model's accuracy continued to increase throughout the training which is a great sign that the model learned and did not overfit. Each epoch took approximately 100 seconds to run, which was efficient given the size of the dataset and nature of the model architecture.

Evaluation:

The model's performance on the held-out test set produced a test accuracy of 92.86%, demonstrating that the model generalises well to different data. The low test loss of 0.2093 also indicates the model performed a reliable value. In terms of performance by class, the model reported precision of 92.53%, therefore making few false-positive predictions. The recall score of 90.39%, shows the model was able to identify a majority of the positive cases correctly. The mean F1 score of 91.29% balanced precision and recall measures. These values not only show the model is an accurate predicter, but provides confidence that the model will also be robust in terms of being able to distinguish between classes, and would be suitable for real-world deployment as a medical image diagnostic application.

Graphs:



The accuracy and loss plots provide an excellent summary of how the model learned through the five training epochs. By reviewing their shapes and values, we can see that the model was effective, and generalization was retained, as shown by the evaluation of the accuracy and loss criteria. In the accuracy plot, the training accuracy starts around 65% and steadily rises to around 88%. The validation accuracy starts at around 70%, and steadily rises to 91%, where it runs above or parallel to the training accuracy— this shows a good amount of generalization without overfitting. The blue dashed line for the test accuracy at 92.86% shows that the model performed very well on unseen data.

In summary, both accuracy plots evidenced effective learning. The loss plot evidenced consistent reductions in both training and validation loss throughout the epochs with training loss going from about 0.86 to 0.34, and validation loss from 0.75 to 0.28, while tracking each other. The test loss (again, the blue dashed line) at 0.2093 is lower than both final training and validation losses. Overall, both accuracy and loss metrics depict trends that

show that the model has learned well, generalized well, and performed well on the test set; thereby confirming the efficacy of the architecture overall, as well confirming an effective training pipeline.

Model 2 : ResNet18

The architecture is based on two primary objects: the ResidualBlock class and the main, ResNet18, class. A Residual Block consists of two convolutional layers (3×3 kernels), batch normalization, and ReLU activation with the most important aspect of the block being the skip connection which allows adding the input (identity) directly to the output of the block. If the input and output shapes are different because a stride has been defined or the channels have changed, the residual block will down sample the input to the same shape, typically with a 1×1 convolution.

The ResNet18 class first executes an initial Convolutional layer (with 7×7 kernel and stride 2), followed by batch normalization, ReLU activation, and max pooling, this will help to reduce the spatial dimensions early on. The main part of the network is done in the final 4 residual layers using make layer() function. Each residual layer is made up of 2 residual blocks, starting from 64 filters and reaching 512, with the spatial resolutions reducing along the way. The hierarchical structure implies the use of the parameter to learn more abstract features progressively as it goes through each layer. The resulting feature map from the residual layers has global average pooling implemented to allow for better translation invariance, followed by adaptive average pooling (to 1×1) before being flattened into a vector and passed through a fully connected layer to generate logits corresponding to the required number of classes, in this case 3.

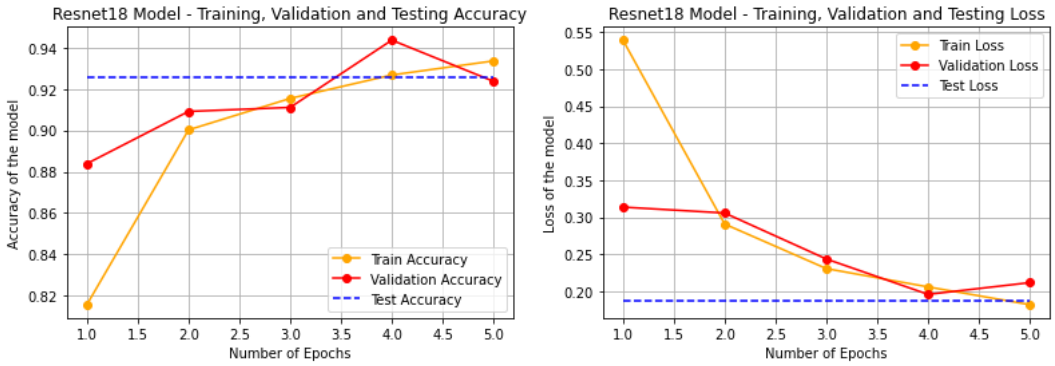
Training :

The training results demonstrated across five epochs indicate that the model: ResNet-18 learned appropriately and was able to generalize its learning to never-before-seen examples. The model began in epoch 1 with a training accuracy of 81.56% and validation accuracy of 88.39% and improved progressively with epoch 5 results of 93.36% training accuracy and 92.37% validation accuracy. The validation loss and training loss were also generally reliable and decreasing with epoch 4 producing the lowest validation loss value of 0.1956. ResNet-18 training times also provided great computational efficiency averaging 65 seconds. Overall, the training phase was reliable and stable; it showed strong convergence, very little overfitting and outstanding generalization. Therefore ResNet-18 is suitable for chest X-ray classification jobs.

Evaluation:

The model obtained a testing accuracy of 92.55%, indicating that the model successfully classified the test-given images correctly in a lot of the cases. The model also had a test loss of 0.1875 and is quite low since the lower the loss the model has, the more confident and less erroneous the prediction(s). The precision score of 91.04% means, when it predicted (for example) COVID-19 or pneumonia, there was a 91% chance it was actually correct, which means a low false positive score. The model's recall of 94.29% demonstrates its ability to identify nearly all relevant cases meaning that it had a low false negative rate, and a low false negative rate is essential for medical cases (i.e., a lack of positive detection when there is a positive case). The F1 score of 92.35% is another measure of balanced performance corresponding to both precision and recall - all classes were performing well as a whole. The scores indicate that the ResNet-18 model generalizes well, as well, that it is a reliable and stable enough model for detecting COVID-19 and pneumonia in X-ray images.

Graphs :



The plots for training and validation provide an illustrative overview of ResNet-18's performance across five epochs. The accuracy plot shows that training accuracy started at 81.56% for epoch 1 and had steadily increased to 93.36% by epoch 5. The validation accuracy plot shows that validation accuracy was also consistent, starting at 88.39% and rising to 94.36% in epoch 4 before a slight decrease to 92.37% at the final epoch. The dashed blue line in the upper left corner of the accuracy plots represents test accuracy at 92.55%. All three accuracy plots suggest the model is generalizing well. The fact that validation accuracy and test accuracy closely match suggests the model has a good generalization ability. Also, there were no signs of substantial overfitting.

This pattern is also present in the loss plot. The training loss started at 0.5393 in epoch 1 and decreased steadily to 0.1816 after 5 epochs, which indicates effective learning. The validation loss started at 0.3136 then had it's lowest value of 0.1956 in epoch 4 then increased to 0.2114 at the end of the last epoch. The test loss was also low at 0.1875 which is better than both final training and validation losses, is also consistent with the model's ability to generalize. Overall, the plots demonstrate that the model learned effectively and performed well in training, validation, and testing while maintaining a reasonable balance between bias and variance throughout the training process.

Weight initialization he in ResNet18 Model:

He initialization is a weight initialization method developed specifically for deep neural networks using the ReLU (Rectified Linear Unit) activations. It attempts to solve the vanishing/exploding gradient issue, by keeping the variance of activations the same between layers (and in practice, to make this variance similar between layers is done primarily with deep networks such as ResNet-18).

By calling the function weights_initialisation(resnet18_model, "he"), the model's weights (convolutional, fully connected) are initialized with weight values from a distribution appropriate for ReLU layers. This helps the convergence rate during training and ultimately provides better model stability and accuracy during inference, which is particularly important in deeper networks. When you are training models from scratch, this is especially important because it provides an appropriate start for the optimizer to begin efficient learning.

Training:

The model in Epoch 1 shows a training accuracy of 85.48% and a validation accuracy of 88.34% with training and validation losses at 0.3715 and 0.2972. This indicates a strong initial learning has taken place. By Epoch 2, the training accuracy rises to 91.74% and the validation accuracy increases to 90.52% with lower training and validation losses, indicating better generalization.

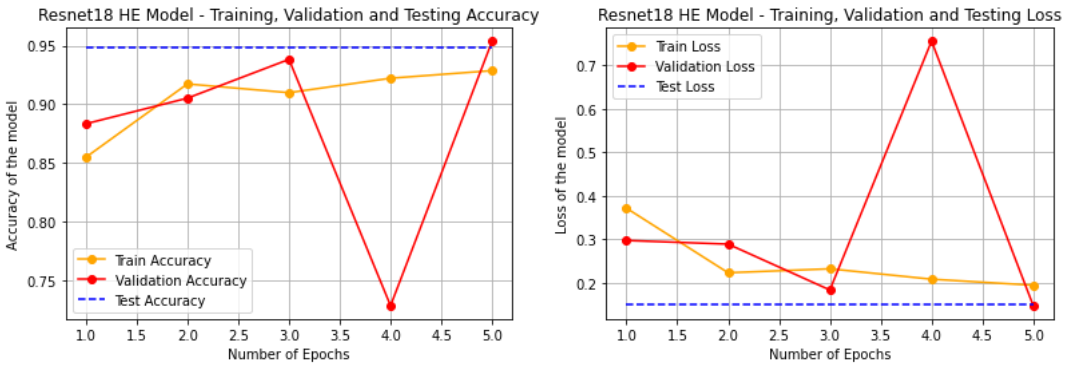
In Epoch 3 the training accuracy is still strong at 90.99% but the validation accuracy jumps to 93.83% with a validation loss of 0.1834 suggesting the model is learning highly discriminative features. In Epoch 4 there is a large upward spike for validation loss at 0.7554 with a large downward spike in validation accuracy of 72.84%

even though the training accuracy has improved to 92.22%. This indicates some form of temporary overfitting or instability in the model due to a batch anomaly or some type of 'fluctuation' in the optimizer. By Epoch 5 our model has a very good rebound achieving 95.38% validation accuracy with a very low validation loss of 0.1473 and a solid training accuracy of 92.87%.

Evaluation:

The model yielded a testing accuracy of 94.88%, which indicates it correctly classified almost 95 out of every 100 chest X-ray images. The test loss of 0.1487 is low, indicating that it can claim with a high level of confidence and very low error in its predictions. The precision score of 93.99% indicates that when the model makes a prediction of a given class (e.g. COVID-19 or Pneumonia), it is correct most of the time having a low false positive rate. The recall score of 94.63% indicates that the model is also good at identifying actual positives resulting in very low false negatives. Together these scores combine to an excellent F1 score of 94.30%, which is the harmonic mean of precision and recall, providing a balanced level of model performance. These results show that the model is not only accurate, but it is also precise and sensitive across all classes therefore can be trusted in the real-world clinical diagnostic context of detecting conditions such as COVID-19 and pneumonia from chest X-rays.

Graphs:



The training accuracy (yellow line) demonstrates a linear upward trend as it increases from approximately 85% to nearly 93%. This shows the model is learning properly. The validation accuracy (red line) also improves from epochs 0-3, peaking in epoch 3 (approximately 95% accuracy) before linearly decreasing in epoch 4 (to significantly less than 60% accuracy) before recovering in epoch 5 (the fifth epoch). The test accuracy (blue dashed line) is consistent and high (94.88%) indicating that the model seems to generalize well despite the major decline in validation accuracy.

In the loss plot, it is not surprising that the training loss (yellow line) decreases so consistently on a downward trend, as it is a representation of successful learning. Additionally, the validation loss (red line) decreases across the first three epochs - but the validations loss spikes at epoch 4 exactly where validation accuracy fell off. The spike in validation accuracy and loss is likely due to an error at the batch-level system, an outlier from data shuffle, or a fluctuation associated with overfitting. In epoch 5, validation loss drops back down to its lowest point - even lower than the test loss - this confirms good recovery and optimum model performance. Finally, the test loss (blue dashed line) remains consistently low.

Model 3: MobileNet V3

With transfer learning, the MobileNetV3-Large model has been modified for chest X-ray classification tasks. MobileNetV3-Large is first loaded with pretrained weights using models.mobilenet_v3_large(pretrained=True), which will allow the model to instantiate weights from the original mobilenet model. The pretrained weights are the model's parameters from training on the ImageNet dataset. These weights are creating general visual characteristics (such as edges, textures, and shapes) that can be generalized into using medical imaging. Pretrained weights will transfer and not leave the model without a large amount of labeled dataset and training time.

In order to work with the task of classifying chest X-rays into a predetermined number of classes (e.g. COVID-19, Pneumonia, Normal), we will replace the last layer of the classifier, mobilenet_v3_model.classifier[3], with a new linear layer using nn.Linear. The output features will equal the number_of_classes. This modification allows the new model to produce output predictions relevant to the new dataset. At this point, the model is moved to either the cpu or the gpu hardware using .to(device) to enable efficient model training and inference.After moving the model, the weights_initialisation(mobilenet_v3_model, "he") function applies He initialization to the model and is executed on any new trainable layers, particularly the classifier added to the model.

Training:

The model achieved 72.72% training accuracy in Epoch 1, but the validation loss is exceedingly high at 634.9761, and the validation accuracy is relatively low with 66.42%. This immense loss means either: (1) the numerical instabilities are severe (for example: weights were not initialized, too high of a learning rate, or incompatible layer structures), or (2) the loss function is misaligned during the early phases of model training. In Epoch 2, the model is not only better in training (80.56% accuracy and 0.4645 training loss), but the model also has a lower validation loss (24.4947), even if it displayed erratic behavior, and the model had a considerably lower validation accuracy (24.68%).

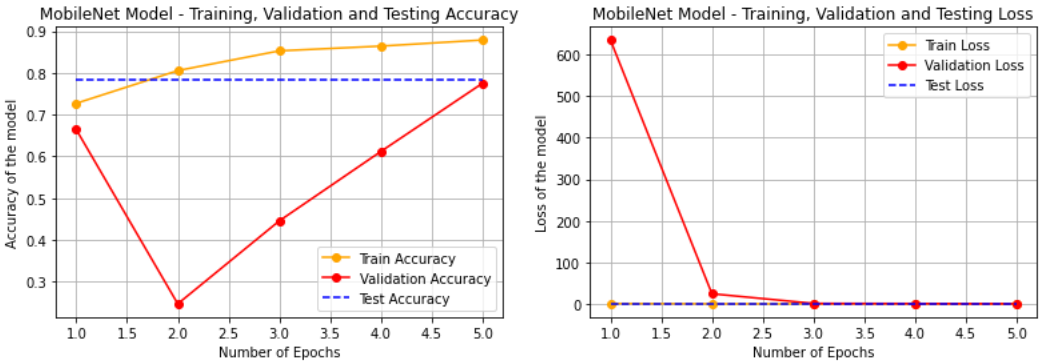
In addition, starting in Epoch 3, the training begins to stabilize. The training accuracy has reached 85.29%, and continues to improve through Epoch 5 (87.88%), while the training loss continues to drop. The validation loss oscillated early on, but began a consistent downward trend (1.2670 - 0.9307 - 0.5253); the final epoch had improved validation accuracy (77.50%) from the first few epochs. Overall, we see the model beginning to generalize better than some of the earlier epochs of training, and we can draw some conclusions about how long it takes the new classifier layer to adapt appropriately.

Evaluation:

The model achieved a testing accuracy of 78.49%, meaning it correctly predicted the class labels for roughly 4 out of every 5 chest X-ray images in the unseen test dataset. While this suggests decent overall classification ability, the relatively high test loss of 0.5029 indicates that the model's confidence in predictions may be inconsistent, and it is still making significant errors, particularly on harder or minority-class samples.

The precision score of 83.74% shows that when the model predicts a given class (e.g., COVID-19 or Pneumonia), it is correct most of the time. This implies a low false positive rate**,** which is valuable in avoiding unnecessary alarms in medical applications. However, the recall score is lower at 70.46%, meaning the model fails to identify nearly 30% of actual positive cases, which is a critical issue in healthcare scenarios where missing a true case (false negatives) can be dangerous. The F1 score of 69.25%, which balances both precision and recall, confirms that the model’s performance is skewed, possibly due to class imbalance, insufficient fine-tuning, or limited learning capacity in the new classification head.

Graphs:



The graphs showing the training and validation data summarize the learning dynamics of the MobileNetV3-Large architecture model over five epochs. Starting with the training accuracy, the model dramatically improves its performance level from roughly 72% to 88%, indicating that the model is leveraging the training data to learn. In stark contrast, the validation accuracy drops significantly after the first epoch, falling below 25% in epoch 2, and then gradually recovers to an increase of 78%. The instability represented by the drop and subsequent recovery of the validation accuracy suggests that there was an early learning segmentation based on an abrupt jet, data batches that were unbalanced in terms their composition, or errant initialization weights. The test accuracy as represented by the horizontal blue dashed line close to the final validation accuracy of 78.49% enables acceptable generalization.

