

EE228 Teamwork:MedMNIST

Yicheng Song, Changqi Zhao

June 15, 2021

Contents

1	Completion Description	3
1.1	Classification Results	3
1.2	Method Description	3
1.3	Main Code Framework	3
1.4	Highlights	4
1.5	GitHub Link	4
2	Problem Description	4
2.1	Targets	4
2.2	Requirements	4
3	Model Design	4
3.1	ResNet-18	5
3.2	ResNet-50	5
3.3	Loss Function	6
4	Performance Analysis	6
4.1	ResNet-18 and ResNet-50	6
4.2	Model Parameters	6
5	Tricks in Design	7
5.1	Data Augment	7
5.1.1	Simple Transformations	7
5.1.2	AutoAugment [1]	8
5.1.3	DCGAN [2]	9
5.2	Model Training	9
6	Discussion	9
6.1	Challenges	9
6.2	Possible Improvements	10

1 Completion Description

1.1 Classification Results

Method	PathMNIST	ChestMNIST	DermaMNIST	OCTMNIST	PneumoniaMNIST
Res-18 AUC	0.98071	0.69807	0.90152	0.95088	0.96157
Res-18 ACC	0.87549	0.94524	0.70324	0.75226	0.84295
Res-50 AUC	0.98179	0.69391	0.89062	0.94371	0.95404
Res-50 ACC	0.86335	0.94537	0.71334	0.77201	0.87512

Method	RetinaMNIST	BreastMNIST	OrganMNIST_A	OrganMNIST_C	OrganMNIST_S
Res-18 AUC	0.71389	0.49603	0.99597	0.98904	0.96778
Res-18 ACC	0.54000	0.73077	0.91758	0.88652	0.74755
Res-50 AUC	0.71253	0.86967	0.99434	0.99037	0.96828
Res-50 ACC	0.46500	0.80128	0.91233	0.89506	0.72795

Table 1: Results Comparison on *Breastmnist* Dataset (ResNet-18) between DCGAN Method ,Original Method and Data Augment Method

On Resnet-18:

Method	Original	Augmented	DCGAN
Best Epoch	0	7	73
Test AUC	0.49603	0.64536	0.81725
Test ACC	0.73077	0.64744	0.78846

On ResNet-50:

Method	Original	Augmented	DCGAN
Best Epoch	99	37	42
Test AUC	0.86967	0.59273	0.86299
Test ACC	0.80128	0.65385	0.80128

Table 2: Results Comparison on *Breastmnist* Dataset (ResNet-18) between DCGAN Method ,Original Method and Data Augment Method

1.2 Method Description

In this project,we implement the Resnet-18 and Resnet-50 model and complete training and testing on the MedMNIST dataset.

In order to improve classification results,we realized three main data augment methods: simple transformations, AutoAugment and DCGAN.We also compare advantages and disadvantages between various methods.

1.3 Main Code Framework

ResNet-18,ResNet-50,DCGAN

1.4 Highlights

- We applied general data augment methods and managed to improve the result.
- We try DCGAN method on the dataset and make a difference in classification.

1.5 GitHub Link

<https://github.com/ammoniaavortex/EE228-medmnist>

2 Problem Description

This is a Medical image classification problem using ResNet methods on the open-source data set: MedMNIST [3].

2.1 Targets

1. Familiar with the composition of medical pictures
2. Familiar with the construction of neural network
3. Master the classification method of medical pictures

2.2 Requirements

1. According to the given code example, build Resnet-18 and Resnet-50 network
2. Expound the differences between Resnet-18 and Resnet-50 in classification results, and analyzes the reasons for the differences
3. Find out the reasons for the influence of model input resolution on the results are analyzed
4. Offer our own improvement scheme on how to improve the classification effect

3 Model Design

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10 ⁹	3.6×10 ⁹	3.8×10 ⁹	7.6×10 ⁹	11.3×10 ⁹

First, the first layer convolution uses a 7 * 77 * 7 template with a step size of 2 and padding of 3. Then *BN*, *relu* and *maxpool* were performed. These constitute the first part of convolution module

conv1.

There are 4 layers, with make in the code *make_layer()*.

```
class ResNet(nn.Module):

    def __init__(self, block, num_blocks, in_channels=1, num_classes=2):
        self.inplane = 64

        super(ResNet, self).__init__()
        ...
        self.layer1 = self.make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self.make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self.make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self.make_layer(block, 512, num_blocks[3], stride=2)
        ...

    def make_layer(self, block, planes, num_blocks, stride):
        ...
        return nn.Sequential(*layers)

    def forward(self, x):
        ...
        return out
```

There are several modules in each stage. Each module is called building block.

Resnet18 = [2,2,2,2], there are eight building blocks. And Resnet50 = [3,4,6,3].

Notice that he has two modules, **Basicblock** and **Bottleneck**.

Resnet18 use **Basicblock** and resnet50 use **Bottleneck**.

Both **Basicblock** and **Bottleneck** modules use the shortcut connection mode.

