IEMS5709 Image and Video Processing

Group12 Final Project Report

overview

In this task, we were asked to do scene recognition on SUN Attribute dataset. This dataset contains 14340 images from 707 scene categories and each image has 102 discriminative attributes. We do several experiments and we would discuss our work in three parts: baseline model test, hyper parameters setting and DenseNet implement.

In the first section, we test the baseline model with default parameters and we also show our best model with the score and hyper parameters value in the first section directly.

In the second section, we would discuss the experiment details especially the hyper parameters setting.

In the third section, we would show the performance of DenseNet on SUN dataset.

All python matplotlib graphs can be found in the graph.ipynb.

1.baseline test

In this section, we firstly test the baseline model with default parameters and we regard the test result as the baseline of our experiment. Then we adjust the hyper parameters on it and do some simple comparsion between this two architecture. In the end of this section, we show our best model with the score and hyperparameters value.

In addition, I want to declear my standard about whether a model is good or not. We measure the model from 3 aspect: precision, recall and time cost. We would consider these 3 criterions on training set and test set respectively.

1.1 VGG-Net

1.1.1 Background

VGG-Net is a famous convolutional neural network proposed in 2015. Their main work is to prove that increasing the depth of the network can affect the final performance of the network to a certain extent. VGG has two structures, VGG16 and VGG19. There is no essential difference between them except the network depth. The network architecture is shown as Figure 1.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 1: VGG Architecture

1.1.2 Result with Default Parameters

We regard the VGG model with default parameters as the baseline model. The hyper parameters values are as Table 1:

Table 1: Default VGG Hyper Parameters

Hyper Parameters	Value
Batch Size	32
Depth	16
Epochs	20
Learning rate	0.01
Dropout	0.5
Momentum	0.9
Optimizer	SGD
Weight Decay	0.0005

The result is shown as Table 2. The Precision and Recall on the training set is picked from the last epoch and on the test set is picked the best precision from each batch epoch and also the corresponding recall. We found the result is bad and it also means the model has great potential.

Table 2: Result of Default VGG

Dataset	Precision	Recall	Time cost
Training	57.15%	47.47%	84.66min
Test	82.52%	14.90%	3.01min

I also draw down the precision and recall value via matplotlib packet in the training processing.

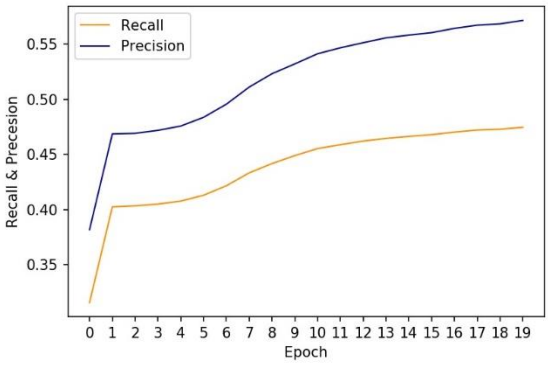

Figure 2: Result in Default VGG Training Process

We can see two strange curve. The yellow one the Recall and blue one is the precision.

1.2 Res-Net

1.2.1 Backgroud

Res-Net is proposed in 2015 by Kaiming He. The highlight of this model is their idea of "shortcut connections". Shortcut connections are those skipping one or more layers. In their case, the shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers. Identity shortcut connections add neither extra parameter nor computational complexity.

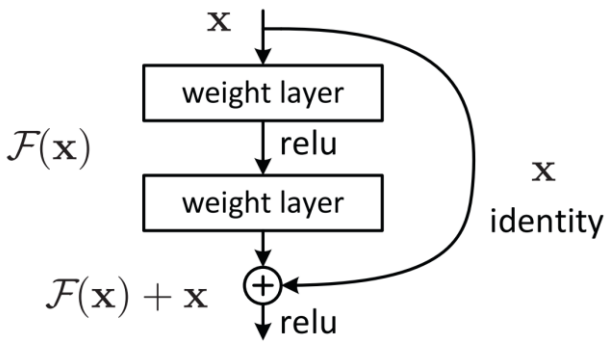

Figure3: Residential Learning

1.2.2 Result with Default Parameters

As for ResNet is also provided, we still regard the ResNet with default parameters as the baseline model. The hyper parameters values are as Table 3:

Table 3: Default ResNet Hyper Parameters

Hyper Parameters	Value
Batch Size	32
Depth	18
Epochs	20
Learning rate	0.01
Dropout	0.5
Momentum	0.9
Optimizer	SGD
Weight Decay	0.0005

The result is shown as Table 4.

