Curtin University

COMP3010 Machine learning Assignment

Name: Polawat Srichana

ID: 20696984

Semester1, 2022

Tutor: Senjian An

Start Dates: 26-Apr-2022

Due Dates: 20-May-2022

Post Dates: 30-Jun-2022

Assignment

Polawat Srichana 20696984

Introduction

This report is the Assignment of COMP3010 machine learning, which involves studying and analyzing classifying images by examining the quality of each method of prediction and the quality of each model with different architecture after the task is done, it requires good analysis and conclusions, and therefore this report shows the results of each model and the conclusions obtained.

Task 1

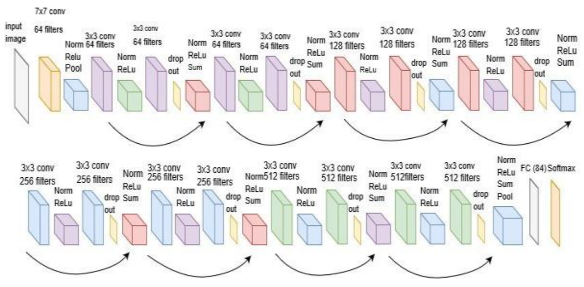
Task 1: Comparing the performance of fine tuning and training from scratch

We start by checking the quality of Training from scratch and Fine tuning. First, we need to know the meaning of Training from Scratch and Fine tuning. Training from scratch is to develop a new model of prediction and train that model with the dataset where all parameters are randomized and model will suit to dataset that we choose, and Fine tuning is the technique that use pretrained model to test the dataset, the computer will remember the weight of that model, then we can adjust parameters such as learning rate, number of epochs, regularization parameter, etc. to get the best results. The dataset we choose is the MNIST dataset. We need to know how detailed the MNIST dataset is. MNIST dataset has a trainset of 60,000 small square 28x28 pixel grayscale images with a single digit between 0 and 9, with 10 Classes, μ and σ are 0.1307, 0.3081 respectively.

Fine tuning

We start with Fine tuning, the researchers selected ResNet18, which was used for the prediction of images measuring 224x224 and 18 layers deep, by downloading the pretrained model. But since the model requires 3 channels for training and an RGB (Red, Green, Blue) pattern so we must modify MNIST dataset by repeating the channel from 1 to 3 channels.

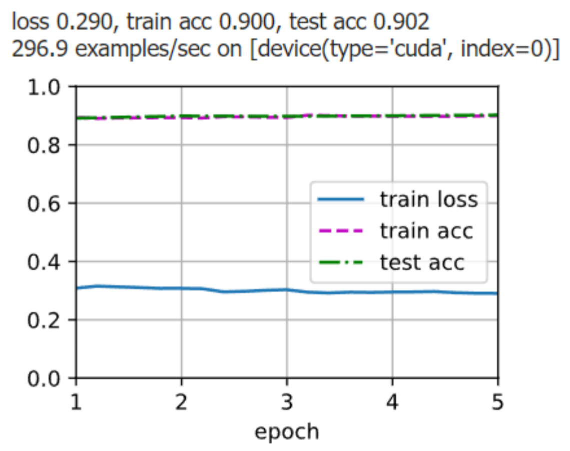
ResNet18 - Architecture



Source: https://www.researchgate.net/figure/Proposed-Modified-ResNet-18-architecture-for-Bangla-HCR-In-the-diagram-conv-stands-for_fig1_323063171

Researcher performed 10 layers of repeating channels of MNIST dataset and features. In the first step, we have to set parameters with high values and just gradually reduce its value for check if the accuracy change on better way or not so we set a batch-size of 256 and a learning rate at 5e-4 and Number of epochs at 5 times, but achieved a precision of 88.91%, which is not a satisfactory result so we tune parameters to a batch size of 128 and a learning rate at 5e-5 which are smaller parameters than the previous. The predictive model produced better results at 90.2%. We can set command as pretrained model = True so the program will remember the weights of model then we can tune all parameters until we get the best results, and it will be faster for training than the previous training.

Fine tuning Accuracy (ResNet18)

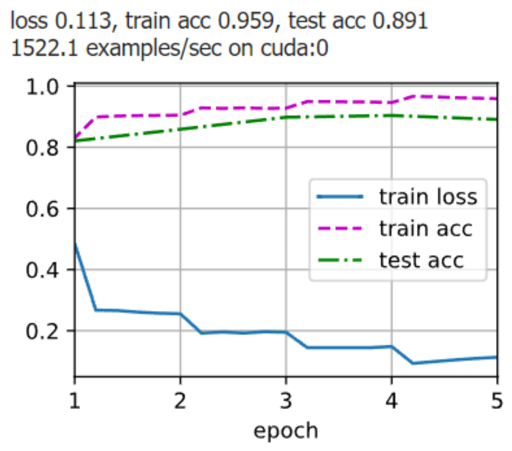


```
loss 0.290, train acc 0.900, test acc 0.902
296.9 examples/sec on [device(type='cuda', index=0)]
```

Training from scratch

Next, Training from scratch, the method should develop follow ResNet18 model based on its architecture, with 18 layers divided into 5 blocks 64,64,128,256,512, respectively, and the last layer with linear (number of features, 10) based on the number of features by parameters. All parameters were randomized because we do not know the best parameter for using it so starting with choose large parameter with a learning rate at 0.05, Number of Epochs at 5 and Batch size at 256 then we add command to multiply learning rate with 10 if training in fully connected layer, which give us a Train accuracy at 95.9% and a Test accuracy at 89.1%, where the model showed a very low Training loss at 0.113, that causes the randomization of that parameter give the appropriate value to the prediction. But in fact, researcher have low chance to random parameter and get the best result in first time so after choosing large parameters, researcher should decrease the parameter until it gives the best result. From our work, we can reduce Batch size to 32, 64 and decrease learning rate to 0.01 so it will give us better results.

