INT305 Machine Learning (2021-22)

Assignment 2 – Report

Student Name: Tianlei Shi

Student ID Number: 1824152

Novermber 30, 2021

I. PART 1

In this section, we will describe and analyze the convolutional kernel and the loss function used in the framework with mathematic equations.

A. Convolutional kernel

As an important part of neural network, convolution kernel is responsible for feature extraction and dimension raising (or dimension reducing). Two 3x3 convolution kernels with a stride of 1 are used in framework. The formula of convolution operation is shown as equation (1),

$$x_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)} \tag{1}$$

where x_{ij} represents the value with coordinates of (i, j) in feature map, m represent the size of the convolution kernel, ω_{ab} represents the weight with coordinates of (a, b) in convolution kernel, and $y_{(i+a)(j+b)}$ represents the value with coordinates of $(i + a, j + b)$ in the input image.

The essence of convolution operation is the weighted accumulation of pixels in a region that as same size as convolution kernel of the input image. The detailed operation process is shown as equation (2).

input image convolution kernel feature map

$$\begin{bmatrix} x_{11}^c & x_{12}^c & x_{13}^c & x_{14}^c & x_{15}^c \\ x_{21}^c & x_{22}^c & x_{23}^c & x_{24}^c & x_{25}^c \\ x_{31}^c & x_{32}^c & x_{33}^c & x_{34}^c & x_{35}^c \\ x_{41}^c & x_{42}^c & x_{43}^c & x_{44}^c & x_{45}^c \\ x_{51}^c & x_{52}^c & x_{53}^c & x_{54}^c & x_{55}^c \end{bmatrix} \times \begin{bmatrix} w_{11}^{cn} & w_{12}^{cn} & w_{13}^{cn} \\ w_{21}^{cn} & w_{22}^{cn} & w_{23}^{cn} \\ w_{31}^{cn} & w_{32}^{cn} & w_{33}^{cn} \end{bmatrix} = \begin{bmatrix} z_{11}^n & z_{12}^n & z_{13}^n \\ z_{21}^n & z_{22}^n & z_{23}^n \\ z_{31}^n & z_{32}^n & z_{33}^n \end{bmatrix} \tag{2}$$

The above equation takes a 3x3 convolution kernel and 5x5 input image as an example, where in input image, x represents pixel, c represents channel; in convolution kernel, w represents weight, c represents corresponding channel of image, n represents number of kernel; and in feature map, z represents value in feature map, n represents number of feature map.

Taking z_{11}^n as an example, the calculation process is shown in equation (3)

$$\begin{aligned} z_{11}^n = {} & w_{11}^{cn} x_{11}^c + w_{12}^{cn} x_{12}^c + w_{13}^{cn} x_{13}^c + w_{21}^{cn} x_{21}^c + w_{22}^{cn} x_{22}^c \\ & + w_{23}^{cn} x_{23}^c + w_{31}^{cn} x_{31}^c + w_{32}^{cn} x_{32}^c + w_{33}^{cc} x_{33}^c \end{aligned} \tag{3}$$

B. Loss function

The loss function used in framework is cross-entropy loss function, the equation is shown as equation (4),

$$L = -\sum_{i=1}^{N} y^{(i)} * \log \hat{y}^{(i)} \tag{4}$$

where N represents the number of categories in the current dataset, $y^{(i)}$ represents the label of class i ($y^{(i)}$ is 1 if data belongs to class i, and 0 otherwise), and $\hat{y}^{(i)}$ represents the confidence of class i in the model prediction results.

Moreover, according to the source code in PyTorch, the F.cross_entropy() function is implemented by combining the 'log_softmax' and the 'nll_loss'. Log-softmax is to perform an extra log operation after softmax, thus its equation is shown as equation (5),

$$\text{logsoftmax}\,(y) = \log \frac{e^{y_k}}{\sum_j e^{y_j}} \tag{5}$$

where y represents prediction results of model, the numerator of the fraction represents the Euler exponent of the k-th y, and the denominator represents the sum of all Euler exponent of y.

The equation of nll_loss (negative log-likelihood) is shown as equation (6),

$$nll(x, t) = -\sum_{i=0}^{N} (x_t)_i \tag{6}$$

where x represents input value, t represents label, N represents number of labels, and $(x_t)_i$ represents the t-th element in the i-th input.

Therefore, the equation of cross-entropy loss function in PyTorch is shown as equation (7).

$$L(y,t) = \text{nll}\,(\text{logsoftmax}, t) = -\sum_{i=0}^{N} \left[\left(\log \frac{e^{y_k}}{\sum_j e^{y_j}}\right)_t\right]_i \quad (7)$$

II. PART 2

The framework is trained on MNIST dataset [1]. The batch size was set to 64, and trained 30 epochs. The learning rate was initially set to 0.1, and decays by 0.7 times per epoch. For the convenience of later comparison and analysis, we take framework as baseline.

The final accuracy performance of baseline is 97.61%, the loss and accuracy chat and confusion matrix chat are shown as Fig.1 and Fig.2.

