# MedMamba: Vision Mamba for Medical Image Classification

Yubiao Yue

10 June 2024

**Abstract**

MedMamba is a hybrid Convolutional Neural Network (CNN) and State Space Model (SSM) for medical image classification, first proposed by the journal mentioned in this report's title. Three variants of Medmamba models was proposed: MedMamba-Tiny, MedMamba-Small, and MedMamba-Base. PneumoniaMNIST dataset is used for training and validation, with model weights generated at the end of training. The model weights are then tested using CPN X-ray dataset. Seven models were assessed and Medmamba-s achieved the best set of results across training, validation and test datasets. The metric scores returned from Medmamba-s for the test dataset are Overall accuracy: 0.9373, Sensitivity: 0.9428, Specificity: 0.9317, F1: 0.9376, AUC: 0.9815.

**Keywords:** Medical Image Classification, State Space Models, Mamba, Transformer

## 1 Introduction

State Space Models are mathematical frameworks used to model systems that evolve over time, with a latent state vector summarizing the system's history. Its system dynamics are determined by two equations: 1. State Transition Equation, which defines how the hidden state evolves, often incorporating dynamics and noise. 2. Observation Equation, which links the hidden state to observed data, modeling how the state generates observations.

Mamba is a deep learning architecture designed to process long data sequences efficiently using Structured State Space Models (SSSMs). By combining state-space dynamics with Convolutional Neural Networks (CNN), Mamba excels at capturing long-range dependencies. Architecturally, it employs structured parameterization and efficient Fourier domain computations, enabling scalability to long sequences while maintaining computational efficiency.

Mamba can be used for image classification by treating images as sequences, such as rows, columns, or patches, and leveraging its ability to model long-range dependencies across these dimensions. Its structured state-space architecture captures spatial relationships effectively, making it suitable for tasks requiring an understanding of global and local image features.

In 2024, Yubiao Yue from Guangzhou Medical University proposed *MedMamba*, a generalized medical image classification model using Mamba architecture that introduced a novel hybrid basic block named SS-

Conv-SSM, consisting of parallel convolutional and state space layers. Three variants of MedMamba were introduced, namely MedMamba-Tiny, MedMamba-Small, and MedMamba-Base.

## 2 Related works

### 2.1 MedViT

In 2023, Omid Nejati Manzari introduced MedViT, another deep learning model specialised for medical image classification. MedViT is a hybrid vision transformer that combines characteristics of CNN. It composes a patch embedding layer, transformer blocks and a series of stacked convolution in each stage, which follows the traditional hierarchical pyramid architecture.

Firstly, an input image is split into fixed-size patches, each of which is linearly embedded into a vector. These vectors are then processed by a series of transformer encoder blocks, which capture global information and contextual relationships within the image. To preserve spatial information that is typically captured by CNNs, MedViT incorporates positional encodings. These are added to the patch embeddings to allow the model to understand the spatial arrangement of image patches.

The key feature of the transformer is its self-attention mechanism, which allows the model to focus on different parts of the image depending on the context. This enables MedViT to learn contextual relationships across various regions of medical images that might be far apart in the spatial domain. After the transformer encoder layers, a classification head is applied to the output of the transformer's last layer. The head typically uses a global average pooling operation to aggregate the features before passing them through fully connected layers to produce the final predictions.

While the key motivation for this report comes from the use of MedMamba as a newly introduced medical image classifier in 2024, MedViT is also another deep learning model that deserves coequal attention in considering the selection of a suitable image classifier for medical research purposes. Three MedViT variants shall be experimented: MedViT-small, MedViT-base and MedViT-large.

### 2.2 Vim

In 2024, Ali Nasiri-Sarvi introduced two variants of Mamba architecture, Vim and VMamba that was used for classification of breast ultrasound images. ploys bidirectional Mamba blocks, which utilize state space models (SSMs) to capture both local and global dependencies in visual data. In Vim, images are divided into smaller patches, each projected into a patch embedding. Its bidirectional Mamba model these patches by considering both previous and next tokens. Bidirectional scanning is performed to achieve multi-directional feature extraction across the patches. Positional encoding is added to each token to retain spatial information about neighboring patches. Figure 1 illustrates the pipeline of Vim.

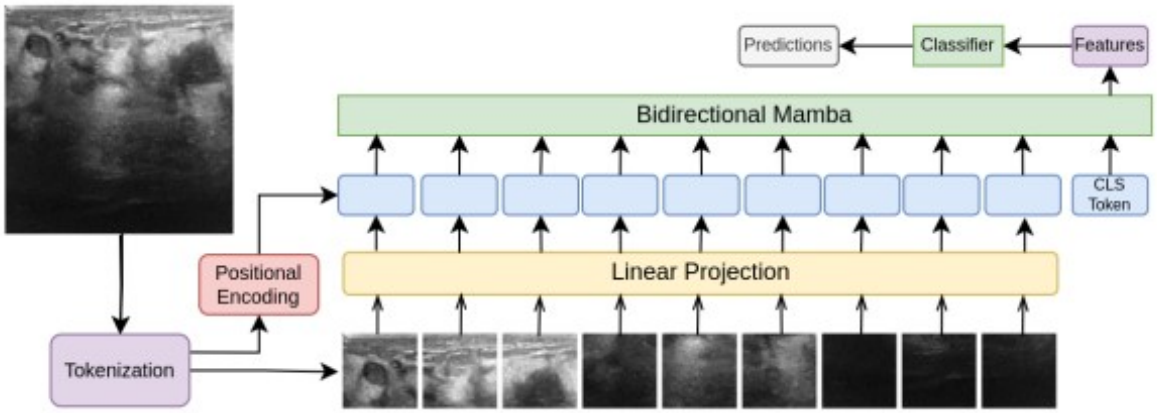


Figure 1. Pipeline of Vim.

