Birds classification implement by DenseNet

SUN Zhen
LIN Di
GUO Peiyuan

Contribution:

SUN Zhen (43%):

1.Apply the DenseNet model to birds classification

2.Use ImageNet pre-trained models to improve the accuracy of classification.

3.Improved the accuracy of classification by modifying the network structure of the DenseNet model and adding the SENet structure

4.Try to enhance the data from the direction of different transformations of the image, finding a suitable way to improve the accuracy of classification.

5.Assist in the completion of the report.

LIN Di (35%):

1.Study the ResNet model and proposed ideas for improving the application of the DenseNet model in birds classification

2.Assist with tasks such as applying DenseNet models to birds classification and improving classification accuracy through pre-training.

3.Through reading a large number of literature, provide ideas for the final improvement of the model, and analyze the principle of each improvement to improve the accuracy.

4.Complete most of the task of writing the report.

GUO Peiyuan (22%):

1. Assist with the implementation of the model.

2. Assist with report writing.

Index

1.Abstract

Nowadays Residual neural network (ResNet), a variant of Convolutional Neural Network which use the skip or residual connections, is widely used for improving performance of fine-grained classification problem, such as birds classification. We refer related references and documentation, apply Dense Convolutional Network (DenseNet) to improve test accuracy on birds classification. After implement our DenseNet model with PyTorch, we also find four ways to improve accuracy of our birds classification model, which are using ImageNet to pre-training model, choosing better DenseNet structure, adding Squeeze-and-Excitation (SE) Networks and try some extra operations such as rotation on training images. Finally, the maximum test accuracy of our model is 0.862.

2.Birds classification implement by DenseNet

2.1 Introduction of DenseNet

Compared to traditional convolutional feed-forward, ResNet add a skip-connection that the gradient can flow directly from later layers to the earlier layers, as the figure (1) shows blow:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}. \tag{1}$$

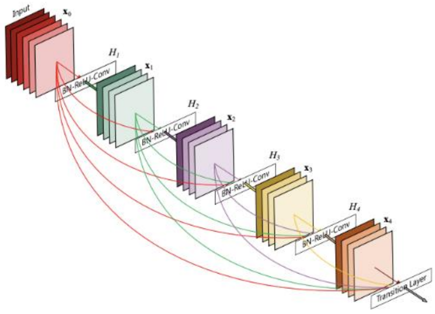
However, the gradient flow may be impeded because the output is combined by summation. To further improve the information flow between layers, Dense Convolutional Network (DenseNet) direct connections from any layer to all subsequent layers, the layer receives the feature-maps of all preceding layers as input [1], as the figure (2) shows blow:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]), \tag{2}$$

By design, DenseNet divides layers to several blocks, all layers spread their weights over many inputs within the same block.

2.2 Build DenseNet model with PyTorch

PyTorch is an open-source machine learning framework for production deployment, the framework supports Dense Convolutional Network (DenseNet), figure (3) shows the structure of PyTorch DenseNet and how weights spread between layers [2].



(3)

PyTorch also provides four DenseNet structures, which are densenet121, densenet161, densenet169 and densenet201. As the figure (4) shows, PyTorch DenseNet divides layers to four blocks, the main differences between those four structures are the number of layers in each block and the number of filters added to each layer and learned in the first convolution layer.

```python
class DenseNet(nn.Module):
    r"""Densenet-BC model class, based on
    `"Densely Connected Convolutional Networks" <https://arxiv.org/pdf/1608.06993.pdf>`_.

    Args:
        growth_rate (int) - how many filters to add each layer (`k` in paper)
        block_config (list of 4 ints) - how many layers in each pooling block
        num_init_features (int) - the number of filters to learn in the first convolution layer
        bn_size (int) - multiplicative factor for number of bottle neck layers
          (i.e. bn_size * k features in the bottleneck layer)
        drop_rate (float) - dropout rate after each dense layer
        num_classes (int) - number of classification classes
        memory_efficient (bool) - If True, uses checkpointing. Much more memory efficient,
          but slower. Default: *False*. See `"paper" <https://arxiv.org/pdf/1707.06990.pdf>`_.
    """

    def __init__(
        self,
        growth_rate: int = 32,
        block_config: Tuple[int, int, int, int] = (6, 12, 24, 16),
        num_init_features: int = 64,
        bn_size: int = 4,
        drop_rate: float = 0,
        num_classes: int = 1000,
        memory_efficient: bool = False,
    ) -> None:
```

(4)

At the beginning, we choose the densenet121 structure to build our birds classification model, which uses default values of growth_rate, block_config and num_int_features. We also noticed that the default value of num_classes in PyTorch DenseNet is 1000, as the figure (4) shows. As our birds image dataset which has about 200 classes, we change the parameter on the last classification layer, as the figure (5) shows.

```python
# build densenet121, and change the args of the last classification layer
bird_densenet = torchvision.models.densenet121(weights=DenseNet121_Weights.DEFAULT)
bird_densenet.classifier = nn.Linear(bird_densenet.classifier.in_features, 200)
bird_densenet.to(DEVICE)
```