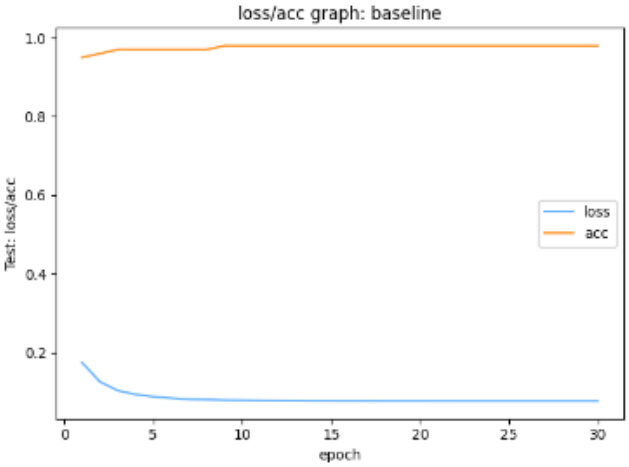


Fig.1. loss and accuracy chat of baseline

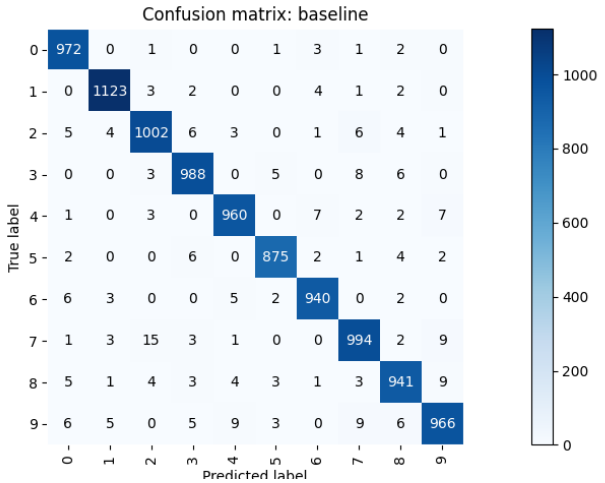


Fig.2. confusion matrix chat of baseline

What's more, to show the performance of the model more clearly, we select a well-classified and a mis-classified example from each category respectively, and shown as Fig.3 and Fig.4.

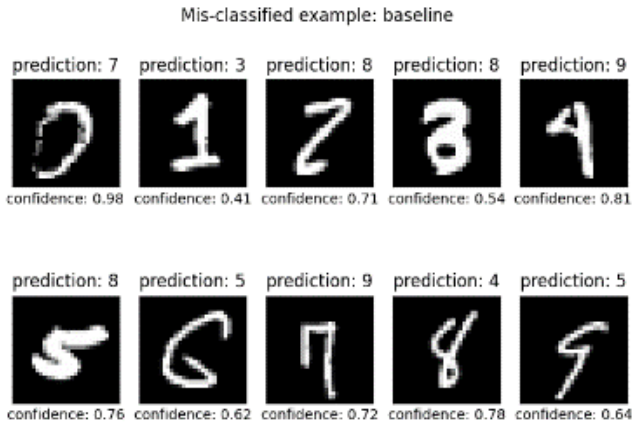


Fig.3. mis-classified example from each category

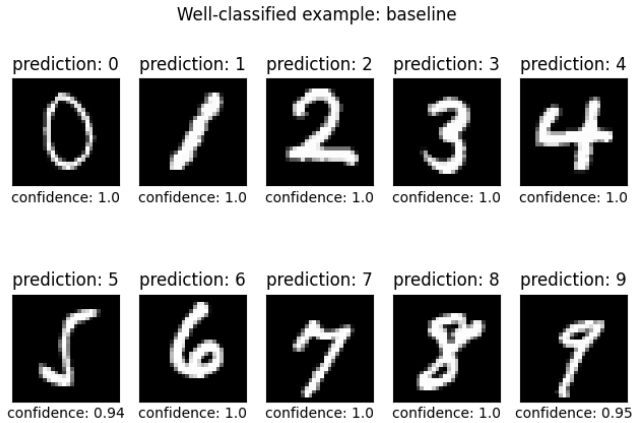


Fig.4. well-classified example from each category

The total parameters of baseline is 149,418, and the estimated total size is 0.70 MB.

III. PART 3

In this section, we propose a network of pruning-SEResnet that can improve the accuracy performance and reduce the size of model effectively, the improvement process of the method is as follows. Additionally, in the process of improving the network, we also conducted ablation experiments to verify that each additional module was effective, the experiment results is displayed after each module.

A. Shortcut connection and batchnorm

Due to the performance of baseline is not satisfactory, and given its structure, we think it is possible that the simple structure of the model limits its performance [2]. Therefore, we added more convolution kernels to the baseline to increase the depth and representation power of the model, and added shortcut connection [3] and batch norm [4] to improve model performance.

Deeper models, which mean better nonlinear representation, can learn more complex transformations and thus fit more complex feature inputs. However, deeper networks are harder to train and suffer from gradient instability and network deterioration. Shortcut connection can solve the above problems.

When the network deterioration, shortcut connection can map the features from shallow layer with better training effects to the deep layer, so as to ensure that the deep layer

training effects do not worse than shallow layer, so as to solve the network degradation. Moreover, batch norm keeps the inputs of each layer of neural networks in the same distribution during training, so that it can speed up convergence, improve the generalization ability of the model, and prevent overfitting in training.

For the convenience of later comparison and analysis, we name this model (or step) as resnet (or shortcut connection+baseline). The batch size was set to 64, and trained 30 epochs. The learning rate was initially set to 0.1, and decays by 0.1 times per 10 epochs.

The final accuracy performance of baseline is 99.40%, the loss and accuracy chat and confusion matrix chat are shown as Fig.5 and Fig.6.

