Pneumonia Classification

SUTD 50.039 Theory and Practice of Deep Learning

Win Tun Kyaw 1005265, Varshini Harthachitramalogan 1005185, Shaun Hin Fei 1005446

1. Introduction

For this project we receive grayscale X-ray images of the chest area and attempt to classify whether the patient is healthy (class 0), or has pneumonia (class 1). Childhood pneumonia is the number one cause of childhood mortality due to infectious diseases (Rudan et al. ,2008). A quick diagnosis and urgent treatment is needed in order to save the child. A Deep learning model that can detect signs of pneumonia can help to provide rapid diagnosis and referrals for these children (Kermany et al, 2018).

In the repository, there are notebooks numbered from 0 to 7.

Notebook 1-5: Training notebooks for different models
Notebook 6: Notebooks to load model weights and test
Notebook 7: To load model weights and test best performing model for each architecture

Although they have been seeded, their outcomes may not be deterministic due to batch normalization and dropout techniques. Hence, while notebooks 0 to 5 can be run, they may not produce all the necessary files to run all notebooks numbered 6. They are used for training purposes. Therefore, we advise against running the notebooks numbered 6 and have included the notebook numbered 7 to verify our best performing models whose weights can be downloaded from the following link: https://drive.google.com/drive/folders/1zcXmKO0L9nvmTLk23JpvpBmgLlQouqLa?usp=sharing

2. Dataset

This dataset of chest X-ray (anterior-posterior) images was created by researchers who wanted to test the generalisability of their AI system in detecting common diseases such as pneumonia. They were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou (Kermany et al., 2018). It contains a total of 5856 images, in which 1583 are Normal images and 4273 are Pneumonia images.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert. (Kermany et al., 2018). The original dataset used is found here: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

3. Data Augmentation

The original dataset was first redistributed into an 80-10-10% train-test-validation split. With just 1267 Normal images compared to 3419 Pneumonia images in the training set, any models would likely be biased towards class Pneumonia. The approach taken to address this issue was to randomly horizontally flip (with a 50% chance) all 1267 Normal images in the training set, then generate a plausible adversarial sample for each of them through random noising (see GenerateImages.ipynb).

This has the combined effect of introducing sample variety, making our models more robust to noisy data and generalizing better as a result, and expanding the Normal dataset to 2534. Images were not taken from the validation and test sets. This ensures that our models do not see any variations of those images, and thus enables us to make more accurate judgments of their robustness and generalizability.

The images are explicitly set to grayscale since they came in RGB format despite appearing to already be in grayscale. They were also resized to be 32x32 (from 64x64) so it will take up less memory and we are able to train larger models using the images. All the images undergo simple data augmentation where the below techniques are applied on the images to improve robustness and training stability of the model.

Techniques used:

1) RandomHorizontalFlip
2) Normalization

4. Methods

We implement 2 models: CNN and Transformer. We chose a convolutional neural network as the convolutional layer works well for images; the convolution operation incorporates information from the image while preserving the spatial dependency and homophily properties of the x-ray images. A transformer was chosen as the second model to investigate if attention mechanisms could produce equally good or better predictions. Additionally, we experimented with an Autoencoder to extract important features from our chest X-ray images before passing them to either of the models.

Training Procedure

A typical training procedure was used, with a stochastic mini-batch gradient descent (where the data is shuffled before drawing each batch). The added randomness not only helps to address the remaining slight imbalance in data samples (2534 Normal vs 3419 Pneumonia) by shuffling them together, but it also helps to reduce bias and improve robustness of the model by preventing it from memorizing the order of images.

Cross entropy was chosen as the loss function due to ease of use and suitability for binary classification tasks. Adam optimizer was selected as it already implements learning rate control and momentum, and requires minimal hyperparameter tuning to achieve good performance. An early stopping mechanism (see EarlyStopper in helpers.py) was used to limit overfitting; training stops when validation loss continuously increases for a specified number of iterations (we refer to this as patience).

Additionally, we experimented with alternating between ReLU and LeakyReLU activation functions, as they are computationally efficient while achieving good performance. We also experimented with Xavier initialization compared to the default He initialization for linear layers in our models. Ultimately, it was concluded that there was an insignificant difference in performance. This could be because the binary classification task is still simple enough to not require such fine adjustments.

Convolutional Model (CNN)

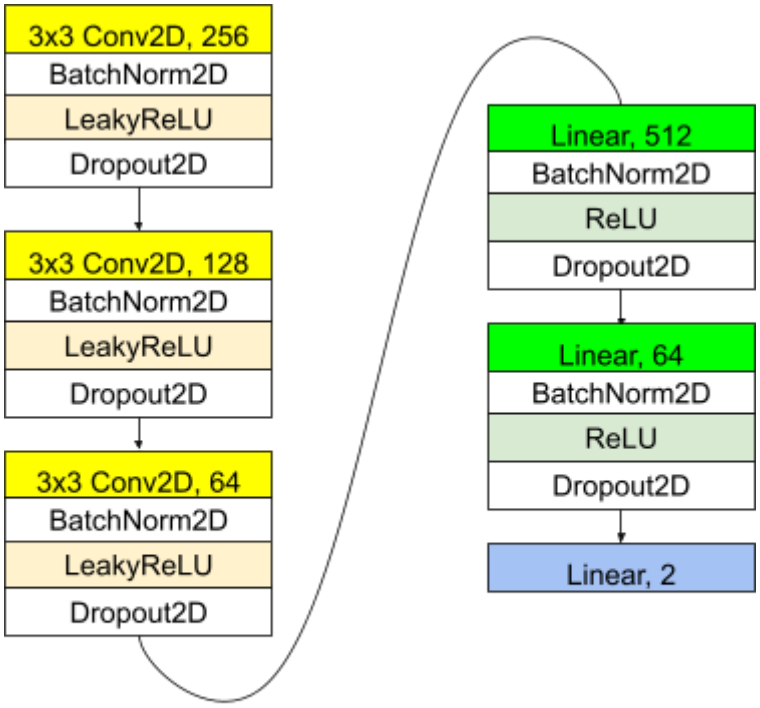


Fig. 1 CNN Model Architecture

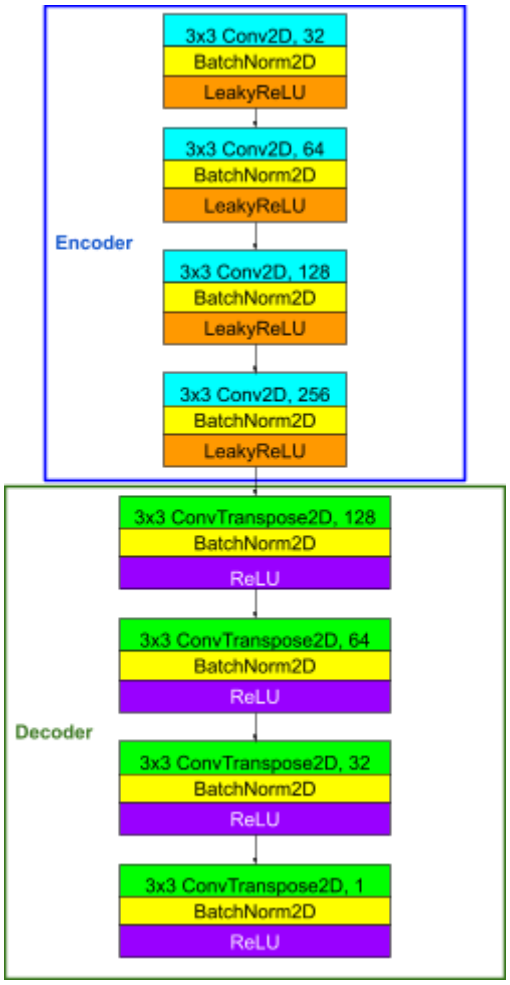
The first iteration of our model had 3 convolution layers with output to 2 linear layers in sequence (see CNNArchitecture.svg), which gave an initial test accuracy of 86.84% on the testing set.