3.1 ResNet-18

Resnet18 represent 18 layers with weight, including convolution layer and full connection layer, excluding pooling layer and BN layer.

```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        ...
        self.conv2 = conv3x3(planes, planes)
        ...

    def forward(self, x):
        ...
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

In **Basicblock** architecture, two convolutions of 3x 3 are used, and then *BN* is used.

3.2 ResNet-50

Resnet50, layers=[3,4,6,3]

```
class Bottleneck(nn.Module):
```

```

expansion = 4

def __init__(self, inplane, midplane, stride, downsample=None):
    super(Bottleneck, self).__init__()

    self.conv1 = nn.Conv2d(inplane, midplane, kernel_size=1, stride=stride, bias=False)
    self.bn1 = nn.BatchNorm2d(midplane)
    self.conv2 = nn.Conv2d(midplane, midplane, kernel_size=3, stride=1, padding=1, bias=False)
    self.bn2 = nn.BatchNorm2d(midplane)
    self.conv3 = nn.Conv2d(midplane, midplane * self.expansion, kernel_size=1, stride=1, bias=False)
    self.bn3 = nn.BatchNorm2d(midplane * self.expansion)
    self.relu = nn.ReLU(inplace=False)
    ...

def forward(self, x):
    ...
    out += self.shortcut(x)
    out = F.relu(out)
    return out

```

3.3 Loss Function

```

if task == "multi-label, binary-class":
    criterion = nn.BCEWithLogitsLoss()
else:
    criterion = nn.CrossEntropyLoss()

```

4 Performance Analysis

4.1 ResNet-18 and ResNet-50

Generally speaking, ResNet-50 has better classification result, especially when the data set is not big enough.

In addition to the difference of network depth, the choice of kernel is also different.

We assume that Over fitting exist when the training data set is big enough, which leads to not so satisfying results on ResNet-50.

4.2 Model Parameters

- Depth
layers of neural network
- Width
number of channels per layer
- Resolution
refers to the resolution of the feature map in the network

1. Deeper network has better nonlinear expression ability, can learn more complex transformation, thus can fit more complex features, deeper network can learn complex features more simply.

Network deepening will bring the problems of gradient instability and network degradation. Too deep network will reduce the shallow learning ability. When the depth reaches a certain level, the performance will not be improved, and it may decline.

2. Enough width can ensure that each layer can learn rich features, such as texture features of different directions and frequencies. The width is too narrow, the feature extraction is not enough, the learning is not enough, and the performance of the model is limited.

Width contributes to a large amount of network computation, too wide network will extract too many duplicate features, increasing the computational burden of the model.

3. To improve the network performance, we can start from the width, improve the utilization rate of each layer channel, use the information of other channels to supplement the narrower layer, find the lower limit of the width, and get better performance with the least amount of calculation

5 Tricks in Design

5.1 Data Augment

The role of data augment:

1. Increase the amount of training data, improve the generalization ability of the model
2. Noise data is added to improve the robustness of the model

5.1.1 Simple Transformations

Implementation on our own:

Geometric Transformation:

Flipping, Shifting up and down, left and right, Rotating the image in a certain angle

Pixel Content Transformation:

Cropping, Color jittering, Adding noise

Method	Original	Augmented
Best Epoch	0	7
Test AUC	0.49603	0.64536
Test ACC	0.73077	0.64744

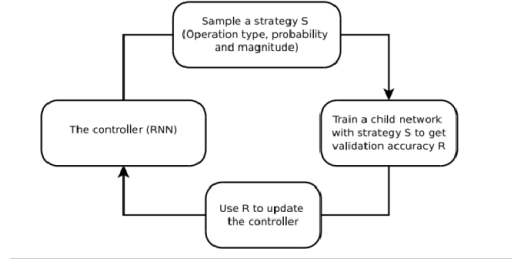
Table 3: Results Comparison on *Breastmnist* Dataset (ResNet-18) between original Method and Data Augment Method

Method	Original	Augmented
Best Epoch	78	25
Test AUC	0.96157	0.94362
Test ACC	0.84295	0.80609









Table 4: Results Comparison on *Pneumoniamnist* Dataset (ResNet-18) between original Method and Data Augment Method

5.1.2 AutoAugment [1]

The problem of finding the best data enhancement strategy is formalized as a discrete search problem.



In the search space, a strategy consists of five sub strategies, each sub strategy contains two image processing operations which are applied in turn.

	Original	Sub-policy 1	Sub-policy 2	Sub-policy 3	Sub-policy 4	Sub-policy 5
Batch 1						
Batch 2						
Batch 3						
		Equalize, 0.4, 4 Rotate, 0.8, 8	Solarize, 0.6, 3 Equalize, 0.6, 7	Posterize, 0.8, 5 Equalize, 1.0, 2	Rotate, 0.2, 3 Solarize, 0.6, 8	Equalize, 0.6, 8 Posterize, 0.4, 6

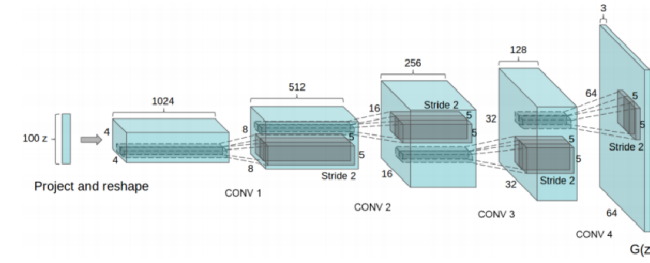
And each operation is also related to two super parameters:

- 1) the probability of application operation;
- 2) the amplitude of operation.

Policy	Original	ImageNet	CIFAR-10	SVHN
Best Epoch	4	1	1	2
Train AUC	0.86832	0.83553	0.83571	0.87301
Train ACC	0.62222	0.56701	0.58819	0.65127
Val AUC	0.80850	0.75999	0.74616	0.76377
Val ACC	0.55000	0.50313	0.47083	0.50625
Test AUC	0.71389	0.68577	0.69558	0.69440
Test ACC	0.54000	0.49812	0.49188	0.49875

Table 5: Results Comparison on Retinamnist Dataset between Different AutoAugment Policies and Original Method

5.1.3 DCGAN [2]



Compared with Gan or ordinary CNN, the improvement of dcgan includes the following aspects:

1. Using convolution and deconvolution instead of pooling layer
2. Batch normalization operation is added in generator and discriminator
3. The full connection layer is removed and the global pooling layer is used instead
4. The output layer of the generator uses the *tanh* activation function, and the other layers use *relu*
5. All layers of the discriminator are activated by *leakyrelu*

5.2 Model Training

Change the input format (choose model ResNet18 or ResNet50 in console).

6 Discussion

6.1 Challenges

Most challenging part is where to find enough computing resources.
We use our own Laptops,kaggle and Google Cloud to complete the MachineLearning project.

On Resnet-18:

Method	Original	Augmented	DCGAN
Best Epoch	0	7	73
Test AUC	0.49603	0.64536	0.81725
Test ACC	0.73077	0.64744	0.78846

On ResNet-50:

Method	Original	Augmented	DCGAN
Best Epoch	99	37	42
Test AUC	0.86967	0.59273	0.86299
Test ACC	0.80128	0.65385	0.80128

Table 6: Results Comparison on *Breastmnist* Dataset (ResNet-18) between DCGAN Method ,Original Method and Data Augment Method

6.2 Possible Improvements

We can follow the idea from google researchers [1], applying PPO or other reinforcement learning method on the *MedMNIST* data set.

A DCGAN

Model part:
Generator:

```
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        ...

        self.conv_blocks = nn.Sequential(
            ...
        )

    def forward(self, z):
        ...
        return img
```

Discriminator:

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        def discriminator_block(in_filters, out_filters, bn=True):
            ...
            return block

        self.model = nn.Sequential(
            ...
        )

        ...
```

```
def forward(self, img):
    ...

    return validity
```

Training part:

The training method of DCGAN is the same as that of GAN, which can be divided into the following three steps:

1. For K steps: train d to maximize the value of formula:
 $\log D(x) + \log(1 - D(G(z)))$ (G keeps still)
2. Keep D unchanged and train g to make the value of formula $\log D(G(z))$ reach the maximum
3. Repeat step (1) and step (2) until G and D reach Nash equilibrium

```
for epoch in range(opt.n_epochs):
    for i, (imgs, _) in enumerate(dataloader):

        # Adversarial ground truths
        valid = Variable(Tensor(imgs.shape[0], 1).fill_(1.0), requires_grad=False)
        fake = Variable(Tensor(imgs.shape[0], 1).fill_(0.0), requires_grad=False)

        real_imgs = Variable(imgs.type(Tensor))

        optimizer_G.zero_grad()

        # Sample noise as generator input
        z = Variable(Tensor(np.random.normal(0, 1, (imgs.shape[0], opt.latent_dim))))

        # Generate a batch of images
        gen_imgs = generator(z)

        # Loss measures generator's ability to fool the discriminator
        g_loss = adversarial_loss(discriminator(gen_imgs), valid)

        g_loss.backward()
        optimizer_G.step()

        optimizer_D.zero_grad()

        # Measure discriminator's ability to classify real from generated samples
        real_loss = adversarial_loss(discriminator(real_imgs), valid)
        fake_loss = adversarial_loss(discriminator(gen_imgs.detach()), fake)
        d_loss = (real_loss + fake_loss) / 2

        d_loss.backward()
        optimizer_D.step()
    ...
```

References

- [1] Ekin D. Cubuk*, Dandelion Mané Barret Zoph†, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. 2018.
- [2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.
- [3] Jiancheng Yan, Rui Shi, Bingbing Ni, and Bilian Ke. Medmnist classification decathlon: A lightweight automl benchmark for medical image analysis. 2020.