Table 4: Result of Default ResNet

Dataset	Precision	Recall	Time cost
Training	61.77%	50.82%	104.46min
Test	86.69%	15.70%	3.26min

I also draw down the precision and recall value via matplotlib packet in the training processing.

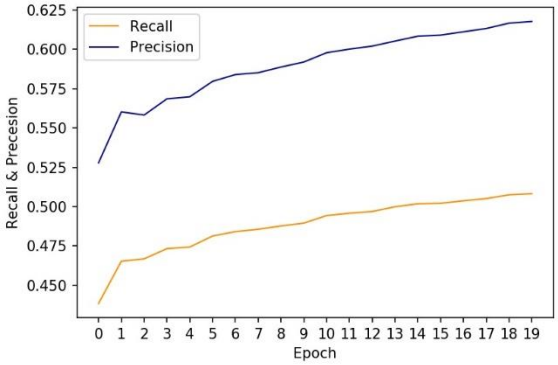

Figure 4: Result in Default RES Training Process

These two curves are more smooth than VGG one but it is still different with the common line chart. And we can also find that there is a big gap between precision and recall not matter in the VGG model or in the Res model. We infer the reason about why these 4 curves (including VGG) are strange is because the epoch we set is too short. We suppose the training for these two model is not enough. We need much more epochs to train these two model and there experiment is shown in the next part, 1.3 .

1.3 result

This part would show the best model with the hyper parameters value. Although we use the DenseNet

achieving the best score in our experiment, we still want to post the best result of VGG-Net and Res-Net because they are also perform well.

1.3.1 Best VGG and RES

In this section I would post the best parameters for VGG and ResNet respectively, the result and the comparison graph. The hyper parameters values are as Table 5:

Table 5: Best Hyper Parameters

Parameters	VGG	ResNet
Batch Size	32	32
Depth	16	50
Epochs	100	100
Learning rate	0.15	0.15
Dropout	0.5	0.5
Momentum	0.9	0.9
Optimizer	SGD	SGD
Weight Decay	0.0005	0.0005

The result is shown as Table 6.

Table 6: Result of Best VGG

Dataset	Precision	Recall	Time cost
VGGTraining	86.54%	72.36%	79.66min
VGGTest	79.65%	15.74%	0.75min
ResTraining	85.79%	71.85%	82.44min
ResTest	82.66%	18.20%	0.86min

We also draw down the training process and the visualization chart is shown as Figure5. We can see that the result VGG and ResNet are quite similar and also the converge epoch.