We experimented with using different kernels for the convolution layers. Looking at the image samples, we decided to use kernels for sharpening and edge detection, as some of the pneumonia x-ray images had swelling of the lungs which appeared as larger silhouettes in the images when compared to the normal images.

Inspecting the default kernel weights that each CNN layer would use and referencing the PyTorch documentation, each layer was typically initialized using the Kaiming uniform distribution. We used a sharpening kernel for the first convolution layer which had the values [[0., -1., 0.], [-1., 5., -1.], [0., -1., 0.]] to increase the differences in values between neighbouring pixels. We then used 2 Sobel filters in the following convolution layers, one to detect vertical edges, and one to detect horizontal edges, which have the values [[1., 2., 1.], [0., 0., 0.], [-1., -2., -1.]], and [[1., 0., -1.], [2., 0., -2.], [1., 0., -1.]] respectively.

Autoencoder + CNN

Fig.2 AutoEncoder Diagram



We also experimented with building an autoencoder model in combination with the CNN model. We believed that compressing the images into smaller vectors will only record the most important features of the images and using the encoded images to train the CNN model can allow the model to be more accurate in its predictions.

We created an Encoder class composed of Conv, BatchNorm and Leaky Relu for each layer; a Decoder class of ConvTranspose, BatchNorm and Relu for each layer. Then both encoder and decoder layers are combined to form an AutoEncoder. There is a get_features function that returns the encoding of the images, which will be saved and used as inputs for the CNN models.

The images are fed as inputs for the AutoEncoder model and trained for the specified number of epochs. We extract the encoded images. The encoded images are fed as inputs into the CNN model for training (see AutoencoderCNN.svg).

Transformer

Our transformer architecture starts off with several convolutional layers with batch normalization, leaky ReLU activation, and dropout, followed by max pooling and finally a flattening operation. This was to produce the input ‘embeddings’ of our images while preserving the spatial dependency and homophily properties of images. Subsequently, a TransformerBlock was defined to execute the following operations.

TransformerBlock (multi-head self-attention with feedforward)

1. Save a copy of the input embeddings, A.
2. Attention layer - Inputs are passed through a multi-head self-attention layer to learn the importance of each pixel relative to other surrounding pixels. Input sizes are preserved.

3. Skip connection - Outputs of the attention layer are added together with A, then passed through a batch normalization layer.
4. Save a copy of the normalized outputs, B.
5. Highway layer - Normalized outputs are passed through highway layers to learn selective retention of important information. Input sizes are preserved.
6. Skip connection - Outputs of the highlayer layer are added together with B, then passed through a batch normalization layer.
7. Output - Normalized outputs are passed to a final linear layer with leaky ReLU activation, batch normalization, and dropout to reduce the input size to a specified output size.

The multi-head self-attention layer was implemented exactly as highlighted in Appendix A. Meanwhile, the highway layer was implemented with several linear layers as follows.

$$input = x$$
$$transform = t = \sigma(W_2 x + b_2)$$
$$output = t \odot (W_1 x + b_1) + (1 - \mathrm{t}) \odot x$$

Figure 3: Highway layer implementation

Several such TransformerBlocks were executed in succession, followed by a final linear linear layer with batch normalization, leaky ReLU activation and dropout to produce an output size equal to the number of labels (in this case, 2). The following figure is an overview of the Transformer architecture used.

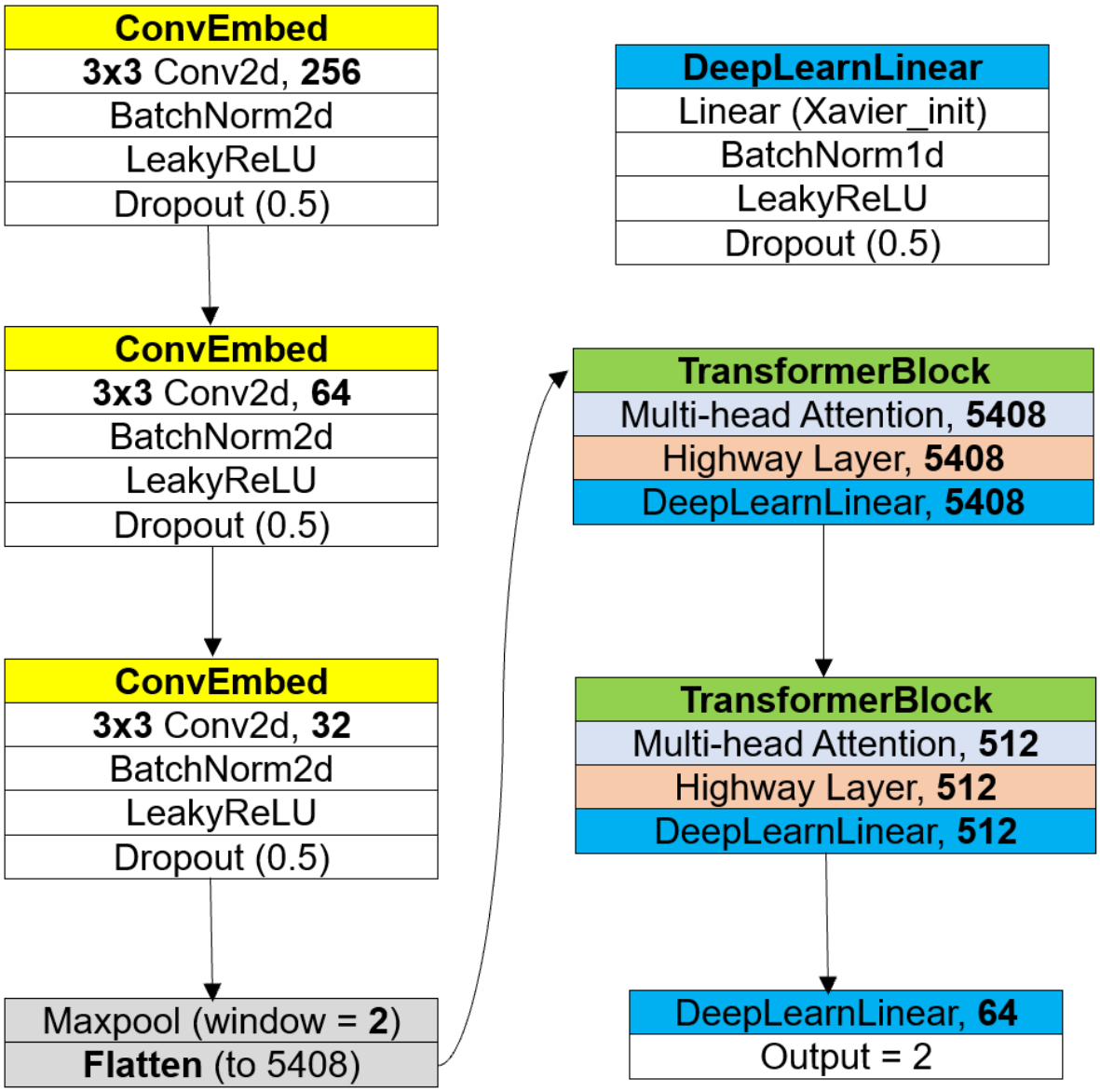


Figure 4: Transformer Overview

Autoencoder + Transformer

As with the convolutional neural network, the images were first passed through an autoencoder for feature extraction before being processed by the transformer (see TransformerCNN.svg). Our belief is that this would help the model learn with greater efficiency.

5. Results & Experiments

Initial Results
The following were used when training all the models:
Kernel size: 3
Optimiser: Adam (learning rate=0.001)
Batch size:128
EarlyStopper (patience = 3)
Epochs: 20

CNN
Test loss: 0.3719, Test accuracy: 0.8684
Precision: 0.8723404255319149
Recall: 0.9601873536299765
F1 score: 0.9141583054626533

The following figures display the confusion matrix and training curves of the CNN.