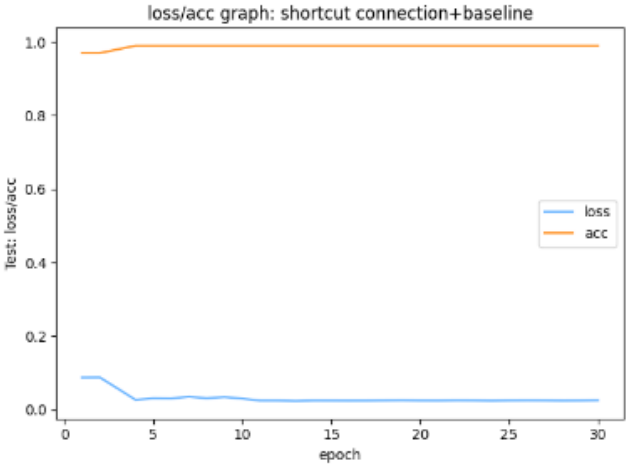


Fig.5. loss and accuracy chat of resnet

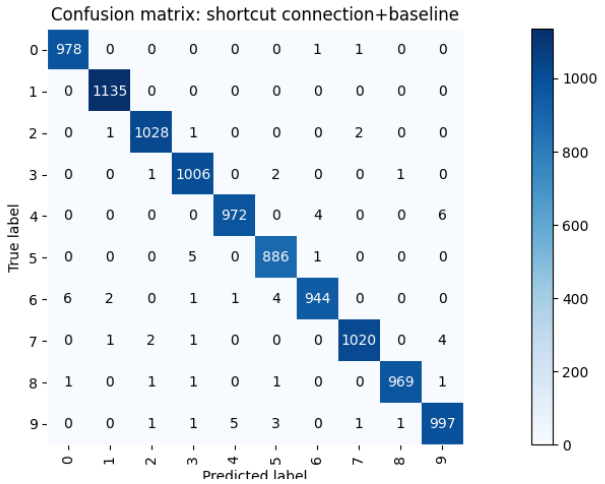


Fig.6. confusion matrix chat of resnet

What's more, to show the performance of the model more clearly, we select a well-classified and a mis-classified example from each category respectively, and shown as Fig.7 and Fig.8.

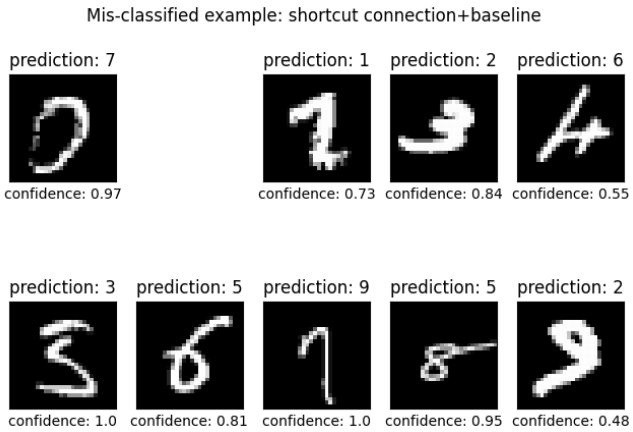


Fig.7. mis-classified example from each category

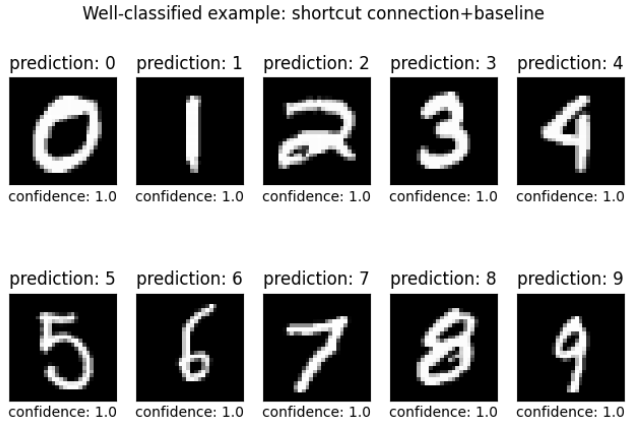


Fig.8. well-classified example from each category

Resnet has improved its accuracy by 1.79% over baseline, the accuracy of classification of the number 1 is 100% especially. The total parameters of resnet is 46,833,226, and the estimated total size is 183.06 MB.

B. 1x1 kernel

Using 1x1 convolution kernel can reduce the size of the network [5]. For example, assume we directly conduct convolutionalize to an 7x7x128 feature map by using 256 3x3 convolution kernels, and the number of network parameters is:

$$3 \times 3 \times 128 \times 256 = 294{,}912 \quad (8)$$

However, if we use 64 1x1 kernels first, then use 256 3x3 kernels, the number of network parameters will be:

$$1 \times 1 \times 128 \times 64 + 3 \times 3 \times 64 \times 256 = 81{,}920 \quad (9)$$

After this operation, the number of parameters are changed to the 0.28 of original.

For the convenience of later comparison and analysis, we name this model (or step) as light resnet (or reduce size+resnet). The batch size was set to 64, and trained 30 epochs. The learning rate was initially set to 0.1, and decays by 0.1 times per 10 epochs.

The final accuracy performance of baseline is 99.39%, the loss and accuracy chat and confusion matrix chat are shown as Fig.9 and Fig.10.

