50.039 Theory of Deep Learning

Pneumonia Detection

Group 23

Done By:

Aishwarya RM Palaniappan 1005762

Namitha Maria Justine 1005388

Content

1 Introduction

1.1 Problem Statement

Pneumonia detection from X-Ray images is important in the field of medical diagnosis, because of pneumonia's potential to allow germs that enter the bloodstream from lungs to spread the infection to other parts of the body causing organ failure if not detected early. This early detection is significantly helpful, and could potentially save lives. However, the manual interpretation/understanding of X-ray images by radiologists is highly time consuming and sometimes prone to error, therefore creating a pressing need for systems/trained models that can detect pneumonia from the X-Ray images. Hence to address this problem we have implemented a model that would be able to predict if a given image of a lung's X-Ray is Normal or has Pneumonia with high accuracy.

1.2 Dataset

The dataset comprises chest X-ray images captured from the pediatric care unit of the Guangzhou Women and Children's Medical Center in Guangzhou, depicting patients aged between one to five years old. These images, obtained through routine clinical procedures, are categorized into two groups: "Pneumonia" and "Normal".

The dataset includes two types of X-Rays, describing different types of pneumonia. The types of pneumonia from the scans are bacterial pneumonia and viral pneumonia. These types can be differentiated by the type of germ that has caused the development of infection in the lungs.

Figure 1. Three X-Ray Scans showing a normal X-ray, a bacterial pneumonia X-Ray and a viral pneumonia X-Ray

The dataset in Kaggle has been split into a training set, validation set and testing set. The training dataset is made up of 1341 Normal Samples and 3875 Pneumonia samples. The training dataset has been used to train the model to learn parameters in order to predict the class of a new input image. The validation dataset has been used to determine the early stopping during training. We have monitored the model performance on the validation dataset during training and stop training if the model performance on validation dataset decreases while the performance on training dataset increases. This would prevent overfitting of the model to the training dataset, and increase its ability to generalize the patterns to be applied for unseen data. The testing dataset is the unseen dataset that the model has not seen before which we have used for evaluating the performance of the model.

Dataset Link: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data

1.3 Task

Task: Predict the class of a lungs X-Ray based on the image of lungs received

Input: Image of lungs X-Ray

Model: Convolution Neural Network to process images

Output: Binary Classification of whether image is Normal or Pneumonia

Dataset: Pneumonia Dataset from Kaggle

2 Model Design

For this Binary classification task, we have implemented three CNN models with different architectures and a fine tuned pretrained Resnet model using the PyTorch Framework. In this section, we will be explaining how we have preprocessed our data, the loss function and optimsers we have used in our models.

2.1 Data Preprocessing

The following pie charts describe the distribution of the training, validation and testing dataset. The orange data represents the Normal samples, while the blue data represents Pneumonia samples. As seen from the training dataset the number of samples belonging to the Pneumonia class is significantly higher than the number of samples belonging to the Normal class. Therefore to ensure that our model also gives equal importance to the minority Normal class we have created a weighted loss function which will be explained in the later in the report.

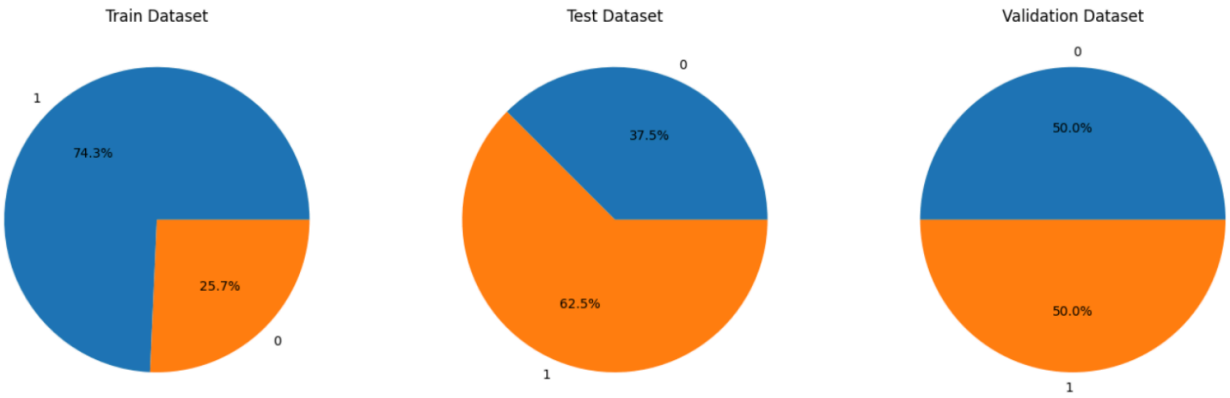

Figure 2: Distribution of Dataset from Kaggle

In the development of our CNN models for the classification of pneumonia from X-ray images, various preprocessing techniques were employed to enhance model training and performance. Preprocessing serves as a critical step to condition the dataset, enabling the models to learn more efficiently and effectively.