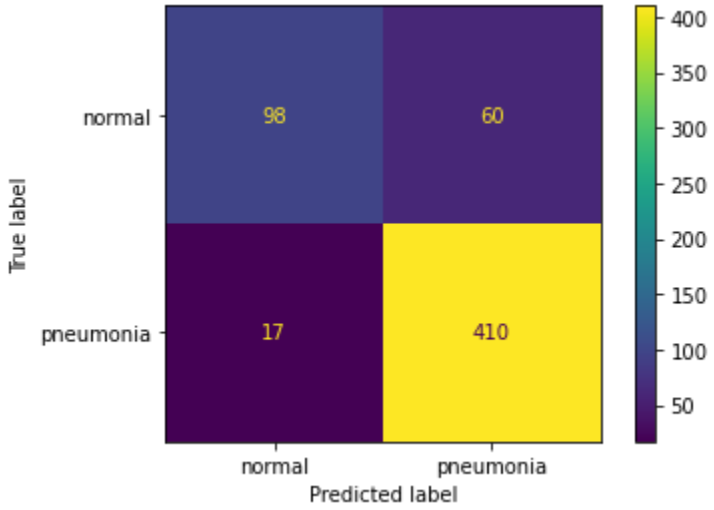


Fig.5 Confusion Matrix for CNN

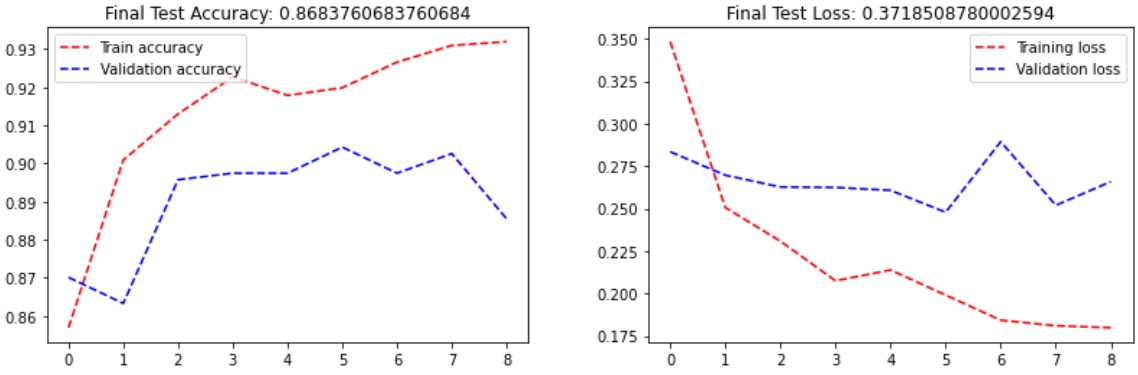


Fig 6. Accuracy and Loss curves for CNN

AutoEncoder + CNN

Test loss: 0.4291, Test accuracy: 0.8308
Precision: 0.8581
Recall: 0.9204
F1 score: 0.8881

The following figures display the confusion matrix and training curves of the CNN trained with features extracted by the AutoEncoder.

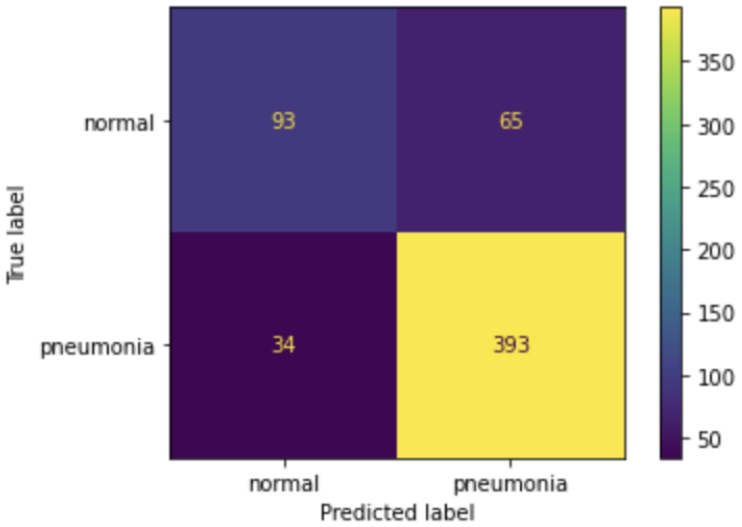


Fig.7 Confusion Matrix for CNN

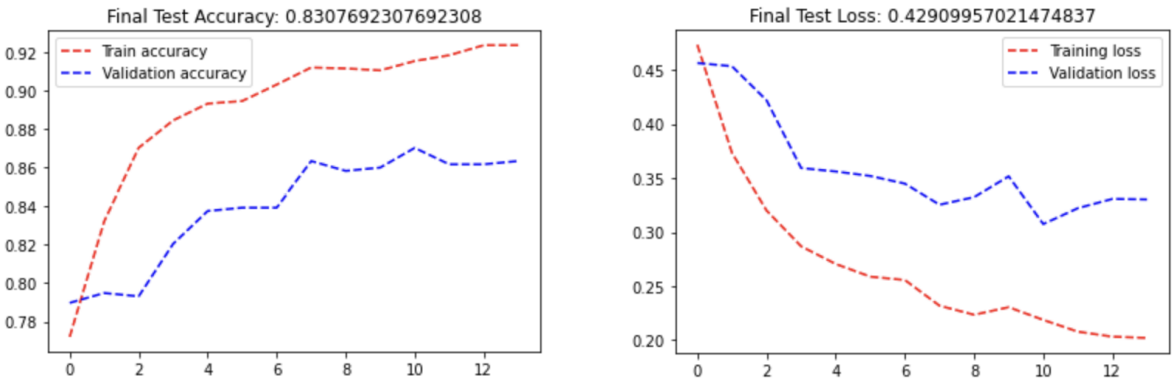


Fig.8 Accuracy and Loss curves for Autoencoder + CNN

Comparing with the confusion matrix produced by the CNN model (Fig.5), it can be seen that both have the same performance in terms of predicting the normal images. The autoencoder model performs worse in predicting pneumonia images, suggesting that the encoding may not have helped improve the model's capabilities.

Training the CNN models with the encoded images did not help to improve the accuracy of this model. Its accuracy (84.4%) is still lower than the accuracy produced by the CNN model (86.8%).

Transformer

Test loss: 0.3059, Test accuracy: 0.8872
Precision: 0.8784
Recall: 0.9813
F1 score: 0.9270

The following figures display the confusion matrix and training curves of the Transformer.