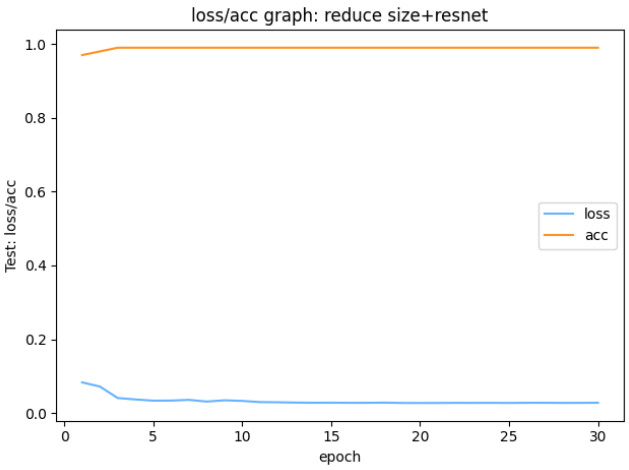


Fig.9. loss and accuracy chat of light resnet

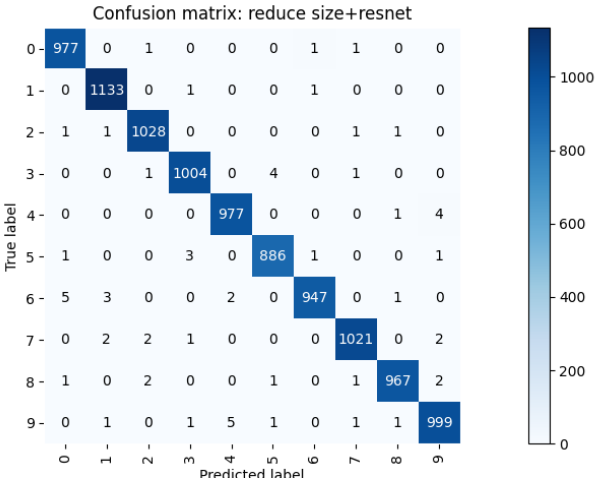


Fig.10. confusion matrix chat of light resnet

What's more, to show the performance of the model more clearly, we select a well-classified and a mis-classified example from each category respectively, and shown as Fig.11 and Fig.12.

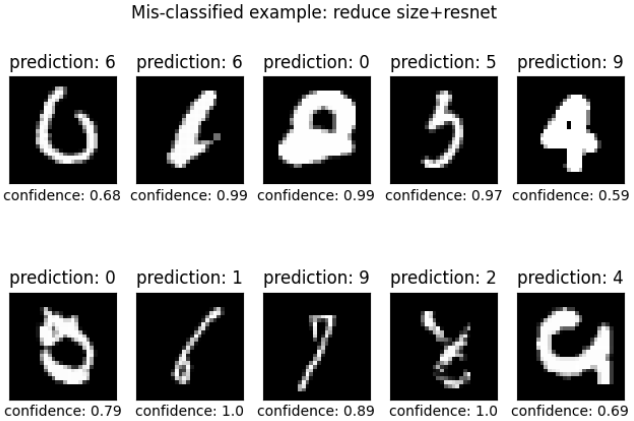


Fig.11. mis-classified example from each category

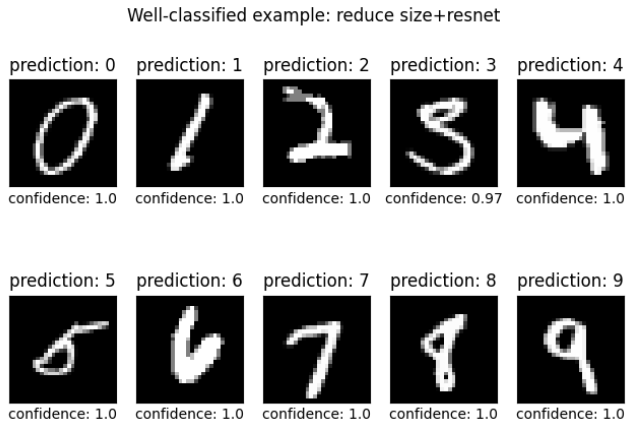


Fig.12. well-classified example from each category

The total parameters of light resnet is 23,522,250, and the estimated total size is 94.92 MB. Compared with resnet, the number of parameters is reduced by 50% and the size of the model is reduced to the 0.52 of original. However, the performance of the model does not change much. Finally, the structural comparison of resnet and light resnet is shown as Fig.13.

resnet	light resnet
7x7, 64, stride 2	7x7, 64, stride 2
3x3 max pool, stride 2	3x3 max pool, stride 2
$\begin{bmatrix} 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
$\begin{bmatrix} 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$
$\begin{bmatrix} 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$
$\begin{bmatrix} 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
average pool	average pool

Fig.13. structural comparison of resnet and light resnet

C. *SENet*

SENet [6] introduces attention model, which can greatly increases the performance of models.

Attention model allow the model to focus on important areas, thus ignoring the less important ones. SENet to get the importance of each feature map by train, and weight each feature map according to the importance, so that the network can focus on certain feature maps to improve model performance.

For the convenience of later comparison and analysis, we name this model (or step) as SE-resnet (or senet+light resnet). The batch size was set to 64, and trained 30 epochs. The learning rate was initially set to 0.1, and decays by 0.1 times per 10 epochs.

The final accuracy performance of SE-resnet is 99.58%, the loss and accuracy chat and confusion matrix chat are shown as Fig.14 and Fig.15.