With respect to loss, the loss graph indicates that the training loss declines from an initial loss of 1.05 to low loss of 0.31, providing certainty that the model is able to continue to minimize error by training on this data. However, the validation loss rapidly spike to 635 in epoch 2, likely from either an outlier or instability that was dependent upon the training, and quickly decrease to stabilization in epochs 3 through 5. Nearing the test loss of 0.5029. This observation shows the model's ability to regain some control or learning trajectory after what was considered an epoch from which it would have not been recoverable the preceding epoch was an outlier and was representative of learning, while characterized by uplift from 635 to falling to a lower score. Overall, while the model appears to learn and generalize well, this learning is accompanied by volatility early on in the model development process and future work to refine learning includes better learning rate scheduling, batch normalization tuning or a longer warm-up phase to create a mapping for convergence.

Model 4: GoogleNet

GoogLeNet (Inception v1) is an image classification architecture that is efficient and impactful. At the high level, it includes a stem block that contains a 7×7 convolution followed by max pooling, and then combines 1×1 and 3×3 convolutions that quickly reduce spatial dimensions while retaining important low-level features. The main content of the architecture is constructed from a sequence of Inception modules, where the input is processed through four parallel paths: one captures a 1×1 convolution; the second a 1×1 and 3×3 convolution; the third a 1×1 and a 5×5 convolution; and the last is a max-pooling layer followed by a 1×1 convolution. The outputs from these branches are concatenated to aggregate features that were extracted from multiple scale structures, which allows the model to learn both the minutiae and the larger structural patterns of images.

The model is organized into three stages: Block 3, Block 4, and Block 5. Each block is comprised of a series of Inceptions using the same process described above. Of the three blocks, Block 4 contains most of the layers, allowing for deep hierarchical representation learning of features. After the last Inception block, the model changes to global average pooling to re-express the feature maps from the output of synthesizing into a 1 × 1 spatial size, followed by dropout for regularization and a fully connected layer with output for the total number of classes. Overall, the modular, multi-scale design of GoogLeNet allows for efficiency, effectiveness, and

accuracy in process such as chest X-ray classification, where the model must learn to identify subtle features and structures that are either small or larger than the subtle features.

Training:

At the start of Epoch 1, the model was clearly still learning the general patterns in the data, considering it had a somewhat high training loss of 1.5345 and a training accuracy of 72.52%. Even so, the model had a substantially lower validation loss of 0.6451 and a reasonably high validation accuracy of 83.67%, which indicated it appeared to have trained reasonably and had reasonably generalized to some extent. In Epoch 2, the model improved just a little more, training loss of 0.3697 and a training accuracy of 86.84% with a validation loss of 0.2547 and a validation accuracy of 91.21%, which confirmed a reasonable amount of learning had occurred as well.

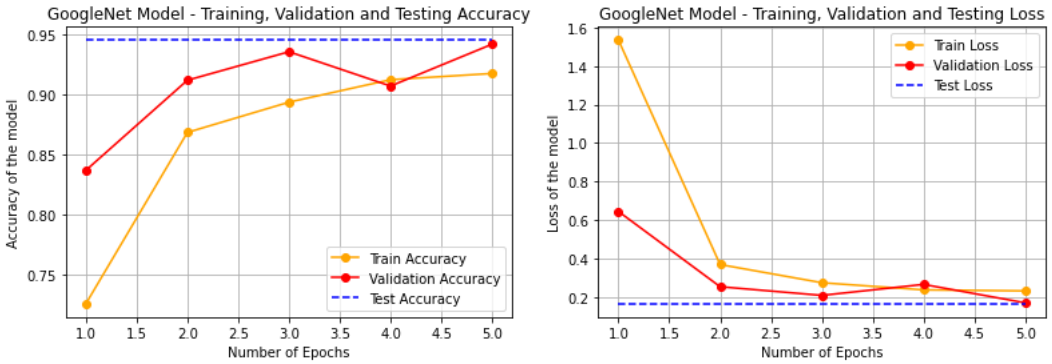
The model performed somewhat better in Epoch 3, again reporting improvement with a training accuracy of 89.37% and a validation accuracy of 93.59%, while at the time the validation loss had finally dropped to 0.2097. In Epoch 4, the model improved the training accuracy again, now to 91.25 % but the validation accuracy had plateaued and dipped to 90.72% while the validation loss apparently went the wrong way again with a marginal rise to 0.2667, so while improving overall, the model may have actually overfit at one moment or there was numerical feedback establishing a small fluctuation in the learning. In Epoch 5, the model had self-corrected itself as well, getting to a training accuracy of 91.77% and a validation accuracy of its all time best to date at 94.22%, and finally setting a validation loss of 0.1709, which meant it was already showing its best performance it achieved throughout the epochs. Overall, the model indicates that there was reasonable progression, learning, convergence, and generalization.

Evaluation:

The model's predictive accuracy on the test dataset of 94.57% means it was able to accurately classify almost 95 of 100 chest X-ray images. This measurement of internal confirmatory validity not only validates the ML model's ability to train on the training dataset, but reinforces the fact that generalization of the pretrained ML model on new data is consistent with the original algorithm design. The test loss of 0.1659 is a relatively small value, which demonstrates the model's confidence level in its predictions and probability estimates of each class is characterized by a small value of error.

The precision score of 95.23% means, when the model predicted a certain class e.g. COVID-19 or Pneumonia, it was correct larger than 95% of the time, thus resulting in a minimal false positive rate - which is an important measurement in medical diagnosis vaccine administration and interoperability, because the consequences of misdiagnosing a healthy patient as diseased. The recall of 92.41% value indicates the model performed well in identifying true positive cases with only a few offending false negatives, this is a particularly important measurement in all medical applications because the incorrect inference of the absence of a case can mean the difference in health and disease for patients. The F1 Score of 93.75% (the harmonic mean of precision and recall) demonstrates a strong balance between sensitivity and specificity of the underlying ML model thus making it a useful tool for all medical applications using medical imaging classification data.

Graphs:



The training accuracy (yellow line) starts at a relatively stable low of 72.5% in the first epoch and while steady improvement occurs as predicted, it reaches the baseline of 91.77% at the fifth epoch, which signals effective learning from the training dataset. The validation accuracy (red line), in contrast, begins at a higher 83.67% which indicates good initial generalization. The section of the graph shows that the validation accuracy reaches a peak of 93.59% in epoch 3 decreasing slightly to 90.72% in compound epoch 4 and then climbing again to ending at a high of 94.22% at epoch 5. Measures of validation accuracy or loss may be impacted by a multiplicity of phenomena, and the slight overfitting seen in epoch 4 could be a result of validation data and/or general data sensitivity or, the sensitivity of the learning rate considering the next result of test accuracy. In summary the test accuracy (blue dashed line) does not fluctuate at all and remains high at 94.57%, which re-affirms good generalization to non-trained data and is closely related to validation performance.

The training loss (yellow line) is high at 1.5345 during epoch 1 and then quickly falls to 0.2330 by epoch 5 after reductions in prediction error were shown during training. The validation loss (red line) decreases from epoch 2 starting at 0.6451 and moving to a low completion state of 0.1709 by epoch 5. The validation loss is shown to spike during epoch 4 slightly but recovers very quickly, however, since we are not concerned with spuriousity during training -please refer to training loss. The test loss (blue dashed line) is recorded at 0.1659 which is very close to the final validation loss, re-affirming that overfitting occurred minimally before recovery, and that generalization potential is quite strong.

Hyperparameter Tuning of GoogleNet:

GoogLeNet model will be trained with a specific set of hyperparameters optimized to achieve the best performance on a classification task such as chest X-ray classification. In this case, a Stochastic Gradient Descent (SGD) optimizer will be used; it is popular due to its simplicity, stability, and effective use for large scale deep learning when learning rates are adjusted correctly. In this model, a batch-size of 32 will be used in training meaning the model updates its weights after ingesting every 32 training samples. This batch-size provides a reasonable balance between speed of training and convergence stability.

The model is likely to have undergone He initialization for weights prior to training, which is an effective weight initialization strategy for deep networks that use ReLU activations/ This can keep steady gradients flowing in the model and improve convergence especially for deep models like GoogLeNet. Once training is complete, the model parameters can be saved using torch.save() which allows the trained weights for inference or further tuning to be reused without having to train from scratch.

Training:

During Epoch 1, the model performed extremely well, achieving a training accuracy of 92.32% and a validation accuracy of 93.34%, along with low training and validation losses of 0.2161 and 0.1870 respectively, indicating

that the model was well initialized and learnt very effectively. During Epoch 2 training accuracy dropped slightly to 91.15%, while the validation accuracy increased at 94.02%, which implies that generalization was still excellent despite a small increase in training loss.

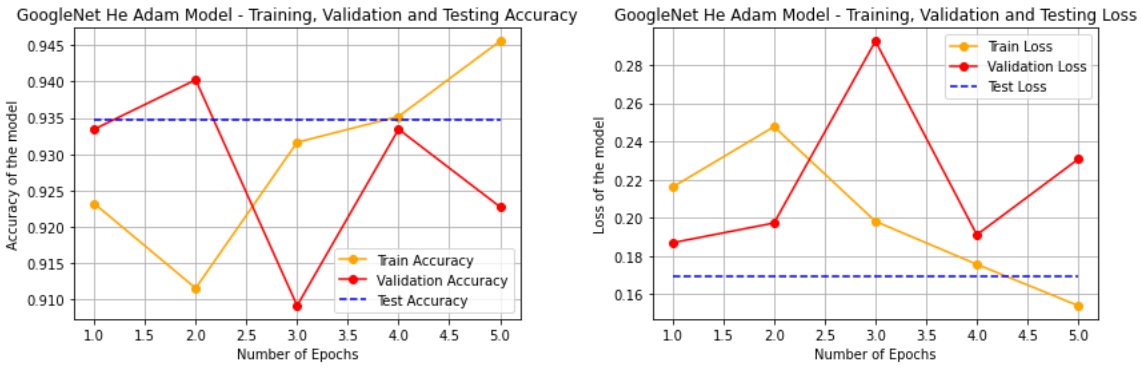
In Epoch 3 training accuracy increased back to 93.16% well above the average training accuracy of Epoch 1 and 2; however, validation loss increased to 0.2925, while the validation accuracy dropped to 90.91%, indicating there may have been a small amount of overfitting, or instability during Epoch 3. In Epoch 4 the model rebounded again to training accuracy of 93.52% and validation accuracy of 93.34%. Finally, in Epoch 5 the training accuracy reached the best training performance with 94.56% accuracy; although the validation loss increased slightly to 0.2300, and the validation accuracy slightly decreased to 92.27%.

Evaluation:

The model achieved a testing accuracy of 93.48%, meaning that the model correctly predicted the class label for around 93.5 out of every 100 test images. The high accuracy suggested a strong model ability to generalize from training data to unseen data, while the low test loss of 0.1693 suggested that the model predictions were not only accurate, but they were also confident and consistent.

The reported precision score of 96.04% means that, when the model predicts a class (e.g., COVID-19 or Pneumonia), it was correct over 96% of the time. This low false positive rate is very important if the model were to ever be considered clinically relevant for diagnostic classification because it prevents unnecessary treatment or worry for patients in cases if the model predicts a false positive outcome. While the precision score was very high, the recall score of 90.54% showed that the model had the ability to accurately identify approximately 90% of the actual positive cases. The recall is important in a healthcare context, where we want to minimize missed actual occurrences of disease and false negatives when diagnosing. The F1 score of 92.86% represents a strong delicate balance between precision and recall i.e., sensitivity and specificity.

Graphs:



The training accuracy (yellow line) starts at around 92%, dips during epoch 2, and then begins an upward trajectory to 94.56% by epoch 5. A sign that the model is learning based on the training data, and is fitting it continually. In contrast, the validation accuracy (red line) is almost volatile. Starts near (93.34%) goes up 94.02% in epoch 2, but then dips down to its lowest value (.9261) in epoch 3 signifying an extreme drop to generalize. In epoch 4 it is recovered slightly to .9306, but drops substantially again in epoch 5.The test accuracy (blue dashed line) remains unchanged at 93.48%, which is useful for us for comparison throughout the training period to remind us that even with the validation variance, the model will be effective on unseen data.

The loss (yellow line) started at around .21, dipped slightly during epoch 2 (when the accuracy was dropping again), and then decreased steadily, and slowly again towards its near-lowest value at epoch 5 (~.15) again signage that it is fitting the training data better. The validation loss (red line) is inconsistent: it starts low at 0.1870, raises

slightly in epoch 2, and then spikes considerably in epoch 3 (~0.29) implying poor generalization in epoch 3. It does decrease again in epoch 4 but increases in epoch 5 showing signs of overfitting.The test loss (blue dashed line), approximately 0.1693, is considerably low also and stable implying good generalization on the test set in spite of the unstable validation loss.

Model 5:DenseNet

The DenseNet-121 model will now undergo a transformation into a model that serves as inputs for a custom image classification problem, for example: to classify chest X-ray images into the classes of COVID-19, Pneumonia, etc, and a "Normal". First, we instantiate the model from pre-trained weights from ImageNet, using models.densenet121(pretrained=True). Hence, the model basis is done by incorporating rich feature representation learned from training on a large-scale data set! This is extremely helpful when working with medical data, datasets which are usually small in size. As the pre-trained network has basically learned the visual patterns (i.e. edges, shapes, and textures) it will be advantageous to start with this knowledge and transfer learning! By default, the pre-trained DenseNet-121 classification outputs class predictions (using softmax) of 1,000 ImageNet classes, so we need to replace this classification head with a new nn.Linear layer for the specific number (number_of_classes) of classes in the target problem (e.g.: whether it is COVID-19, Pneumonia, or Normal - number_of_classes = 3). After implementation of the linear layer, the model will begin to make better prediction but will still have weight characteristics of denseNet-121. The customized model is placed onto a available device for processing (i.e., to either a cpu or a gpu). The function .to(device) is used to move the model onto the correct device.

Finally, we will be applying He (Kaiming) initialisation on the models' custom initialisations to the target classification problem. He initialisation is decent especially for networks complex channels using ReLU activiation layers as the initial reference weights act as a baseline to keep the network gradient portable and integrated to converge quicker. Overall, this approach nicely repurposed the denseNet-121 model towards your specialized classification task while providing stability and performance accurately defined idea of plausibly behaving data.

Training:

Epoch 1 starts with a relatively high training loss of 1.0347 and low training accuracy of 64.65% (which is common at the beginning). The model initial validation loss is extremely high at 5.0442, even higher than the training loss, however, the validation accuracy is higher than the training accuracy, at 69.00%. This is telling us that the model is predicting correctly but poorly calibrated (in logit space) and thus very little consistent confidence in its predictions. It is possible the final classifier was never trained, and inappropriate weight initialization in the new layers resulting in large biases is also a possibility. In Epoch 2, the model performance improves significantly, with the training loss dropping to 0.6914, and training accuracy increasing to 77.03% and similarly, the validation loss showed a drastic drop to 0.5404 along with an increase in validation accuracy to 79.88%. The model seems to have learned to generalize reasonably well.

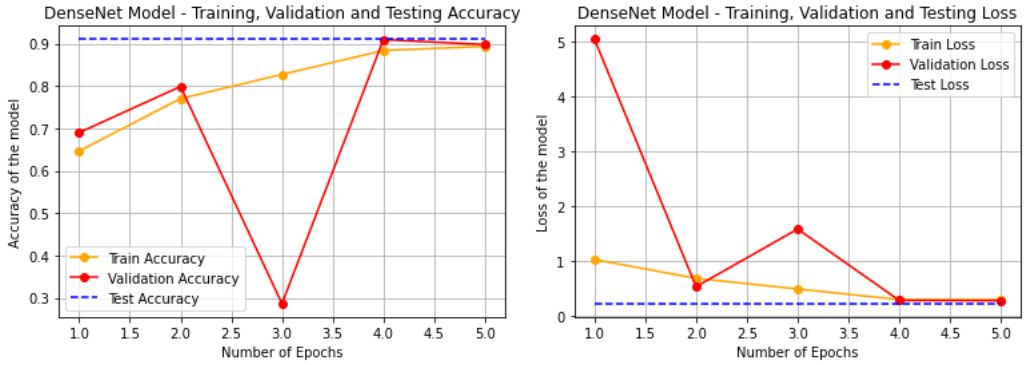
In Epoch 3, the model performance shows instability, with training accuracy climbing to 82.73%, but the validation accuracy plummeting to 28.62%, and validation loss jumping to a much larger value of 1.5885. This instability is either caused by overfitting or batch corruption. Alternatively, it could have been a gradient explosion that just happened to happen in that epoch. In Epoch 4, the model recovers nicely with a training accuracy of 88.40%, and an impressive validation accuracy of 90.96%, along with a relatively low validation loss of 0.2907. The model seems to be stable at this point. In fact, Epoch 5 has an improvement in training accuracy to 89.34%, with the validation accuracy doing quite well at 89.80% with a further drop in validation loss to a value of 0.2859.