VMamba is an adaptation of Mamba's SSM into a vision backbone consisting of Visual State-Space (VSS) blocks. It consists of a single state space model backbone similar to Medmamba's SS2D found within its SSM-branch, but without any CNN components. Its SS2D module traverses images along multiple scanning routes to effectively gather contextual information. VMamba is designed to process visual data with linear time complexity, enhancing efficiency in handling high-resolution images. Figure 2 illustrates the pipeline of VMamba.

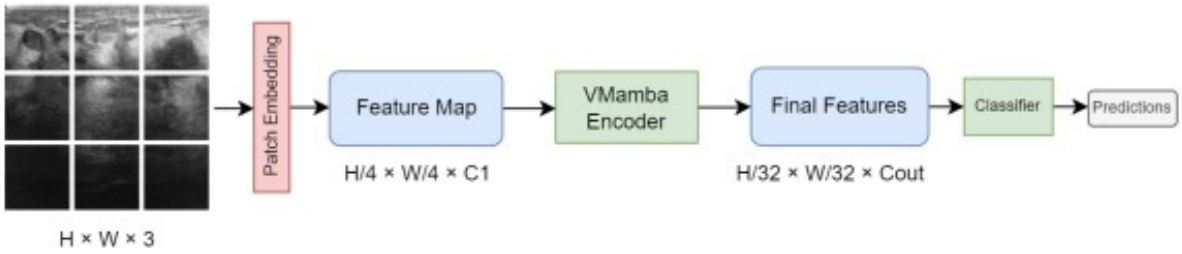


Figure 2. Pipeline of VMamba.

## 3 Method

### 3.1 Medmamba

The architecture of MedMamba consists of a patch embedding layer, alternately stacked SS-Conv-SSM blocks and patch merging layers for down-sampling, and a feature classifier at the end, as shown in the upper figure of Figure 3 below. Each SS-Conv-SSM block consist of convolutional layers (Conv-branch), which capture local spatial features from images, and a parallel State Space Model branch (SSM-branch) that process global dependencies over an input feature map, as shown in the lower figure of Figure 3 below.

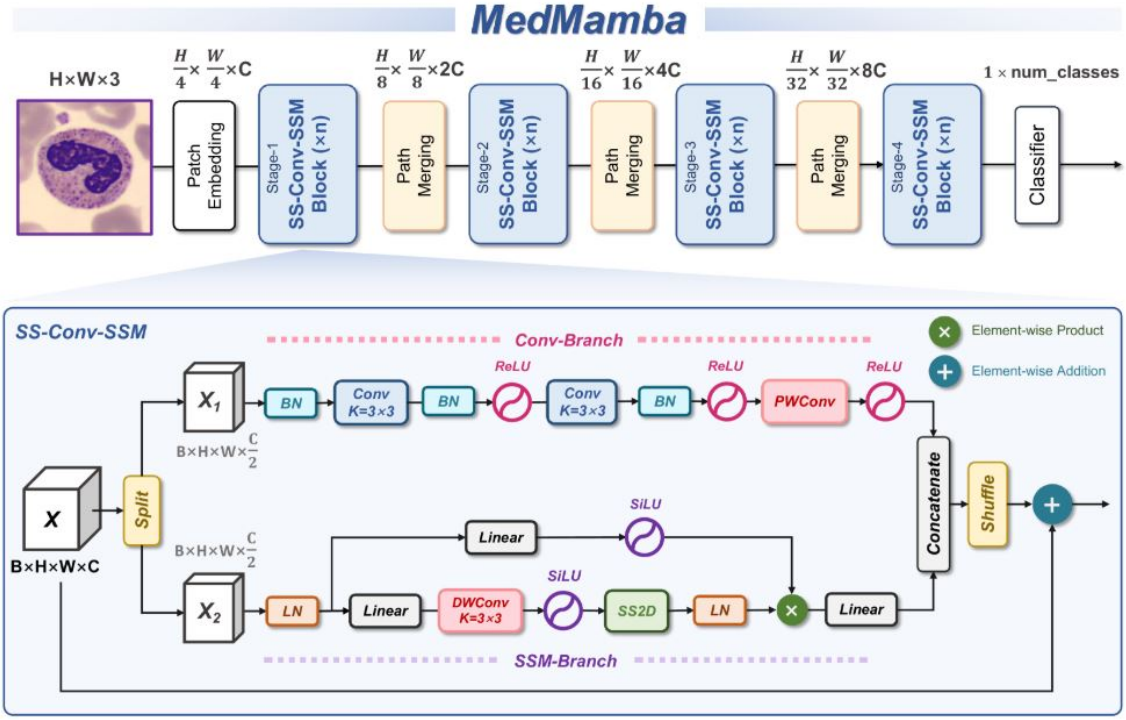


Figure 3. Upper figure shows the architecture of MedMamba. Lower figure shows the architecture of an SS-Conv-SSM block. BN, LN, Linear, PWConv and DWConv represent Batch normalization, layer normalization, Linear layer, Point-wise convolution and Depth-wise convolution, respectively.

The SSM-branch processes input feature maps by scanning them selectively along multiple spatial routes. In the SSM-branch, selective scanning is performed by traversing through the 2D grid of features using predefined patterns or routes. Unlike simple row-wise or column-wise traversals, it uses multiple, adaptive scanning paths to encode the spatial dependencies across the image. For each route, the feature map is unrolled into a sequence via 2D to 1D Transformation, enabling the SSM to process the data efficiently using its state-space computations. The unrolled sequences are processed by the SSMs to model long-range dependencies. The results are then reassembled back into a 2D format, preserving the original 2D grid structure of the input image.