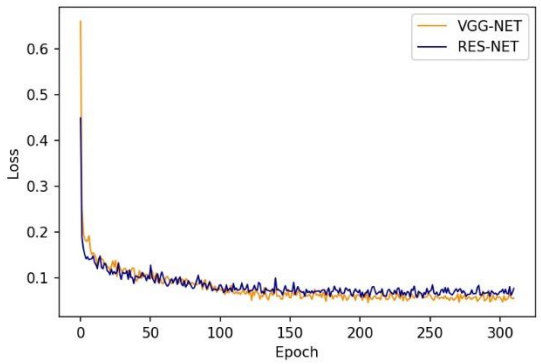

Figure 5(a): Loss Comparison

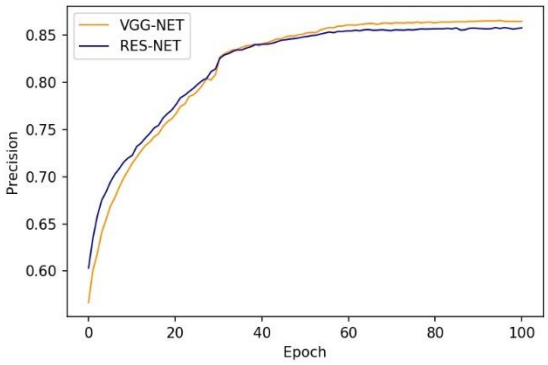

Figure 5(b): Precision Comparison

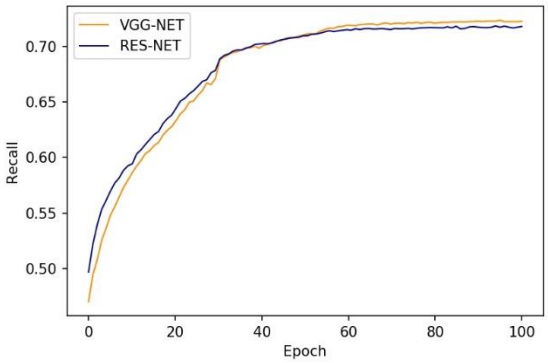

Figure 5(c): Recall Comparison

1.3.1 Best Model

We get best precision on DenseNet169. The parameters setting is shown is the tabel.

Table 3: Best Hyper Parameters

Hyper Parameters	Value
Batch Size	32
Depth	169
Epochs	80
Learning rate	0.15
Dropout	0.5
Momentum	0.9
Optimizer	SGD
Weight Decay	0.0005

The result is shown in the tabel.

Table 6: Result of Best Model

Dataset	Precision	Recall	Time cost
Training	91.02%	77.26%	94.91min
Test	94.66%	26.94%	1.24min

The change of recall and precision in training process is shown in figure6.

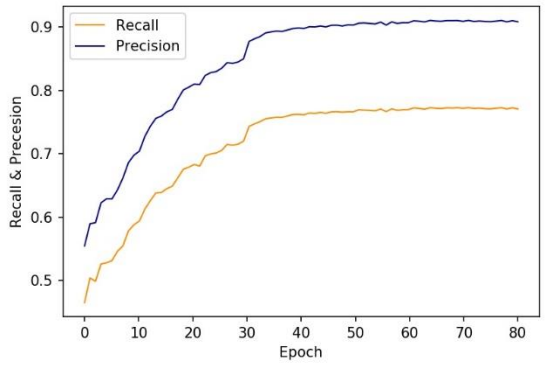

Figure 6: Best Model Training Process

2.hyper parameters

In machine learning, a hyperparameter is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training. In this section, we mainly talk about the effect of learning rate scheduler, network depth, mini-batch size and learning rate.

2.1 learning rate scheduler

In training deep networks, it is helpful to reduce the learning rate as the number of training epochs increases. This is based on the intuition that with a high learning rate, the deep learning model would possess high kinetic energy. As a result, it's parameter vector bounces around chaotically. Thus, it's unable to settle down into deeper and narrower parts of the loss function. If the learning rate, on the other hand, was very small, the system then would have low kinetic energy. Thus, it would settle down into shallow and narrower parts of the loss function. Therefore we need to adjust learning rate dynamicly. In this section I would compare the effect of two learning rate scheduler.

2.1.1 SGD with Momentum

The introduction of momentum is to speed up the learning process, especially for high curvature, small but consistent gradient, or noise gradient. The main idea of momentum is to accumulate the moving average of the prior exponential decay of gradient. The most intuitive understanding is that if the current gradient direction is consistent with the cumulative historical gradient direction, then the current gradient will be strengthened, and this step of decline will be greater. If the current gradient direction is inconsistent with the cumulative gradient direction, the current descending gradient range will be weakened

We test SGD with learning rate 0.15, momentum 0.9 and weight decay 0.005 on ResNet-50. The training and test result is shown in table. Notice that, the time cost is the 100 epoch time cost. Actually, the model has already converged in 40 epochs.

Table 7: SGD Result

Dataset	Precision	Recall	Time cost
Training	85.79%	71.85%	82.44min
Test	82.66%	18.20%	0.86min

2.1.2 Adam

The Adam algorithm is different from the traditional stochastic gradient descent algorithm. Random gradient descent keeps a single learning rate (alpha) to update all weights, and the learning rate will not change during the training process. Adam designs independent adaptive learning rates for different parameters by calculating the first and second moment estimates of gradients.

We test Adam with learning rate 0.15 and weight decay 0.0005 on ResNet-50. The training and test result is shown in table.

Table 8: Adam Result

Dataset	Precision	Recall	Time cost
Training	49.95%	41.84%	81.90min
Test	45.66%	16.70%	0.88min

The result of Adam is not good. We cannot know the reason only from the result. In the next section, I compare two learning rate scheduler and print out the training process. I suppose we can get more information from the graph.