For **CNN1** and **CNN2**:

- We have resized the image to 160 by 160 by taking into consideration the computational efficiency and the level of detailed image. For example a higher dimensional image would enable the model to receive a more detailed image and extract more features, however this means that it would be computationally more intensive. Hence we have chosen 160 by 160 striking a balance between resolution of image and computational efficiency.
- Converted to grayscale image hence the number of channels reduced to 1 which enables faster computations because of reduced dimensions
- We have implemented data augmentation by randomly horizontally flipping the images and randomly rotating the images by 15 degrees to ensure that the model learns invariant features and improves its robustness to different orientations.
- Converted the 2D image into a PyTorch tensor so that the tensors can run on the GPU to enable faster computations.
- We have also normalized the pixel value ranges from [0-255] to [0-1] which improves stability and convergence of the model

For **CNN3**:

- In the initial stage of our image processing sequence, dimensions were adjusted to a uniform square aspect of 150 by 150 pixels. This dimension was meticulously selected to optimize between the computational load and the richness of details within the images, essential for feature extraction.
- Converted to grayscale similar to CNN1 and CNN2 to reduce the number of channels to 1
- To bolster the model's capacity to discern relevant patterns irrespective of orientation, we incorporated a random horizontal flip within our augmentation strategy. This approach is designed to fortify the model's ability to recognize pertinent features across varying image presentations. In this model, we skipped the rotating of images by 15 degrees.
- Converted the 2D image to PyTorch tensor similar to CNN1 and CNN2

- The pixel intensity values were standardized from their original span to a centered scale. By adopting a normalization centered around a mean of 0.5 with an equivalent spread, the neural network is primed for more dynamic learning efficacy and accelerated convergence during the training process.

For **ResNet**:

- Every pixel value is rescaled to a range between 0 and 1 by dividing by 255. This normalization helps in speeding up the training by keeping input values small.
- Images are randomly sheared with an intensity of 10 degrees. Shearing changes the shape of the object in an image by displacing parts of the image along the X-axis or Y-axis, simulating a viewing angle change.
- Random zooming of images by up to 30% (zoom_range=0.3). This can simulate objects being closer or farther away from the viewpoint.
- Images are randomly flipped horizontally. This augmentation helps if the orientation of objects in your images does not convey different classes.
- Similarly, images are also randomly flipped vertically. This is useful if upside-down objects in your images still belong to the same class.
- The brightness of images is randomly adjusted, with a range of [0.5,2.0]. This simulates various lighting conditions and can help your model generalize better to images taken under different lighting.
- Images are randomly translated horizontally by up to 20% of the image width. This helps the model learn to recognize objects regardless of their horizontal position in the image.
- Images are randomly rotated by up to 20 degrees. This allows the model to handle images of objects in various orientations.
- When applying transformations such as rotation or width/height shifts, 'nearest' fill mode is used to fill in newly created pixels, which may introduce artifacts at the borders of transformed images.

2.2 Optimiser

We have experimented with Adam and Adagrad optimisers to check which optimiser would be more suitable for our model. Through multiple experiments and hyper tuning of parameters,

Adagrad proved to be a more suitable optimizer that was able to update the parameters of our model such that it is able to escape the local minimum loss and finalise the optimal parameters such that a best minimum loss is achieved.

2.3 Loss Function

We have used a weighted Cross Entropy Loss function because we noticed that in our dataset we have an unequal number of samples belonging to Normal Class and Pneumonia Class. There is a significantly higher number of samples belonging to Pneumonia Class as compared to Normal class hence the Model may tend to optimize for the majority class(Pneumonia) and perform poorly on minority class. Therefore to address this problem we have used a weighted loss function that assigns higher weightage to the Normal class and lower weightage to the Pneumonia class. The Cross Entropy Loss function has both the softmax layer and negative log likelihood function to compute the Loss value after each forward pass.

Formula:

Weight for the class Normal = total number of samples/(number samples belonging to Class Normal*2)

```
weight_for_class_normal = 5291 / (1341 * 2)
weight_for_class_pneumonia = 5291 / (3875 * 2)

class_weights_tensor = torch.tensor([weight_for_class_normal, weight_for_class_pneumonia], dtype=tor
class_weights_tensor = class_weights_tensor.to(device)


# weighted loss function using to ensure that the model gives prority to the normal samples as well
#although they are much lesser in quantity
weighted_loss_function = nn.CrossEntropyLoss(weight=class_weights_tensor)
```

Figure 2: Weighted Loss function

3 Explored models/Architectures

In this section we will be discussing the four models that we have created to model the Pneumonia detection classification task. We have implemented three CNN models with different architectures and a pretrained Resnet model that has been fine-tuned to suit the classification needs of this model to do a state of art comparison between the performance of our CNN models and the Resnet Model. Besides different models, in each model we have hypertuned various parameters in our experimentation:

- Optimiser
- Learning rate
- Dropout
- Early Stopping
- Activation Function
- Batch Size

In our CNN models we have used numerous convolutional layers, Batch Normalisation Layers, sigmoid/ReLU activation function, Dropout layers, Max Pooling and a final linear layer to predict whether the lungs in the image are Normal or have Pneumonia.
The purpose of each layer is stated as follows:

Convolutional Layer:
The purpose of Convolutional layers is to remove noise from data and for feature detection. Through sliding the filter kernels across the entire image, it detects the presence of edges, textures and shapes from the input image hence extracting relevant information from the data.