(5)

2.3 Data preparation

After choosing densenet121 as our model to predict birds classification, now we need to prepare training dataset and testing dataset. The work mainly contains two parts, the first part is the basic preparation of dataset. As the figure (6) shows, this part of work including calculating the number of birds species and how many pictures in each birds species, changing the birds species name to numeric values, disorder birds picture paths and birds species, etc.

```python
def get_picture_path(dir_path):  # 获取所有鸟类图片的路径

    bird_dir_list = os.listdir(dir_path)
    bird_species = dict()
    picture_paths = []
    each_bird_species = []  # 储存每一个鸟类对应的种类
    for each_bird in bird_dir_list:
        bird_name = each_bird.split('.')[1]
        this_bird_picture_path = glob.glob(dir_path + '/' + each_bird + '/*.jpg')
        for i in range(len(this_bird_picture_path)):
            each_bird_species.append(bird_name)
        picture_paths.append(this_bird_picture_path)
        bird_species[bird_name] = len(this_bird_picture_path)
    bird_statistics = pd.DataFrame(list(bird_species.items()), columns=['species', 'picture_number'])  # 统计

    picture_paths = list(np.concatenate(picture_paths))
    return picture_paths, each_bird_species, bird_statistics


def digitize_bird_tags(each_bird_species, bird_statistics):  # 将鸟的种类名称数字化
    digital_bird_species = []
    for bird in each_bird_species:
        digital_bird_species.append(bird_statistics[bird_statistics.species == bird].index.tolist()[0])
    return digital_bird_species


def change_list_random(a1, a2):  # 将列表随机打乱
    np.random.seed(117)
    random_list_number = np.random.permutation(len(a1))
    a1 = np.array(a1)[random_list_number]
    a2 = np.array(a2)[random_list_number]
    return a1, a2
```

(6)

The second part is to build training dataset and testing dataset. We divide dataset to training dataset and testing dataset at the ratio of 9:1. As the figure (7) shows, before normalizing, we convert ImageNet's mean pixel value to range [0,255] for padding training images, and we also crop, vertical flip, horizontal flip training images randomly. Then we normalize training tensor images and testing tensor images with mean and standard deviation from ImageNet [3].

```python
# fill padded area with ImageNet's mean pixel value converted to range [0, 255]
fill = tuple(map(lambda x: int(round(x * 256)), (0.485, 0.456, 0.406)))
# pad images to 500 pixels
max_padding = transforms.Lambda(lambda x: pad(x, fill=fill))

transforms_train = transforms.Compose([
    # max_padding,
    max_padding,
    transforms.RandomOrder([
        transforms.RandomCrop((375, 375)),
        transforms.RandomVerticalFlip(),
        transforms.RandomHorizontalFlip()
    ]),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
])
transforms_eval = transforms.Compose([
    # max_padding,
    transforms.RandomCrop((375, 375)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

(7)

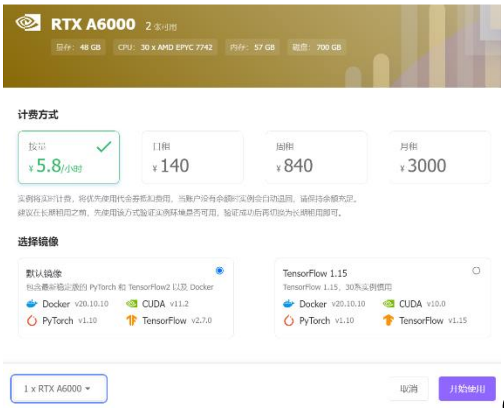
2.4 Start training and testing densenet121 model

After building our training dataset and testing dataset, now we could define related arguments and start training and testing our birds classification model. As the figure (8) shows, we define that every training and testing will load 50 samples and the model will train and test for 35 rounds. At first, we want to train our model for 50 rounds, but after serval tests, we observe that the training accuracy and testing accuracy have low growth since about fifth round, so we let the model train 35 rounds to reduce cost. Also, at each round of training cost and testing cost, we will record loss and accuracy as standards for accessing and improving our model.