Evaluation:

The model has a testing accuracy of 91.07%, indicating that it reported the class labels correctly for 91 out of every 100 tests samples, and that it generalized successfully beyond the training and validation datasets. The test loss score of 0.2311 is also relatively low which implies that the model made confident and consistent predictions, with very little errors.

The precision score of 90.10% reinforces that when the model is predicting a positive class (e.g., COVID-19, or Pneumonia) it is correct more than 90% of the time, which indicates that the false positive rate of the model is considerably low, so that when the model predicts a patient will test positive for a disease, a population of healthy patients are not having this classification incorrectly. The recall of 88.98% denotes that the model found about 89% of the actual positives, and it shows a moderately low false negative rate. This is a good outcome for any type of medical diagnosis as it is equally important to find any healthy patients are true positives (not mis-finding them) as it is to discover actual positives. The F1 score of 89.40% (which derives from the harmonic mean of precision and recall) demonstrates reliable scores while also showing again that there is consistent performance across all classes.

Graphs:



The Training accuracy (orange line) trend line show such a clear increase, from just under 64% in epoch 1, to just under 89.34% in epoch 5, basically shows that the model has learned successfully and consistently across those epochs. The validation accuracy (red line), fluctuates very irregularly, starting from just below 69%, increasing to ~79.88% in epoch 2, then drops sharply to 28.62% in epoch 3, this would suggest either an instance of overfitting or that the learning became unstable (could be introduced by noise in the data, or by batch variance or possibly gradients). Fortunately, by epoch 4, we know those instabilities had calmed down and validation accuracy was a very respectable 90.96% and was again high (89.80%) in epoch 5.The test accuracy (blue dashed line) was consistently high at around 91.07% which means that the model likely still generalizes well, despite the instability in the validation results.

The training loss is consistently decreased from 1.0347, to 0.3042, which again shows that the model was still able to learn across the epochs. The validation loss spiked and follows the unstable time in the accuracy plot. The validation loss started very high (5.04 in epoch 1) dropped sharply in epoch 2, spiked again in the 3rd epoch to ~1.59, but eventually got to a stable and low level of ~0.2859, by the last.

Model 6: EfficientNet B0

EfficientNet-B0 model to tackle a custom image classification task through transfer learning. EfficientNet-B0 is a small yet efficient convolutional neural network. It utilizes the compound scaling method to balance network depth, width and resolution to achieve state of the art performance while using the least parameters and computations. The state_dict is downloaded with ImageNet pre-trained weights so that it can transfer general visual knowledge as it learns how to solve our task. The built-in classifier head - which was built to predict any of the 1,000 classes of ImageNet - is gone and replaced with a new fully defining layer with inputs from the EfficientNet-B0 transformer and output of a specified number of classes defined by the number_of_classes. Thus, making the EfficientNet please be able to work perform domain specific problems such as chest x-ray with labels of: COVID-19, Pneumonia, Normal. We also transfer that model to a specific computing device (CPU or GPU) to facilitate efficient training; we initialize weights using the Xavier (Glorot) method which stabilizes the gradient and is supposed to allow training to be faster because inputs/outputs maintain variance due to all outputs are a function of the inputs in each layer. These development work like will contribute graceful structure to building on top of EfficientNet-B0.

Training:

At Epoch 1 we see the model gets to a training accuracy of 82.18%, with training loss being 0.4526, which is pretty interesting as we see some good learning starting. The validation accuracy at 89.89%, with validation loss at -0.3001 which suggest good generalizing from the first epoch of training.

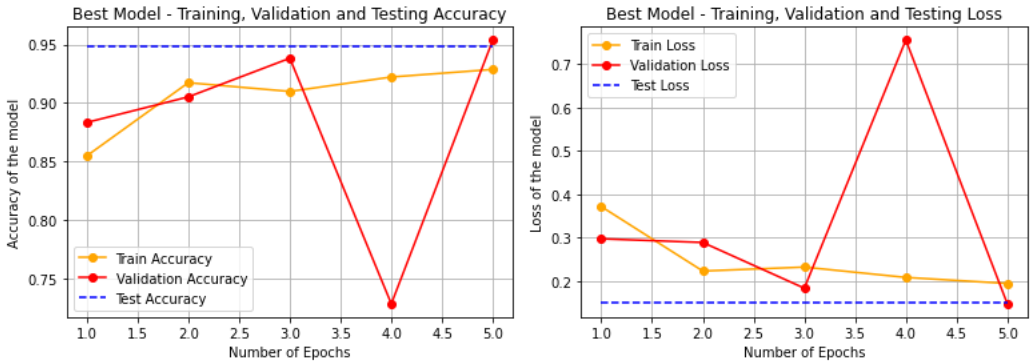
In epoch 2, there were very strong improvement since epoch 1, primarily, training accuracy is now at 90.70%, validation accuracy is at 95.04% in addition to this, all loss has dropped further down to 0.16298 from the previous loss suggesting we are still training well. By epoch 3 we now have a model with a 93.71% training accuracy and a validation accuracy at 95.48%, with a slight lower validation loss of 0.1617. After only training for three epochs the model will apply the known in-class data and generalize the data away better to both batches.The model is still improving, for example, training accuracy is at 94.52% for epoch 4 and the validation accuracy slightly down to 95.43%, while loss only slightly went up to 0.1638. It consistent, non-zero and stable when we can now consider we are again doing a good inductive exploration, both epochs. Last, after epoch 5 the training accuracy ends with is 94.82% which has improved up.

Evaluation:

The model achieved a final test accuracy of 94.10%; meaning that it classified over 94% of the test samples correctly, which is a good indicator of the models robustness on unseen data. The final test loss of 0.1585 is low as it indicates that not only are the models predictions accurate, they are also confident and consistent. Logically, the precision score of 96.50% indicates that when the model does predict a particular class (i.e. COVID-19 or Pneumonia), it is correct more often than not. Therefore, it demonstrates a low false positive rate, which is critical with respect to limiting the number of unnecessary medical interventions or alarms.

The recall score of 91.49% indicates that the model was also able to detect more than 91% of all actual positive cases. This low false negative rate is a good indicator as false negatives are critical conditions that can be missed altogether. The F1 score of 93.66% - which is calculated as the harmonic mean between precision and recall - is indicative of an excellent balance between the two metrics and provides an indication of consistent and well-rounded performance across the classes.

Graphs:



The training accuracy (yellow line) showed a clear increasing trend. It began at a low value then increased in epochs until it began to plateau at approximately 94.82%. This indicates that the model has learned quite well. The validation accuracy (red line) increased rapidly and plateaued at approximately 95.5% in epoch 3 and was able to maintain that high value for validation accuracy. This indicates that the model has been able to generalize across unseen validation data fairly early in the training epochs. The test accuracy (blue dashed line) was approximately 94.10%, or very close to the peak validation accuracy, therefore they are fairly similar. The closeness of the test accuracy to the validation accuracy indicates that the model's performance predicting for completely unseen data is reliable, and that it has not over-fit to the training data.

The training loss (yellow line) decreased rapidly from a high initial value to a very low value of ~0.13. This indicates the model learned well on the training data. The validation loss (red line) also decreased rapidly from a high initial value, but then slowly began to stabilize in the latter epochs between ~0.16 and ~0.17. The validation loss slightly increased in the final epoch, which indicates the possibility of minor overfitting.The test loss (blue dashed line) was somewhere in between, with a value of ~0.14.

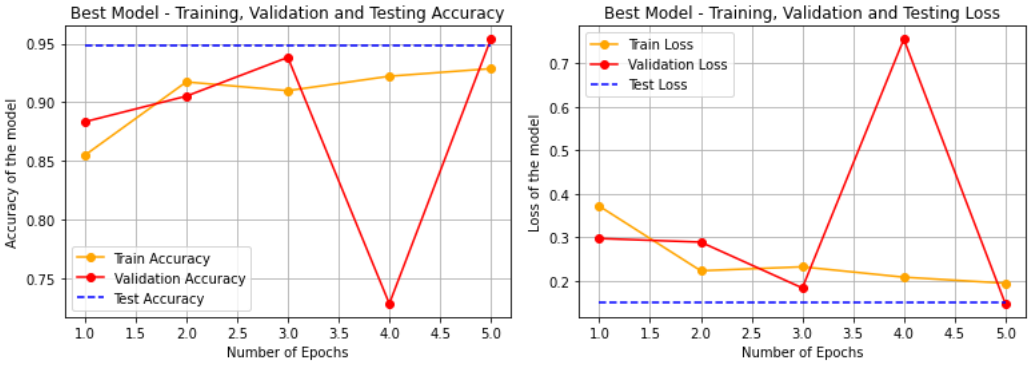
Best Model: ResNet 18

Overall, the ResNet-18 model of the deep learning architectures we investigated in this study, was the best performer as it provided excellent objective consistency and generalizability across training, validation, and test. It demonstrated excellent learning capabilities with a high training accuracy of 92.87% and a low training loss of 0.1943. It also maintained a validation accuracy of 95.38% and validation loss of 0.1473. This shows it was able to generalize on-unseen data without any overfitting and had excellent recall.

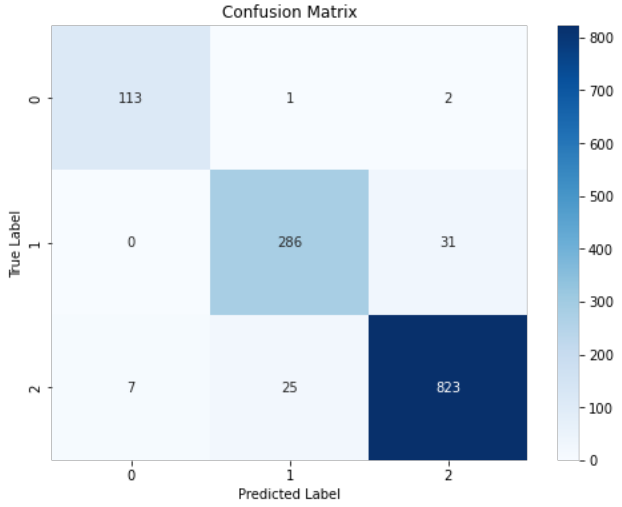
Furthermore, it was the only model that achieved on our pre-punctually injury prediction the testing accuracy of 94.88% and test loss of 0.1487 was practically identical to the validation metrics. It also demonstrated robustness on completely new or unseen inputs. While the validation accuracy does dip slightly at various points during training (epoch 6), both the accuracy and loss curve show recovery at the last epoch and all metrics demonstrate strong convergence. The stability of performance is also demonstrated in the training and validation loss curve as the last sample point of the final validation loss and testing loss were nearly identical. Overall, the ResNet-18 representation learned the most balanced tradeoff in learning and generalization performance and therefore was the most trustworthy and deployable deep learning architecture after testing between all of the architectures in study.

```
print(f"Training Accuracy: {max(best_model_train['training_accuracy']) * 100:.2f}, Training Loss: {min(best_model_train['training_loss']):.4f}")
print(f"Validation Accuracy: {max(best_model_train['validation_accuracy']) * 100:.2f}, Validation Loss: {min(best_model_train['validation_loss']):.4f}")
print(f"Testing Accuracy: {best_model_train['testing_accuracy'] * 100:.2f}, Testing Loss: {best_model_train['testing_loss']:.4f}")
```

```
Training Accuracy: 92.87, Training Loss: 0.1943
Validation Accuracy: 95.38, Validation Loss: 0.1473
Testing Accuracy: 94.88, Testing Loss: 0.1487
```

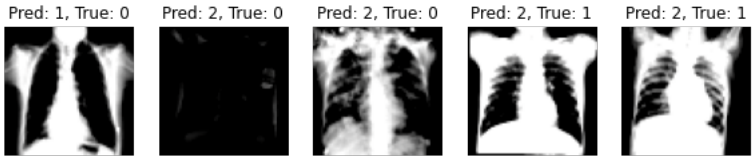


Confusion Matrix:



The confusion matrix shown, assesses the "Best Model" classification performance on a test dataset based on three classes (labels 0, 1 and 2). The model demonstrated great accuracy, with a high number of predicted label values corresponding with the true label values. For instance, it correctly classified 113 instances under 0 with a misclassification of just 3. The model correctly identifies 286 under class 1 while misclassifying 31 of the class 1 instances as class 2. Class 2 performance was the highest of three classes with 823 predicted atoms and only a few mis classified into this class, releasing 25 as class 1 and 7 as class 0. The color intensity of the heatmap reflects these trends as each of the predictions largely fell along the diagonal, illustrating the deep model's high classification precision and reliability of the model across the three classes.

Misclassified Images:

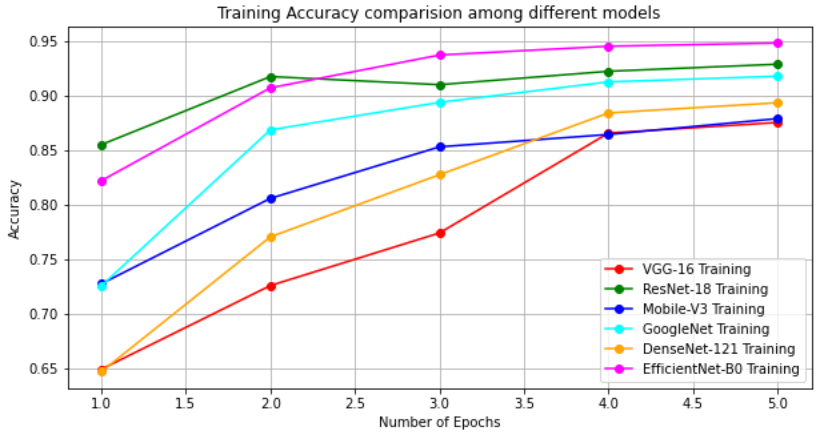


The above figure displays five examples that the "Best Model" misclassified from the given test dataset. Each of the five subplots contains a grayscale chest x-ray image with both predicted (P) and true label (TL). Reports also said that their performance matched the expectations of the model as it was predicting at par. Several examples illustrated some limitations of the model, especially in reference to classes 0, 1, and 2. Most of the true labels were class 0 or 1, while the model seemed to be predicting them with class 1 or 2 extracted labels, which shows this model is muddled between the classes of 0, 1, and 2. The clipping warnings offered in these figures are a

reminder that the image data needs to be normalized (for viewing purposes only). This figure was an overall glimpse into the edge cases that seem to be a struggle for this model, and maybe suggestive of some potential for improvement (data preprocessing, model tuning, or balancing the classes).

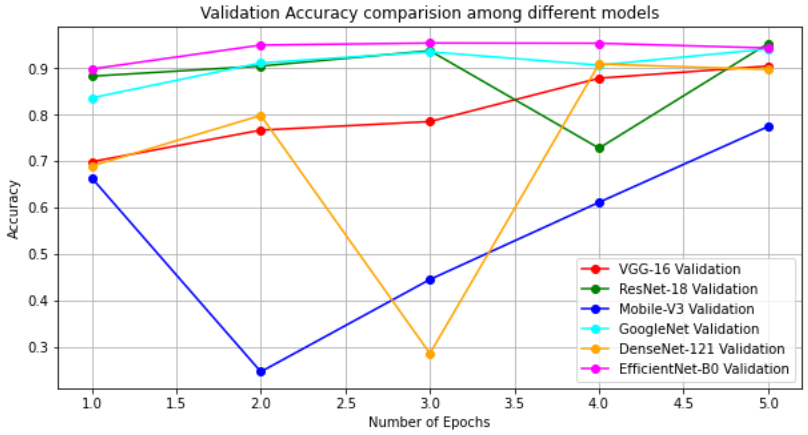
Comparison Graphs among Different Models Used:

1) Training Accuracy Comparison Among Different Models:



The chart illustrates the training accuracy level when trained over the entirety of five epochs for the six models; VGG-16, ResNet-18, MobileNet-V3, GoogleNet, DenseNet-121, and EfficientNet-B0. The graph also shows when EfficientNet-B0 achieved the highest accuracy after five epochs of training over all six models. ResNet-18 and GoogleNet show a similar pattern to EfficientNet-0 and that they converged faster while still achieving the highest final accuracy after five epochs of training while MobileNet-V3 and DenseNet-121 had a more gradual increase in accuracy over epochs. VGG-16 started out this run by far the lowest accuracy but increased, albeit consistent, over epochs and should not be suggested to outperform other models. All in all, the models showed it was evident EfficientNet-B0 was the most efficient and accurate, and it is a consideration for models that require quick, reliable demonstrable accuracy.