Batch Normalisation Layer:
Batch Normalisation normalises the activations produced by each layer for each mini-batch to reduce the internal covariate shift. This leads to a more stabilised training, so faster convergence promoting better performance of the model.

Dropout Layer:

We have included Dropout layers in the model Architecture to ensure that the model does not overfit to the training data and is able to generalize the patterns and perform well on unseen data. By setting the dropout rate we can control the percentage of neurons that will be dropped when moving onto the next layer. This is to ensure that the neurons are not too dependent on the other neurons but are still able to learn generalized patterns. This dropout is only applied during training, not during the inference step when we are evaluating the performance of the model.

Max Pooling:

Max pooling reduces the spatial dimensions of feature maps keeping the important information. This would significantly reduce the computations required by the model resulting in a more efficient training.

3.1 CNN Architecture 1

This CNN model starts with the preprocessed data with dimensions 160 by 160 as an input into the 2D convolutional layer followed by Batch normalization to reduce the internal covariate shift and a Sigmoid Activation Function. Thereafter another convolutional layer, Batch Normalisation followed by Max pooling with kernel size=2 and Sigmoid. Since max Pooling reduces the spatial dimensionality by half, now the dimensionality of the image becomes 80 by 80 only which makes computations more efficient while keeping the most important information in the feature map. This is followed by another convolutional layer, Batch Normalisation, Sigmoid Function and Linear Layer. The output from the Sigmoid function is flattened before being sent into the Fully Connected Layer(Linear Layer). This linear layer is the final layer that predicts the class of an image with the features that it has learnt from the previous layers.

3.1.1 Model Architecture

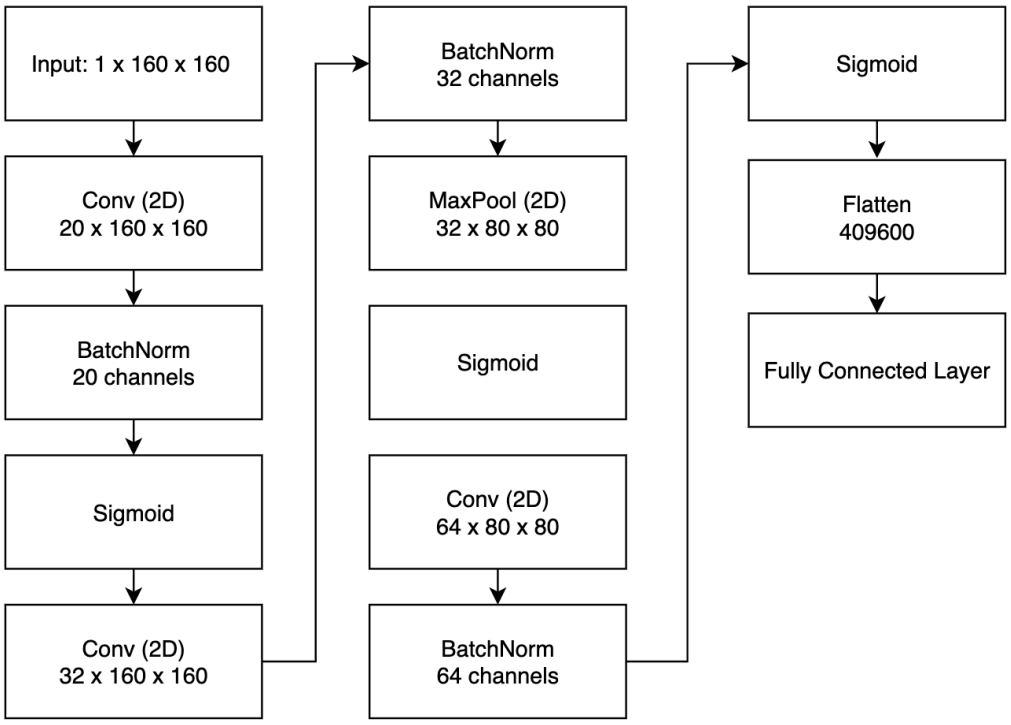

Figure 3: CNN1 Architecture

Here, the output size is indicated for the convolutional and max pooling layers.

3.1.2 Experiment Results

	Activation functions	Learning Rate	Optimisers	Batch Size	Early Stopping	Validation Accuracy	Test Accuracy
1	ReLU	0.001	Adam	128	True	0.75	0.7624
2	Sigmoid	0.001	Adam	128	True	0.875	0.7744
3	Sigmoid	0.001	AdaGrad	128	True	0.9358	**0.8542**
4	Sigmoid	0.001	AdaGrad	128	False	0.625	0.8141
5	Sigmoid	0.001	AdaGrad	64	True	0.9197	0.7436
6	Sigmoid	0.001	Adam	64	True	0.625	0.6843
7	Sigmoid	0.01	AdaGrad	128	True	0.9375	0.7932

Chosen **final model is 3** as it has achieved the highest accuracy of 0.8542 in the testing dataset. It has used sigmoid as its activation function, with a learning rate of 0.001 and the optimiser used is Adagrad with a batch size of 128 and the early stopping set to true.