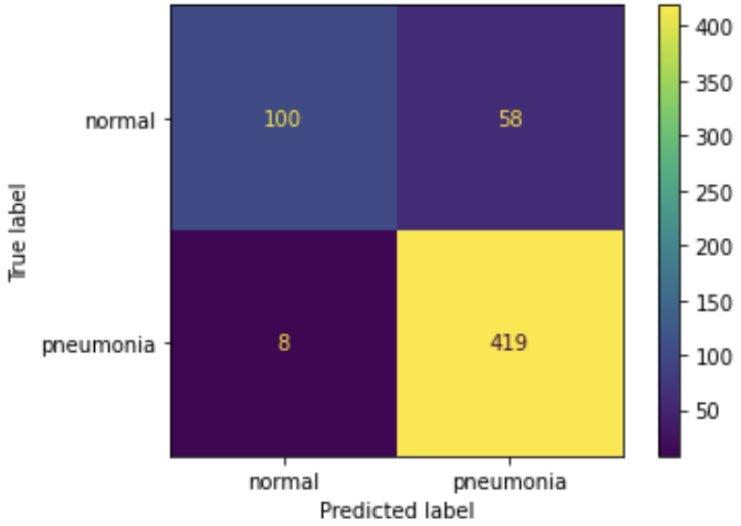


Fig. 9 Confusion Matrix for Transformer

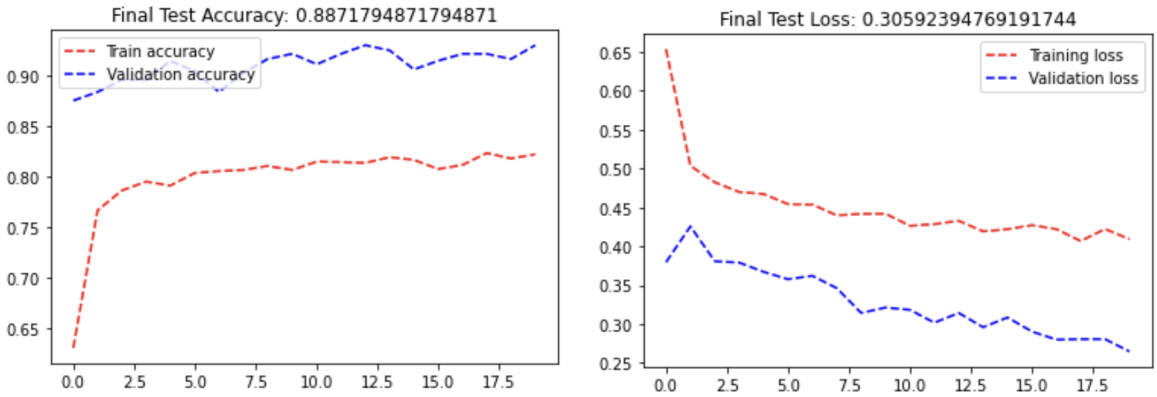


Fig. 10 Accuracy and Loss curves for Transformer

Notably, the validation loss continued decreasing for the entire 20 epochs even with an early stopping mechanism. Furthermore, the resulting model weights were very large (~2 GB) compared to the other models. This is unsurprising, as the Transformer is a complex architecture with many trainable parameters.

It also performs much better than the CNN models, it is able to correctly predict more images - 419 for Pneumonia, 100 for Normal (Fig. 9) as compared to CNN's - 410 for Pneumonia, 98 for Normal (Fig.5) .

AutoEncoder + Transformer

Test loss: 0.3202, Test accuracy: 0.8855
Precision: 0.8896
Recall: 0.9625
F1 score: 0.9246

The following figures display the confusion matrix and training curves of the Transformer trained with features extracted by the AutoEncoder.

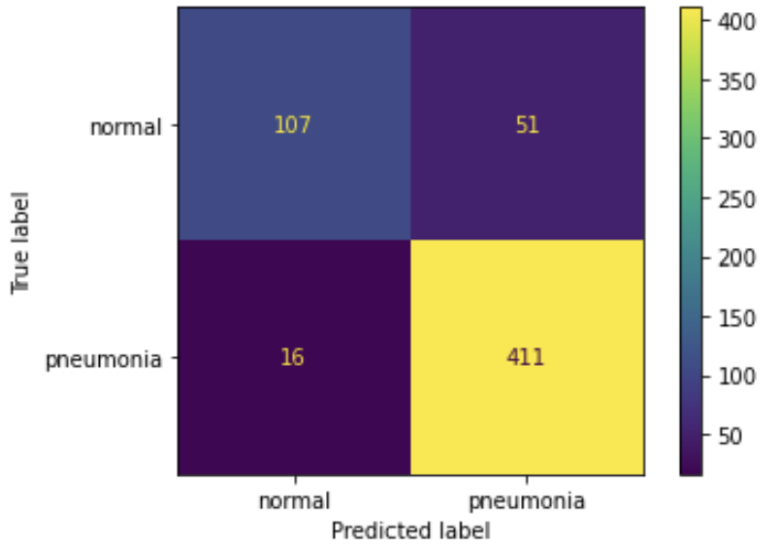


Fig. 11 Confusion Matrix for Autoencoder and Transformer

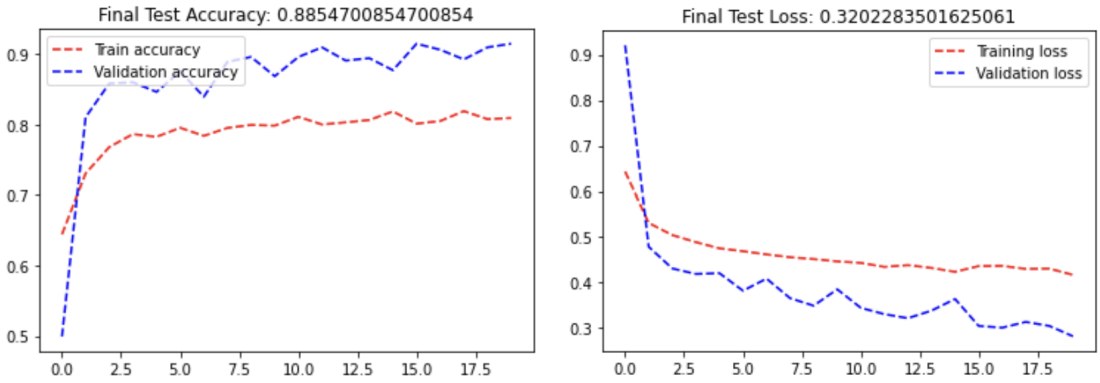


Fig. 12 Accuracy and Loss curves for Autoencoder and Transformer

As with the base Transformer, the validation loss continued decreasing for the entire 20 epochs. While the AutoEncoder didn't seem to make much difference, the resulting model weights were significantly smaller (~500 MB) than without the AutoEncoder. This is likely because the encoder part of the AutoEncoder has produced more sparse feature representations that caused some parameters to be set to zero.

Hyperparameter Tuning for CNN

We also perform hyperparameter tuning on the models using these parameters:

1) *hidden_list=[[256, 256, 128, 64], [256, 512,128,64], [256,512,128]]*
   (number of hidden convolutional layers and neurons in each layer)

2) *kernel_list=[3,5]* (kernel size in convolutional layers)

3) *window_list=[2,3,5]* (maxpool kernel size)

4) *hidden_linear_list= [[flattened, 512, 64], [flattened,1024, 512], [flattened,1024,512,256]]*
   (number of hidden linear layers and neurons in each layer)

5) *lr_list=[0.01,0.005,0.001]* (learning rate of optimiser)

6) *dropout_list=[0.3,0.5,0.7]* (dropout probability of Dropout2D layer)

From our experiments, it could be observed that having a smaller kernel size (*kernel size=3*), more layers and more neurons in each layer and a *dropout = 0.5* gave better accuracy and f1 scores.

This is an expected behaviour as having a smaller kernel size means the images do not shrink quickly as they go through the layers, retaining more image features; having more layers and neurons allows the model to learn complex patterns; having dropout prevents overfitting of the model. For the other parameters, each model (CNN and Transformer) had different best performing parameters.

For CNN, a max pooling *window size = 3* and *learning rate=0.01* were the best performing parameters in terms of accuracy and f1 score. We thought there would be significant changes in the accuracy and loss values but there were very minimal differences caused by all the hyperparameter tuning. All changes in hyperparameters yielded an accuracy between 88% and 90%, with a loss value of around 0.2 to 0.4 (Appendix B-G). It seems that the model has reached a plateau where no matter what changes are implemented on it, it doesn't go past a certain accuracy.

Continued Training for Transformers

Since both Transformers trained for the entire 20 initial epochs without being stopped early, their training was continued to check if their performance could be improved. Hence, both models were trained to 29 epochs (which was when training was stopped early). By loading the checkpoint made at 26 epochs, when the validation loss was lowest, we get our best performing models for both the base Transformer and the Transformer trained with the AutoEncoder's features.

Transformer (Weights filename = Transformer_checkpoint, for steps refer to 9.3.2)

Test loss: 0.2814, Test accuracy: 0.9077
Precision: 0.9011
Recall: 0.9813
F1 score: 0.9395
The following figure displays the confusion matrix of the Transformer trained until 26 epochs.