```python
for epoch in range(50):
    output = open('result_now.txt', 'a', encoding='utf-8')
    # train
    bird_densenet.train()
    running_loss = 0.0
    train_acc = 0.0
    for step, data in enumerate(train_dl, start=0):
        images, labels = data
        optimizer.zero_grad()

        train_y = bird_densenet(images.to(DEVICE))
        loss = loss_function(train_y, labels.long().to(DEVICE))

        train_predict_y = torch.max(train_y, dim=1)[1]
        train_acc += (train_predict_y == labels.long().to(DEVICE)).sum().item()
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        # print train process
        rate = (step + 1) / len(train_dl)
        a = "*" * int(rate * 50)
        b = "." * int((1 - rate) * 50)
        print("\rtrain loss: {:^3.0f}%[{}->{}]{:.4f}".format(int(rate * 100), a, b, loss), end="")
    train_accurate = train_acc / train_num
```

(8)

To meet demands of running DenseNet model, such as the high demand of graphics card, we find a machine learning platform named Featurize [4], on which we could rent servers which meet demands of DenseNet, and run our DenseNet model on that remote machine, as the figure (9) shows.



(9)

2.4 Performance of densenet121 model and ideas for improvement

As the figure (10) shows, we find the best testing accuracy in 35 rounds is only 0.254, which is a bad performance. To improve performance of model, we refer to related references and documentation, and finally find four possible ways. The first is to pre-training our model using ImageNet. The second is to choose a better DenseNet model structure from densenet121, densenet161, densenet169 and densenet201. The third way is to add Squeeze-and-Excitation (SE) blocks on four blocks of DenseNet model. Last, we try some extra operations such as rotation on training images to see whether it would have an improvement. In the other parts of this report, we will introduce principle, implement and effect of these four ways for you.

```
[epoch 1] train_loss: 1094.709 test_loss: 120.043 train_accuracy: 0.021 test_accuracy: 0.039
[epoch 2] train_loss: 1016.106 test_loss: 113.743 train_accuracy: 0.045 test_accuracy: 0.051
[epoch 3] train_loss: 956.083 test_loss: 110.189 train_accuracy: 0.072 test_accuracy: 0.065
[epoch 4] train_loss: 908.378 test_loss: 105.471 train_accuracy: 0.096 test_accuracy: 0.079
[epoch 5] train_loss: 868.455 test_loss: 101.333 train_accuracy: 0.119 test_accuracy: 0.090
[epoch 6] train_loss: 832.210 test_loss: 97.645 train_accuracy: 0.139 test_accuracy: 0.111
[epoch 7] train_loss: 798.569 test_loss: 94.488 train_accuracy: 0.160 test_accuracy: 0.124
[epoch 8] train_loss: 767.584 test_loss: 91.547 train_accuracy: 0.181 test_accuracy: 0.147
[epoch 9] train_loss: 736.370 test_loss: 90.297 train_accuracy: 0.203 test_accuracy: 0.162
[epoch 10] train_loss: 706.693 test_loss: 88.081 train_accuracy: 0.229 test_accuracy: 0.165
[epoch 11] train_loss: 678.716 test_loss: 87.854 train_accuracy: 0.247 test_accuracy: 0.174
[epoch 12] train_loss: 653.182 test_loss: 85.612 train_accuracy: 0.272 test_accuracy: 0.184
[epoch 13] train_loss: 626.737 test_loss: 83.640 train_accuracy: 0.292 test_accuracy: 0.201
[epoch 14] train_loss: 603.444 test_loss: 85.952 train_accuracy: 0.320 test_accuracy: 0.170
[epoch 15] train_loss: 579.765 test_loss: 82.964 train_accuracy: 0.342 test_accuracy: 0.194
[epoch 16] train_loss: 558.409 test_loss: 81.366 train_accuracy: 0.362 test_accuracy: 0.194
[epoch 17] train_loss: 536.057 test_loss: 84.662 train_accuracy: 0.383 test_accuracy: 0.182
[epoch 18] train_loss: 512.965 test_loss: 83.610 train_accuracy: 0.415 test_accuracy: 0.185
[epoch 19] train_loss: 493.815 test_loss: 81.648 train_accuracy: 0.429 test_accuracy: 0.198
[epoch 20] train_loss: 471.519 test_loss: 81.838 train_accuracy: 0.457 test_accuracy: 0.211
[epoch 21] train_loss: 452.780 test_loss: 85.009 train_accuracy: 0.473 test_accuracy: 0.198
[epoch 22] train_loss: 431.256 test_loss: 85.751 train_accuracy: 0.504 test_accuracy: 0.192
[epoch 23] train_loss: 411.034 test_loss: 84.172 train_accuracy: 0.527 test_accuracy: 0.221
[epoch 24] train_loss: 393.589 test_loss: 82.839 train_accuracy: 0.547 test_accuracy: 0.209
[epoch 25] train_loss: 373.343 test_loss: 82.575 train_accuracy: 0.573 test_accuracy: 0.221
[epoch 26] train_loss: 357.416 test_loss: 86.039 train_accuracy: 0.590 test_accuracy: 0.212
[epoch 27] train_loss: 343.344 test_loss: 86.086 train_accuracy: 0.607 test_accuracy: 0.229
[epoch 28] train_loss: 324.246 test_loss: 82.552 train_accuracy: 0.629 test_accuracy: 0.252
[epoch 29] train_loss: 309.158 test_loss: 83.530 train_accuracy: 0.645 test_accuracy: 0.240
[epoch 30] train_loss: 294.473 test_loss: 83.433 train_accuracy: 0.666 test_accuracy: 0.254
[epoch 31] train_loss: 281.159 test_loss: 87.268 train_accuracy: 0.684 test_accuracy: 0.243
[epoch 32] train_loss: 264.903 test_loss: 89.143 train_accuracy: 0.703 test_accuracy: 0.237
[epoch 33] train_loss: 253.940 test_loss: 92.154 train_accuracy: 0.716 test_accuracy: 0.238
[epoch 34] train_loss: 239.233 test_loss: 90.576 train_accuracy: 0.738 test_accuracy: 0.234
[epoch 35] train_loss: 222.669 test_loss: 89.631 train_accuracy: 0.764 test_accuracy: 0.238
Finished Training.Best test accuracy is 0.254
```

(10)

3.Pre-training DenseNet model on ImageNet

3.1 Introduction

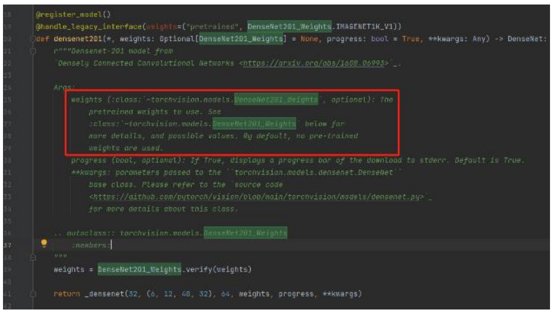
ImageNet has about 1.2 million labeled images and 1000 distinct object classes, which forces the network to learn a hierarchy of generalizable features [5]. We could compare ImageNet with our birds images dataset, 1.2 million images of ImageNet from lots of different categories as figure (11) shows, which contain large various shallow and deep features while our training birds image dataset only has about 10,000 images which may not contain enough features to fine-grained classification.

(11)

We think the weights pre-trained on ImageNet can extract features well because ImageNet has about 1.2 million labeled images and 1000 distinct object classes, many of these classes are also similar. We believe that at least some features extracted from pre-training on ImageNet can also use for our birds classification, solving the problem that our dataset is small and do not have enough features for fine-grained classification.

3.2 Implement

After looking up PyTorch DenseNet source code, we find that all four DenseNet structures in PyTorch have already support pre-training as figure (12) and figure (13) show.



(12)