Figure 4: Accuracies and Loss Across Epochs for Model 1

3.1.3 Analysis

- Using 0.001 learning rate provided highest accuracy, because it was not too large resulting in the model to overshoot the optimal solution or too low making the model take a long time to converge to an optimal solution. The other learning rate of 0.01 was too large and overshot the optimal solution and hence could not provide the most optimal parameters.
- By using early stopping we were able to prevent overfitting of the data.
- By changing the optimiser to adagrad we were able to ensure that the model is able to increase the and gain an higher accuracy on the test dataset and
- By using the sigmoid activation function we were able to get better results as compared to ReLU because this is a binary classification model making it more suitable. Additionally sigmoid is less prone to vanishing gardens as compared to ReLU because it mostly has non-zero gradients which helps in backpropagation promoting more stable training resulting in an improved performance.

3.1.4 Files and notebooks

The architecture and source code of the best model (3) used above is defined in CNN1.ipynb.

The trained model is located in the Trained Models folder:
Best_model_CNN1_Sig_adagrad_128_0.001.pt.

To test the model, we can use the Testing_CNN1.ipynb file located at the Testing folder. Run all cells in the Testing File this will print out the accuracy, precision and F1score of the model on testing data. Functionality of testing on a single sample is also available.

Access the testing files through github or the testing files have been uploaded to google collaboratory with the test data: 🗀 DL Submission

3.2 CNN Architecture 2

In this CNN model we aimed to develop a deeper neural network to be able to capture more complex patterns using receptive fields and we employed techniques such as Dropout layers to ensure not to overfit the training data.

During the convolutional layers we have gradually increased the receptive field size (filter kernel size). For example, in this model we have decreased the kernel size from 5 to 3 and adjusted the padding accordingly. Using the larger receptive fields in the initial layer allows the network to capture low level features such as edges, shapes and textures and followed by using the smaller fields in the later layers allows us to capture the hierarchical representations and understand the shapes. This model architecture has a deeper network where there are many more layers as compared to the other CNN architectures as it aims to be able to learn more patterns from the training dataset.

However this would result in a larger computation time hence we have increased the number of max pooling layers to 2 in order to reduce the spatial dimension twice while keeping the important information still intact. We have two max pooling layers with kernel size of 2 which means that the feature map dimensions would be halved. Reducing the spatial dimensions would decrease the computational time required by the model to train and learn parameters.

We have used a dropout layer with 0.2 dropout rate which drops 20% of the neurons in order to ensure that the neurons are not too dependent on other neurons but are able to generalise data patterns.