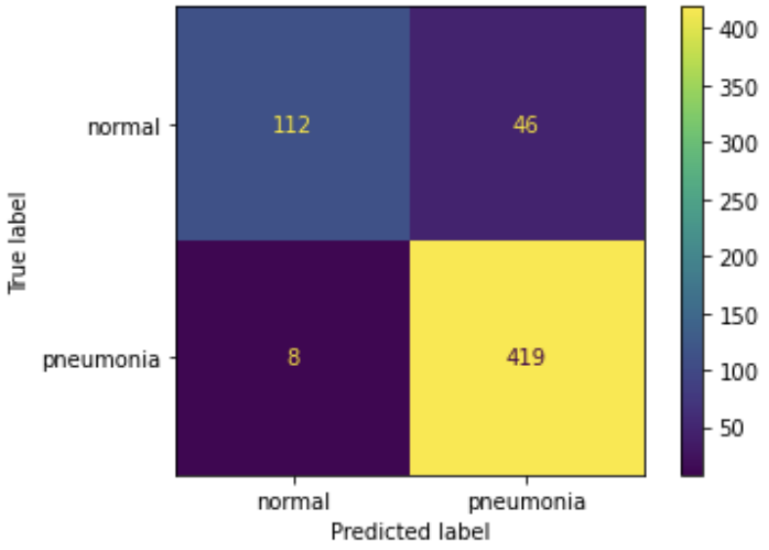


Fig. 13 Confusion Matrix for Transformer

AutoEncoder + Transformer (Weights filename= Transformer_checkpoint_withAE, for steps refer to 9.3.2)

Test loss: 0.2975, Test accuracy: 0.9009
Precision: 0.9073
Recall: 0.9625
F1 score: 0.9341
The following figure displays the confusion matrix of the Transformer trained with features extracted by the AutoEncoder until 26 epochs.

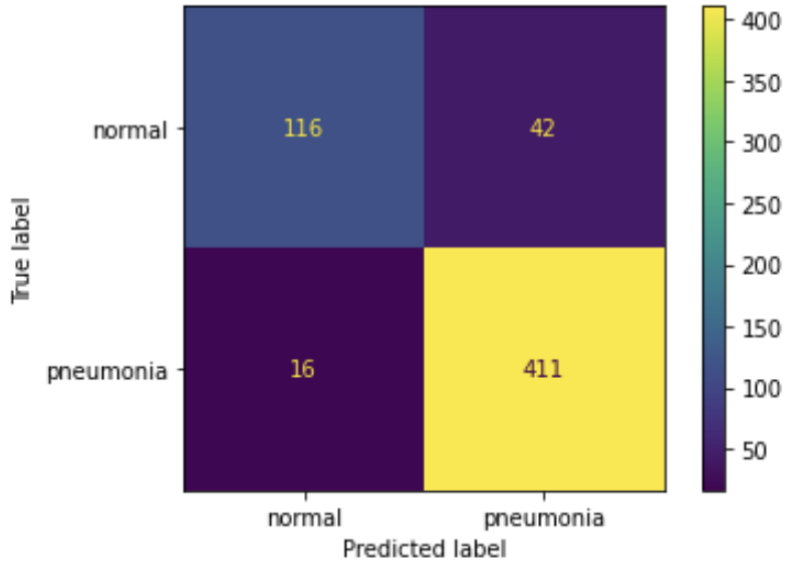


Fig. 14 Confusion Matrix for Autoencoder and Transformer

Continued training of the Transformers did not improve the recall significantly, but noticeably improved the precision and f1 score of both Transformers. In particular, the base Transformer saw an improvement in precision from 0.8784 to 0.9011, and f1 score from 0.9270 to 0.9395. As for the Transformer trained with the AutoEncoder, there was an improvement in precision from 0.8896 to 0.9073, and f1 score from 0.9246 to 0.9341.

Performance Comparison (ResNet)

We conducted a comparison against the ResNet34 model, as ResNet is one of the more popular and advanced models used for Computer Vision problems. It managed to obtain an accuracy of 90.6%, with similar recall, precision and f1 scores to our Transformer architecture. The Transformers we trained thus achieved comparable performance to Resnet34.

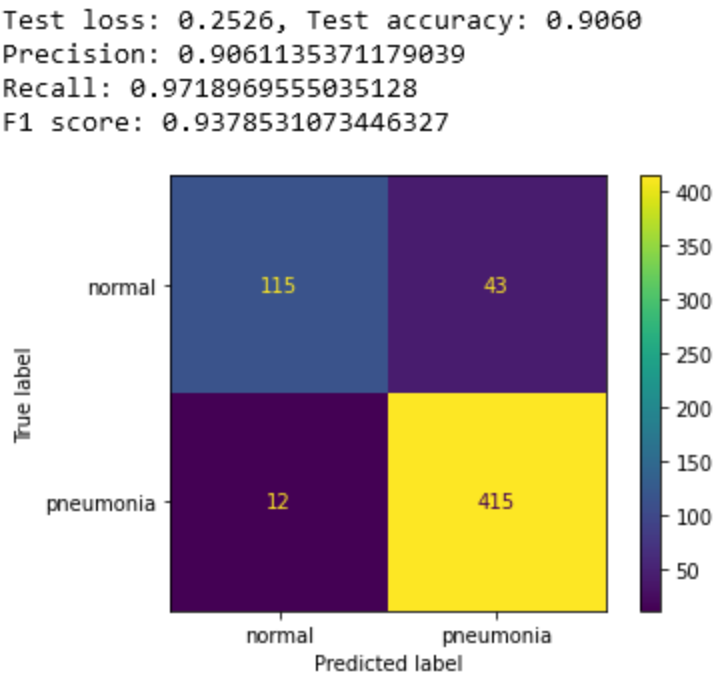


Fig. 15 Confusion Matrix for ResNet

6. Analysis

The models can't pass a certain threshold (~90%) in terms of accuracy. Even in the study for which these datasets were created, their model only managed to obtain an accuracy of 92.8% when trained over 100 epochs. We believe the reasons behind this could be:

1) **Distinguishing images:** Some images from the normal set and the pneumonia are too similar and difficult for even humans to distinguish. For example, below are 2 figures: the top one shows the images correctly predicted by a Transformer model, and the bottom shows the images wrongly predicted by the same model.

   As can be seen from the pictures, it is difficult to distinguish a normal picture from a pneumonia picture, therefore it is understandable that even the model also is unable to predict at a higher accuracy.