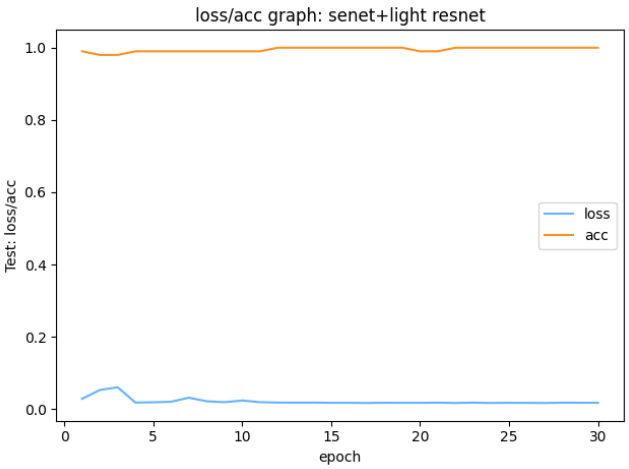


Fig.14. loss and accuracy chat of SE-resnet

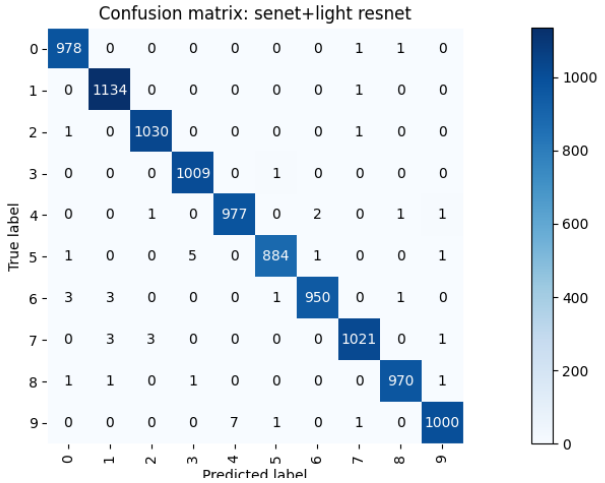


Fig.15. confusion matrix chat of SE-resnet

What's more, to show the performance of the model more clearly, we select a well-classified and a mis-classified example from each category respectively, and shown as Fig.16 and Fig.17.

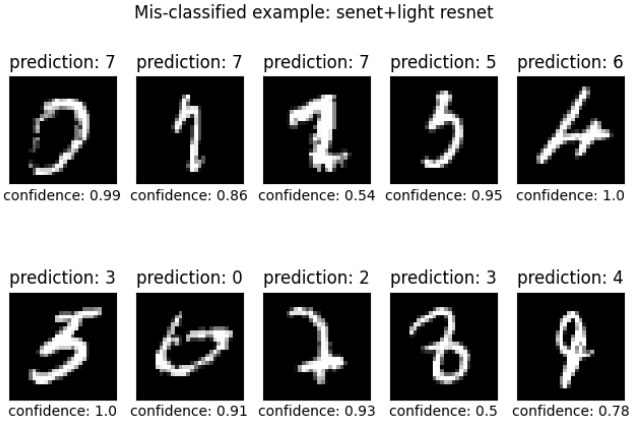


Fig.16. mis-classified example from each category

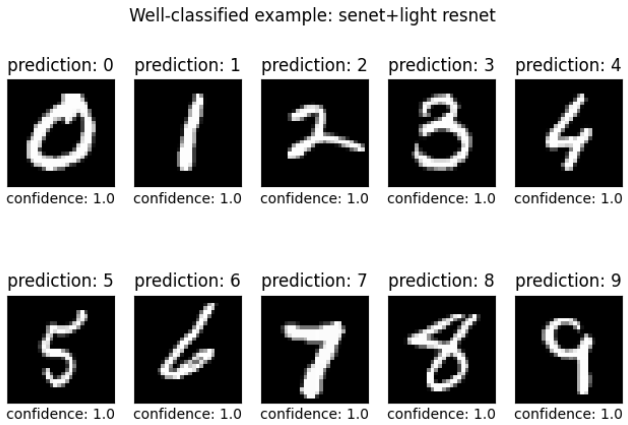


Fig.17. well-classified example from each category

SE-resnet has improved its accuracy by 0.19% over light resnet under the same training strategy. The total parameters of SE-resnet is 21,439,478, and the estimated total size is 96.63 MB.

D. Network pruning

Pruning [7] is one of the main methods to reduce model size. Its principle is to remove the unimportant connections in the model to reduce the model size and speed up the calculation.

For the convenience of later comparison and analysis, we name this model (or step) as pruning-SEResnet. We conducted the network pruning to the model of SE-resnet whose accuracy of 99.58%, and tuned the pruned-model 10 epochs to obtain higher performance. The batch size was set to 64, and the learning rate was set to 0.1.

The final accuracy performance of pruning-SEResnet is 97.62%, the loss and accuracy chat and confusion matrix chat are shown as Fig.18 and Fig.19.

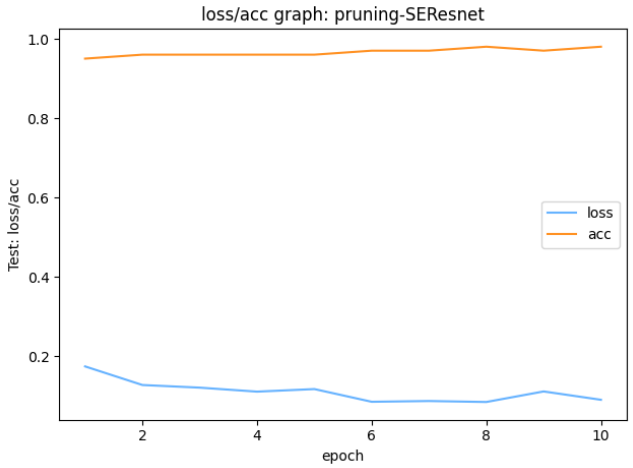


Fig.18. loss and accuracy chat of pruning-SEResnet

	baseline	Our network				Compared network		
		resnet	light resnet	SE-resnet	pruning-SEResnet	VGGnet	GoogLenet	Vision Transformer
Accuracy	97.61%	99.40%	99.39%	99.58%	97.62%	99.45%	99.18%	98.58%
Total params	149,418	46,833,226	23,522,250	21,439,478	19,190,848	128,825,290	5,603,882	85,076,746
Estimated total size (MB)	0.70	183.06	94.92	96.63	81.55	496.15	22.51	411.63

Table 1. comparison of our network and classic network

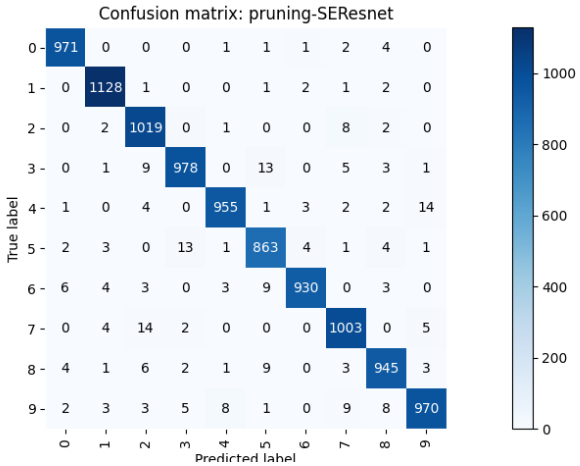


Fig.19. confusion matrix chat of pruning-SEResnet

What's more, to show the performance of the model more clearly, we select a well-classified and a mis-classified example from each category respectively, and shown as Fig.20 and Fig.21.