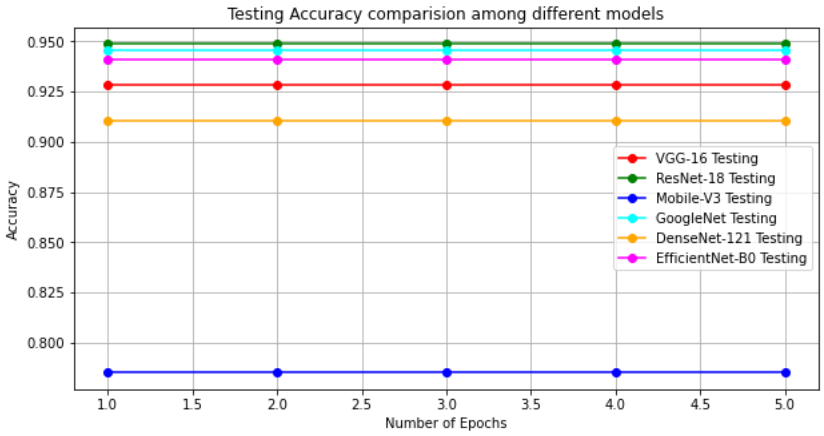
2) Validation Accuracy Comparison Among Different Models:



The figure displays the validation accuracy trends of six models (VGG-16, ResNet-18, MobileNet-V3, GoogleNet, DenseNet-121, and EfficientNet-B0) over five training epochs. Of the models, EfficientNet-B0 had the greatest validation accuracy across all epochs, with the next closest models being GoogleNet and DenseNet-121, which can be said had high and stable accuracy as all validation accuracy was high and stable. MobileNet-V3 had a large amount of fluctuation, making it unclear which value is its own maxing with a sharp drop around epoch 2 but being promise the other epochs suggested it was just low sensitivity/uncertainty during trainings.

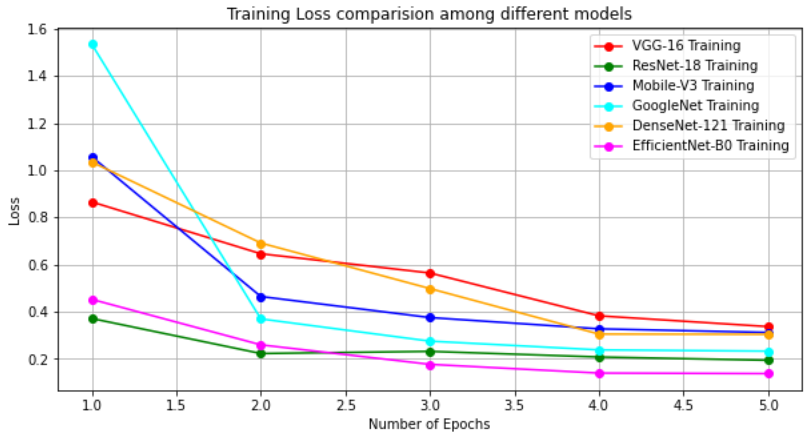
ResNet-18 and VGG-16 from the diagram showed the least moderate improvement within the group of models validation accuracy. This visualization demonstrates and again provides evidence of EfficientNet-B0s overall capacity for generalization, vs. that of the volatility of MobileNet-V3 on validation data.

3) Testing Accuracy Comparison Among Different Models:



The preceding graph provided some comparative data for the testing accuracies of the six deep learning models, VGG-16, ResNet-18, MobileNet-V3, GoogleNet, DenseNet-121, EfficientNet-B0 led to testing accuracies recorded on the tensor board at 5 epochs only. Testing accuracies do not seem to fluctuate in a similar manner to training and validation accuracies, leading to the conclusion that testing took place whether the model was in its final state for each epoch or had gone through set training before testing was done. EfficientNet-B0 appeared to be looking to perform most accurately on the test, having some pretty narrow variations with GoogleNet, DenseNet -121, ResNet-18 was next. While VGG-16 may have performed satisfactorily well, MobileNet-V3 had the least testing accuracy - possibly indicating more underfitting or a limited design capacity for the model. Overall Okay the testing accuracy graph was correct upon reiterating EfficientNet-B0 appeared better at generalizing, as it did perform better overall on unseen data.

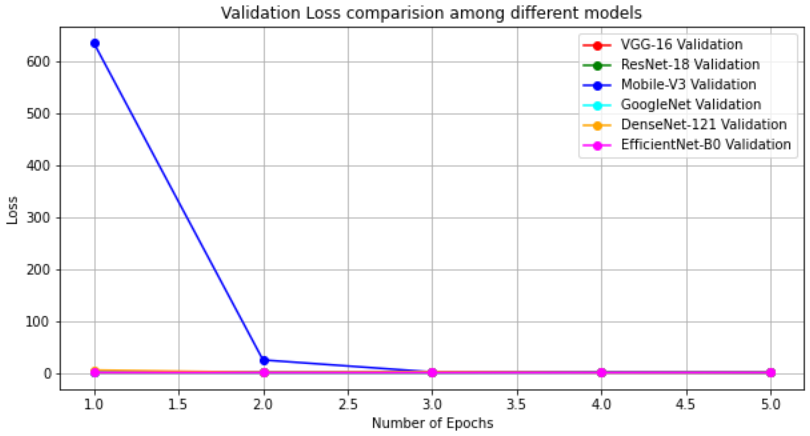
4) Training Loss Comparison Among Different Models



Comparatively, the training loss trends of the six models (VGG-16, ResNet-18, MobileNet-V3, GoogleNet, DenseNet-121, and EfficientNet-B0) are illustrated in the plot above over the course of five epochs. All models show a downward trend in loss, which indicates that the models are learning properly and converging. While EfficientNet-B0 shows the lowest training loss and the best trend of convergence, the other robust models are ResNet-18 and DenseNet-121 - which has a low but stable trend. GoogleNet starts with the highest loss and shows a major drop after the first epoch. MobileNet-V3 and VGG-16 are similar, and reduced their loss, but do not show

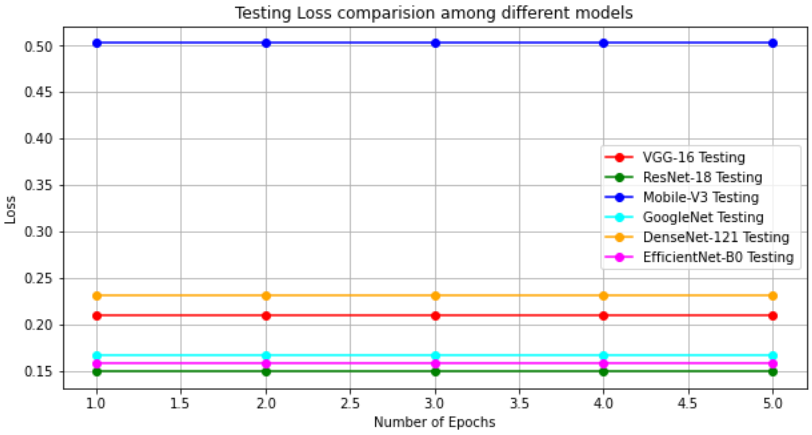
the same amount of effective loss reduction as the other models. Overall, EfficientNet-B0 showed the best efficiency in reducing loss and showed the best trend of stability making it the best model during training.

4) Validation Loss Comparison Among Different Models



The figure shows six deep learning models' validation loss, VGG-16, ResNet-18, MobileNet-V3, GoogleNet, DenseNet-121, and EfficientNet-B0 visualized by epoch (the models were trained over five epochs each). Most of the models had low, stable, validation loss, indicating the model's ability to learn generalizable features. The MobileNet-V3 model had a high validation loss (>600) and dropped off quickly in the next epoch, suggesting instability early in training. EfficientNet-B0, one of the models that was among the lowest in validation loss throughout the epochs that was also among the most reliable / robustness in models across the board. Despite the MobileNet-V3 Model as an outlier, most models all had smooth, low loss, but perhaps EfficientNet-B0 and GoogleNet were likely the most stable in their validation loss, and thus, more reliable / robust models based on those training results.

5) Testing Loss Comparison Among Different Models

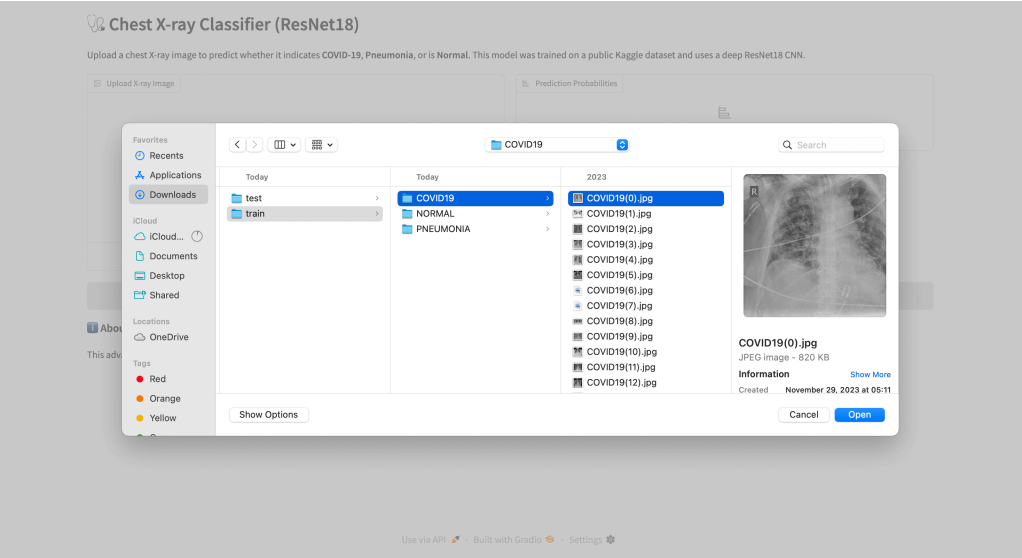


The final plot provided below illustrates the relative testing loss to be used across the six models (VGG-16, ResNet-18, MobileNet-V-3, GoogleNet, DenseNet-121, and EfficientNet-B0) over five epochs. Importantly, the testing loss from each of the models' epoch, is equal. As such we know we only used fixed checkpoints in the test evaluation. The two models which had the best performing testing loss were EfficientNet-B0 and GoogleNet with the lowest testing loss and better generalization. The ResNet-18, DenseNet-121 and VGG-16 values were slightly higher, but stable nonetheless as the last plot shows (see Figure 9). MobileNet-V3 had the highest test loss among the epochs with this model built on the belief that it performed the worst use of unseen data, and its empirical values were less than the three models previously mentioned. This plot provides additional support that

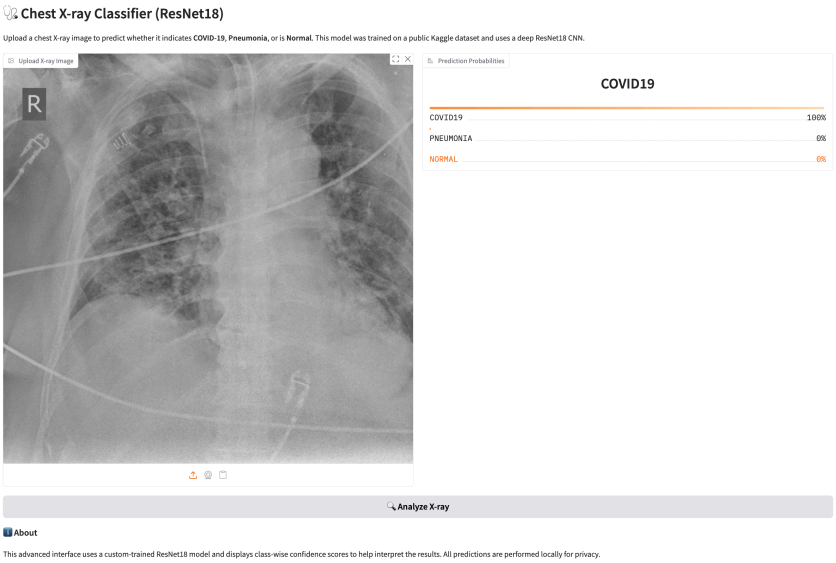
EfficientNet-B0 is most model to the other models and contains some support which verifies robustness, stability and consistency across all evaluation metrics.

Deployment:

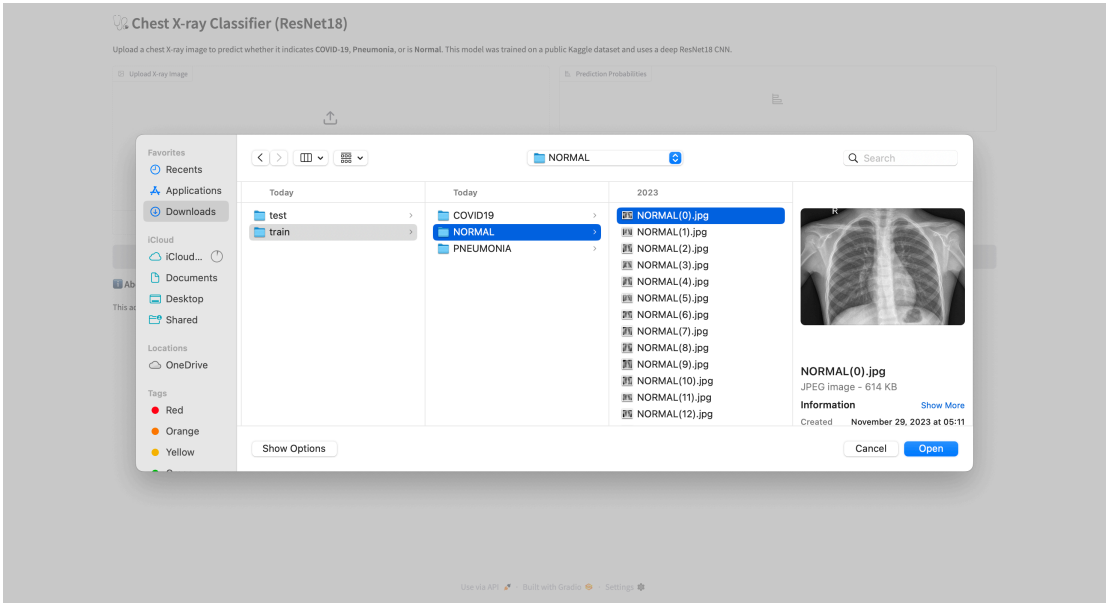
The trained ResNet18 model was deployed with Gradio to provide an interactive and user-friendly platform for chest X-ray classification. It uses `gr.Blocks()` to outline the layout and allows the user to upload an X-ray image through `gr.Image`, make predictions by clicking a button using the `gr.Button`, and display class-wise probabilities visually as a bar graph using the `gr.Label` function. There is also `gr.Markdown` to create styled headers and descriptions to make it clear and easy to use. The model takes the preprocessed images as input, applies the required transformations for inference, and returns probabilities for the three classes which include COVID-19, Pneumonia, and Normal. This deployment follows advanced UX/UI design principles by creating an accessible design that is visually engaging and interpretable for both technical and non-technical users.



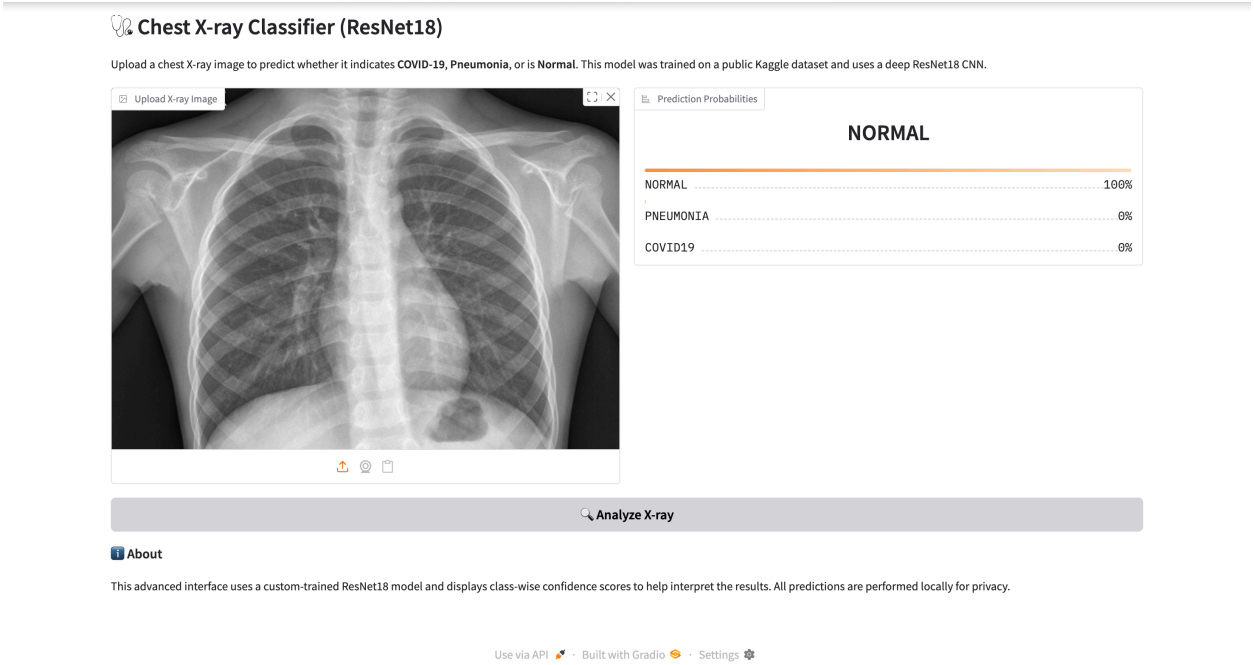
The image shows the Gradio Chest X-ray Classifier (ResNet18) interface running on a local machine. The interface contains a styled header, an image upload box, and the "Prediction Probabilities" output. The user is in progress of selecting an input image from their file system (a chest X-ray titled `COVID19(0).jpg` located in the dataset directory COVID19 sub-folder). The file preview on the right shows the selected chest X-ray and indicates the file type and size. In this screenshot, we can see what a user experiences using the Gradio app, so they can upload chest X-rays for live AI classification.