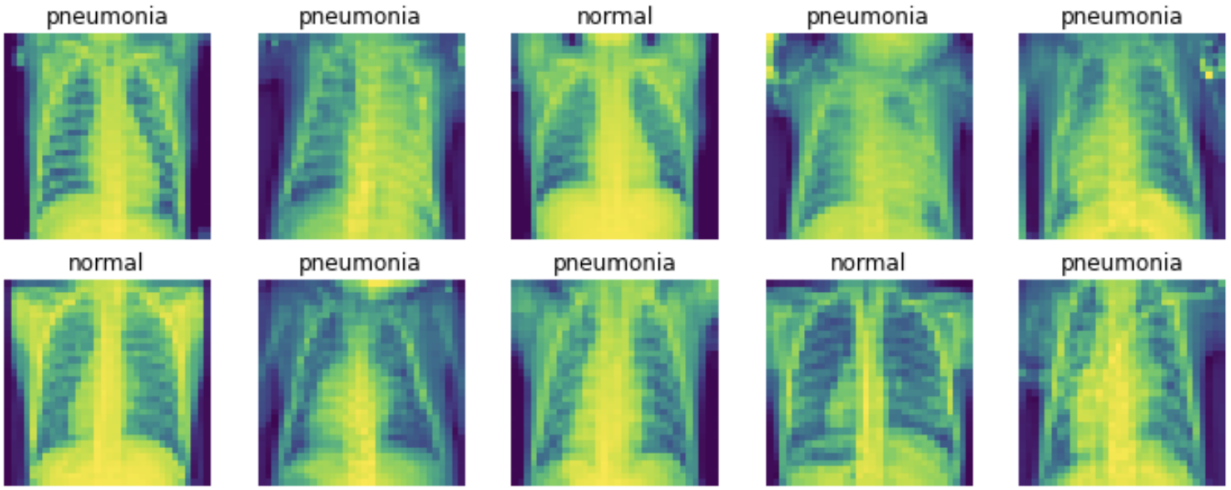


Fig.16 Correct list

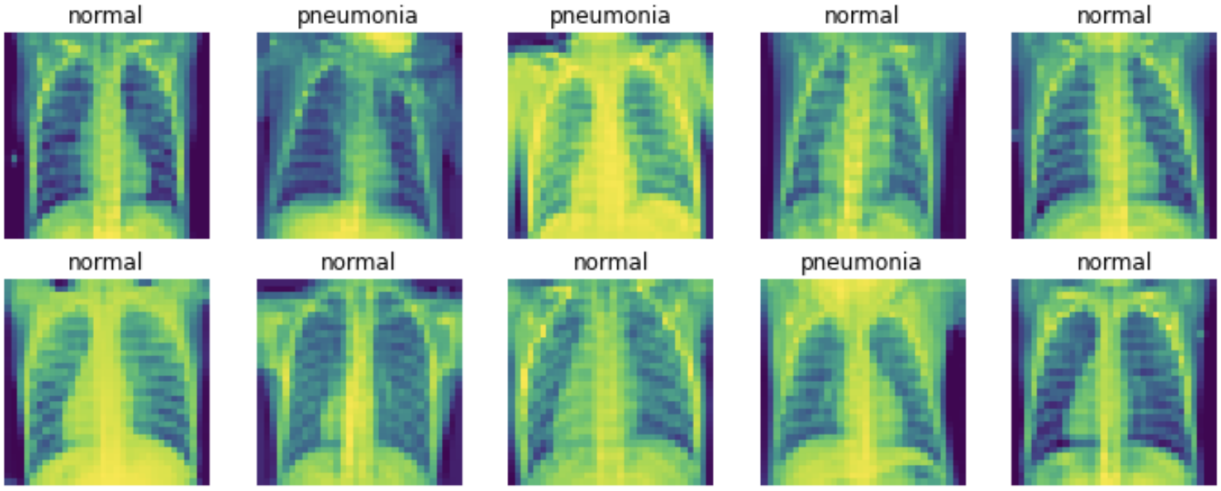


Fig.17 Wrong list

2) **Shallow Network:** There may not be enough layers and neurons to capture the complex patterns in these pictures. The maximum neurons we could use were 512 and a maximum of 4 layers before our workstations were unable to support the training of the model. Having a bigger CNN could potentially allow the model to detect the complex patterns and improve its performance, as seen by ResNet having a slightly better performance than the models we trained.

3) **Imbalanced dataset:** More Normal images were needed. All the models tend to perform much better in predicting Pneumonia images as compared to Normal images. This could be just because of there being lesser Normal training images, so the model has not learnt enough about Normal pictures to predict it more accurately. However, such a dataset may not be feasible to obtain as x-ray imaging would be performed on patients presenting with pneumonia-like

symptoms, hence healthy patients would be x-rayed less frequently explaining the relative lack of normal images available.

Additionally, CNN models trained with the encoded images also performed slightly worse than when the model was trained with the images. This could be due to the fact that when encoded the images are compressed into its most essential components, and this may cause some image features to be lost. That said, the encoded images enabled the Transformer to achieve similar performance to the base Transformer with a smaller weight file. Therefore, we would overall prefer to have the Transformer trained with features extracted by the AutoEncoder.

7. Conclusion

We implemented, and tested the CNN model, autoencoder architecture, and transformer architecture to classify x-ray images of healthy patients and those with pneumonia, then compared our results against the state-of-the-art ResNet34 model as the standard. Since our final transformer model with early stopping produced comparable results to the state-of-the-art model, we conclude that our model is capable of performing the classification task to a high standard. Although we could not achieve accuracy beyond the ~90% threshold, further improvements can be made by addressing the limitations in our analysis section.

8. Individual Contributions

Win Tun Kyaw:
Convolutional Neural Network & Transformer architecture, generating adversarial samples, helper functions, report

Varshini: Autoencoder + CNN architecture, hyperparameter tuning, report

Shaun: CNN architecture, report

9. How to run the model

In order to achieve the results we have shown in sections 4 and 5, please follow the below steps.

1.CNN

1) Open 1_CNNModel.ipynb

2) Run all the cells under ‘Imports & Setup’, ‘Prepare Dataset’
3) Go to the ‘Initialize Model’ section. Change any parameters if needed. Then run all the cells in this section ‘Initialize Model’ and ‘Training model’ sections
4) Run the ‘Testing Model and Performance Curves’ to view the testing results
5) Save the model weights with your preferred filename

2.AutoEncoder + CNN

1) Open 2_Autoencoder.ipynb
2) Run all the cells under ‘Imports & Setup’, ‘Prepare Dataset’ sections
3) Under the ‘Initialize Autoencoder’ section, run the 1st cell. Change the number of hidden layers and their sizes or kernel sizes if needed. Run the remaining cells in the ‘Initialize Autoencoder’ section
4) Go on to ‘Train the Autoencoder’ section. Change the number of epochs if needed. Run the cells under this section
5) Run the ‘Visualize AutoEncoder Outputs’ to view the decoded images
6) Save the autoencoder model weights
7) Open 3_CNNModelWithAutoencoder.ipynb
8) Run all the cells under ‘Imports & Setup’, ‘Prepare Dataset’
9) Go to the ‘Load the Autoencoder’ section. Change filename if needed. If not, use default filename = ‘autoencoder256_4epochs’. Run all the cells in this section.
10) Go to the ‘Initialize Model’ section. Change any parameters if needed. Then run all the cells in this section
11) If doing hyperparameter tuning, skip to step 14
12) Run all the cells under ‘Training model’ and ‘Testing Model and Performance Curves’ sections to view the testing results
13) Save the model weights with your preferred filename

For a bit more streamlined hyperparameter tuning, move on to ‘Hyparam Tuning’ section

14) Change any hyperparameters needed.
15) The lines to edit are commented on the 2nd cell in this section. Once changed, run all the cells in this section

3.Transformer

1) Open 4_Transformer.ipynb
2) Run all the cells under ‘Imports & Setup’, ‘Prepare Dataset’
3) Go to the ‘Initialize Model’ section. Change any parameters if needed. Then run all the cells in this section ‘Initialize Model’ and ‘Training model’ sections
4) Run the ‘Testing Model and Performance Curves’ to view the testing results
5) Save the model weights with your preferred filename

4. Autoencoder + Transformer

1) Open 5_TransformerWithAutoencoder.ipynb
2) Run all the cells under ‘Imports & Setup’, ‘Prepare Dataset’
3) Go to the ‘Load the Autoencoder’ section. Change filename if needed. If not, use default filename = ‘autoencoder256_4epochs’. Run all the cells in this section.
4) Go to the ‘Initialize Model’ section. Change any parameters if needed. Then run all the cells in this section
5) If doing hyperparameter tuning, skip to step 14
6) Run all the cells under ‘Training model’ and ‘Testing Model and Performance Curves’ sections to view the testing results
7) Save the model weights with your preferred filename

For a bit more streamlined hyperparameter tuning, move on to ‘Hyparam Tuning’ section