3.2.1 Model Architecture

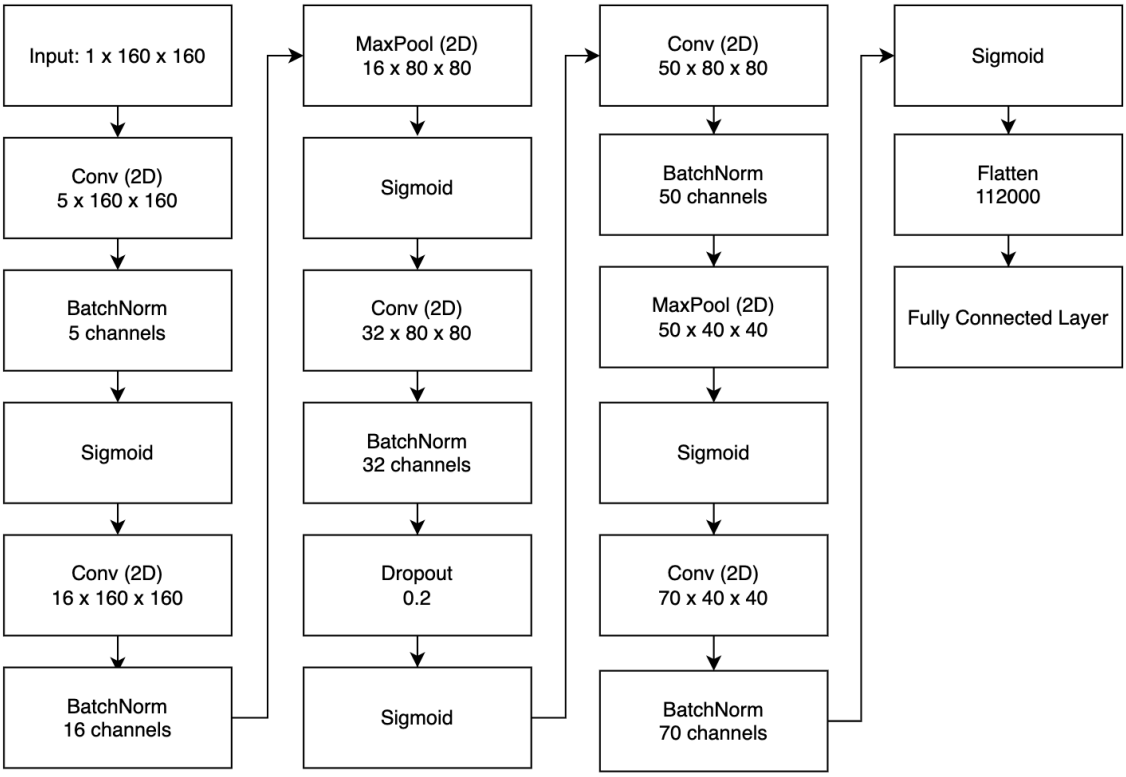

Figure 5: CNN2 Architecture

3.2.3 Experiment Results

	Activation functions	Optimisers	Dropout Size	Early Stopping	Validation Accuracy	Test Accuracy
1	ReLU	Adagrad	0.2	True	0.9592	0.8558
2	Sigmoid	Adagrad	0.2	True	0.9415	**0.8846**
3	Sigmoid	Adam	0.2	True	0.9285	0.8526
4	Sigmoid	Adam	0.5	True	0.9162	0.8446
5	Sigmoid	AdaGrad	0.5	True	0.9214	0.8654

Chosen **final model is 2** as it has achieved the highest accuracy of 0.8846 in the testing dataset that has used sigmoid as its activation function, with a learning rate of 0.001 and the optimiser used is Adagrad with a batch size of 128, the early stopping set to true and dropout size is 20%.

This is the chosen finalised model amongst all other CNN Architectures.

Figure 6: Accuracies and Loss Across Epochs for Model 2

3.2.3 Analysis

- Using 0.001 learning rate provided highest accuracy, because it was not too large resulting in the model to overshoot the optimal solution or too low making the model take a long time to converge to an optimal solution. The other learning rate of 0.01 was too large and overshot the optimal solution and hence could not provide the most optimal parameters.
- Comparing models 3 and 4 with all variables kept constant Dropout 0.2 provided better accuracy than 0.5. This is possibly because dropping 50% of neurons could have meant that important information learnt from the training dataset was dropped resulting in lower validation and testing accuracy.
- By using early stopping we were able to prevent overfitting of the data.
- By changing the optimiser to adagrad we were able to ensure that the model is able to increase and gain an higher accuracy on the test dataset

- By using the sigmoid activation function we were able to get better results as compared to ReLU because it is more suitable for this binary classification model. Additionally sigmoid is less prone to vanishing gardens as compared to ReLU because it mostly has non-zero gradients which helps in backpropagation promoting more stable training resulting in an improved performance.

3.2.4 Files and notebooks:

The architecture and source code of the best model (2) used above is defined in CNN2.ipynb.

The trained model is located in the Trained Models folder:

Chosen_Best_model_CNN2_Sig_adagrad_0.2_128_0.001.pt.

To test the model, we can use the Testing_CNN2.ipynb file located at the Testing folder. Run all cells in the Testing File this will print out the accuracy, precision and F1score of the model on testing data. Functionality of testing on a single sample is also available.

Access the testing files through github or the files have been uploaded to google collaboratory with the test data: 🗀 DL Submission

3.3 CNN Architecture 3

This CNN model, comprises successive convolutional layers paired with ReLU activations to extract features, interspersed with max pooling for dimensionality reduction. After feature extraction, the model transitions via a flattening step to a series of fully connected layers, culminating in a classification layer with a number of outputs matching the desired class count. This design encapsulates both feature learning and classification in one model, leveraging convolutional layers for spatial feature detection and dense layers for decision-making based on these features.