Given an input image $x \in \mathbb{R}^{H\times W\times 3}$, the model partitions it into non-overlapping patches of size 4 × 4 using a patch embedding layer, subsequently mapping the channel dimensions of the image to C, without further flattening the patches into a 1-D sequence. Patch embedding layer results in a feature map with $H/4\times W/4\times C$ without compromising the 2D structure of the input image. MedMamba then repeatedly stacks SS-Conv-SSM blocks to build Stage-1 that further process the feature map without changing its dimension size. To construct hierarchical representations, MedMamba utilizes a patch merging layer to down-sampling the feature map. Stage-2, Stage-3 and Stage-4 repeat the above process, and two patch merging layers are used for down-sampling the outputs of Stage-2 and Stage-3, resulting in two outputs with resolutions of $H/16 \times W/16 \times 4C$ and $H/32 \times W/32 \times 8C$, respectively. At the end of the network, a classifier consisting of an adaptive global pooling layer and a linear layer is used to calculate the category of the input image. The default size for MedMamba's input is set as 224 x 224 x 3.

Three different scales of MedMamba were developed, i.e., MedMamba-Tiny, MedMamba-Small, and MedMamba-Base (referred to as MedMamba-t, MedMamba-s, and MedMamba-b, respectively). Detailed

architectural specifications are outlined in Table 1. The FLOPs (Floating point operations per second) for all models are assessed using a 224 × 224 input size.

| **Layer name** | **Output Size** | **MedMamba-Tiny** | **MedMamba-Small** | **MedMamba-Base** |
|---|---|---|---|---|
| Patch-E | 56×56 | conv 4×4, 96, stride 4 | conv 4×4, 96, stride 4 | conv 4×4, 128, stride 4 |
| | | Channel number → 96 | | Channel number → 128 |
| Stage 1 | 56×56 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 |
| Path-M | 28×28 | Channel number → 192 | | Channel number → 256 |
| Stage 2 | 28×28 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 |
| Path-M | 14×14 | Channel number → 384 | | Channel number → 512 |
| Stage 3 | 14×14 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 4 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 8 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 12 |
| Path-M | 7×7 | Channel number → 768 | | Channel number → 1024 |
| Stage 4 | 7×7 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 | [Channel Split, Conv-SSM, Channel Concat, Channel Shuffle] × 2 |
| Classifier | 1×1 | Adaptive global pooling, Linear (768/1024→num-classes), Softmax | | |
| Parameter Size(M) | | 15.2 | 23.5 | 48.1 |
| FLOPs(G) | | $2.0 \times 10^9$ | $3.5 \times 10^9$ | $7.4 \times 10^9$ |

**Table 1**: Architectural overview of MedMamba variants. Patch-E and Patch-M represents Patch-Embedding and Patch-Merging, respectively. The results for parameter size and FLOPs are calculated when the number of classes is equal to 1000.

### 3.2 MedViT

MedViT architecture consists of an initial Convolutional layer, known as the stem followed by four stages of Patch embedding and vision transformers. The stem extracts edges, textures, and simple patterns from initial input, to reduce its spatial resolution. Patch embedding follows suit, which divide the image into smaller patches and then map them into a higher-dimensional space. Patch embedding ensures that the image is appropriately represented for transformer-based processing, where each patch is treated as an individual token, capturing both local and global features.

After each round of spatial embedding, the images are passed onto a series of MedVit, consisting of specialized transformer MedViT blocks designed to learn local spatial dependencies and long-range contextual relationships. Each MedViT block is made up of three components: Efficient Convolution Block (ECB), Local Transformer Block (LTB) and Transformer Augmentation Block (TAB).

The ECB consists of a Locally Feed Forward Network (LFFN) which introduces local representation of features into the network using depth-wise convolution, and a Multi-Head Convolutional Attention (MHCA) for the mixing of tokens. The LTB captures and mixes multi-frequency signals for better representation of tokens at the global level. Finally, TAB consists of a Patch Momentum Changer (PMC) which blends feature normalization with data augmentation at training time for a pair of images at token level. The architecture of MedViT is shown in Table 2 below.

| **Stages** | **Output size** | **Layers** | **MedViT-T** | **MedViT-S** | **MedViT-L** |
|---|---|---|---|---|---|
| **Stem** | $\frac{H}{4} \times \frac{W}{4}$ | Convolution Layers | Conv $3 \times 3, C = 64, S = 2$ | | |
| | | | Conv $3 \times 3, C = 32, S = 1$ | | |
| | | | Conv $3 \times 3, C = 64, S = 1$ | | |
| | | | Conv $3 \times 3, C = 64, S = 2$ | | |
| **Stage 1** | $\frac{H}{4} \times \frac{W}{4}$ | Patch Embedding | Conv $1 \times 1, C = 96$ | | |
| | | MedViT Block | $[\text{ECB} \times 3, 96] \times 1$ | | |
| **Stage 2** | $\frac{H}{8} \times \frac{W}{8}$ | Patch Embedding | Avg_pool, $S = 2$ | | |
| | | | Conv $1 \times 1, C = 192$ | | |

| Stages | Output size | Layers | MedViT-T | MedViT-S | MedViT-L |
|---|---|---|---|---|---|
| | | MedViT Block | $\begin{bmatrix} \text{ECB} \times 3, 192 \\ \text{LTB} \times 1, 256 \end{bmatrix} \times 1$ | | |
| **Stage 3** | $\frac{H}{16} \times \frac{W}{16}$ | Patch Embedding | Avg_pool, $S = 2$ | | |
| | | | Conv $1 \times 1, C = 384$ | | |
| | | MedViT Block | $\begin{bmatrix} \text{ECB} \times 4, 384 \\ \text{LTB} \times 1, 512 \end{bmatrix} \times 2$ | $\begin{bmatrix} \text{ECB} \times 4, 384 \\ \text{LTB} \times 1, 512 \end{bmatrix} \times 4$ | $\begin{bmatrix} \text{ECB} \times 4, 384 \\ \text{LTB} \times 1, 512 \end{bmatrix} \times 6$ |
| **Stage 4** | $\frac{H}{8} \times \frac{W}{8}$ | Patch Embedding | Avg_pool, $S = 2$ | | |
| | | | Conv $1 \times 1, C = 768$ | | |
| | | MedViT Block | $\begin{bmatrix} \text{ECB} \times 2, 768 \\ \text{LTB} \times 1, 1024 \end{bmatrix} \times 1$ | | |

**Table 2**: Architectural overview of MedViT variants. C and S denotes number of channels and stride of convolution for each stage.