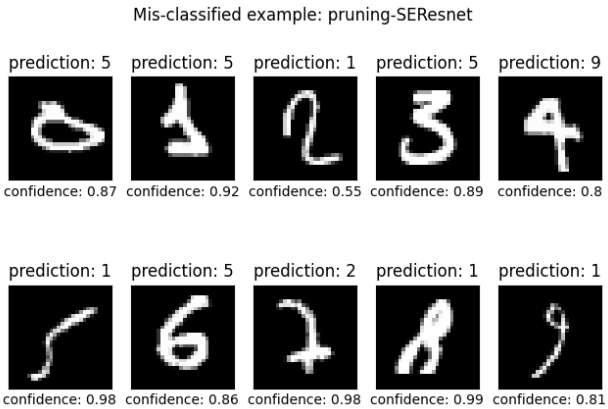


Fig.20. mis-classified example from each category

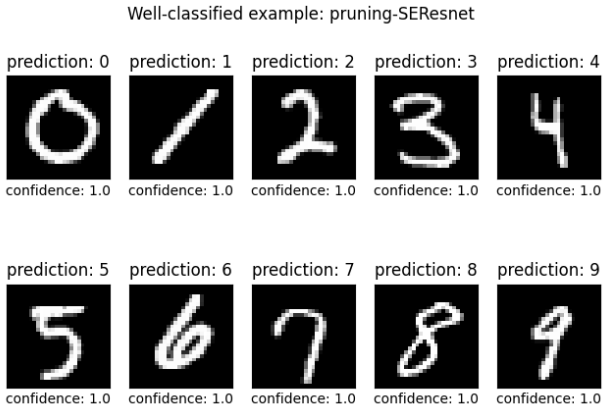


Fig.21. well-classified example from each category

The total parameters of pruning-SEResnet is 19,190,848, and the estimated total size is 81.55 MB. Compared to SE-resnet, pruning-SEResnet sacrificed 1.96% accuracy, reducing the size of the model by 16%.

EVALUATION

To prove the performance of our method, we trained the classic classification network of VGGnet [2], and GoogLenet [5] and the latest Vision Transformer (ViT) [8] on MNIST dataset, and compare their test results to our methods. The comparison results are shown in Table 1.

Therefore, through ablation experiments and comparisons, we can prove the classification performance of our network.

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," 1998.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016.

[4] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *ArXiv Learn.*, 2015.

[5] C. Szegedy *et al.*, "Going deeper with convolutions," 2015.

[6] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-Excitation Networks," 2018.

[7] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015.

[8] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ArXiv Comput. Vis. Pattern Recognit.*, 2020.

APPENDIX

The whole source code of this project at: https://github.com/Stanley-Yi/Y4S1-INT305-Machine-Learning-Assignment2