3.3.1 Model Architecture

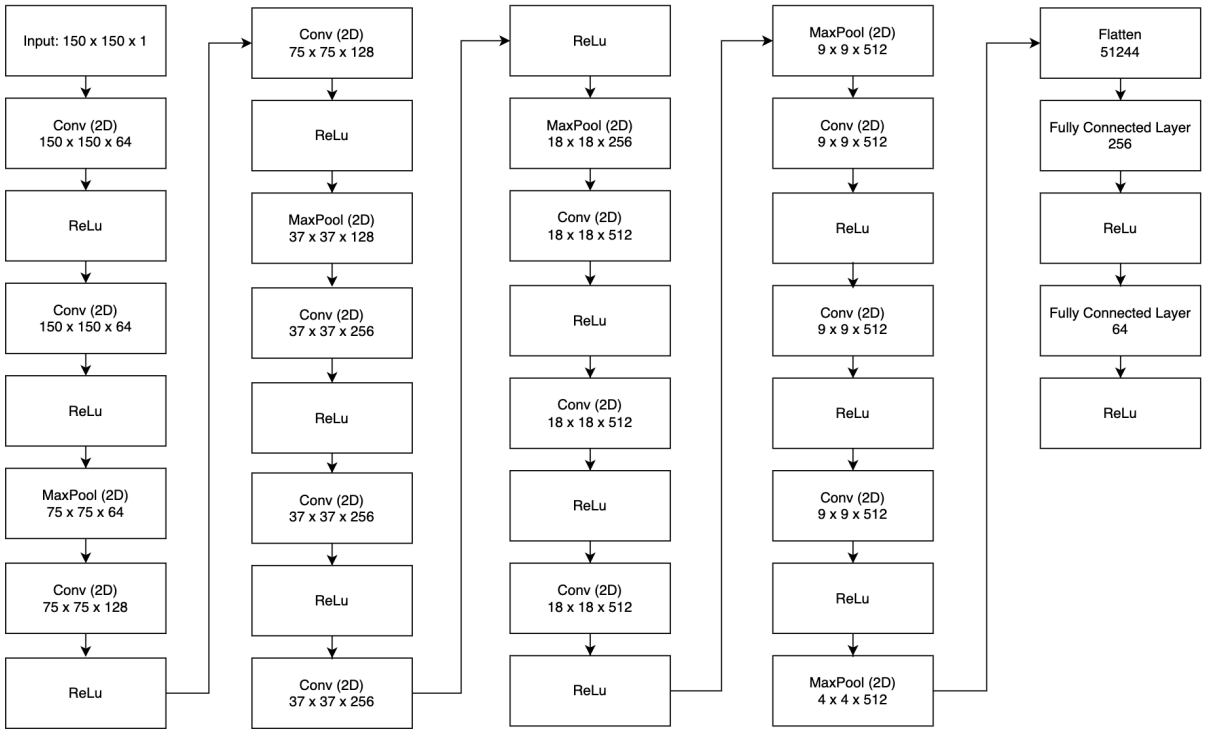

Figure 7: CNN3 Architecture

3.3.2 Experiment Results

	Activation Functions	Learning Rate	Optimizers	Batch Size	Early Stopping	Val Accuracy	Test Accuracy
1	ReLu	0.0001	Adagrad	64	Yes	0.875	0.7965
2	ReLu	0.001	Adagrad	64	Yes	0.875	0.8141
3	ReLu	0.0001	Adagrad	128	Yes	0.5625	**0.8798**
4	ReLu	0.00001	Adagrad	64	Yes	0.5	0.6250

5	Sigmoid	0.0001	Adagrad	64	Yes	0.5	0.6250

Chosen **final model is 3** as it has achieved the highest accuracy of 0.8750 in the testing dataset that has used ReLu as its activation function, with a learning rate of 0.0001 and the optimiser used is Adagrad with a batch size of 128 and the early stopping set to true.

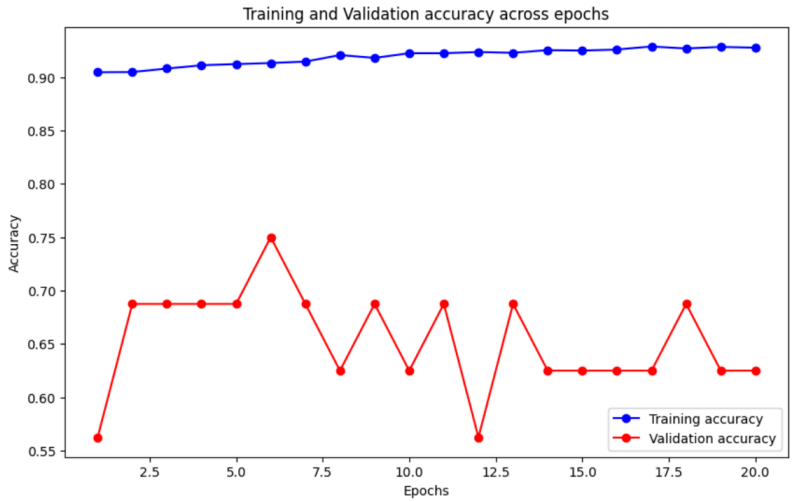

Figure 8: Accuracies Across Epochs for Model 3

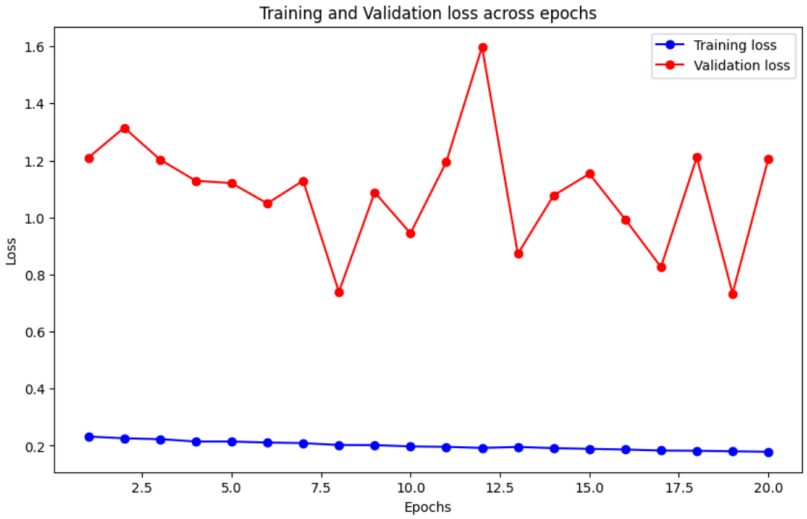

Figure 9: Losses Across Epochs for Model 3

3.3.3 Analysis

- By using early stopping we were able to prevent overfitting of the data.
- By changing the optimiser to adagrad we were able to ensure that the model is able to gain a higher accuracy on the test dataset

3.3.4 Files and notebooks

The architecture and source code of the best model (3) used above is defined in CNN3.ipynb.