### 3.3 Evaluation Metrics

Overall Accuracy, Sensitivity, Specificity, F1-score, and Area Under the receiver operating Curve (AUC) are used as standard evaluation metrics for training, validation and test datasets. **Overall Accuracy (OA)**: The proportion of all correct predictions (both true positives and true negatives) out of the total number of predictions. **Sensitivity**: Also known as Recall, sensitivity measures the proportion of true positive predictions made by the model out of all actual positive cases. **Specificity**: The proportion of true negative predictions out of all actual negative cases. **F1-score**: The harmonic mean of Precision and Sensitivity, balancing the contributions between two metrics. **AUC**: Area Under Curve quantifies the overall performance of a binary classifier by measuring the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate at various threshold settings. A higher AUC indicates better model performance in distinguishing between the positive and negative classes.

The calculation formulas for Overall Accuracy, Precision, Sensitivity, Specificity, and F1-score are as follows:

$$Overall\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{2}$$

$$Specificity = \frac{TN}{TN + FP} \tag{3}$$

$$F1 - score = \frac{2 * Precision * Sensitivity}{Precision + Sensitivity} \tag{4}$$

The loss function determines the difference between the predicted output of the model and the actual target value, and is applicable only for training and validation datasets. Training loss is the calculated error when the model makes predictions on the training data. Validation loss is calculated upon the validation dataset, and determines how well the trained model generalizes to new data. As this is a classification problem, cross entropy loss function is applied.

# 4 Implementation details

## 4.1 Datasets

Two datasets are used for this report: MedMNIST and CPN X-ray. MedMNIST is a publicly available MNIST-like collection of standardized biomedical images, including 12 datasets for 2D and six datasets for 3D. For this report, PneumoniaMNIST is used. PneumoniaMNIST is a 2D dataset consisting of 5,856 normal and pneumonia lung X-ray images. It is offered as 28 × 28 for default resolution, with options for 64 x 64, 128 128 and 224 x 224. CPN X-ray is another publicly available dataset consisting of 1626 COVID, 1802 normal and 1800 pneumonia X-ray images. For this report, only normal and pneumonia X-ray images are used from CPN X-ray dataset.

## 4.2 Comparing with the released source codes

Source code for MedMamba and its training script maybe found at Yubiao Yue's github repository. As for MedViT, it can be found at Omid-Nejati's github repository. Adaptations to their training scripts are made in the use of the DataParallel library and corresponding adjustments in its training and validation loop. At the same time, the results for overall accuracy (simply called accuracy), sensitivity, specificity, F1 and AUC are printed out at the end of the training/validation cycle and also when the test batch is run. For each of the metrics listed, regression graphs are also generated to track corresponding changes in its metric scores for both the training and validation datasets.

## 4.3 Experimental environment setup

The GPUs used were NVIDIA Geforce RTX 4080 and Geforce RTX 2080 Ti, which have maximum capacities of 24,564 MiB and 11,019 MiB respectively. WindTerm 2.5.0 was used to access the GPUs, which runs on Ubuntu 18.04-SMP (Symmetric MultiProcessing) environment. The training, validation and test runs are performed in a Conda environment, and Python libraries used for runs are found in the requirements.txt file, together with the training scripts.

Seven models were used: MedMamba-t, MedMamba-s, MedMamba-b, MedViT-small, MedViT-base, MedViT-large and ResNet50. While MedMamba is the main model investigated for this report, MedViT is also

chosen considering that the author also chose this model for his methodology, and also to serve as a comparison between the effectiveness of state space-based model and vision transformer-based model. ResNet50 is taken from PyTorch library and used as a baseline model.

Batch sizes of 128 and 100 training epochs were set for MedViT-small, MedViT-base, MedViT-large and ResNet50 models, as these were training parameters mentioned in the journal. Batch sizes of 64 and 150 training epochs were set for MedMamba-t, MedMamba-s, MedMamba-b, as the GPU (Graphical Processing Unit) returned an error message of having insufficient memory space when batch size of 128 was attempted using MedMamba-b. Although MedMamba-s and MedMamba-t is able to run with batch size of 128 and 100 epochs, the same configuration is applied to both models for the sake of consistency in performance comparison. DataParallel library, which splits tensors across multiple GPUs, does not work in the case of MedMamba.

For all models mentioned, Adam optimiser is used, using its default learning rate at 0.001. Adam optimiser works by keeping track of the gradient's direction and builds on past gradients and adjusting the learning rate for each parameter dynamically, allowing faster convergence. MultistepLR is used to regulate the learning rate, to facilitate convergence and prevents the model from getting stuck or overshooting the optimal solution. The 50th to 75th epoch point is chosen as it is determined that most features have been trained into the model and encourage convergence between both datasets at this point.

The MedMNIST library is imported and loaded onto the local environment using Pytorch Dataloader, and split into the training and validation datasets on a 80:20 ratio. Default resolution was used for training and validation, considering the limitations in computational resources for larger models. Larger batch size of 128 could be used for MedViT, as its tensors could be split into multiple devices using PyTorch's DataParallel library. The same arrangement could not be experimentally applied for MedMamba, hence smaller batch size of 64 with additional 50 training epochs were applied as a fallback.