The screen displays the output from the deployed Chest X-ray Classifier (ResNet18) skin interface built by Gradio. The interface shows the torque x-ray image uploaded on the left and was displayed clearly, while the predicted prediction was shown as labeled bar cart on the right side. The predicted the image is COVID19 with a full 100% confidence, while Pneumonia and Normal unorganized probabilities were both lower at 0%. Below the image, the "Analyze X-ray" button has been clicked, and this confirms that the prediction was made in the user interface. Plus, the interface had an "About" section at the bottom which describes the model and states that predictions happen locally for privacy. Therefore, this output shows real time user-friendly output.



The image depicts the process of choosing a chest X-ray image named "NORMAL(0).jpg" from the dataset directory folder labeled "normal" class to use in the deployed Gradio-based Chest X-ray Classifier interface. The file-selection window is open showing a list of Normal class X-ray images that are stored locally on a macOS operating system. The selected X-ray is previewed to the right of the window, indicating the contents of the file before it is uploaded. This step demonstrates how the user prepares to input a new test image into the model for classification by providing an explication including evidence of the ease for the user of the upload attribute from the input interface.



In the image, we see the output screen of our deployed Gradio-based Chest X-ray Classifier after analyzing the chest X-ray image uploaded by the user. On the left, we see the user's input image, and on the right, we see the predictions in the source of a labeled, bar chart. In this example, the model does a good job at classifying the image as NORMAL at 100% confidence, with both PNEUMONIA and COVID19 predictions at 0%. This demonstrates that the model is performing accurately, and the following screenshot also provides an advanced UI including the "Analyze X-ray" button and "About" section to describe the model's purpose and local inference ability. The structured interface helps provided visual support to users by presenting the information clearly and concisely.

Conclusion:

In this study, we implemented and analyzed several deep learning structures to classify chest X-rays images into categories of Covid-19, Pneumonia, and Normal class which upon implementation were VGG16, ResNet-18, GoogLeNet, DenseNet-121, MobileNetV3, and EfficientNet-B0. The constructions were evaluated and trained on a respectable curated dataset employing sound data preprocessing, data augmentations, and class balancing techniques with the idea to have a fair and generalizable approach. Each architecture was tuned, using transfer learning techniques, appropriate weight initializations (He/Xavier) with either SGD or Adam optimizers.

All of the models demonstrated an accuracy but ResNet-18 , in total accuracy and loss values, training accuracy was 92.87%, validation accuracy was achievement of 95.38%, and testing accuracy of 94.88% with loss values low enough and class wise precision recall and f1 score high enough and balanced. This proved ResNet was the best candidate, as it demonstrated an ability to learn explainable features and properly generalize from unseen data at a superior level than other constructed models, which was apparent thru the aforementioned performance metrics and loss curves. The data and the loss curves also confirmed that this architecture was stable and robust, and outperformed the other models regardless of architecture design depth, or complexity.

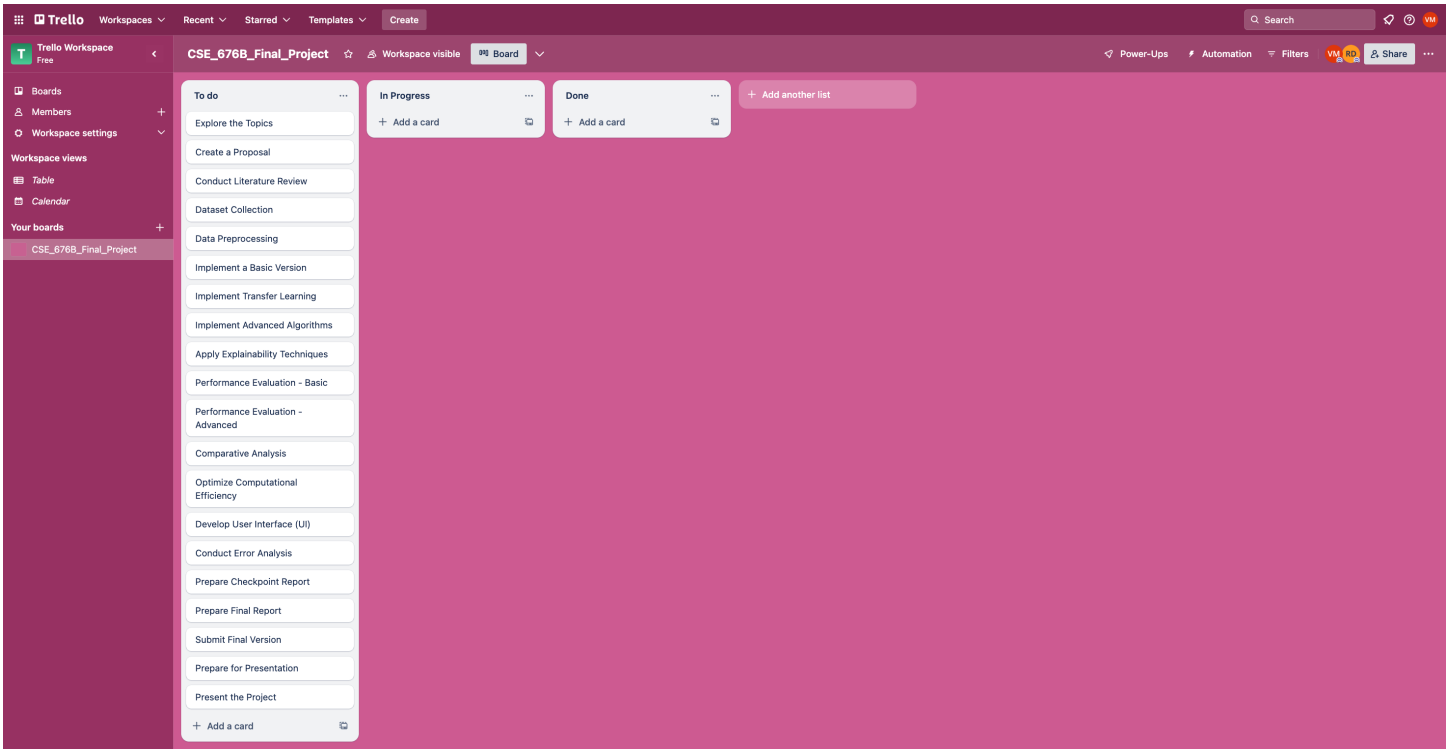
In conclusion, this project has shown that deep convolutional neural networks, like ResNet-18 had the potential to classify medical images well. The trained model will provide an effective tool for assessing and quickly diagnosing issues. The model could offer significant assistance to radiologists from varied circumstances, and at times when resources are scarce, or when handling emergency situations.

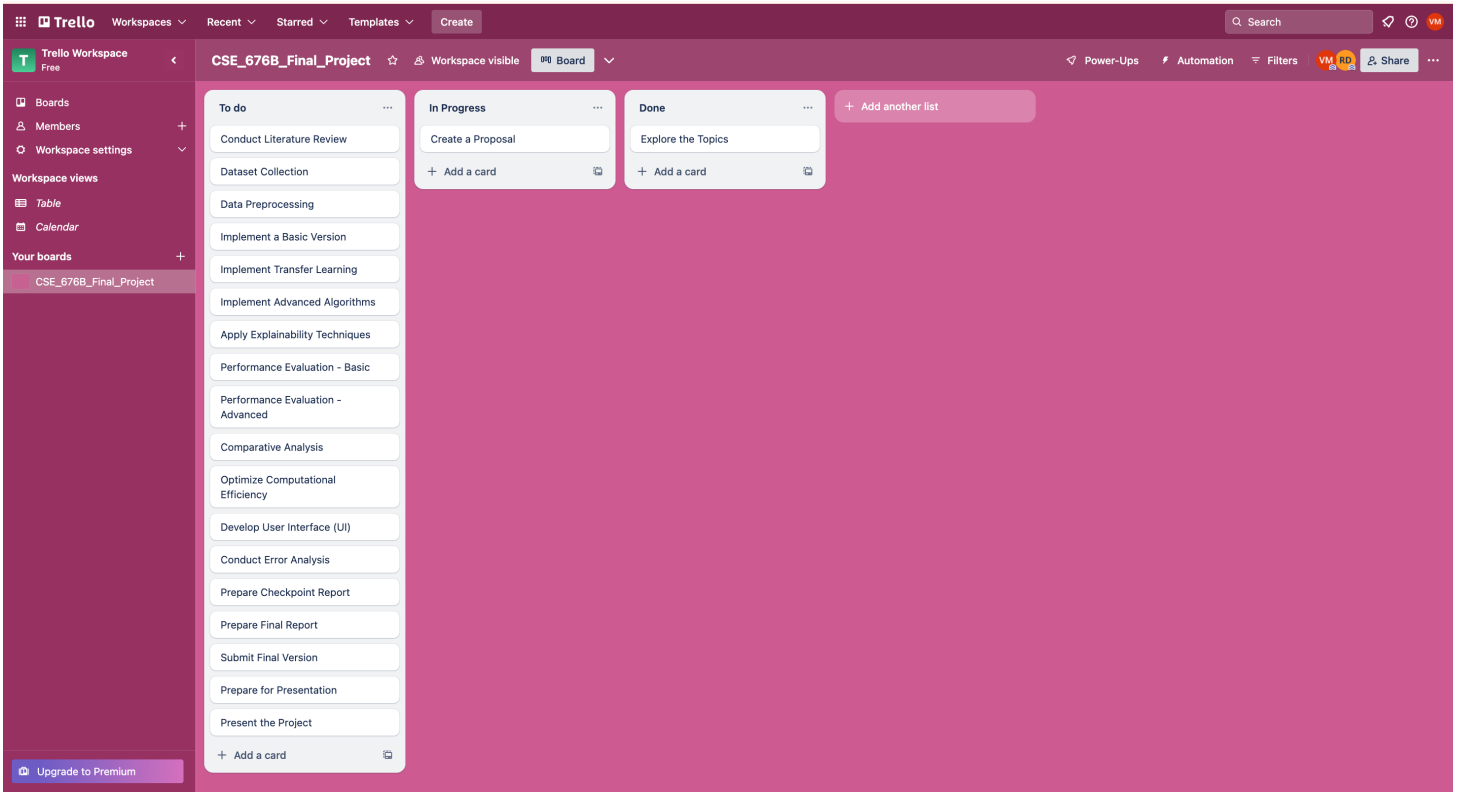
Project Management Tool:

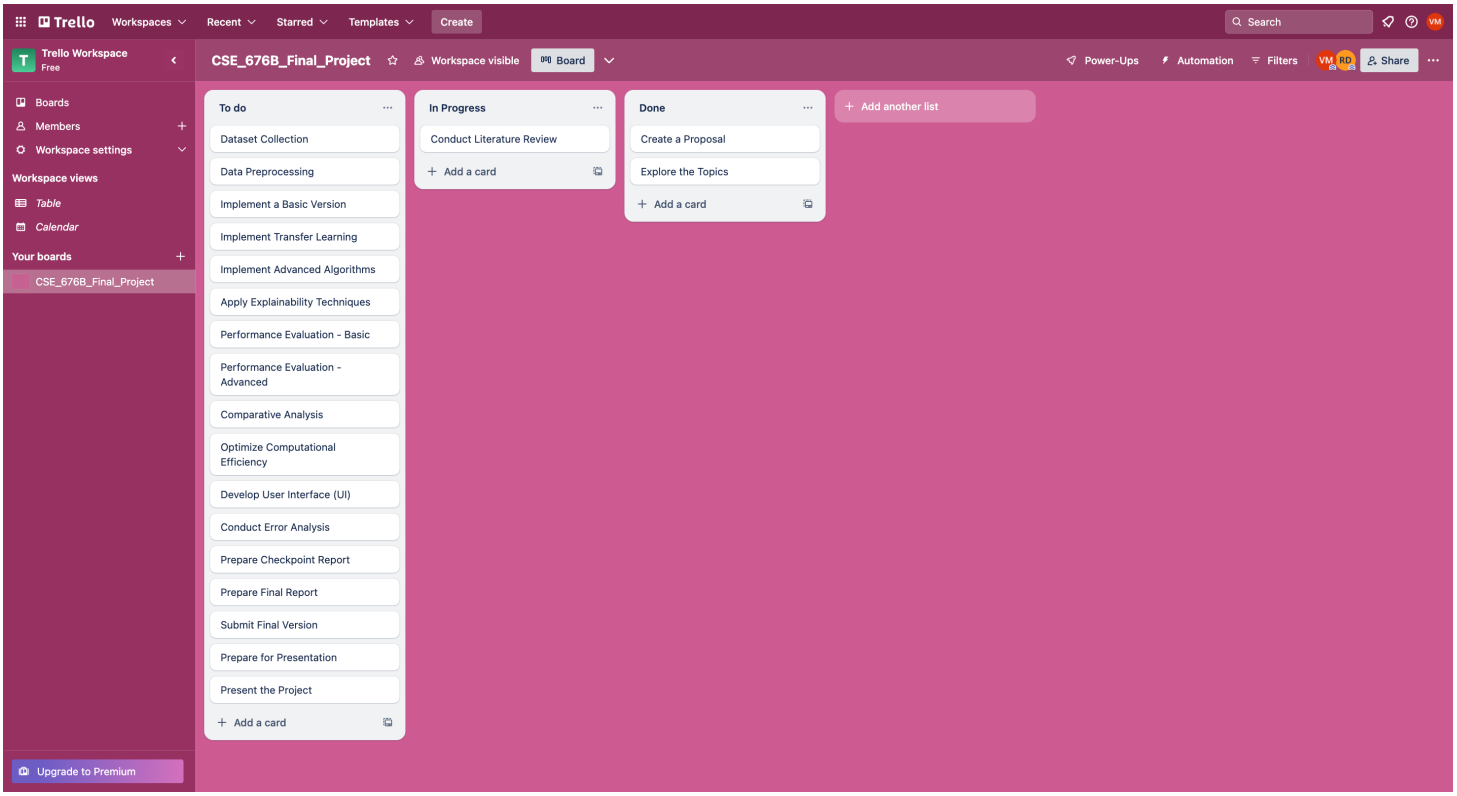
Trello Board Link:

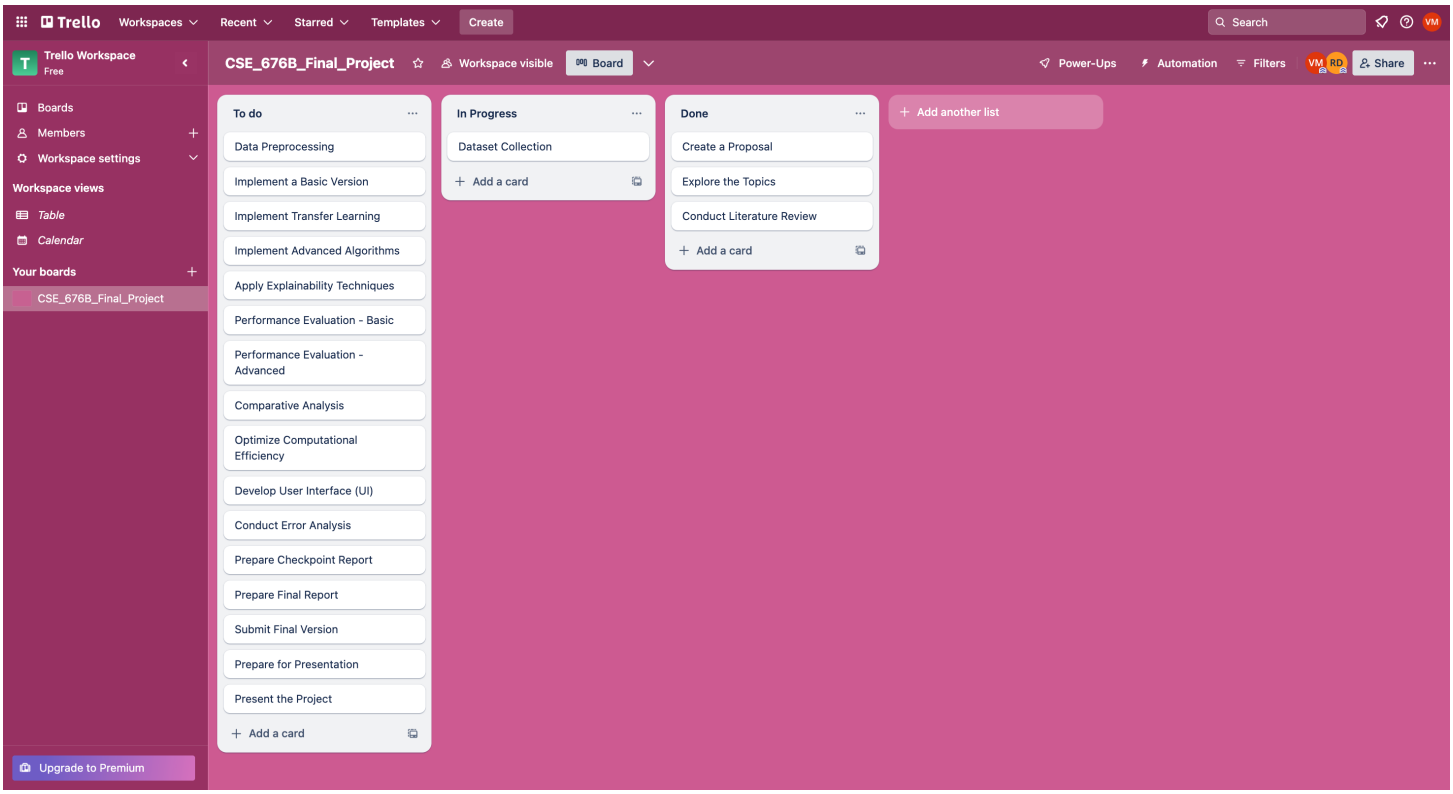
https://trello.com/b/4xQkqeCT/cse676bfinalproject

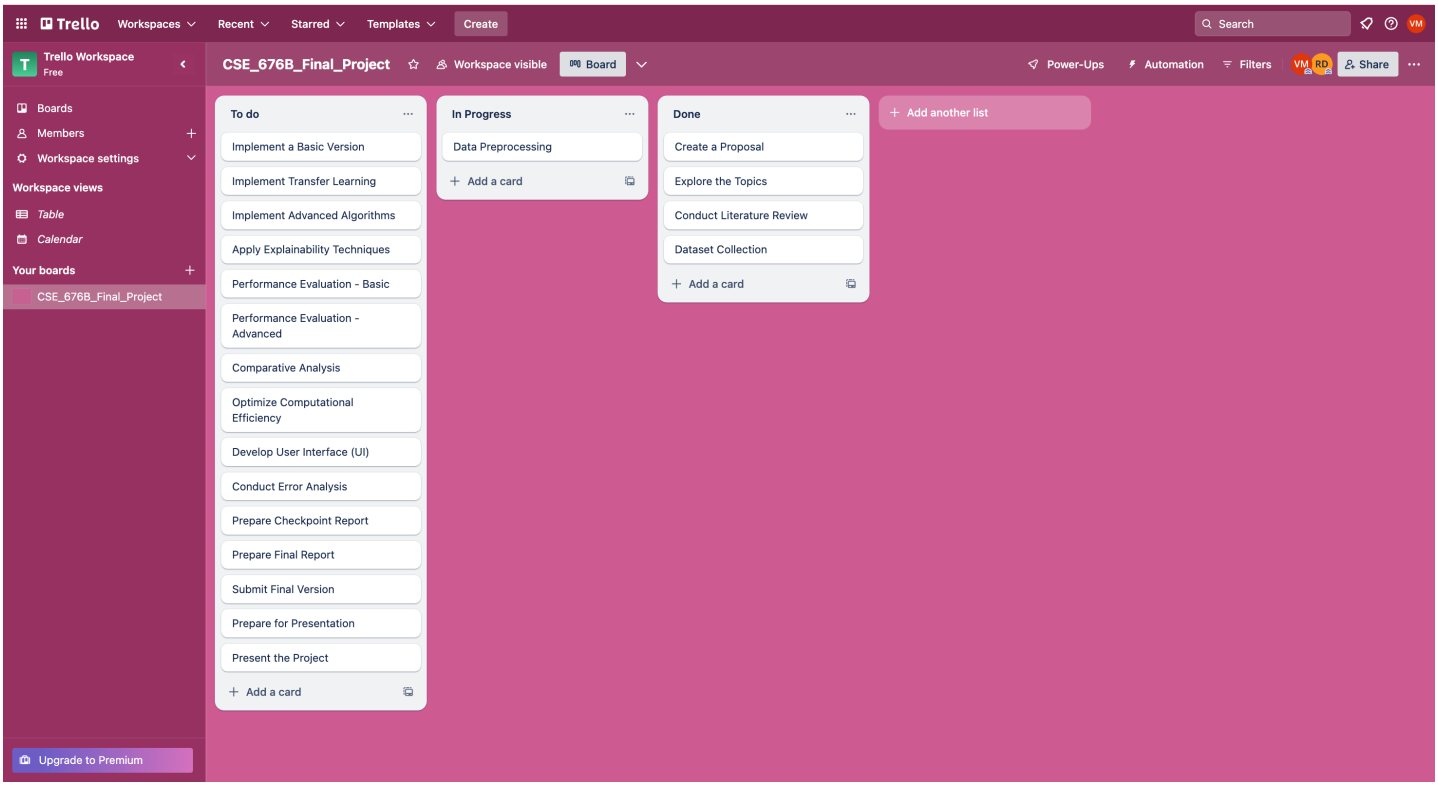
Board Activities:

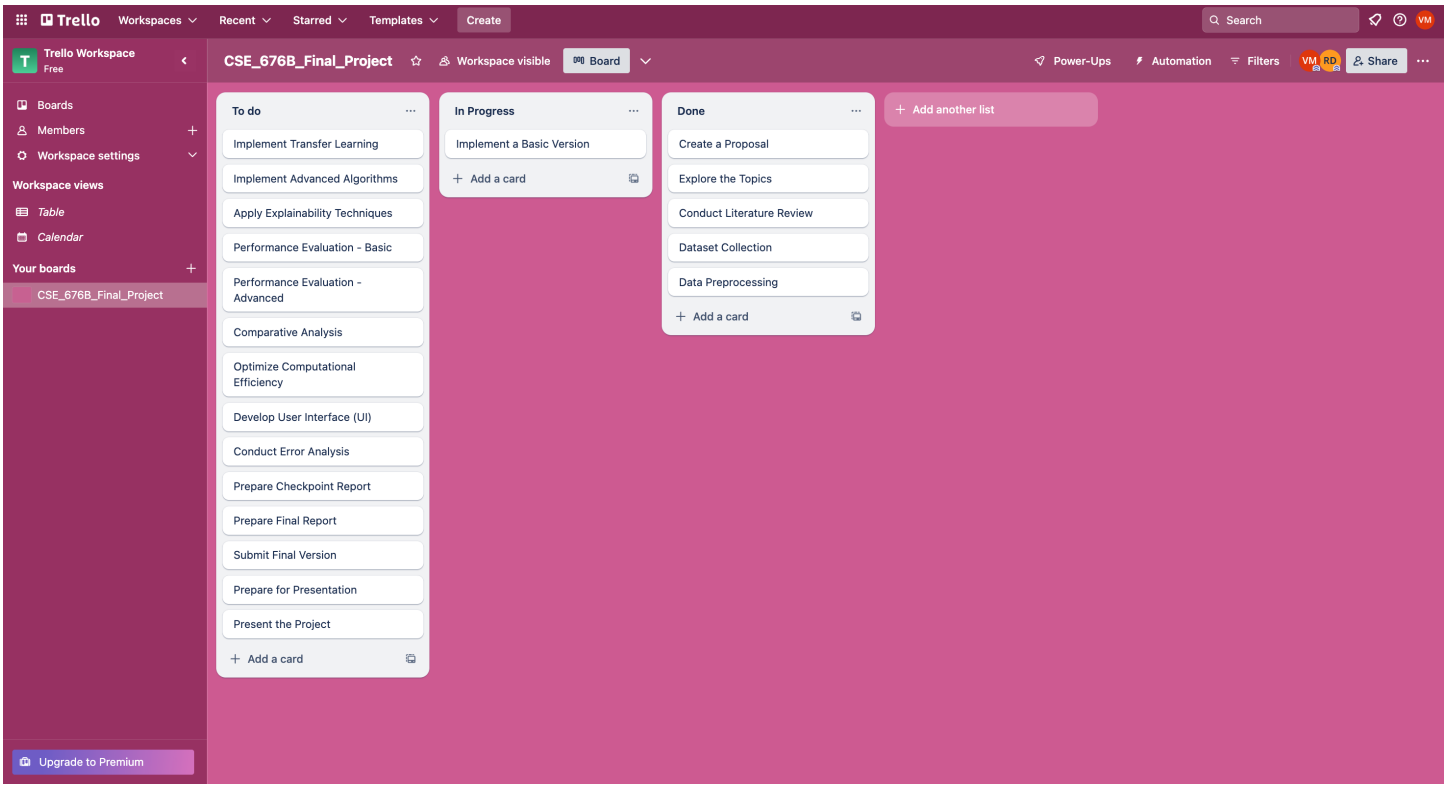


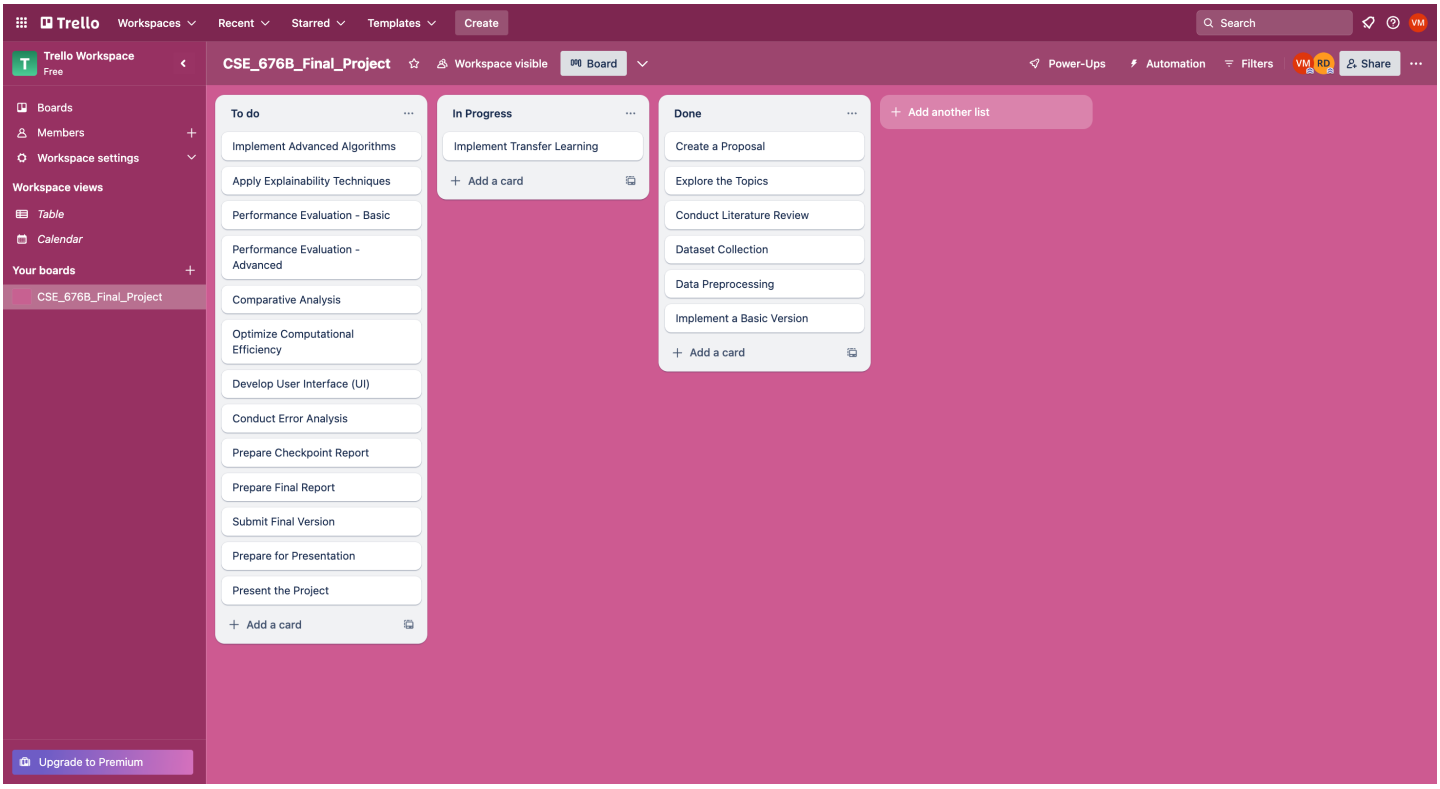


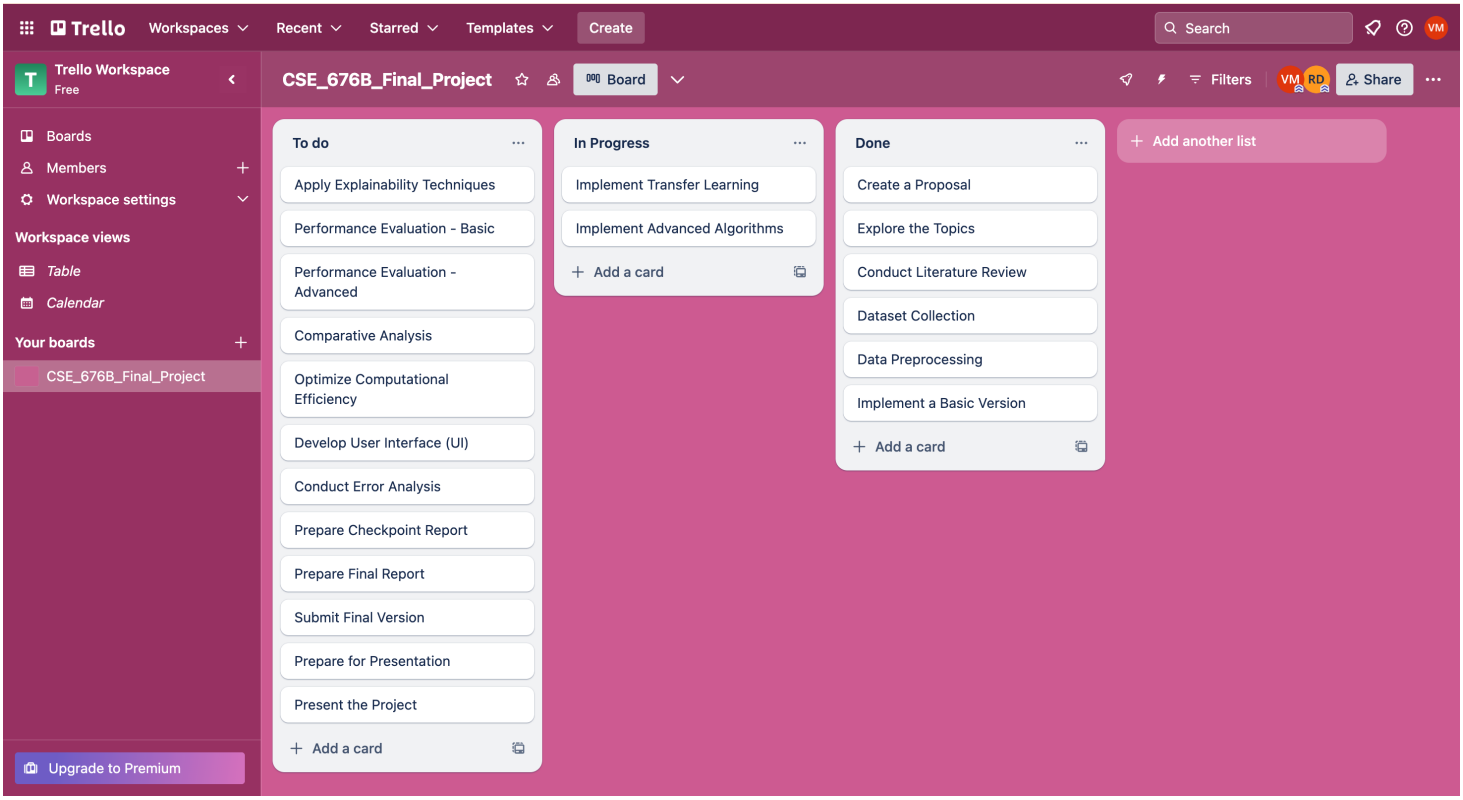

Trello
Workspaces
Recent
Starred
Templates
Create
Search
Trello Workspace
Free
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
To do
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Conduct Literature Review
Add a card
Done
Create a Proposal
Explore the Topics
Add a card
Add another list