2.1.3 Comparasion

The change of loss and precision is shown as figure7. We can find that SGD can find a better way to optimize the loss function and get a lower loss but Adam perform a little bit worse. This is the intuitive information we can get from the figure7(a) but we still do not know why they are different.

Form the figure7(b) we can easily find that the precision-epoch curve of SGD is much more smooth than Adam. Adam has a huge fluctuation in the first 50 epoch and then becomes even. We suppose because

of the large learning rate the precision changed heavily at first. And then the Adam algorithm decrease the learning rate, which the learning rate even becomes 0, so the precision do not change at the end.

We suppose if we give the Adam algorithm a better initial learning rate. It would be better. In order to confirm our assumption, we test different initial learning rate on Adam algorithm and the result is shown in the next section.

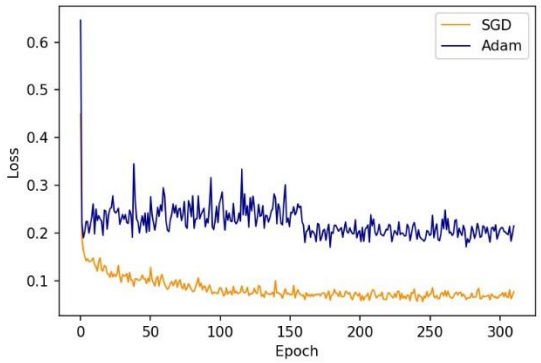

Figure 7(a): Loss Comparison

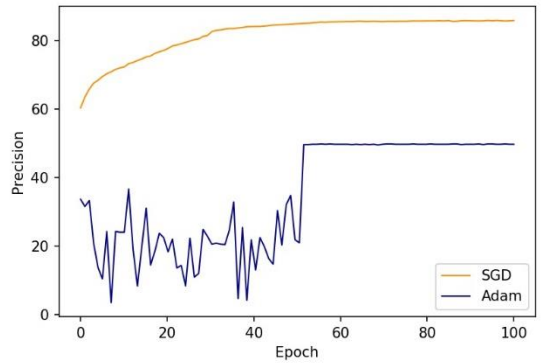

Figure 7(b): Precision Comparison

2.2 Learning Rate

We set the learning rate with 0.001, 0.01, 0.1 and 0.15 respectively on Adam and the result is shown as the table .We test every learning rate with 100 epoch. Theoretically, the time cost should not be different so much. The reason maybe because of the different GPU work loader.

Table 9: Different Learning Rate Result

lr	Precision	Recall	Time cost
0.001	61.95%	50.65%	113.06min
0.01	52.82%	44.03%	114.52min
0.1	46.53	40.15%	114.14min
0.15	49.85%	41.84%	83.92min

The change of precision is shown as figure8(a) . Nearly all precision with different learning rate change little after 50 epochs. This problem is none about the learning rate. In order to get the best model, we didn't spend too much time on "Adam". We try to find best model via SGD with momentum.