Model weights for each of the seven models are generated after training and validation. A seperate test script is then used to call each of the model weights upon the CPN X-ray normal and pneumonia X-ray images, which are henceforth used as the test dataset. Standard evaluation metric scores mentioned in the Evaluation Metrics subsection are generated at the end of each test cycle.

## 5 Results and analysis

Table 3 and 4 shows the training and validation results applied onto the PneumoniaMNIST dataset, which was split into the training and validation datasets:

| **Model** | **OA** | **Sensitivity** | **Specificity** | **F1** | **AUC** | **Loss** |
|---|---|---|---|---|---|---|
| Medmamba-b | 0.9488 | 0.9702 | 0.8871 | 0.9657 | 0.9857 | 0.1321 |
| Medmamba-s | 0.9446 | 0.9674 | 0.8789 | 0.9628 | 0.9851 | 0.1341 |
| Medmamba-t | 0.9492 | 0.9702 | 0.8888 | 0.9659 | 0.9883 | 0.1184 |
| MedViT-large | 0.9386 | 0.9631 | 0.8682 | 0.9588 | 0.9791 | 0.1557 |

| Model | OA | Sensitivity | Specificity | F1 | AUC | Loss |
|---|---|---|---|---|---|---|
| MedViT-base | 0.9465 | 0.9691 | 0.8814 | 0.9641 | 0.9855 | 0.1324 |
| MedViT-small | 0.9407 | 0.9665 | 0.8666 | 0.9603 | 0.9812 | 0.1480 |
| Resnet50 | 0.9460 | 0.9685 | 0.8814 | 0.9638 | 0.9834 | 0.1385 |

**Table 3**: Train results of deep learning models applied on training dataset of PneumoniaMNIST. OA = Overall Accuracy, AUC = Area Under Curve.

| Model | OA | Sensitivity | Specificity | F1 | AUC | Loss |
|---|---|---|---|---|---|---|
| Medmamba-b | 0.9752 | 0.9794 | 0.9630 | 0.9832 | 0.9953 | 0.0736 |
| Medmamba-s | 0.9676 | 0.9949 | 0.8889 | 0.9785 | 0.9961 | 0.0819 |
| Medmamba-t | 0.9523 | 0.9383 | 0.9926 | 0.9669 | 0.9963 | 0.1541 |
| MedViT-large | 0.9618 | 0.9692 | 0.9407 | 0.9742 | 0.9924 | 0.1212 |
| MedViT-base | 0.9542 | 0.9460 | 0.9778 | 0.9684 | 0.9962 | 0.1376 |
| MedViT-small | 0.9828 | 0.9897 | 0.9630 | 0.9884 | 0.9957 | 0.0669 |
| Resnet50 | 0.9408 | 0.9820 | 0.8222 | 0.9610 | 0.9813 | 0.1539 |

**Table 4**: Validation results of deep learning models applied on validation dataset of PneumoniaMNIST. OA = Overall Accuracy, AUC = Area Under Curve.

The training and validation metric scores for the seven models listed above are within the range of $>0.8$ and above. Performance wise, all seven models have training and validation loss of $<0.16$ and below. A slight divergence maybe observed in the Specificity scores for the trained models with the exception of Medmamba-s and Resnet50; in the training dataset, the specificity scores of the models are in the range of 0.85 to 0.89, whereas in the validation dataset they are in the range of 0.94 to 1.00, an indication of overfitting. Nevertheless, the overfit is not severe, as there have been cases where the difference between the metric score of the training and validation datasets in a given trained model is $>0.3$.

Among all the models compared, Medmamba-s appeared to have the best training and validation results; primarily because the difference in metric scores, between the training and validation datasets is less than 0.05 across the board. Table 5 shows test results applied onto the CPN X-ray dataset, with training weights from PneumoniaMNIST taken from each of their respective models:

| Model | OA | Sensitivity | Specificity | F1 | AUC |
|---|---|---|---|---|---|
| Medmamba-b | 0.8612 | 0.9839 | 0.7386 | 0.8763 | 0.9832 |

| Model | OA | Sensitivity | Specificity | F1 | AUC |
|---|---|---|---|---|---|
| Medmamba-s | 0.9373 | 0.9428 | 0.9317 | 0.9376 | 0.9815 |
| Medmamba-t | 0.7735 | 0.9906 | 0.5566 | 0.8138 | 0.9746 |
| MedViT-large | 0.9245 | 0.9356 | 0.9134 | 0.9253 | 0.9786 |
| MedViT-base | 0.8129 | 0.9744 | 0.6515 | 0.8388 | 0.9619 |
| MedViT-small | 0.7846 | 0.9917 | 0.5777 | 0.8214 | 0.9698 |
| Resnet50 | 0.6835 | 0.9772 | 0.3901 | 0.7553 | 0.9381 |

**Table 5**: Test results of deep learning models applied to CPN X-ray dataset. OA = Overall Accuracy, AUC = Area Under Curve.

The test results showed varying performances for each of the models used for testing: Medmamba-s returned the best scores across all five metrics assessed, followed by MedViT-large as a close second. Variation in metrics is most obvious between the sensitivity and specificity scores; taking Resnet50 for example, it returned sensitivity of 0.9772, but has a much lower specificity score of 0.3901.

In the realm of medical imaging, sensitivity measures how many samples are correctly identified as positive and may arguably be considered as the most important metric. For this problem, the goal is to identify how many samples identified as Pneumonia cases is a true positive case; the implication of having too many false positives returned in a learning model would call into question its reliability and crediibility as a useful diagnostic tool. On this aspect, all models tested have good sensitivity scores of $>$0.9.