```python
class DenseNet121_Weights(WeightsEnum):
    IMAGENET1K_V1 = Weights(
        url="https://download.pytorch.org/models/densenet121-a639ec97.pth",
        transforms=partial(ImageClassification, crop_size=224),
        meta={
            **_COMMON_META,
            "num_params": 7978856,
            "_metrics": {
                "ImageNet-1K": {
                    "acc@1": 74.434,
                    "acc@5": 91.972,
                }
            },
        },
    )
    DEFAULT = IMAGENET1K_V1
```

(13)

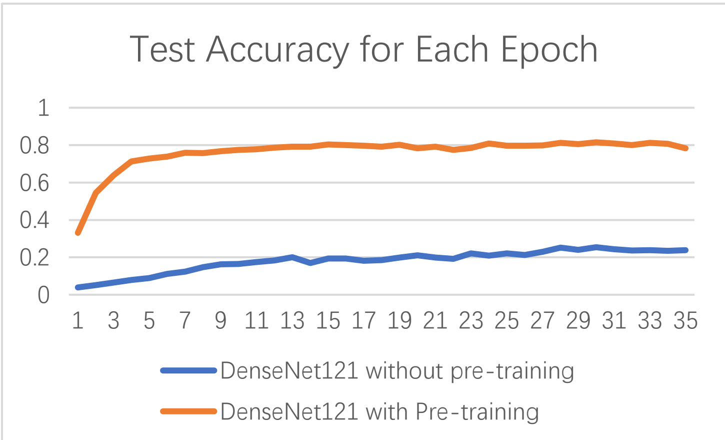
As figure (14) shows, we can change our program code to rebuild our densenet121 model by setting the pre-training argument so that our densenet121 model could use part of ImageNet pre-trained weights.

