Companion Paper: MeDiANet Implementation and Reproducibility Details

Dipayan Dewan[1[0000−0002−3314−353X]], Asim Manna[1[0000−0001−7617−9762]], Dyutit Mohanty[2[0009−0006−6377−2923]], and Debdoot Sheet[1[0000−0001−9046−149X]]

[1] Indian Institute of Technology Kharagpur, 721302, India
[2] Manipal Institute of Technology, 576104, India
{{diiipayan93,asimmanna17}@kgpian, debdoot@ee}.iitkgp.ac.in, dyutit.mohanty@learner.manipal.edu

Abstract. This companion paper focuses on the reproducibility of our previously accepted work on the proposed MeDiANet architecture, a lightweight computationally efficient architecture designed for medical image classification. We outline key practices, including consistent random seed initialization and mixed-precision training with TensorFlow. By detailing these essential steps, we provide a clear framework to enable other researchers to replicate the findings, ensuring transparency and reliability in deep learning experiments.

Keywords: Convolutional neural network · Computationally efficient · Medicla image classification

1 Introduction

Reproducibility is a critical aspect of validating deep learning research, ensuring that results can be consistently replicated by others. This companion paper is focused on the reproducibility of our previously accepted paper "MeDiANet: A Lightweight Network for Large-scale Multi-disease Classification of Multi-modal Medical Images using Dilated Convolution and Attention Network", designed for medical image classification using the MedMNIST dataset. While the original paper explored model performance and architectural innovations, this paper aims to provide a detailed account of the steps taken to ensure the reproducibility of our results.

While providing code is helpful, there are often additional considerations, such as the effect of specific hyperparameter choices or the rationale behind selecting particular building blocks. In this paper, we delve deeper into these aspects by discussing the architectural components of MeDiANet and the strategies used for training section 2. The dataset preparation details are given in section 3. We also explore the influence of key parameters on the model's performance and present the exact implementation details in section 4. Furthermore, we provide some ablation studies to support the principle behind the network architecture design in section 5. The reproducibility steps of the paper is given in section 6.

This paper seeks to provide a comprehensive guide for replicating the model and its results, enabling other researchers to confidently reproduce the experiments.

2 Network Architecture

The network was implemented using both PyTorch and TensorFlow. Details on the implementations of the network building blocks are provided below.

2.1 Modified residual block(`Res(·)`)

The network module uses pre-activated residual blocks [2], however, all activations are Mish [3] instead of ReLU. Mish is a new activation function introduced in [3], and is smooth and non-monotonic. Mish has demonstrated improved performance in various deep learning tasks by offering better generalization and convergence compared to traditional activation functions such as ReLU[4] and Leaky ReLU[6]. Mathemetically it can be defined as:

$$\text{Mish}(x) = x \cdot \tanh(\text{softplus}(x))$$

where softplus is defined as:

$$\text{softplus}(x) = \ln(1 + e^x)$$

The tensorflow implementation of above is given below

```
class Mish(Layer):
    def __init__(self, **kwargs):
        super(Mish, self).__init__(**kwargs)
    def call(self, inputs):
        return inputs * tf.math.tanh(tf.math.softplus(inputs))
```

Below the the implementation of the modified residual block employing Mish activation layer is given. The PyTorch version of this implementation can be found in the Github Repository [4].

```
def residual_block(input, input_channels=None,
output_channels=None, kernel_size=(3, 3), stride=1, drop_prob=0):
    if output_channels is None:
        output_channels = input.shape[-1]
    if input_channels is None:
        input_channels = output_channels // 4
    strides = (stride, stride)
    x = BatchNormalization()(input)
    x = Mish()(x)
    x = Conv2D(input_channels, (1, 1))(x)
```

```
    x = BatchNormalization()(x)
    x = Mish()(x)
    x = Conv2D(input_channels, kernel_size, padding='same',
            strides=stride)(x)
    x = Dropout(0.3)(x)
    x = BatchNormalization()(x)
    x = Mish()(x)
    x = Conv2D(output_channels, (1, 1), padding='same')(x)
    input = Conv2D(output_channels, (1, 1),
                padding='same', strides=strides)(input)
    x = Add()([x, input])
    return x
```