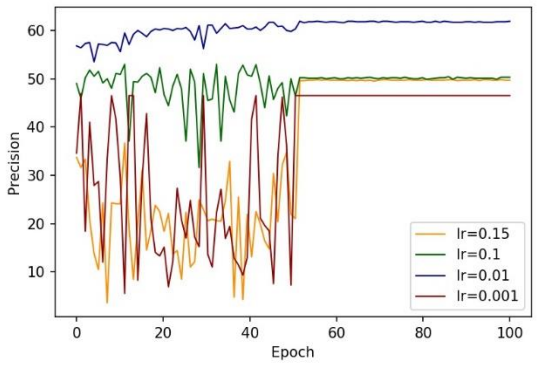

Figure 8(a): Precision on Different Learning Rate

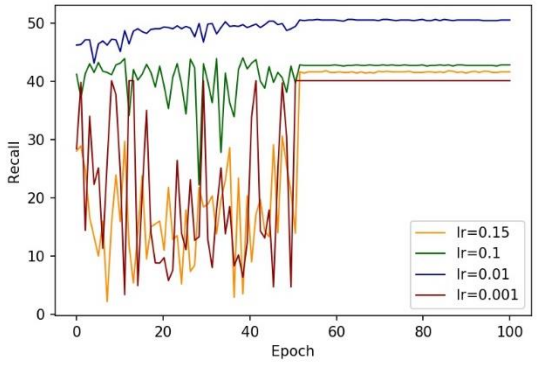

Figure 8(b): Recall on Different Learning Rate

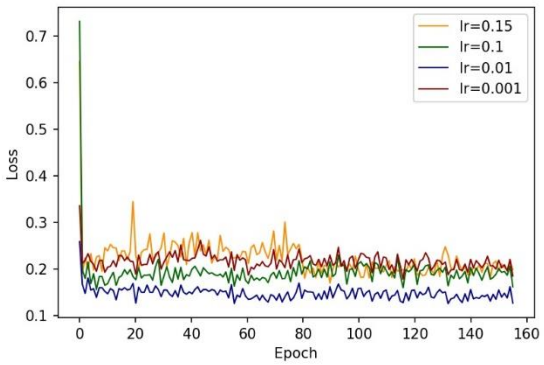

Figure 8(c): Loss on Different Learning Rate

2.3 Depth

Different depth always lead to different architecture of the model and also lead to different results. In this experiment we test different depth on Res-Net. To specific, we test Resnet18, Resnet34, Resnet50 and

Resnet152. Note that Resnet152 is not in the baseline model and it is imported from torchvision.

The experiment result is shown as table. We can see that deeper model need more time for one epoch. More comparison could be found in the figure9.

Table 10: Different depth ResNet

Dataset	Depth	Precision	Recall	Time cost
Training	18	77.77%	64.41%	89.51min
Test	18	89.23%	16.23%	0.89min
Training	34	80.42%	66.96%	124.84min
Test	34	88.63%	16.11%	1.24min
Training	50	85.70%	71.74%	168.86min
Test	50	87.19%	15.78%	1.68min
Training	101	83.45%	69.70%	104.41min
test	101	87.75%	15.81%	1.04min
Training	152	88.31%	74.12%	179.16min
test	152	86.19%	15.45%	1.79min

As the figure9 shown, actually, not deeper model could lead to better result. ResNet50 perform better than ResNet101. So in different task, we can try different deep network rather than pick the deepest one. And we should also trade off the precision and time cost, as deeper network may not lead to better result but must spend more time.

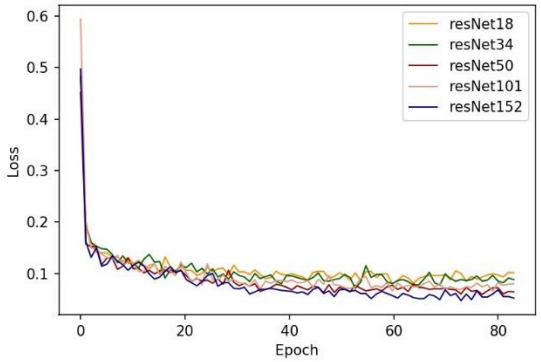

Figure 9(a): Loss on Different Depth ResNet

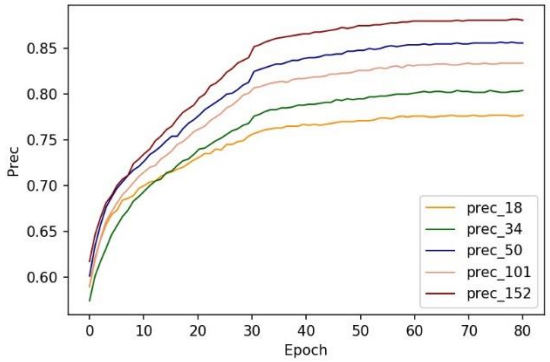

Figure 9(b): Precision on Different Depth ResNet

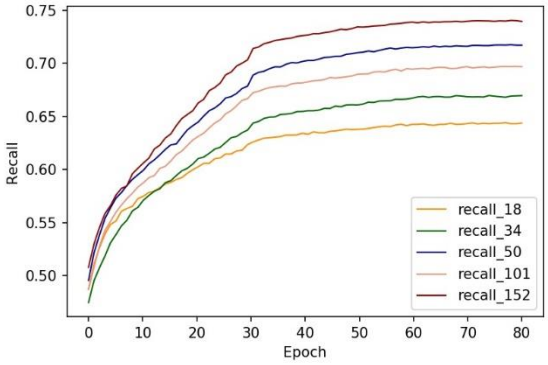

Figure 9(c): Recall on Different Depth ResNet

2.4 Batch Size

Batch gradient descent is a variation of the gradient descent algorithm that calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated. One cycle through the entire training dataset is called a training epoch.