The significance of other metrics cannot be discounted; specificity measures how many samples are correctly identified as negative. In the field of medical imaging, one may wish to consider whether a patient having respiratory distress is experiencing pneumonia. There maybe other underlying causes of respiratory distress, such as pneumothroax and atelectasis. From the models trained, Medmamba-s and MedViT-large have good specificity scores of $>$0.9, while other models have more varied performance.

In the context of considering which is is the best performing Medmamba variant, it can be qualified to conclude that Medmamba-s performs best when analysing X-ray images of the similar type. It has also the best set of metric scores in the test dataset namely OA: 0.9373, Sensitivity: 0.9428, Specificity: 0.9317, F1: 0.9376, AUC: 0.9815. Between Medmamba-t and Medmamba-s, the lower overall accuracy, specificity and F1 scores maybe attributed to the greater VSSM depth in Medmamba-s as compared to Medmamba-t; which allows for enhanced scanning of the image feature map and consequently pick up more features usable for test prediction.

Medmamba-b on the other hand, has an even greater VSSM depth and higher channel number as compared to Medmamba-s, yet performed worse in its metric scores. Comparing the loss and F1 regression graphs across epochs for Medmamba-b and Medmamba-s in Figure 2 and 3, there is a much greater degree of fluctuation in the former model, indicating that Medmamba-b was not able to converge as well as compared to Medmamba-s.

Additionally, it has been experimentally determined that using Adam optimiser with learning rate value of 0.001, and using MultistepLR with gamma value of 0.1, as the scheduler returns the best set of training, validation and test metric scores for Medmamba-s. While increasing or decreasing any combination of these

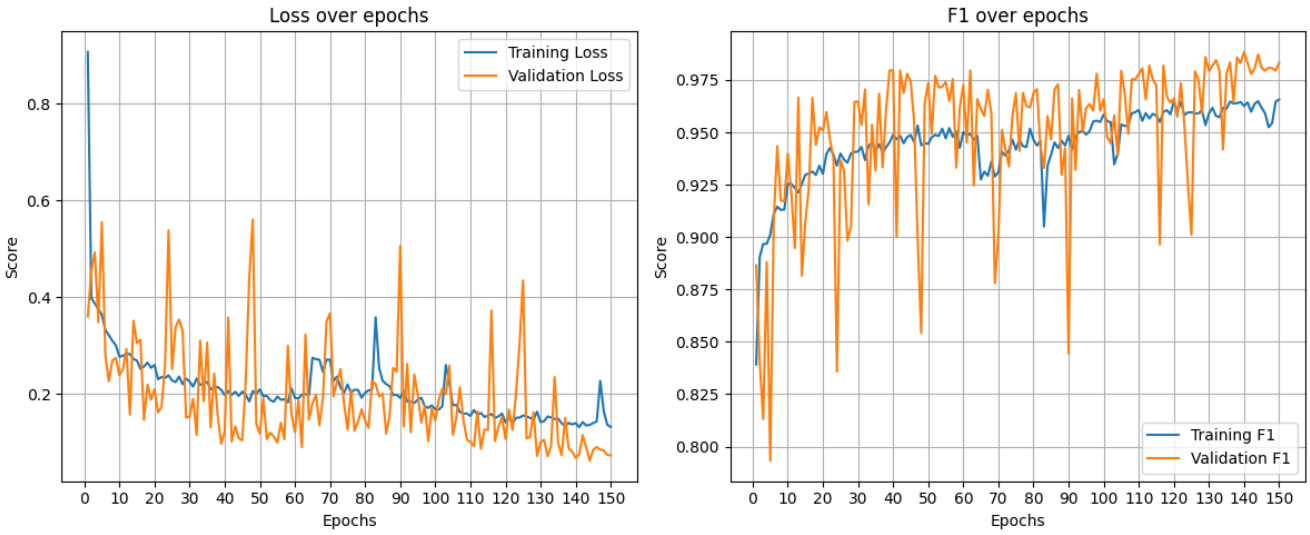


Figure 4. Loss and F1 score charts for Medmamba-b across 150 epochs.

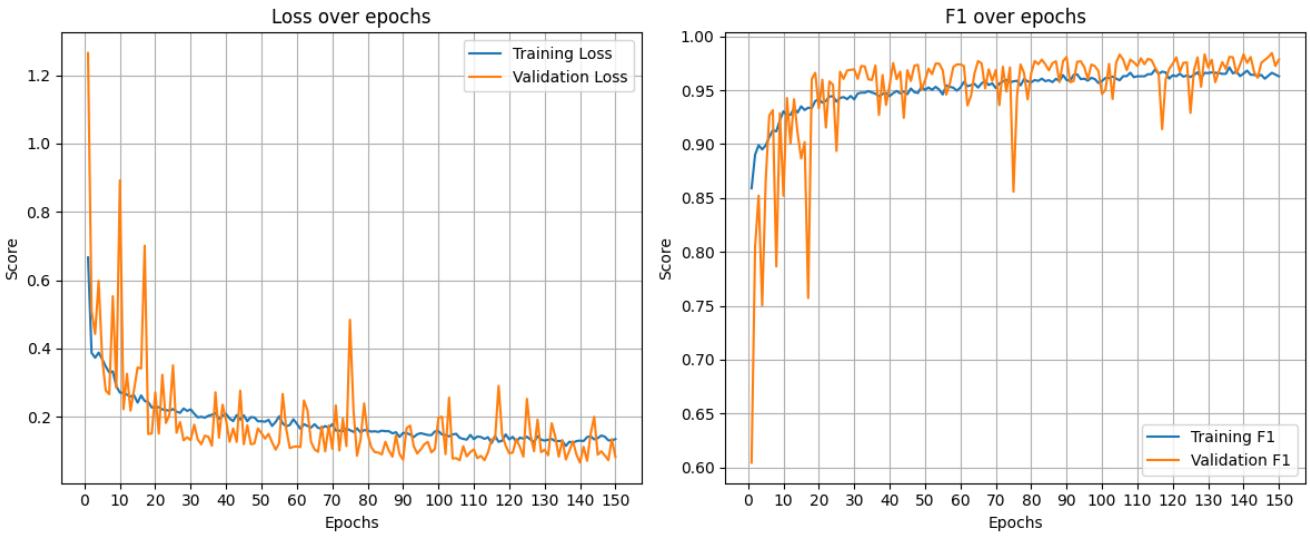


Figure 5. Loss and F1 score charts for Medmamba-s across 150 epochs.