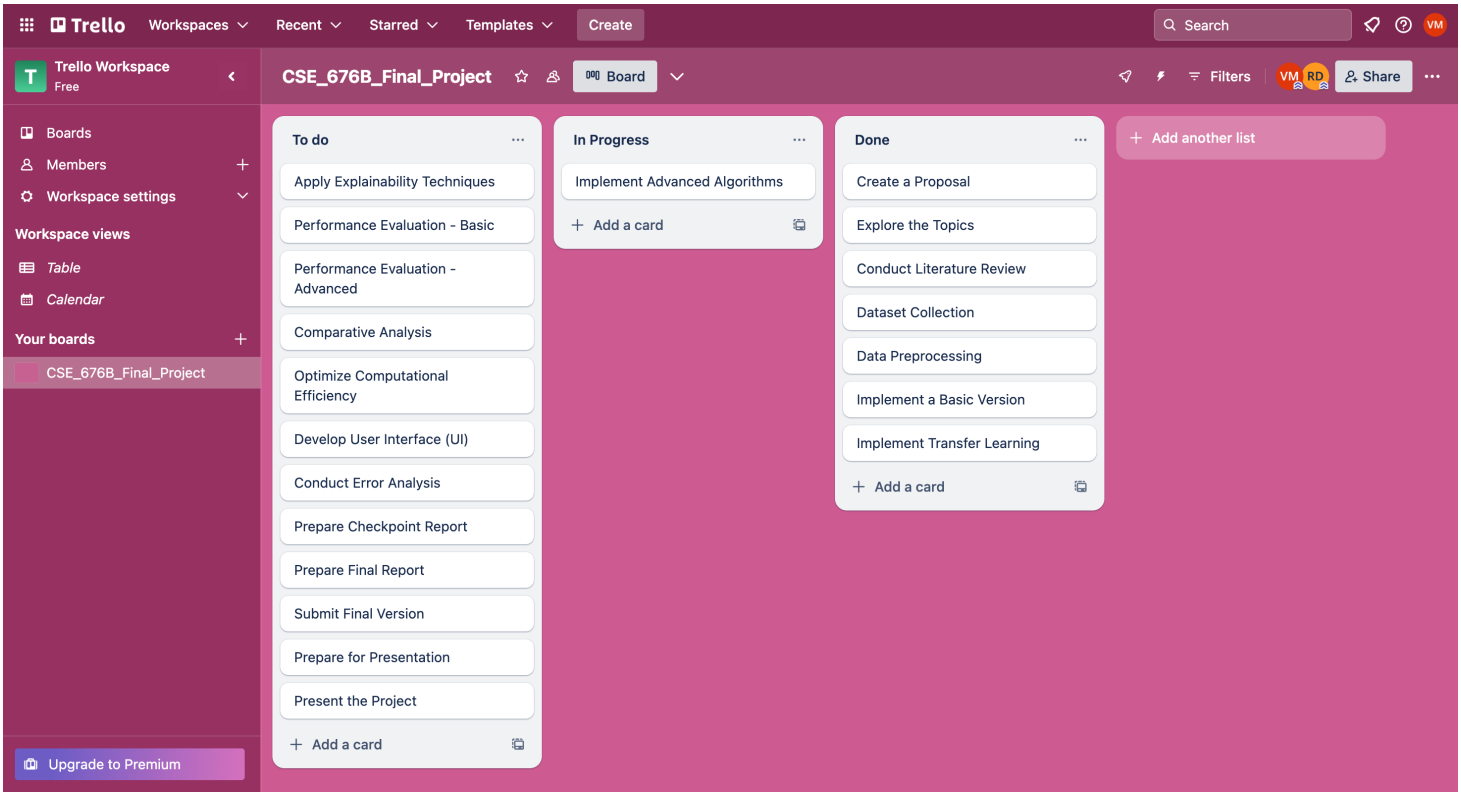

Trello
Workspaces
Recent
Starred
Templates
Create
Search
Trello Workspace
Free
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
To do
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Dataset Collection
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Add a card
Add another list

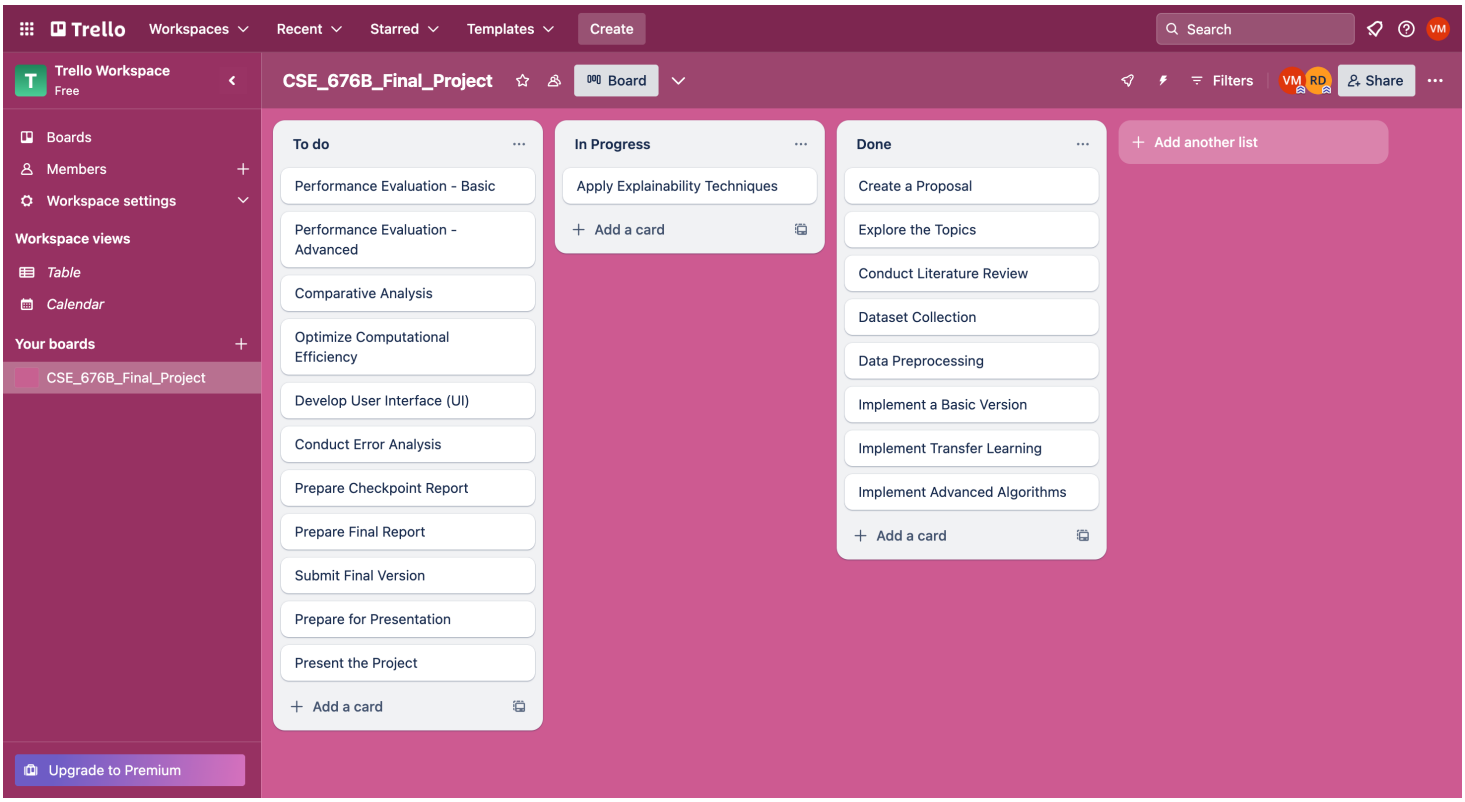

Trello
Workspaces
Recent
Starred
Templates
Create
Search
Trello Workspace
Free
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
To do
Implement a Basic Version
Implement Transfer Learning
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Data Preprocessing
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Add a card
Add another list

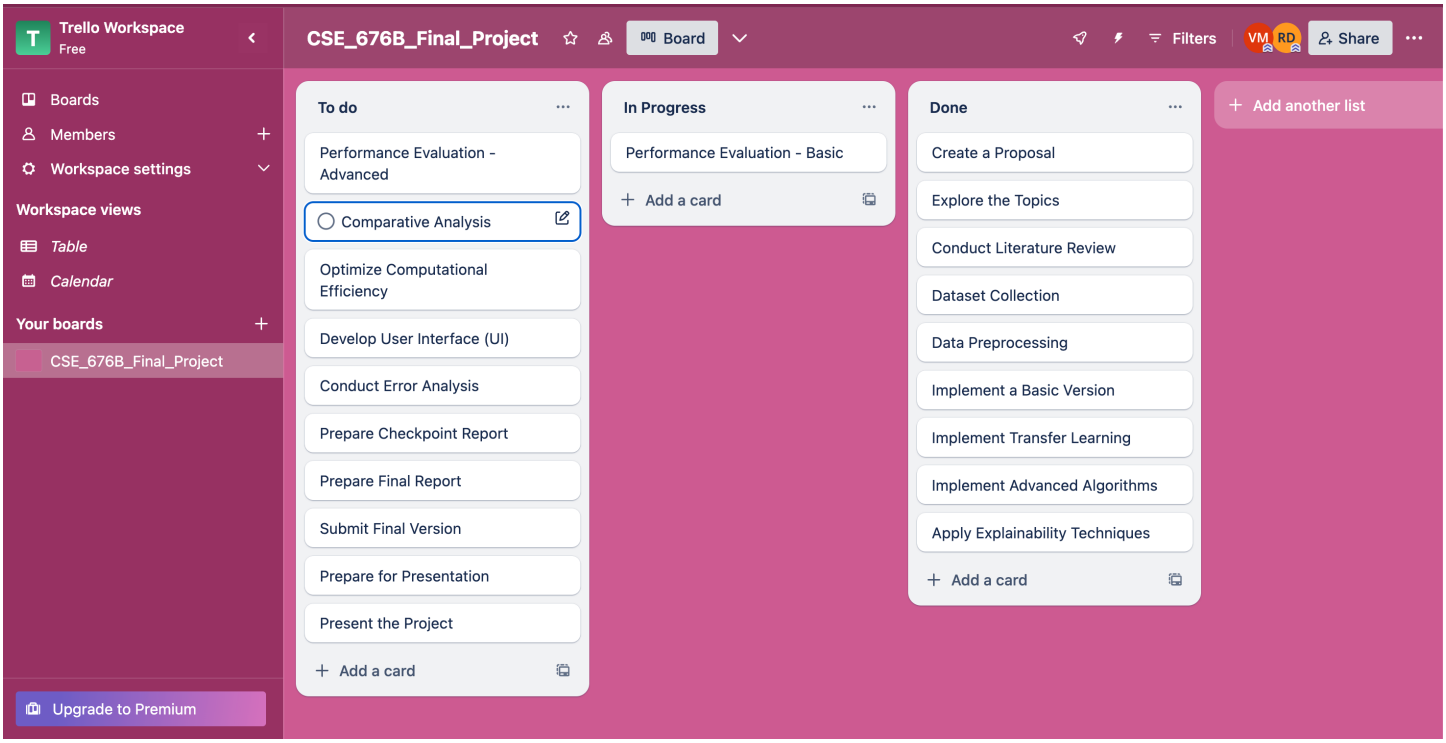

Trello
Workspaces
Recent
Starred
Templates
Create
Search
Trello Workspace
Free
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
To do
Implement Transfer Learning
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Implement a Basic Version
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Add a card
Add another list

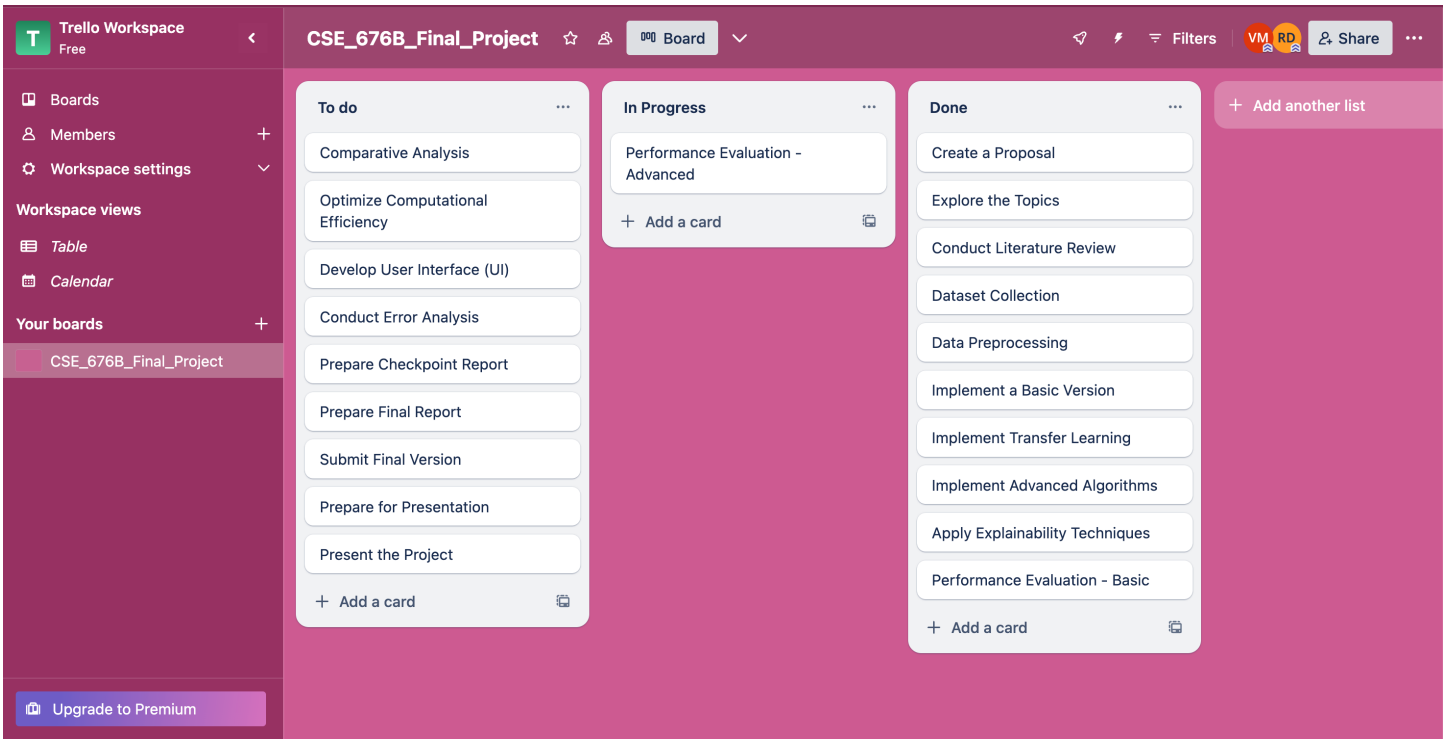

Trello
Workspaces
Recent
Starred
Templates
Create
Search
Trello Workspace
Free
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
To do
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Implement Transfer Learning
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Add a card
Add another list