8) Change any hyperparameters needed. Run all the cells in this section

5. View Best Models

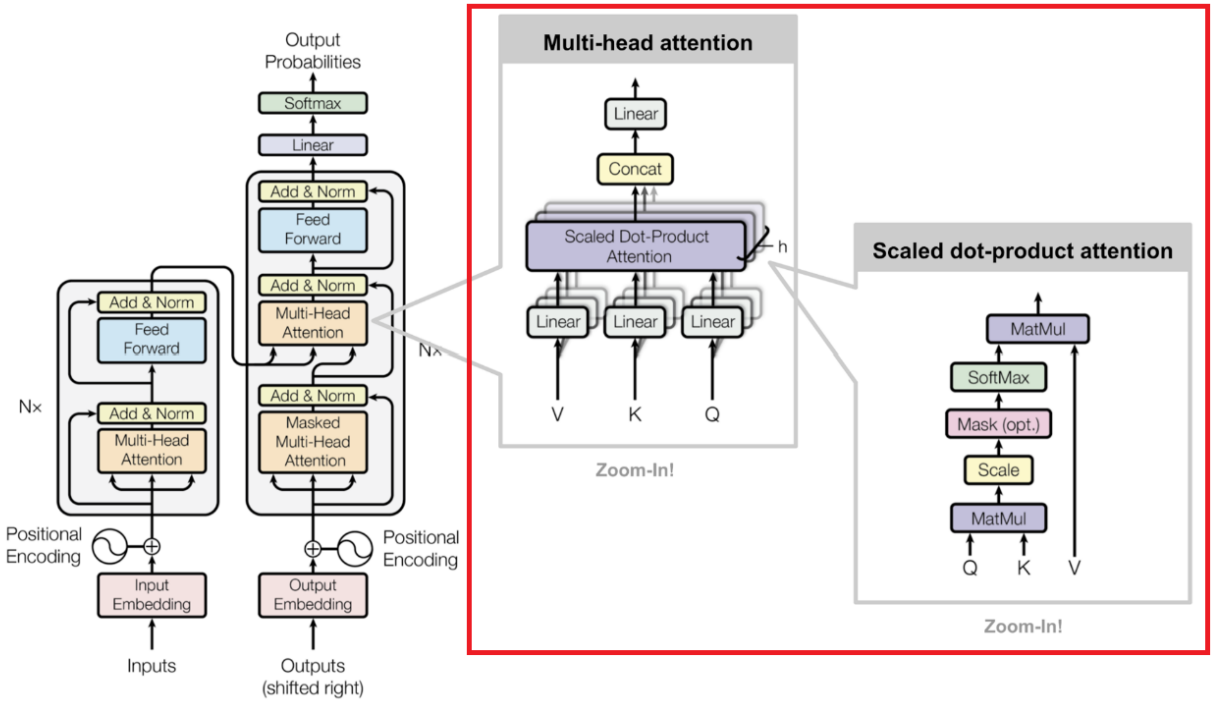
1) Open 7_BestPerformers.ipynb
2) Download the model weights from google drive link and store it in same folder as the jupyter notebook
3) Run all the cells, the weight filenames have already been typed in

References

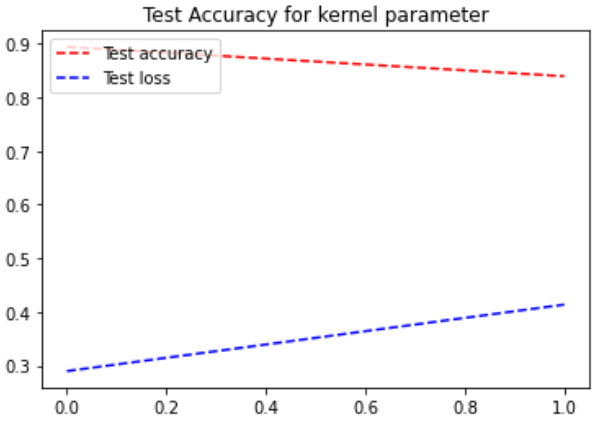
Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), “Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification”, Mendeley Data, V2, doi: 10.17632/rscbjbr9sj.2

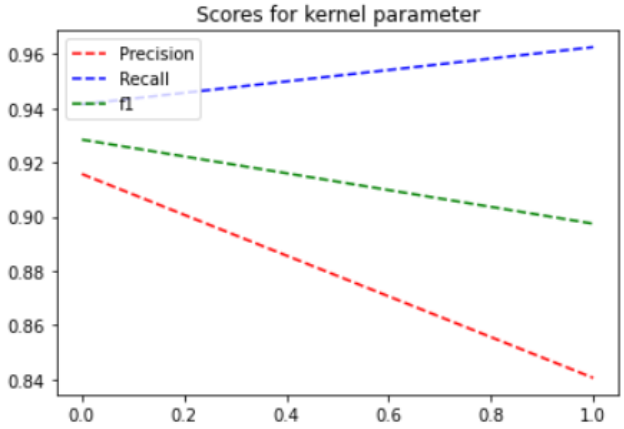
Rudan I., Boschi-Pinto C..Biloglav Z., Mulholland K.,Campbell H.Epidemiology and etiology of childhood pneumonia. *Bull. World Health Organ.* 2008; 86: 408-416

Appendix



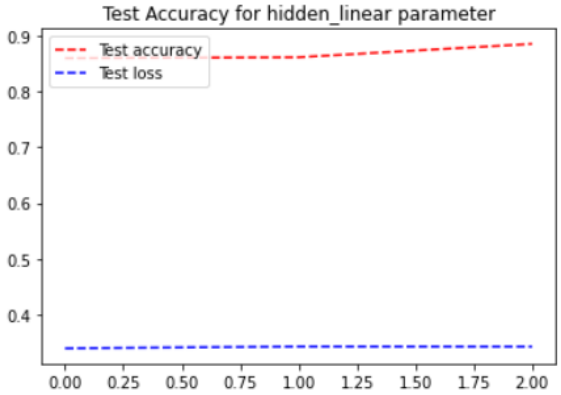
Appendix A: Multi-head self-attention layer implementation

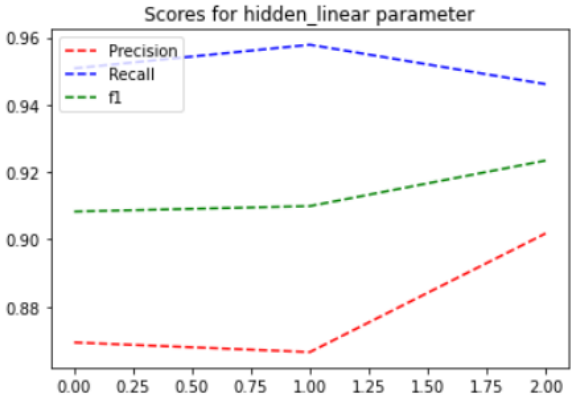




Scores/Kernel_size	3	5
Test accuracy	0.894	0.839
Test loss	0.290	0.414
Precision	0.916	0.840
Recall	0.941	0.963
F1 score	0.928	0.897

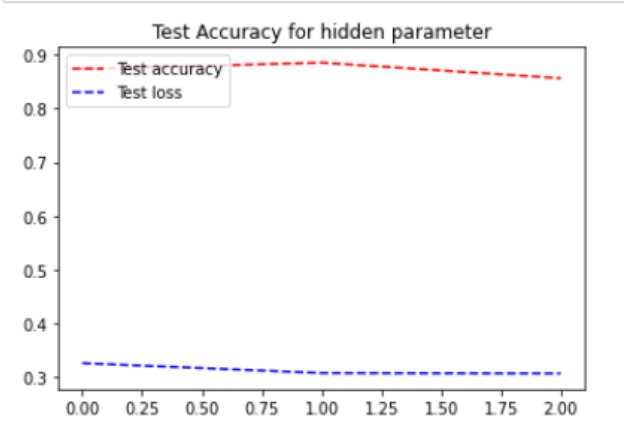
Appendix B. Scores for kernel parameter

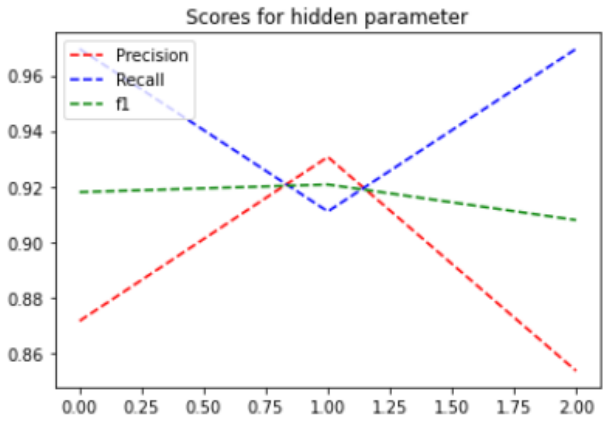




Scores/hidden_linear	[flattened, 512, 64]	[flattened,1024, 512]	[flattened,1024,512,256]
Test accuracy	0.860	0.862	0.885
Test loss	0.339	0.343	0.342
Precision	0.869	0.867	0.902
Recall	0.950	0.958	0.946
F1 score	0.908	0.910	0.923