Mini-batch sizes, commonly called “batch sizes” for brevity, are often tuned to an aspect of the computational architecture on which the implementation is being executed. Such as a power of two that fits the memory requirements of the GPU or CPU hardware like 32, 64, 128, 256, and so on.

Small values give a learning process that converges quickly at the cost of noise in the training process.

Large values give a learning process that converges slowly with accurate estimates of the error gradient.

We test different batchsize with 16, 32 and 64 on ResNet 152. The result is shown in table.

Table 11: Different Batchsize

Dataset	Size	Precision	Recall	Time cost
Training	16	87.72%	73.58%	231.03min
Test	16	86.55%	15.62%	2.31min
Training	32	88.31%	74.12%	179.16min
Test	32	86.19%	15.45%	1.79min
Training	64	85.70%	71.74%	168.86min
Test	64	87.19%	15.78%	1.68min

Although small batchsize spend less time on a single interation, it need more interation in one epoch. The time cost is too high although it achieve a good precision. We can also find the small batchsize would have large fluctuation in the figure10.

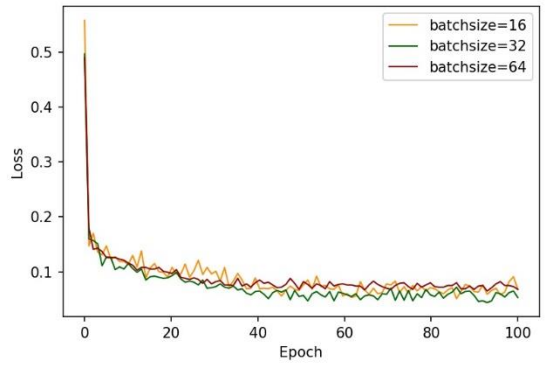

Figure 10(a): Loss on Different Batchsize

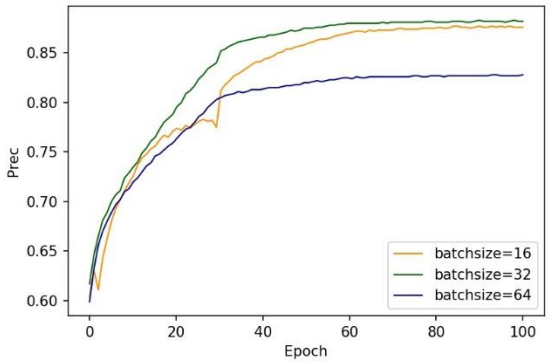

Figure 10(b): Precision on Different Batchsize

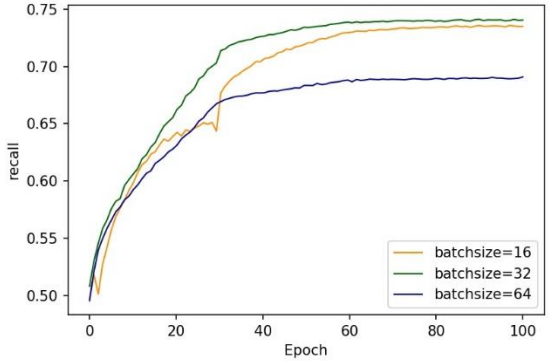

Figure 10(c): Recall on Different Batchsize

3. densenet

3.1 Background

Dense-Net is proposed in 2017 and it has several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. They propose an architecture that distills this insight into a simple connectivity pattern: to ensure maximum information flow between layers in the network, we connect all layers (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Figure 11 illustrates this layout schematically.

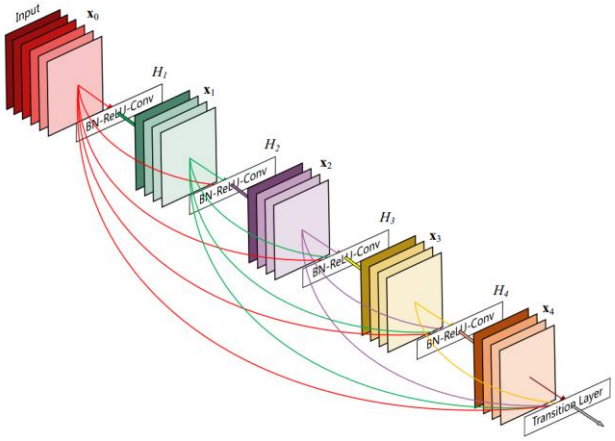

Figure11: Dense-Net Architecture

3.2 Result from Different Depth

We test different depth DenseNet and the result is shown as table. I found although DenseNet 169 has a deep architecture but its time cost is quite reasonable. So I tried different group of parameters and then get the best model which is shown in the first section.