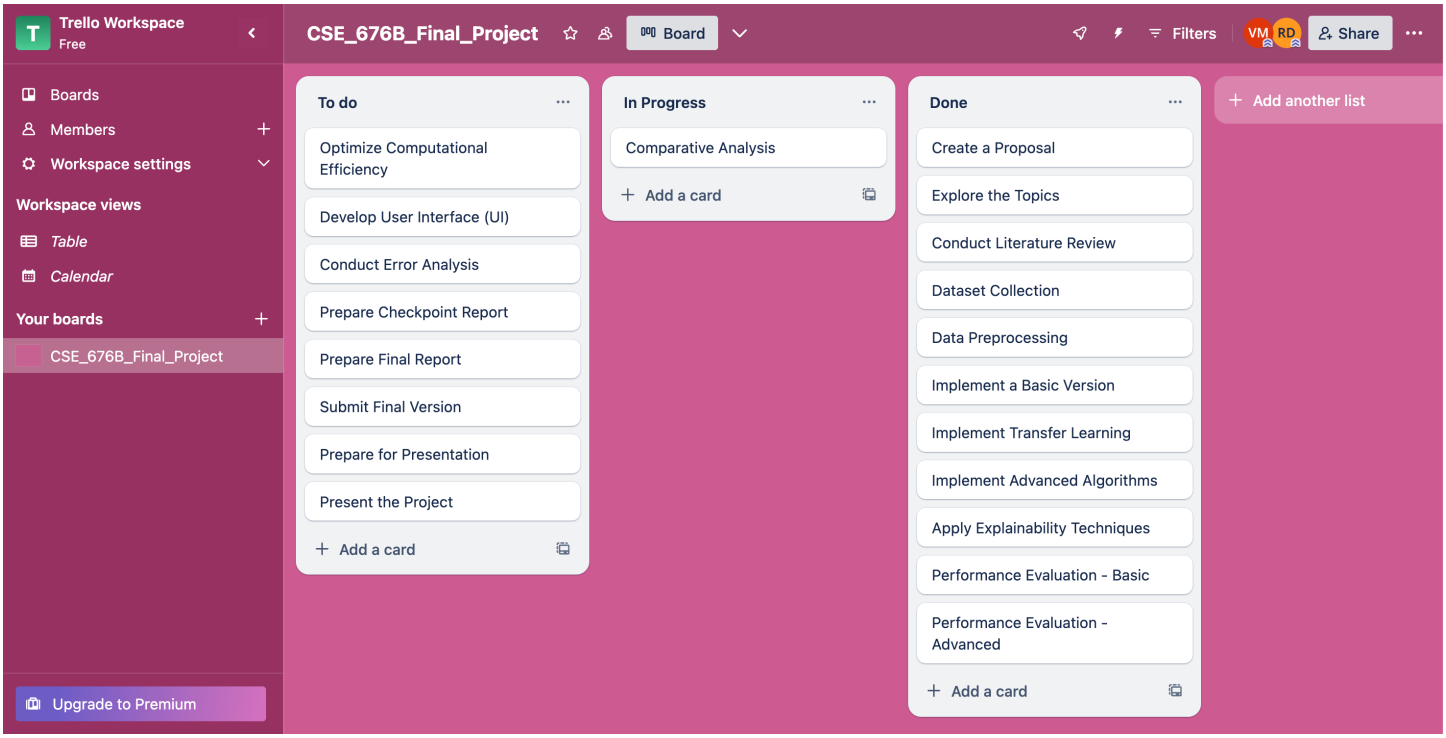

Trello
Workspaces
Recent
Starred
Templates
Create
Search
Trello Workspace
Free
CSE_676B_Final_Project
Board
Filters
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
To do
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Implement Transfer Learning
Implement Advanced Algorithms
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Add a card
Add another list

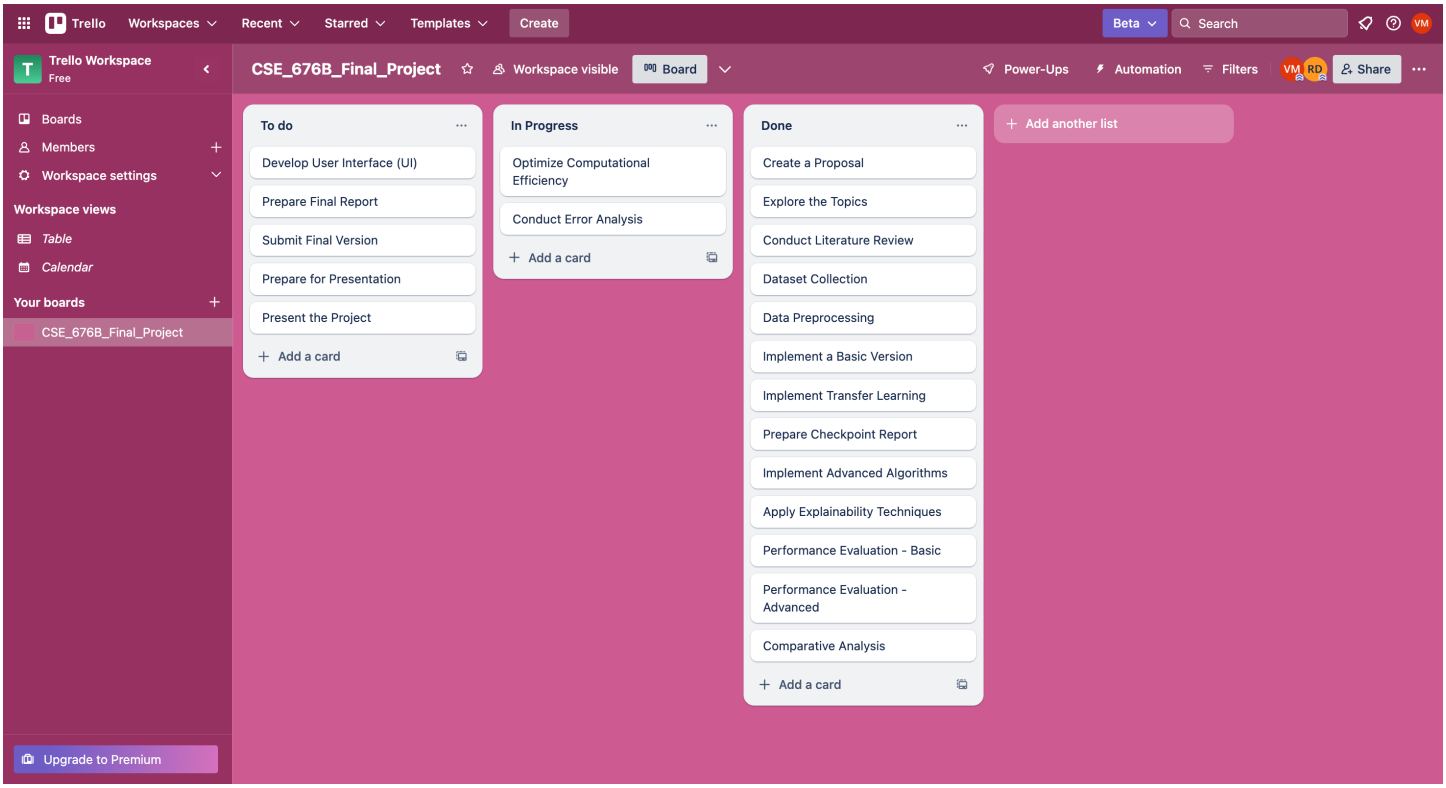

Trello
Workspaces
Recent
Starred
Templates
Create
Search
VM
Trello Workspace
Free
CSE_676B_Final_Project
Board
Filters
VM
RD
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
To do
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Implement Advanced Algorithms
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Add a card
Add another list

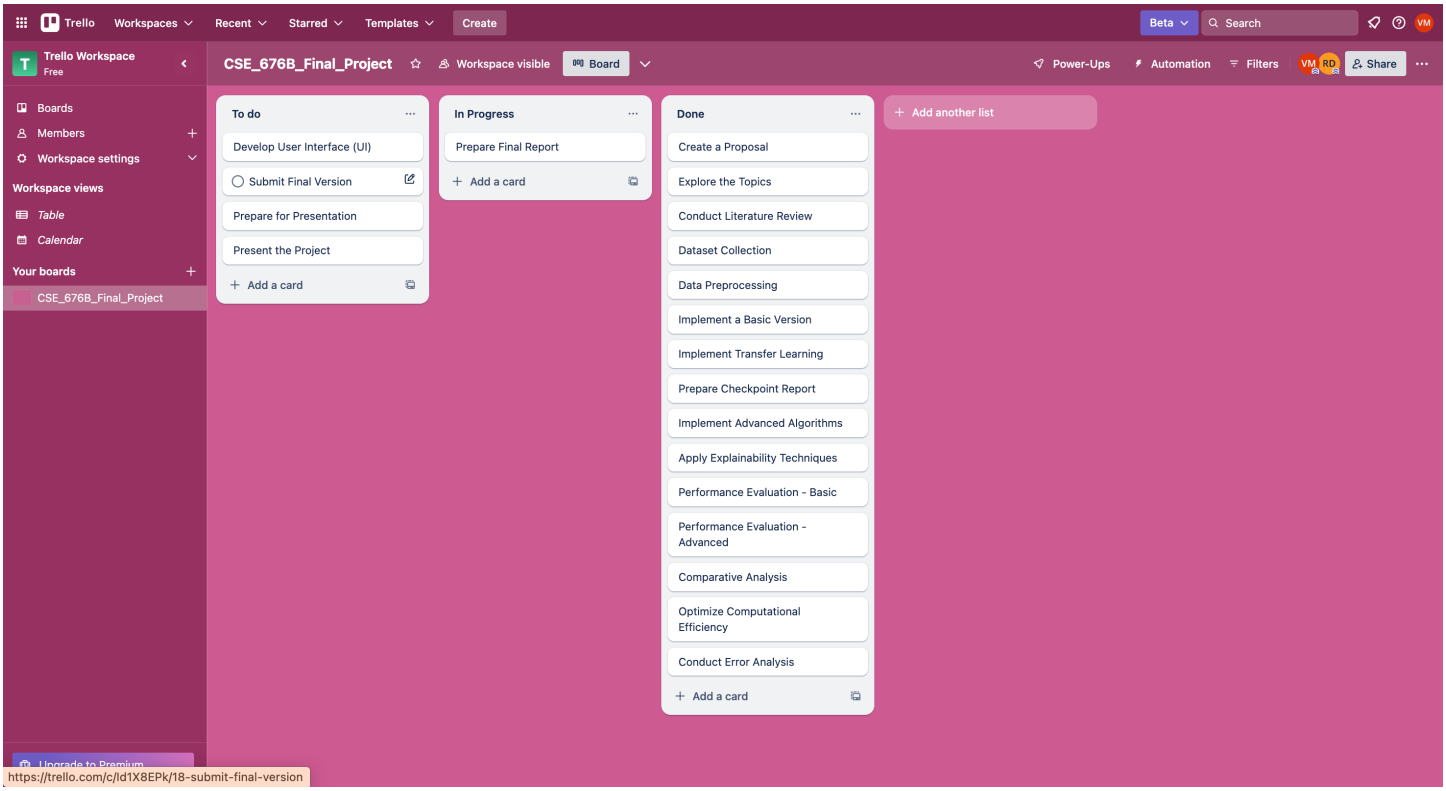

Trello
Workspaces
Recent
Starred
Templates
Create
Search
VM
Trello Workspace
Free
CSE_676B_Final_Project
Board
Filters
VM
RD
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
To do
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Apply Explainability Techniques
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Implement Advanced Algorithms
Add a card
Add another list

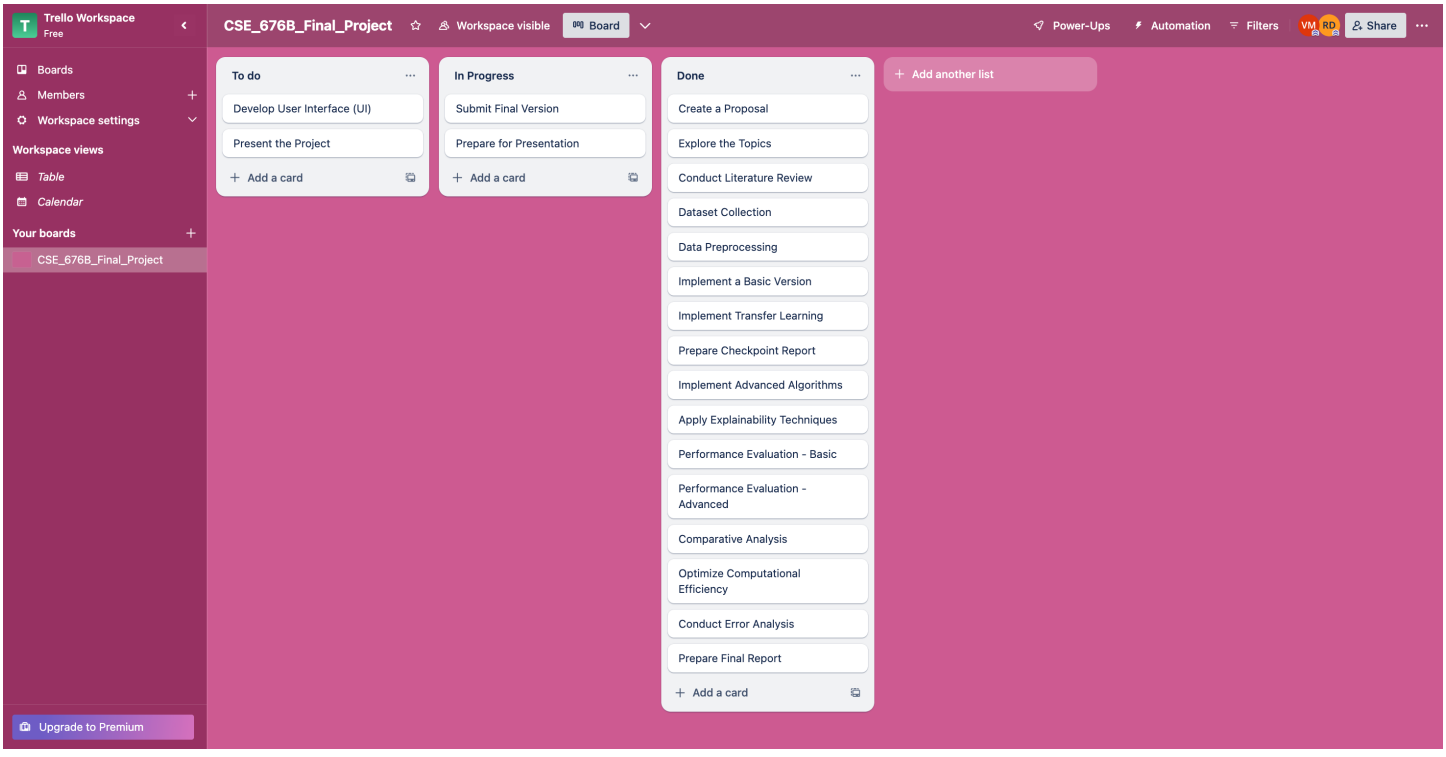

Trello Workspace
Free
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
CSE_676B_Final_Project
Board
Filters
VM
RD
Share
To do
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Performance Evaluation - Basic
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Implement Advanced Algorithms
Apply Explainability Techniques
Add a card
Add another list


Trello Workspace
Free
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
CSE_676B_Final_Project
Board
Filters
VM
RD
Share
To do
Comparative Analysis
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Performance Evaluation - Advanced
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Add a card
Add another list


Trello Workspace
Free
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
CSE_676B_Final_Project
Board
Filters
Share
To do
Optimize Computational Efficiency
Develop User Interface (UI)
Conduct Error Analysis
Prepare Checkpoint Report
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Comparative Analysis
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Add a card
Add another list


Trello
Workspaces
Recent
Starred
Templates
Create
Beta
Search
Trello Workspace
Free
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
Upgrade to Premium
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
To do
Develop User Interface (UI)
Prepare Final Report
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Optimize Computational Efficiency
Conduct Error Analysis
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Prepare Checkpoint Report
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Add a card
Add another list


Trello
Workspaces
Recent
Starred
Templates
Create
Beta
Search
Trello Workspace
Free
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
To do
Develop User Interface (UI)
Submit Final Version
Prepare for Presentation
Present the Project
Add a card
In Progress
Prepare Final Report
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Prepare Checkpoint Report
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Conduct Error Analysis
Add a card
Add another list
Upgrade to Premium
https://trello.com/c/ld1X8EPk/18-submit-final-version


Trello Workspace
Free
CSE_676B_Final_Project
Workspace visible
Board
Power-Ups
Automation
Filters
Share
Boards
Members
Workspace settings
Workspace views
Table
Calendar
Your boards
CSE_676B_Final_Project
To do
Develop User Interface (UI)
Present the Project
Add a card
In Progress
Submit Final Version
Prepare for Presentation
Add a card
Done
Create a Proposal
Explore the Topics
Conduct Literature Review
Dataset Collection
Data Preprocessing
Implement a Basic Version
Implement Transfer Learning
Prepare Checkpoint Report
Implement Advanced Algorithms
Apply Explainability Techniques
Performance Evaluation - Basic
Performance Evaluation - Advanced
Comparative Analysis
Optimize Computational Efficiency
Conduct Error Analysis
Prepare Final Report
Add a card
Add another list
Upgrade to Premium

Contribution:

Team Member	Project Part	Contribution (%)
Venkata Sri Sai Surya Mandava	Preprocessing	50
Rachana Dharmavaram	Preprocessing	50
Venkata Sri Sai Surya Mandava	Model Training	50
Rachana Dharmavaram	Model Training	50
Venkata Sri Sai Surya Mandava	Model Evaluation	50
Rachana Dharmavaram	Model Evaluation	50
Venkata Sri Sai Surya Mandava	Model Comparison	50
Rachana Dharmavaram	Model Comparison	50

References:

1. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html
2. https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html
3. https://en.wikipedia.org/wiki/Early_stopping
4. https://pytorch.org/vision/stable/transforms.html
5. https://en.wikipedia.org/wiki/F-score
6. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html
7. https://pytorch.org/tutorials/beginner/saving_loading_models.html
8. https://www.kaggle.com/datasets/tolgadincer/labeled-chest-xray-images
9. http://arxiv.org/abs/1704.04861
10. https://pytorch.org/vision/stable/models/mobilenetv3.html
11. https://pytorch.org/vision/stable/models.html