```python
# pre-training densenet121 with ImageNet
bird_densenet = torchvision.models.densenet121(weights=("pretrained", DenseNet121_Weights.IMAGENET1K_V1))
bird_densenet.classifier = nn.Linear(bird_densenet.classifier.in_features, 200)
bird_densenet.to(DEVICE)
```

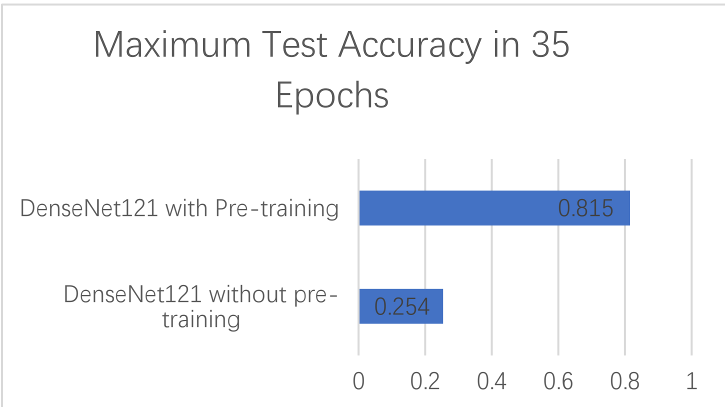
(14)

3.3 Performance Comparison

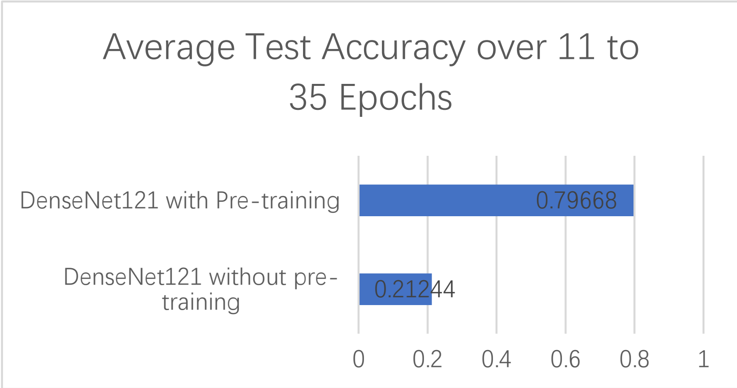
After changing program code to pre-training densenet121 model, now we can train and test our new model which with pre-training on a GPU machine, compare the performance between densenet121 model with pre-training and without pre-training, as figure (15), figure (16) and figure (17) show.



(15)



(16)



(17)

We can obviously see that both Maximum Test Accuracy (MTA) and Average Test Accuracy (ATA) have a great improvement after we pre-training densenet121 model. The MTA in 35 rounds improve from 0.254 to 0.815 while the ATA improve from about 0.21244 to 0.79668, which means that the idea of pre-training DenseNet model with ImageNet has a great effect.

4.Choose a better PyTorch DenseNet structure

4.1 Introduction

As we have mentioned earlier in the report, PyTorch support four DenseNet structures for users to choose. The main differences between four structures are the number of layers in each pooling block and the number of filters added to each layer and learned in the first convolution layer.

Layers	Output Size	DenseNet-121	DenseNet-161	DenseNet-169	DenseNet-201
Convolution	112 x 112	7 x 7 conv, stride 2			
Pooling	56 x 56	3 x 3 max pool, stride 2			
Dense Block (1)	56 x 56	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times6$
Transition Layer (1)	56 x 56	1 x 1 conv			
	28 x28	2 x 2 average pool, stride 2			
Dense Block (2)	28 x28	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times12$
Transition Layer (2)	28 x28	1 x 1 conv			
	14 x 14	2 x 2 average pool, stride 2			
Dense Block (3)	14 x 14	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times24$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times36$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times48$
Transition Layer (3)	14 x 14	1 x 1 conv			
	7x7	2 x 2 average pool, stride 2			
Dense Block (4)	7x7	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times16$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times24$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1\text{ Conv}\\3\times3\text{ Conv}\end{bmatrix}\times32$