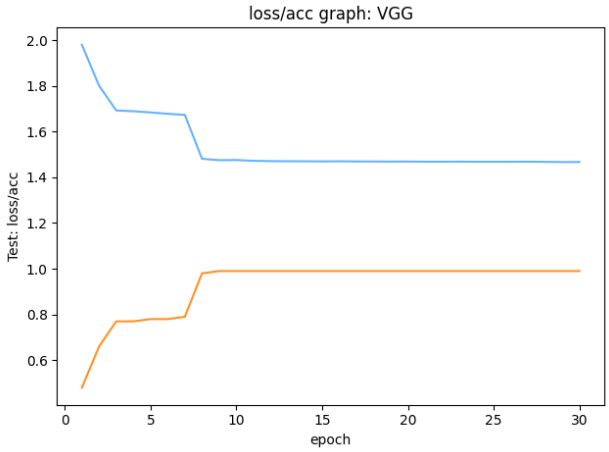
A. *Source Code of pruning-SEResnet*

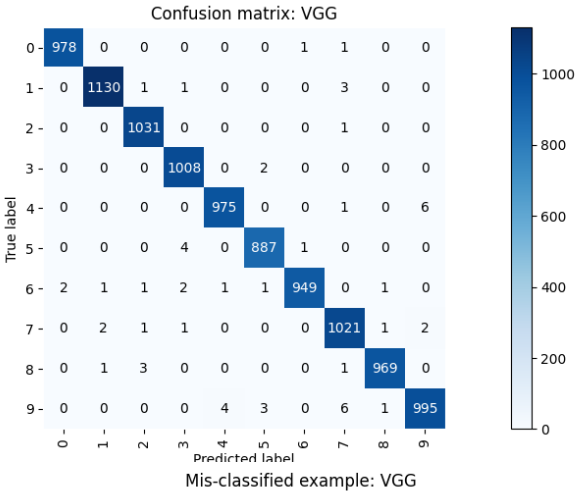
```
class PreActBlock(nn.Module):
    def __init__(self, in_planes,
planes, stride=1):
        super(PreActBlock,
self).__init__()
        self.bn1 =
nn.BatchNorm2d(in_planes)
        self.conv1 =
nn.Conv2d(in_planes, planes,
kernel_size=3, stride=stride,
padding=1, bias=False)
        self.bn2 =
nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes,
planes, kernel_size=3, stride=1,
padding=1, bias=False)

        if stride != 1 or in_planes !=
planes:
            self.shortcut =
nn.Sequential(
                nn.Conv2d(in_planes,
planes, kernel_size=1, stride=stride,
bias=False)
            )

        # SE layers
        self.fc1 = nn.Conv2d(planes,
planes//16, kernel_size=1)
        self.fc2 =
nn.Conv2d(planes//16, planes,
kernel_size=1)

    def forward(self, x):
        out = F.relu(self.bn1(x))
        shortcut = self.shortcut(out)
if hasattr(self, 'shortcut') else x
        out = self.conv1(out)
        out =
self.conv2(F.relu(self.bn2(out)))

        # Squeeze
        w = F.avg_pool2d(out,
out.size(2))
        w = F.relu(self.fc1(w))
        w = F.sigmoid(self.fc2(w))
        # Excitation
        out = out * w

        out += shortcut
        return out


class SENet(nn.Module):
    def __init__(self, block,
num_blocks, num_classes=10):
        super(SENet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(1, 64,
kernel_size=3, stride=1, padding=1,
bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        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)
        self.linear = nn.Linear(512,
num_classes)

    def _make_layer(self, block,
planes, num_blocks, stride):
        strides = [stride] +
[1]*(num_blocks-1)
        layers = []
        for stride in strides:

layers.append(block(self.in_planes,
planes, stride))
            self.in_planes = planes
        return nn.Sequential(*layers)

    def forward(self, x):
        out =
F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out


def Net():
    model = SENet(PreActBlock, [3, 4,
6, 3])
    return model


def updateBN(model,
s ,pruning_modules):
    for module in pruning_modules:
        module.weight.grad.data.add_(s
* torch.sign(module.weight.data))

def get_pruning_modules(model):
    module_list = []
    for module in model.modules():
        if
isinstance(module,PreActBlock):
```

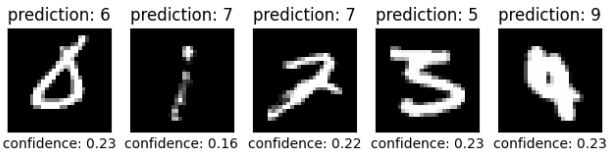
```
module_list.append(module.bn1)

module_list.append(module.bn2)
    return module_list


def
gather_bn_weights(model,pruning_modules
):
    size_list =
[module.weight.data.shape[0] for module
in model.modules() if module in
pruning_modules]
    # print(size_list)
    bn_weights =
torch.zeros(sum(size_list))
    # print(bn_weights)
    index = 0
    for module, size in
zip(pruning_modules, size_list):
        bn_weights[index:(index +
size)] =
module.weight.data.abs().clone()
        index += size

    return bn_weights

def
computer_eachlayer_pruned_number(bn_wei
ghts,thresh, bn_modules):
    num_list = []
    #print(bn_modules)
    for module in bn_modules:
        num = 0

#print(module.weight.data.abs(),thresh)
        for data in
module.weight.data.abs():
            if thresh > data.float():
                num +=1
        num_list.append(num)
    print(thresh)
    return num_list

def prune_model(model, num_list):
    model.to('cuda')
    DG =
tp.DependencyGraph().build_dependency(m
odel, torch.randn(1, 1, 28, 28))

    def prune_bn(bn, num):
        L1_norm =
bn.weight.detach().cpu().numpy()
        prune_index =
np.argsort(L1_norm)[:num].tolist()  #
remove filters with small L1-Norm
        plan = DG.get_pruning_plan(bn,
tp.prune_batchnorm, prune_index)
        print(plan)
        plan.exec()

    blk_id = 0
    for m in model.modules():
        if isinstance(m, PreActBlock):
            prune_bn(m.bn1,
num_list[blk_id])
            prune_bn(m.bn2,
num_list[blk_id + 1])
            blk_id += 2
    return model
```

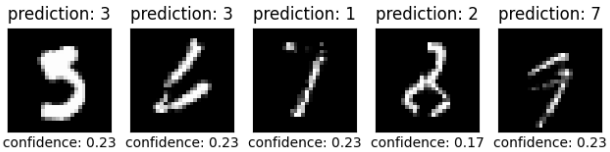
B. Training details of VGGnet, GoogLenet, and ViT