The trained model is

CNN3_adagrad_0001_64_relu.pth

To test the model, we can use the Testing_CNN3.ipynb file located at the Testing folder. Run all cells in the Testing File this will print out the accuracy, precision and F1score of the model on testing data. Functionality of testing on a single sample is also available.

Access the testing files through github or the files have been uploaded to google collaboratory with the test data: 🗀 DL Submission

3.4 ResNet Module

We have fine tuned the pre-trained Resnet-50 Model, a state of art model to compare with our CNN architectures.

ResNet-50 introduces residual learning blocks which help in training very deep neural networks. By using skip connections or shortcuts to jump over some layers, ResNet-50 can avoid the vanishing gradient problem that affects deep networks. This is crucial for learning complex patterns in medical images like X-rays. ResNet-50 can generalize better from the training data to unseen data, which is important for medical diagnoses where the model will encounter a wide variety of X-ray images.

3.4.1 Model Architecture

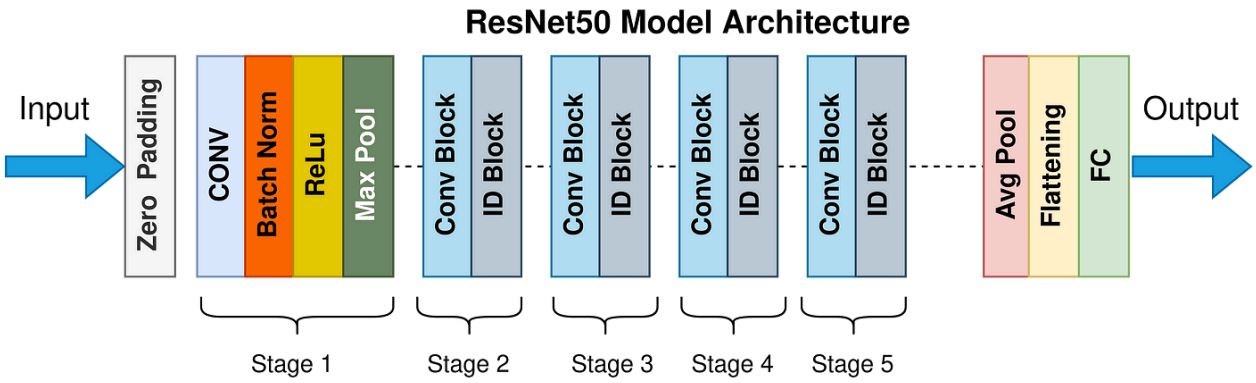

Figure 10: Basic Model Architecture for ResNet-50

With its 50 layers, ResNet-50 is adept at extracting a hierarchy of features – from simple to complex. This multi-scale feature extraction is valuable in medical imaging, where the difference between normal and pathological images may be subtle.

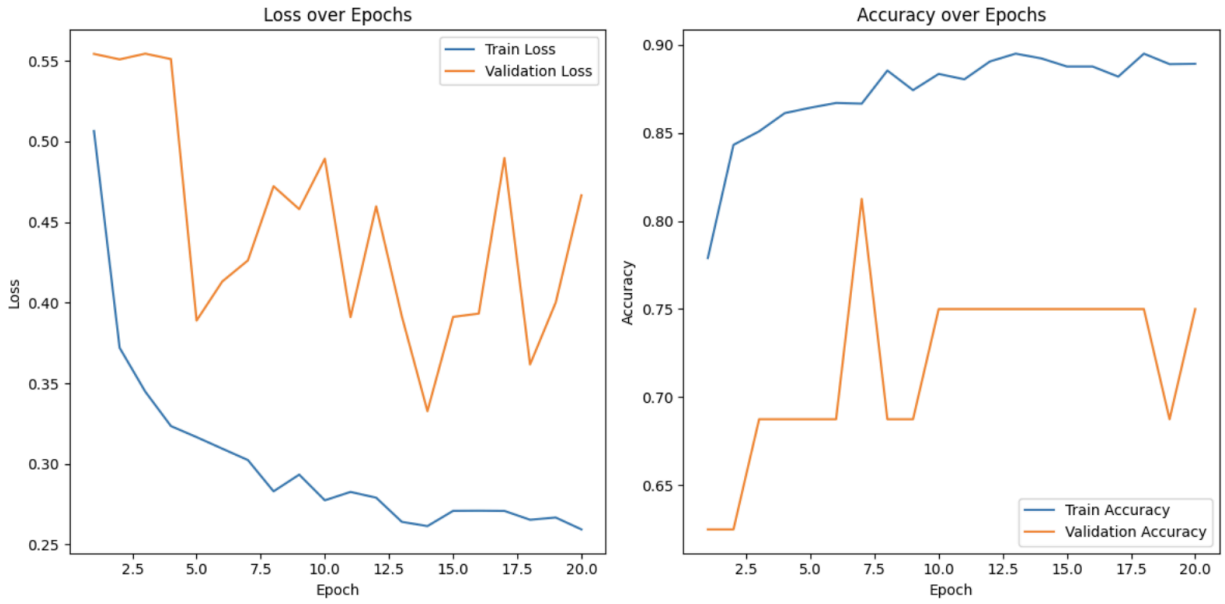

Figure 11: Accuracies and Loss Across Epochs for ResNet-50

4 Comparison between our CNN models and State of Art model ResNet-50

Model	CNN1	CNN2	CNN3	ResNet-50
Accuracy	85.42%	87.98%	87.66%	90.13%
Precision	86.22%	87.67%	86.82%	89.79%
F1 Score	83.69%	86.98%	86.85%	90.45%