The change of loss, precision and recall is shown in the figure12.

Table 12: Different depth DenseNet

Dataset	Depth	Precision	Recall	Time cost
Training	121	69.74%	57.28%	76.06min
Test	121	83.21%	14.88%	0.94min
Training	161	84.05%	70.00%	145.96min
Test	161	81.68%	14.50%	1.82min
Training	169	78.98%	65.34%	95.09min
Test	169	83.07%	14.87%	1.18min
Training	201	80.71%	66.93%	120.07min
test	201	82.54%	14.77%	1.50min

We shouldn't measure different DenseNet only according to the depth. Actually, not only the depth are different, but also other architecture is different, like the kernel size. DenseNet169 has deeper network but its kernel size is small and that's why it can spend less time with such deep network.

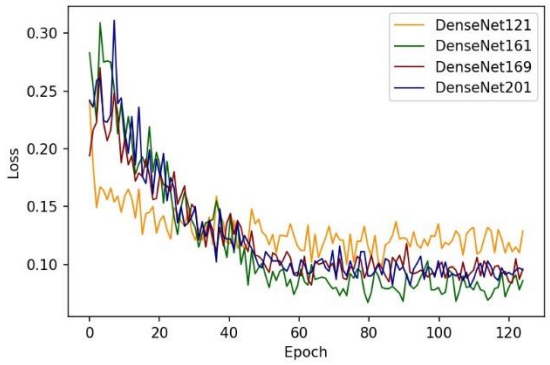

Figure 12(a): Loss on Different Depth DenseNet

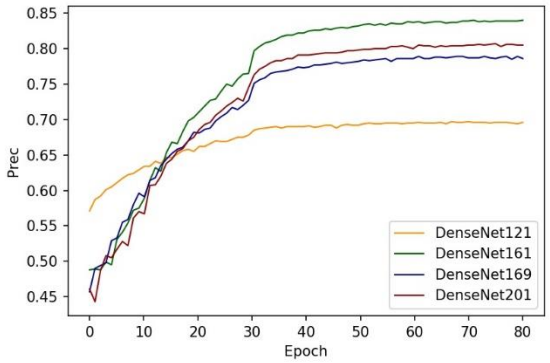

Figure 12(b): Precision on Different Depth DenseNet

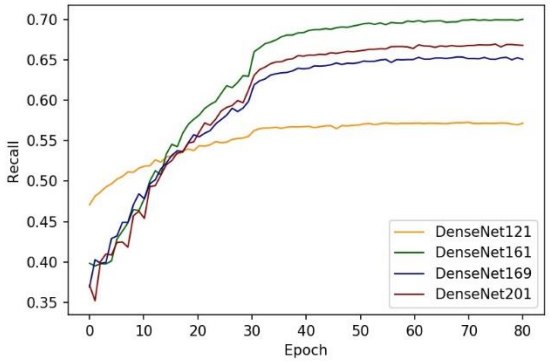

Figure 12(c): Recall on Different Depth DenseNet

Contribution

We have 3 members in our group. The work and contribution is as below:

Wang Zezhong(1155121439):

1.Finish the experiment of DenseNet.

2.Develop 2 matplotlib function to visualize the experiment result.

3.Finish the part of report corresponding the experiment.

4.Gather all members report together and finish the final report.

5.Contritbut 40% for this project.

Ye Luyao(1155118110):

1.Test 2 baseline model.

2.Find the best parameters for VGG and ResNet.

3.Finish the experiment of different depth.

4.Finish the part of report corresponding the experiment.

5.Contritbut 30% for this project.

Xu Dan(1155115261):

1.Finish the experiment of different learning rate scheduler.

2.Finish the experiment of different learning rate.

3.Finish the experiment of different batchsize

4.Finish the part of report corresponding the experiment.

5.Contritbut 30% for this project.

All members agree with the job description above.