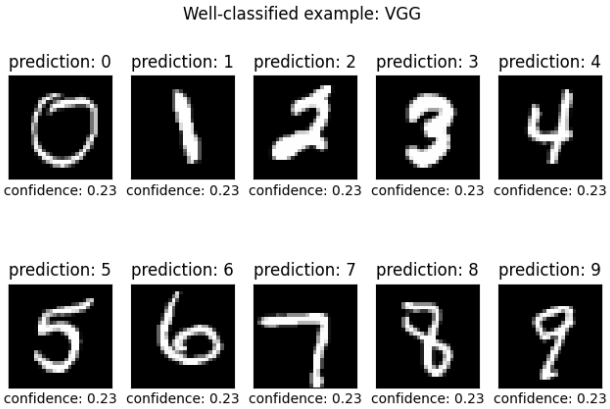
- VGGnet

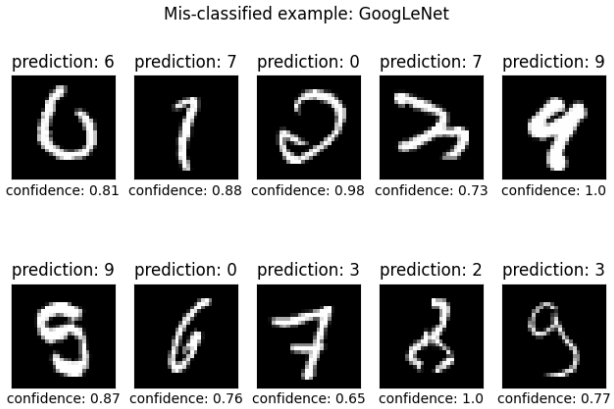




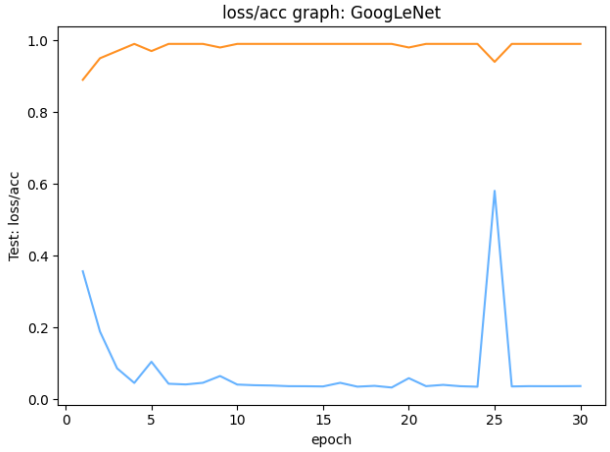


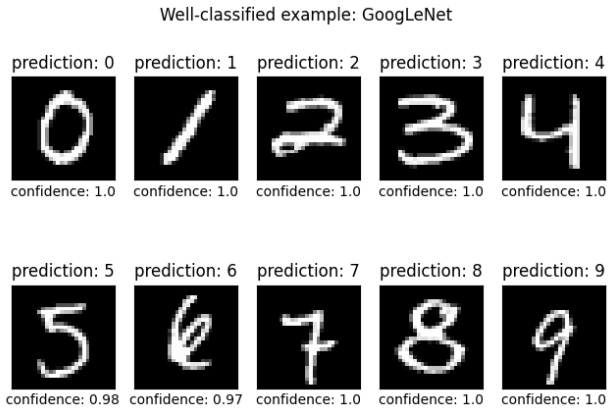


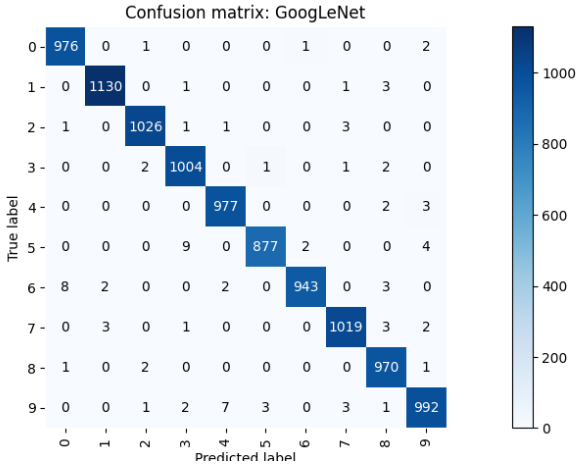




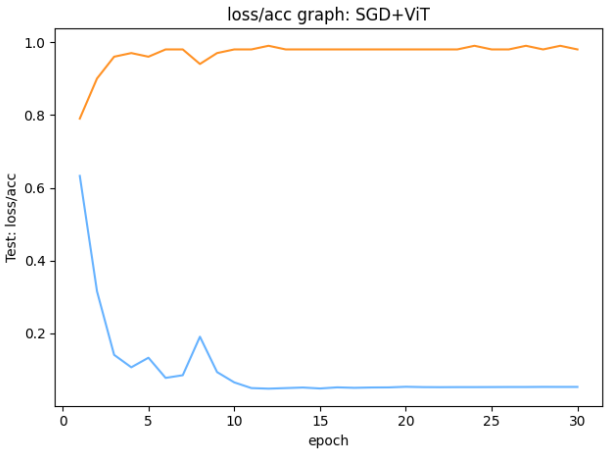
- GoogLenet

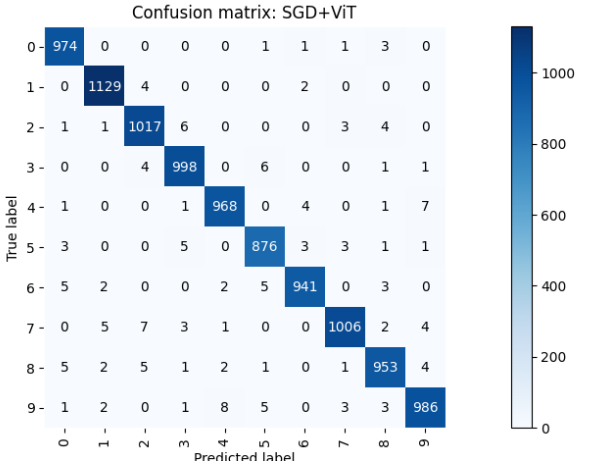




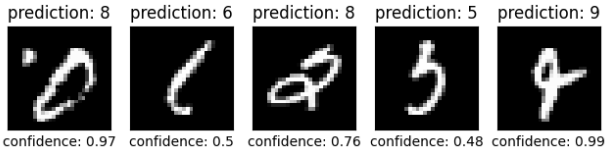


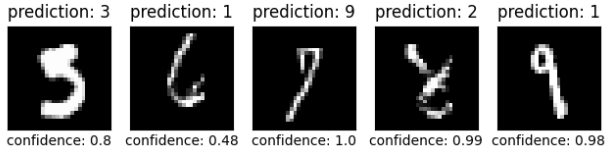
- ViT





Mis-classified example: SGD+ViT





Well-classified example: SGD+ViT