We have used 3 metrics to evaluate the performance of our models:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$Accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ samples}$$

$$F1 = \frac{true\ positives}{true\ positives + 0.5((false\ positives + false\ negatives)}$$

As seen from the above results the model with Architecture 2 is the best performing model with the highest Accuracy, Precision and F1 Score amongst other CNN models which combines convolutional layers, batch normalization, dropout layer and max pooling layers respectively. The accuracy of our CNN2 model is very close to the fine tuned ResNet-50 model showing that our model is robust, able to capture important features and patterns from the dataset, and generalise well. Our CNN2 model has a significantly lesser number of layers than the Resnet-50 model which has 50 layers and has been pre-trained on a large number of images. However it is still able to score an accuracy, precision and F1 score as close to the ResNet model because of the optimal hyperparameters tuning and design of the architecture that is able to capture complex patterns. Hence our CNN2 is a suitable model in terms of both performance and computational efficiency tailored for the case of the Pneumonia Detection.

5 Contribution

Aishwarya RM Palaniappan	CNN1, CNN2, Testing Files, Report
Namitha Maria Justine	CNN3, ResNet, Report

6 How to Run

6.1 Training model from scratch

1. For CNN1 Architecture model
    - Open up the CNN1.ipynb notebook in kaggle Notebook and import the necessary libraries as stated in the first cell
    - Import the Pneumonia Dataset in the input tab in Kaggle
    - If running locally, will need to download entire dataset from kaggle and run pip install torch, torchvision, numpy, pandas
    - Run the entire file and the trained model will be saved in the stated directory and will print out the training and validation accuracy
2. For CNN2 Architecture model
    - Open up the CNN2.ipynb notebook in kaggle Notebook and import the necessary libraries as stated in the first cell
    - Import the Pneumonia Dataset in the input tab in kaggle
    - If running locally, will need to download entire dataset from kaggle and run pip install torch
    - Run the entire file and the trained model will be saved in the stated directory and will print out the training and validation accuracy
3. For CNN3 Architecture model
    - Open up the CNN3.ipynb notebook in kaggle Notebook and import the necessary libraries as stated in the first cell
    - Import the Pneumonia Dataset in the input tab in kaggle
    - If running locally, will need to download entire dataset from kaggle and run pip install torch
    - Run the entire file and the trained model will be saved in the stated directory and will print out the training and validation accuracy
4. For ResNet-50 model
    - Open up the ResNetModel.ipynb notebook in kaggle Notebook and import the necessary libraries as stated in the first cell

- Import the Pneumonia Dataset in the input tab in kaggle
- If running locally, will need to download entire dataset from kaggle and run pip install torch
- Run the entire file and the trained model will be saved in the stated directory and will print out the training and validation accuracy

6.2 Instructions to test the trained model:

Google Collab Link for Testing:
https://drive.google.com/drive/folders/1wHvg2v8h2PAix6bwnLQfUcffqxu8vL_3?usp=drive_link

1) For CNN1 Architecture model
    1. Under the Testing folder open up the Testing_CNN1.ipynb file and import it to Kaggle or open up the Google Collab Link
    2. Google Collab link has the Testing file and imported Test Dataset
    3. Run all the cells and the model will be loaded and can be tested against a test dataset or single images
2) For CNN2 Architecture model
    1. Under the Testing folder open up the Testing_CNN2.ipynb file and import it to Kaggle or open up the Google Collab Link
    2. Google Collab link has the Testing file and imported Test Dataset
    3. Run all the cells and the model will be loaded and can be tested against a test dataset or single images
3) For CNN3 Architecture model
    1. Open up the Google Collab link,it has the Testing file and imported Test Dataset
    2. Under the CNN3 Trained Models folder open up the Testing_CNN3.ipynb file and import it to Kaggle or open in google collab
    3. Run all the cells and the model will be loaded and can be tested against a test dataset
4) For Resnet-50 Model
    1. Open up the Google Collab link,it has the Testing file and imported Test Dataset
    2. Under the ResNet Trained Models folder open up the Testing_ResNet.ipynb file and import it to Kaggle or open in google collab
    3. Run all the cells and the model will be loaded and can be tested against a test dataset

7 References

Mukherjee, S. (2022). *The Annotated ResNet-50*. [online] Medium. Available at: https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758.

www.kaggle.com. (n.d.). *Chest X-Ray Images (Pneumonia)*. [online] Available at: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data.