2.2 Multi dilated residual block(MDiRes(·))

Each multi dilated residual block includes three parallel Conv2D of different dilation rates with the residual connection. The TensorFlow implementation is provided below:

```
def dilated_residual_block(input, input_channels=None,
    output_channels=None, kernel_size=(3, 3),
    dilation = [1,2,3], drop_prob=0, regularization = None):
    regularizer = l2(regularization)

    if output_channels is None:
        output_channels = input.shape[-1]
    if input_channels is None:
        input_channels = output_channels // 4
    x = BatchNormalization()(input)
    x = Mish()(x)
    x = Conv2D(input_channels, (1, 1))(x)
    x = BatchNormalization()(x)
    x = Mish()(x)
    x1 = Conv2D(input_channels, kernel_size,
        padding='same', dilation_rate=(dilation[0]),
        kernel_regularizer = regularizer)(x)
    x1 = Dropout(0.2)(x1)
    x2 = Conv2D(input_channels, kernel_size,
        padding='same', dilation_rate=(dilation[1]),
        kernel_regularizer = regularizer)(x)
    x2 = Dropout(0.2)(x2)
    x3 = Conv2D(input_channels, kernel_size,
        padding='same', dilation_rate=(dilation[2]),
        kernel_regularizer = regularizer)(x)
    x3 = Dropout(0.2)(x3)
    x = Add()([x1, x2, x3])
```

```
x = BatchNormalization()(x)
x = Mish()(x)
x = Conv2D(output_channels, (1, 1), padding='same')(x)
input = Conv2D(output_channels, (1, 1), padding='same')(input)
x = Add()([x, input])
return x
```

2.3 Dilated Residual Attention Block (DiET(·))

The following algorithm can be used to construct a Dilated Residual Attention Block, with adjustments based on the block's depth within the network.

Algorithm 1 DiET(·) Block Creation

Initialize out_input ← residual_block(input)

TRUNK BRANCH:
Set output_trunk ← out_input
Apply dilated_residual_block with dilation_rate to output_trunk
Apply dilated_residual_block with dilation_rate to output_trunk

ATTENTION MASK:
Initialize empty list skip_connections
for each i from 1 to encoder_depth **do**
    Apply MaxPool2D with 'same' padding to out_input, store result in out
    Apply dilated_residual_block with dilation_rate to out
    Apply dilated_residual_block with dilation_rate to out, store result in skip_connection
    Append skip_connection to skip_connections
end for
Reverse skip_connections
Apply dilated_residual_block with dilation_rate to out
Apply dilated_residual_block with dilation_rate to out

DECODER:
for each i from 1 to encoder_depth **do**
    Add out and skip_connections[i]
    Apply dilated_residual_block with dilation_rate to out
    Apply UpSampling2D to out
end for

Apply Conv2D(1×1) to out with input_channels
Apply Conv2D(1×1) to out with input_channels
Apply sigmoid activation to out
Add 1 to out
Multiply out by output_trunk (element-wise multiplication)
Apply residual_block to out
return out

The encoder depth of the above block will change depending on the stage of the network. As mentioned in the original paper two different variants of MeDiANet are proposed based on the number of channels (C_{in}) used in the first convolution layer of the network. The configuration for the different versions of the proposed MeDiANet are given below:

1. MeDiANet$_{base}$69: $C_{in} = 16$, B = (1,1,1,0), R = (1,1,1,2)
2. MeDiANet$_{base}$117: $C_{in} = 16$, B = (1,2,3,0), R = (1,1,1,3)
3. MeDiANet$_{wide}$69: $C_{in} = 32$, B = (1,1,1,0), R = (1,1,1,3)
4. MeDiANet$_{wide}$117: $C_{in} = 32$, B = (2,2,2,0), R = (1,1,1,3)

where B denotes the number of `DiET`$(\cdot)$ and R denotes the number of `Res`$(\cdot)$ in each stage.

3 Dataset Preparation

The dataset is constructed by aggregating publicly available datasets from various sources to create a large-scale medical image benchmark aimed at evaluating performance. The original sources and links are cited in the main paper. Each image is annotated with one of 35 common thoracic pathologies. This combined dataset includes images from four different organs captured across four different imaging modalities. In total, it consists of 148,753 images, with 100,475 images allocated for training, 11,164 for validation, and 37,214 for testing. Since the images come in varying sizes, they are all resized to $3 \times 224 \times 224$ before the training process, and they are stored in '.npy' format. The dataset can be accessed on Zenodo [1] [3]. Upon downloading and extracting the 'Dataset.zip' file, the images are organized into three subfolders: 'train', 'validation', and 'test'.