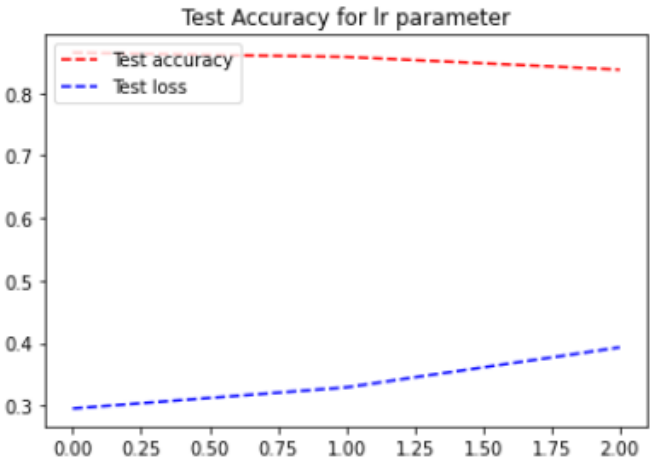
Appendix C. Scores for hidden_linear parameter

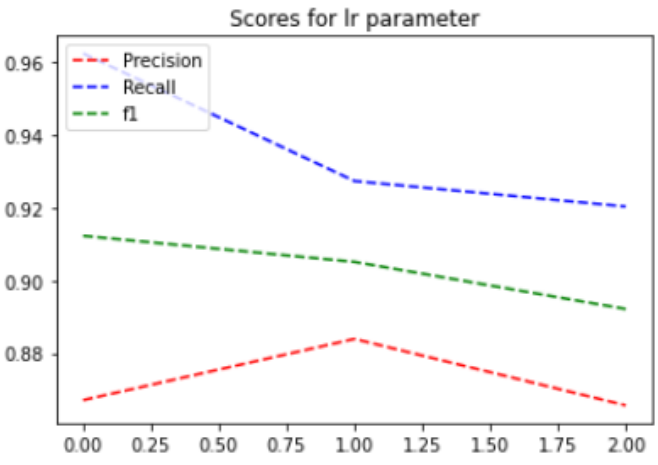




Scores/hidden	[256, 256, 128, 64]	[256, 512,128,64]	[256,512,128]
Test accuracy	0.874	0.885	0.856
Test loss	0.326	0.308	0.307
Precision	0.872	0.931	0.854
Recall	0.970	0.911	0.970
F1 score	0.918	0.921	0.908

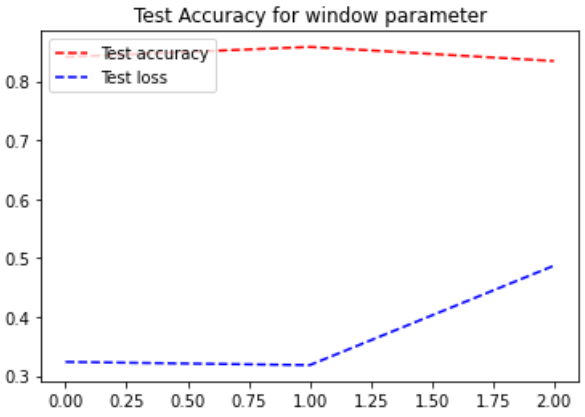
Appendix D. Scores for hidden parameter

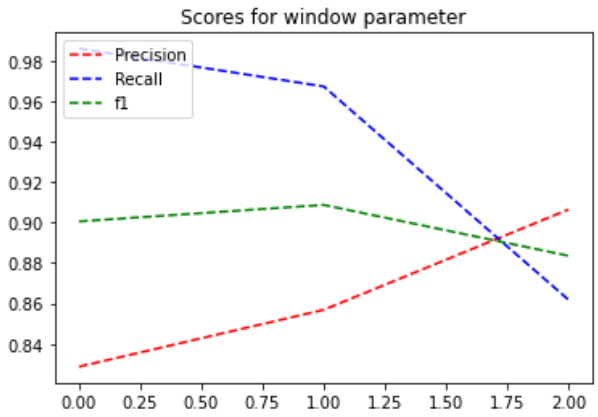




Scores/learning rate	0.01	0.005	0.001
Test accuracy	0.865	0.858	0.838
Test loss	0.295	0.329	0.393
Precision	0.867	0.834	0.866
Recall	0.963	0.927	0.920
F1 score	0.912	0.905	0.892

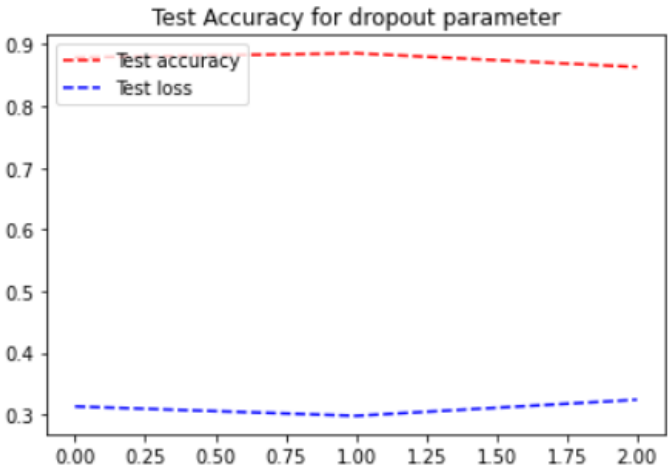
Appendix E. Scores for learning rate parameter

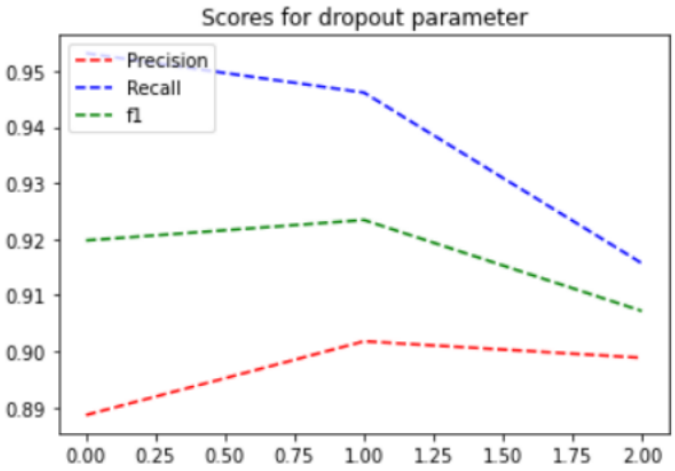




Scores/window	2	3	5
Test accuracy	0.887	0.899	0.850
Test loss	0.271	0.293	0.329
Precision	0.897	0.914	0.863
Recall	0.956	0.951	0.944
F1 score	0.925	0.932	0.902

Appendix F. Scores for window parameter





Scores/dropout	0.3	0.5	0.7
Test accuracy	0.879	0.885	0.863
Test loss	0.313	0.298	0.324
Precision	0.889	0.902	0.900
Recall	0.953	0.946	0.916
F1 score	0.920	0.923	0.907

Appendix G. Scores for dropout parameter