Classification Layer	1x1	7 x 7 global average pool			
		1000D fully-connected,softmax			

(18)

As figure (18) shows, densenet201 structure has more layers in the third dense block and forth dense block than other structures, which means some layers in third block and forth block will receive feature-maps with more information. This is the reason we consider densenet201 model may have a better performance on birds classification prediction.

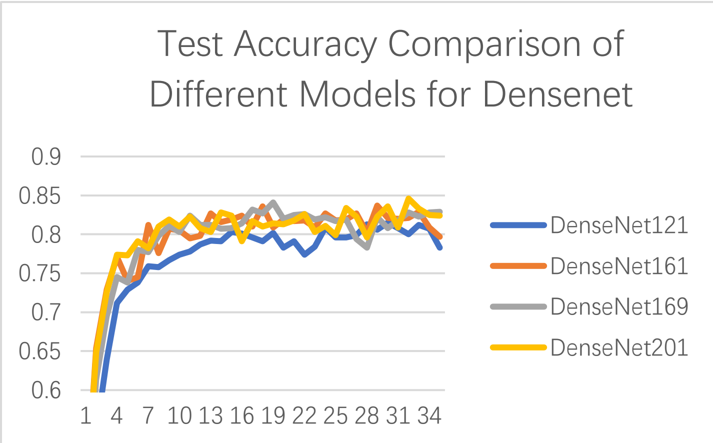
4.2 Implement

When we build DenseNet model in PyTorch, we can use different structures by choosing different functions, as the figure (19) shows.

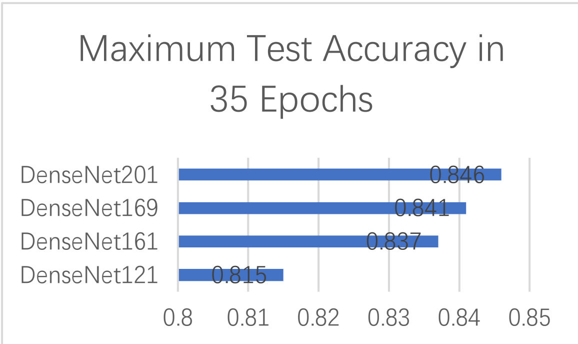
```
# densenet model with different structures
bird_densenet = torchvision.models.densenet121(weights=("pretrained", DenseNet121_Weights.IMAGENET1K_V1))
bird_densenet = torchvision.models.densenet161(weights=("pretrained", DenseNet161_Weights.IMAGENET1K_V1))
bird_densenet = torchvision.models.densenet169(weights=("pretrained", DenseNet169_Weights.IMAGENET1K_V1))
bird_densenet = torchvision.models.densenet201(weights=("pretrained", DenseNet201_Weights.IMAGENET1K_V1))
```
(19)

4.3 Performance Comparison

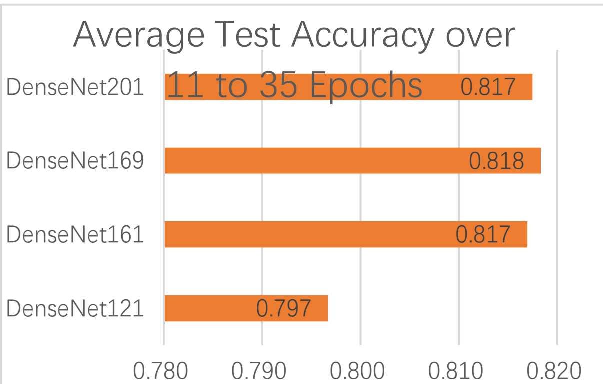
We try all four structures (pre-trained by ImageNet), running in the GPU machine, training, and testing for the same 35 rounds, comparing the performance of four structures, as figure (20), figure (21) and figure (22) show.



(20)



(21)



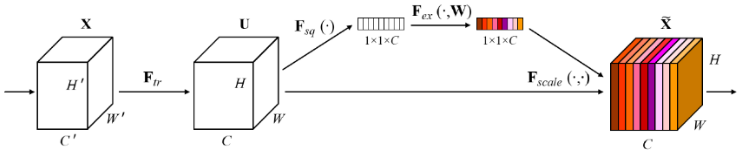
(22)

Comparing the Maximum Test Accuracy (MTA) and Average Test accuracy (ATA) of those four DenseNet structures, we find that densenet201 has the best performance on our birds classification prediction which the MTA is 0.846 and the ATA is 0.817.

5. Add Squeeze-and-Excitation (SE) Networks

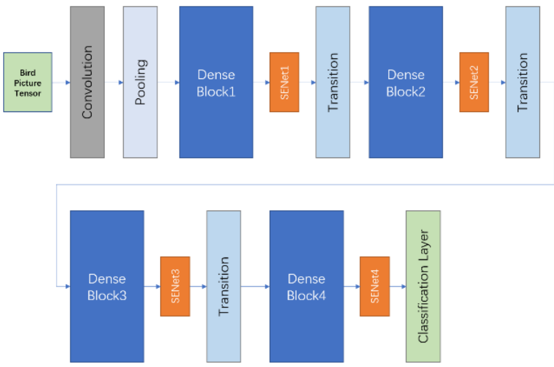
5.1 Introduction

Squeeze-and-Excitation (SE) Networks is widely used for performing feature recalibration [6] as the figure (23) shows.



(23)

We can use SE Networks on densenet201 model, adding SE Networks after each dense block as the figure (24) shows, to perform feature recalibration, reduce useless features and strength useful features.



(24)

5.2 Implement

PyTorch provide a function AdaptiveAvgPool2d() which applies a 2D adaptive average pooling over an input signal composed of several input planes. We can define our SE block by calling this function, as figure (25) shows.