hyperparameters may lead to a one metric score slightly better than the test score reported above, it may lead to a precipitous drop in performance in another metric. Take for example, changing Adam's learning rate from 0.001 to 0.0001; while Sensitivity score improves from 0.9428 to 0.9506, its specificity score decreases from 0.9317 to 0.5477. Decrease in test scores are also observed in OA and F1 when this learning rate value is used. More details of different hyperparameters used maybe found in Appendix section.

Comparing the test metrics between Medmamba-s and MedViT-large, the former model performed marginally as shown in all five metrics scores assessed. While metrics scores is an important factor in considering a suitable learning model for deployment, another factor to be considered is the amount of computational resources needed for training. MedMamba has lower complexity because it uses depthwise separable convolutions and avoids the quadratic overhead of self-attention. On the other hand, MedViT is more computationally expensive due to the self-attention mechanism, despite its use of efficient convolution blocks that is intended to reduce computational resources.

Finally, while the above experiments have deomonstrated that Medmamba-s and MedViT-large are the most suitable for predicting Pneumonia, it cannot be assumed that the same conclusion maybe reached when other datasets are used for training, validation and testing. Taking a different MNIST model, or any other image dataset for training and validation, and then applying the trained and validated model on a third image dataset for testing may report another trained model, with a different set of hyperparameters to be the most optimal.

### 5.1 Comparison with author data

The author's journal contained performance results of various models trained against several databases. However, the source of performance results was not clearly defined; in this case it is assumed to be training results of standard evaluation metrics. There was no mention of model weights generated used for testing. For purposes of comparison, training results generated by ownself is compared against author's performance scores, as shown in Table 6 below:

| **Model** | **OA-Self** | **AUC-Self** | **OA-Author** | **AUC-Author** |
|---|---|---|---|---|
| Medmamba-b | 0.949 | 0.986 | 0.925 | 0.973 |
| Medmamba-s | 0.947 | 0.985 | 0.936 | 0.976 |
| Medmamba-t | 0.949 | 0.988 | 0.899 | 0.965 |
| MedViT-large | 0.939 | 0.979 | 0.991 | 0.921 |
| MedViT-base | 0.947 | 0.986 | 0.973 | 0.925 |
| MedViT-small | 0.941 | 0.9812 | 0.961 | 0.995 |
| Resnet50 | 0.946 | 0.983 | 0.884 | 0.962 |

**Table 6**: The second and third columns shows OA and AUC results done by ownself respectively, whereas fourth and fifth columns shows OA and AUC results done by author.

For the case of PneumoniaMNIST using Medmamba-s model, only OA and AUC results were provided. Results done by ownself with that of author showed minimal variations between each other, although there is a difference of 6 points in OA between Resnet50 between ownself and author, with the former returning a better score. The author also displayed scores from MedViT models as large, small and tiny, which differed from Manzari's journal which classified the three variants of MedViT as large, base and small. For comparison purposes, MedViT-small and MedViT-tiny in Yue's journal is presumed to be referring to MedViT-base and MedViT-small in Manzari's journal respectively.

## 6 Conclusion and future work

This report has demonstrated the performance of Medmamba and MedViT variants in using PneumoniaMNIST for training and validation, and using CPN X-ray images for testing. The model weights are suitable to be used for sensitivity problems, as all sensitivity scores of ¿0.9. However, if other metrics are needed for consideration, Medmamba-s and MedViT-large are suitable models, considering their test metric scores of ¿0.9 and above in this aspect.

As PneumoniaMNIST is a binary dataset, other models have been considered for comparison study to assess the performance of the models mentioned in 3D datasets. Other datasets in the MNIST database are considerably larger and attempts to download their npz files have met with timeout responses. Additionally, consideration also needs to be given to the availability of test datasets when pairing with suitable train/validation datasets.

Even when such datasets are available, desirable training, validation and test results might not necessarily be returned. In this aspect, BreastMNIST for training and validation have been paired with the Breast Ultrasound Images Dataset for test, but much poorer standard metric scores have been returned. As a point of future work, extensive hyperparameter tuning and exploration can be considered.

The journal has also mentioned of using Grad-CAM as a visual interpretation tool to visualize and examine the model's decision-making mechanism. According to the journal's author, Grad-CAM focuses on the lesion areas in each type of image, and rarely pay attention to the background parts that are not related to the prediction results. Preliminary runs have been run on Medmamba-s for normal and pneumonia images on the test dataset, but due to the absence of professional guidance on interpreting X-ray images, no determination can be reached on this as yet, which shall be an area for further study.