4 Training Instructions

The hyperparameters used for model training have been provided in Table 1. These hyper-parameters remain the same for training all the four different versions of MeDiANet. We use the random search method for hyper-parameter tuning using KerasTuner [5] for 200 trials where 10% of the training dataset (randomly sampled) has been used for 400 epochs. Hyper-parameters of the best trial were chosen for the final training with the full dataset.

5 Ablation Study

5.1 Ablation study for Attention Module

In 2 we show the effectiveness of `DiET`$(\cdot)$ in the proposed architecture. First, network performance is evaluated by using `MDiRes`$(\cdot)$ block, only in the Trunk

[3] https://zenodo.org/records/13923240

Parameter	Value
Optimizer	AdamW
Optimizer Momentum	$\beta_1 = 0.9$, $\beta_2 = 0.999$
Batch Size	192
Epochs	400
Initial Learning Rate	0.0016
Warmup Schedule	Linear
Warmup Epochs	40
Warmup Target	0.0007
Learning Rate Schedule	Cosine Decay
Alpha	0.042
Final Learning Rate	0.0007 * Alpha
Label Smoothing	0.1
Weight Decay	0.01

Table 1. Training Hyperparameters

branch of the `DiET`$(\cdot)$. Then we place the `MDiRes`$(\cdot)$ only in attention mask branch of the network as mentioned in the original paper. Finally, we evaluate the model performance by placing `MDiRes`$(\cdot)$ blocks in both Trunk branch and Attention Mask. From the comparison study stated in Table 2, it is clear that using the `MDiRes`$(\cdot)$ block in both branches of the `DiET`$(\cdot)$ gives the best result out of the alternatives.

Model	Accuracy (%)	Parameters (M)
Trunk branch	93.75	0.36
Soft mask branch	93.89	0.36
Trunk + Soft mask + Dilation	93.95	0.38

Table 2. Ablation study for Attention Module

5.2 Ablation study for activation function

In this study we check the performance of our proposed model with the ReLU and Mish activation functions keeping the rest of the network unchanged. From Table 3 it is evident that Mish activation function achieved better performance for both MeDiANet$_{Base}$ and MeDiANet$_{Wide}$, and time to train each epoch is also faster in comparison to ReLU.

Model	Accuracy (%)	Training time/epoch (M)
$MeDiANet_{base}69$+ReLU	93.93	0.88
$MeDiANet_{base}69$+Mish	94.18	0.87

Table 3. Ablation study for Activation function

6 Reproducibility

We outline key reproducibility practices implemented in the original study, including the consistent use of random seeds, detailed data preprocessing pipelines, and TensorFlow's mixed-precision training for computational efficiency. To ensure reproducibility, the random seeds for both the PyTorch and TensorFlow models were set to 153. The Random and NumPy libraries were also set to use a random seed of 153. To train the network, the 'main_mdnet.py' script can be used, along with arguments for the framework and model version to be trained. The pretrained TensorFlow model is available on the GitHub[4]. To execute the code, Python 3.10 is required, along with PyTorch 2.4.1, TensorFlow 2.16.1, and NumPy 1.26.4.

7 Conclusion

This paper presents the implementation details of the proposed MeDiANet architecture, and also provides the training details and evaluation procedure for reproducibility purposes. This paper extensively provides details on the implementation for each of the building blocks used in the MeDiANet. It also provides justification for choosing these blocks by providing ablation studies. Finally to reproduce the result provided in the original paper, we provide all the technical details that need to be followed. The hyperparameter setting has been used for all the four variants is of of MeDiANet$_{base}$69 variant. Though all the hyperparameter setting has been provided for training the network in section 4, the paper does not investigate the different hyper parameter setting for each of the four varient of the proposed MeDiANet which could further improve the performance of the network.

References

1. Dewan, D., Asim, M.: Large scale medical image dataset (Oct 2024). https://doi.org/10.5281/zenodo.13923240, https://doi.org/10.5281/zenodo.13923240

[4] https://github.com/dipayandewan94/MeDiANet

2. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14. pp. 630–645. Springer (2016)
3. Misra, D.: Mish: A self regularized non-monotonic activation function. In: British Machine Vision Conference (2020), https://api.semanticscholar.org/CorpusID:221113156
4. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning. p. 807–814. ICML'10, Omnipress, Madison, WI, USA (2010)
5. O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al.: Kerastuner. https://github.com/keras-team/keras-tuner (2019)
6. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. CoRR **abs/1505.00853** (2015), http://arxiv.org/abs/1505.00853