```python
class SE_Block(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SE_Block, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)  # 全局自适应池化
        self.fc = nn.(
            nn.Linear(channel, channel // reduction, bias=True),
            nn.ReLU(inplace=True),
            nn.Linear(channel // reduction, channel, bias=True),
            nn.Sigmoid()
        )

    def forward(self, x):
        branch = self.avg_pool(x)
        branch = branch.view(branch.size(0), -1)

        weight = self.fc(branch.cpu())

        h, w = weight.shape
        weight = torch.reshape(weight, (h, w, 1, 1))

        scale = weight.cuda() * x
        return scale
```

(25)

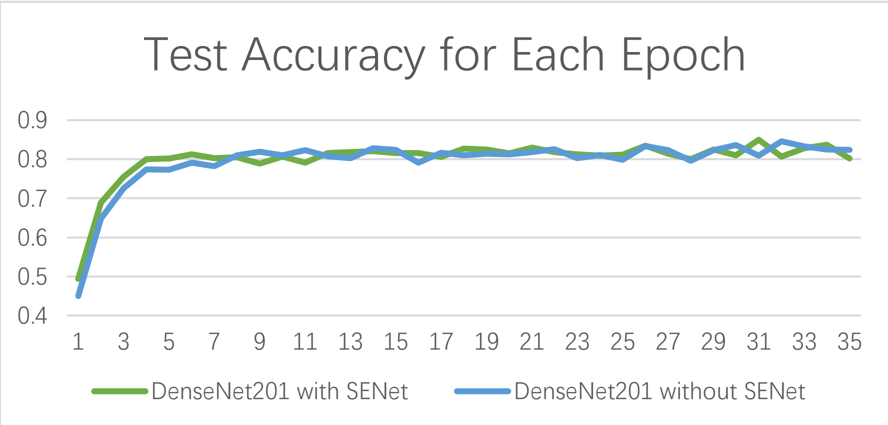
Then we can change codes of function forward() of DenseBlock, adding SE block after each dense block of densenet201 model to perform feature recalibration, as the figure (26) shows.

```python
def forward(self, init_features: Tensor) -> Tensor:
    features = [init_features]
    for name, layer in self.items():
        new_features = layer(features)
        features.append(new_features)
    all_features = torch.cat(features, 1)
    b, c, _, _ = all_features.shape
    se = SE_Block(c)
    all_features = se(all_features)
    return all_features
```

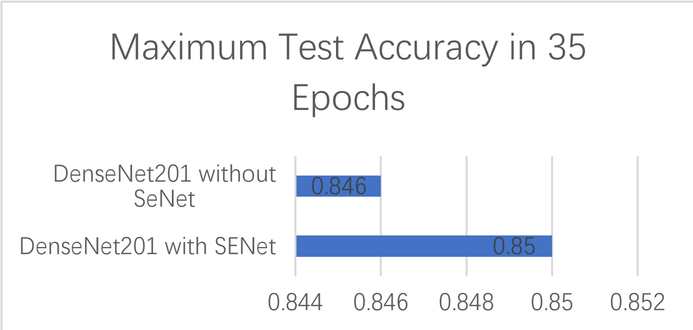
(26)

5.3 Performance Comparison

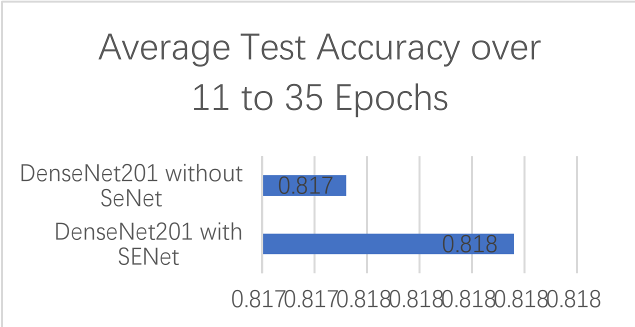
As figure (27), figure (28) and figure (29) show, after adding SE blocks, the Maximum Test Accuracy (MTA) improve to 0.850 and the Average Test Accuracy (ATA) improve to 0.818.



(27)



(28)



(29)

6. Increase training dataset by random rotation

6.1 Introduction

As we have mentioned earlier in the report, our training dataset is a small-scale dataset, which only contains about 10,000 birds images. To solve the problem, we pre-training our DenseNet model on ImageNet for feature reuse. But pre-training cannot solve everything, so we try some extra operations such as random rotation on training birds pictures, to enlarge training dataset, hoping there are more useful features which can improve the performance of densenet201 model (with SE blocks).

6.2 Implement

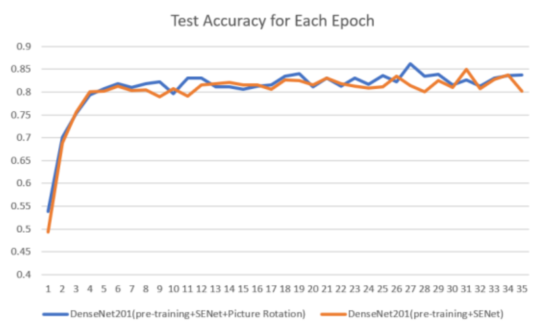
After analyzing on training images, we decide to random rotate training images by 45 angles. As the figure (30) shows, we random rotate training images, and combine old training images and rotated training images to have double-size training dataset.