## References


[1] Yue, Y. MedMamba: Vision Mamba for Medical Image Classification. *arXiv: 2403.03849v5*. 2024. https://arxiv.org/html/2403.03849v5

[2] Manzari, O. N. et al. MedViT: A Robust Vision Transformer for Generalized Medical Image Classification. *arXiv: 2302.09462v1*. 2023. https://doi.org/10.1016/j.compbiomed.2023.106791

[3] Nasiri-Sarvi, A. et al. Vision Mamba for Classification of Breast Ultrasound Images. MedMamba: Vision Mamba for Medical Image Classification. *arXiv: 2407.03552*. 2024. https://arxiv.org/pdf/2407.03552

[4] Yang, J., Shi, R., Wei, D. et al. MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification. *Scientific Data*. vol. 10, no. 41, 2023. https://doi.org/10.1038/s41597-022-01721-8

[5] Yang J. et al. MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis. *IEEE 18th International Symposium on Biomedical Imaging (ISBI)*. 2021. https://doi.org/10.48550/arXiv.2010.14925

[6] Shastri, S., Kansal, I., Kumar, S. et al. CheXImageNet: a novel architecture for accurate classification of Covid-19 with chest x-ray digital images using deep convolutional neural networks. Health Technol. 12, 193–204. 2022. https://doi.org/10.1007/s12553-021-00630-x

[7] Kumar S, Shastri S, Mahajan S, et al. LiteCovidNet: A lightweight deep neural network model for detection of COVID-19 using X-ray images. Int J Imaging Syst Technol. 2022, pp. 1-17. https://doi.org/10.1002/ima.22770

[8] Yue, Y. MedMamba: Vision Mamba for Medical Image Classification, GitHub repository. 2023. [Online]. Available: https://github.com/YubiaoYue/MedMamba. [Accessed: Jan. 6, 2025].

[9] Nejati, O. MedViT: Vision Transformer for Medical Image Classification. GitHub repository. 2023. [Online]. Available: https://github.com/Omid-Nejati/MedViT. [Accessed: Jan. 6, 2025].

# 7 Appendix

Standard conditions for Medmamba-s are as follows:

```
optimizer = Adam(net.parameters(), lr=0.0001)
milestones = [50, 75]  # range(10,100,10)
scheduler = MultiStepLR(optimizer, milestones=milestones, gamma=0.1)
```

Below subsections show the standard metric scores for training, validation and test datasets using Medmamba-s, with variations to hyperparameters used. Deviations from standard conditions mentioned above are listed under the leftmost ”Condition” column.

## 7.1 Training dataset

| **Condition** | **OA** | **Sensitivity** | **Specificity** | **F1** | **AUC** | **Loss** |
|---|---|---|---|---|---|---|
| Adam=0.0001 | 0.9669 | 0.9791 | 0.9316 | 0.9777 | 0.9941 | 0.0820 |
| Adam=0.01 | 0.8269 | 0.9176 | 0.5659 | 0.8872 | 0.8389 | 0.4276 |
| Adam=0.001 + Gamma=0.001 | 0.9562 | 0.9725 | 0.9094 | 0.9706 | 0.9880 | 0.1198 |
| Adam=0.001 + Gamma=0.01 | 0.9471 | 0.9717 | 0.8764 | 0.9646 | 0.9837 | 0.1363 |

| Condition | OA | Sensitivity | Specificity | F1 | AUC | Loss |
|---|---|---|---|---|---|---|
| SGD=0.0001 | 0.8619 | 0.9462 | 0.6194 | 0.9105 | 0.8997 | 0.3402 |
| SGD=0.001 + Gamma=0.001 | 0.9267 | 0.9591 | 0.8336 | 0.9510 | 0.9682 | 0.1903 |
| SGD=0.001 + Gamma=0.01 | 0.9191 | 0.9548 | 0.8163 | 0.9460 | 0.9652 | 0.1999 |

### 7.2 Validation dataset

| Condition | OA | Sensitivity | Specificity | F1 | AUC | Loss |
|---|---|---|---|---|---|---|
| Adam=0.0001 | 0.9599 | 0.9871 | 0.8815 | 0.9734 | 0.9914 | 0.1162 |
| Adam=0.01 | 0.8302 | 0.9409 | 0.5111 | 0.8916 | 0.8948 | 0.3704 |
| Adam=0.001 + Gamma=0.001 | 0.9618 | 0.9537 | 0.9852 | 0.9738 | 0.9962 | 0.1438 |
| Adam=0.001 + Gamma=0.01 | 0.8683 | 0.8226 | 1.0000 | 0.9027 | 0.9948 | 0.4294 |
| SGD=0.0001 | 0.8473 | 0.8175 | 0.9333 | 0.8883 | 0.9504 | 0.3729 |
| SGD=0.001 + Gamma=0.001 | 0.9485 | 0.9769 | 0.8667 | 0.9657 | 0.9884 | 0.1231 |
| SGD=0.001 + Gamma=0.01 | 0.9561 | 0.9589 | 0.9481 | 0.9701 | 0.9888 | 0.1414 |

### 7.3 Test dataset

| Condition | OA | Sensitivity | Specificity | F1 | AUC |
|---|---|---|---|---|---|
| Adam=0.0001 | 0.7490 | 0.9506 | 0.5477 | 0.7910 | 0.9369 |
| Adam=0.01 | 0.7965 | 0.7133 | 0.8796 | 0.7779 | 0.8136 |
| Adam=0.001 + Gamma=0.001 | 0.9045 | 0.8500 | 0.9589 | 0.8989 | 0.9689 |
| Adam=0.001 + Gamma=0.01 | 0.5480 | 0.9983 | 0.0982 | 0.6882 | 0.8985 |
| SGD=0.0001 | 0.5283 | 0.0572 | 0.9989 | 0.1081 | 0.8791 |
| SGD=0.001 + Gamma=0.001 | 0.7271 | 0.9628 | 0.4917 | 0.7791 | 0.9293 |
| SGD=0.001 + Gamma=0.01 | 0.8723 | 0.7894 | 0.9550 | 0.8607 | 0.9488 |