```python
for data_path in train_data_path:
    pic = Image.open(data_path)
    new_train_data_species.append(train_data_species[i])
    transforms1 = transforms.Compose([
        transforms.RandomRotation(45)
    ])
    pic2 = transforms1(pic)
    new_train_data_species.append(train_data_species[i])
    np_picture = np.asarray(pic, dtype=np.uint8)
    np_picture2 = np.asarray(pic2, dtype=np.uint8)
    i += 1

    train_pic.append(np_picture)
    train_pic.append(np_picture2)
    pic.close()
```

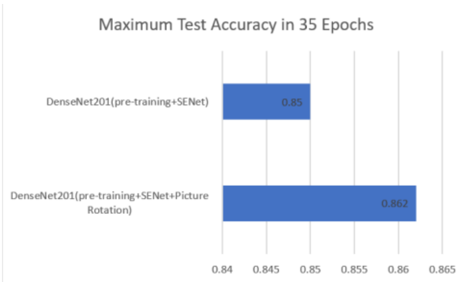
(30)

6.3 Performance Comparison

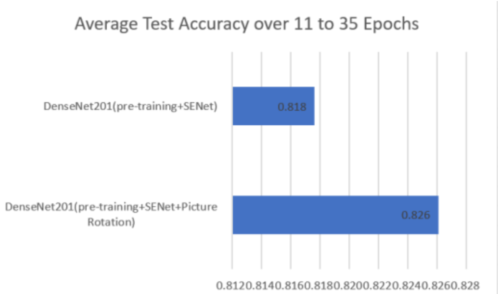
As figure (31), figure (32), figure (33) show, after random rotation on training images, the Maximum Test Accuracy (MTA) promotes to 0.862, while the Average Test Accuracy (ATA) promotes to 0.826.



(31)



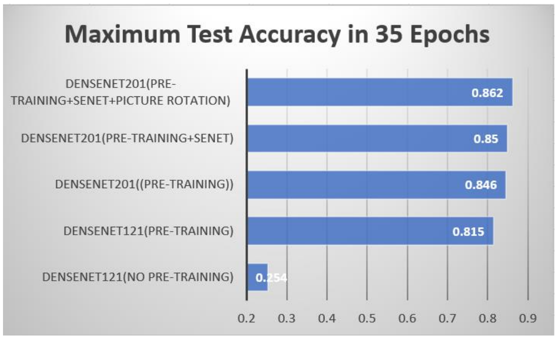
(32)



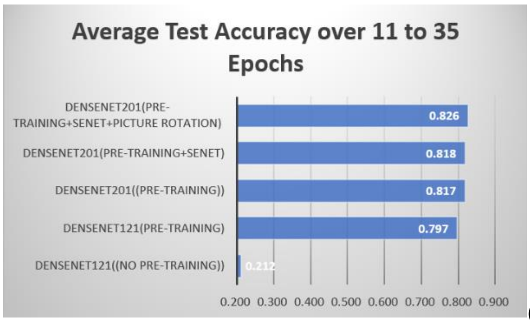
(33)

7 Conclusion

We implement our birds classification model using Dense Convolutional Network (DenseNet) in PyTorch Framework, find four ways to improve test accuracy of our model.



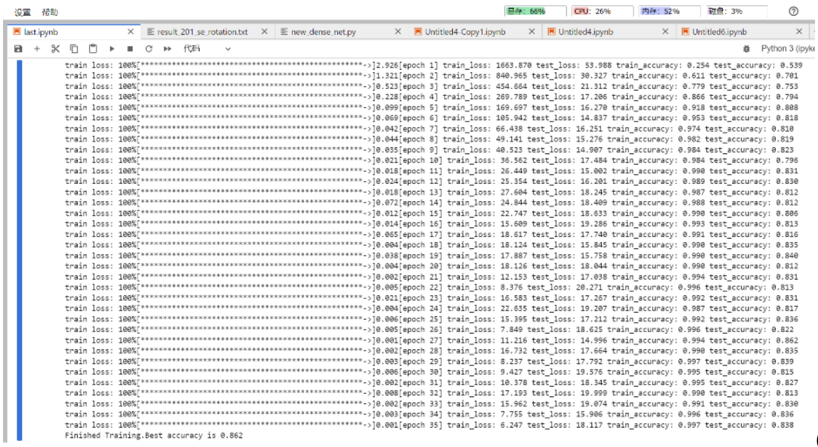
(34)



(35)

As the figure (34) and figure (35) show, each way give our DenseNet model an improvement while pre-training gives the maximum improvement.

Finally, the maximum test accuracy of our model promotes to 0.862, as the figure (36) shows.



(36)

References:

[1] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely Connected Convolutional Networks, 2017

[2] PyTorch DenseNet documentation, https://pytorch.org/hub/pytorch_vision_densenet

[3] ImageNet website, https://www.image-net.org/about.php

[4] Featurize, an online machine learning platform, https://featurize.cn

[5] Minyoung Huh, Pulkit Agrawal, Alexei A. Efros. What makes ImageNet good for transfer learning, 2016

[6] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, Enhua Wu. Squeeze-and-Excitation Networks, 2018