Training from Scratch Accuracy (ResNet18)



```
loss 0.113, train acc 0.959, test acc 0.891
1522.1 examples/sec on cuda:0
```

Task1: Conclusion

The model that we have chosen is Fine tuning because if we observe the results, Fine tuning give better results in Train accuracy and Test accuracy than Training from Scratch, but the loss of training of Fine tuning is significantly higher (Cross Entropy loss) which, in fact downloading pretrained and set it equal to true can make computer to remember weights of that model so we can train it again with new parameters which are smaller than the first one so it will give better in accuracy at the next time of prediction. Moreover, we can use the new dataset with the model that we set pretrained = true so it will take much less time to execute compared to training from scratch which required to modify the model to suit new data set which requires computer resources, time, and money. Although, in this project, training from scratch will take less time for training.

Task 2

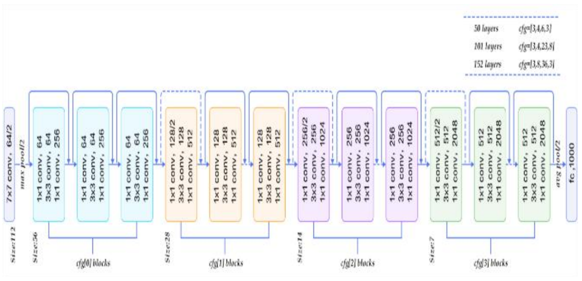
Task 2: Comparing the performance of different pretrained models

From Task 1 we selected Fine tuning as the training method for Task 2. Then we selected the rest of the models. All to compare the quality of each model. (ResNet50, AlexNet includes the following steps to prepare the data)

ResNet50

Starting with ResNet50, although we can download the pretrained model, but we should know the architecture of the model to support Parameters selection. A deep neural network called ResNet50 is a model with 50 layers and 4 blocks. Each block has a convolutional layer consisting of 3,4,6 and 3, respectively, which are integrated with Convolutional layer. The layer attached to the input data and the layer attached to the predicted data out of the model are used for extracting image characteristics. This makes the Dimension higher, so it is necessary to make a Standard Scaler to keep the images in the same standard. The principle of operation is to find a hyperplane to project images from a higher dimension down to a lower dimension hyperplane.

ResNet50 - Architecture

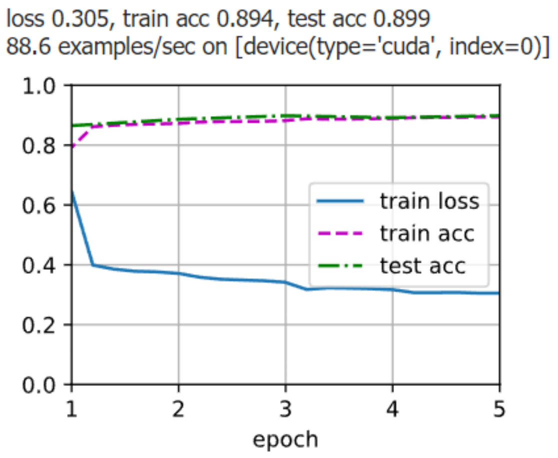


Source: https://blog.devgenius.io/resnet50-6b42934db431

Creating a model and setting the parameters

The researcher downloaded the Pretrained model (ResNet50) and set the parameters to be like ResNet18 (Task 1) so we will start with high values of parameters (Batch size = 256, Learning rate = 5e-4) and set the pretrained model as true so it will easier for Fine tuning. After that we add command to multiply learning rate with 10 when training in fully connected layer. But the high values of parameters give not satisfy accuracy with Train accuracy at 88.4% and Test accuracy at 88.5% so we gradually decrease parameters ‘values with Batch size at 128 and Learning rate at 5e-5 so it gives us better result which illustrate below.

ResNet50 Accuracy



```
loss 0.305, train acc 0.894, test acc 0.899
88.6 examples/sec on [device(type='cuda', index=0)]
```

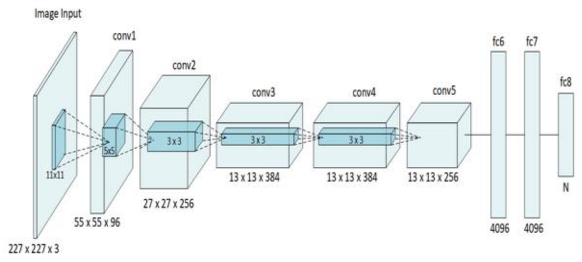
Conclusion

Although, ResNet50 will have more layer than ResNet18, but it gives us worse result than ResNet18. Even though we gradually reduce parameters, but Accuracy cannot get higher than ResNet18. Moreover, if we reduce the values of parameters to the lowest, it will give us very bad result and high of Training loss which mean model cannot predict the right picture at all so we can conclude that ResNet18 is more suitable for training MNIST dataset than ResNet50 by comparing in Test accuracy and Training loss.

AlexNet

AlexNet is the smallest neural network in comparison to all previous models. It consists of 8 convolution layers, where the first 5 are some complex layers followed by the highest convolution layer and the last three are fully connected layers, and non-linearity has been added to the network using the activation function of ReLU with a derivative of 0 when values go from 0 and 1 when positive values make gradient propagation efficient, and with dropouts as regularization, the network forgets random data and makes the model more flexible.

AlexNet -Architecture

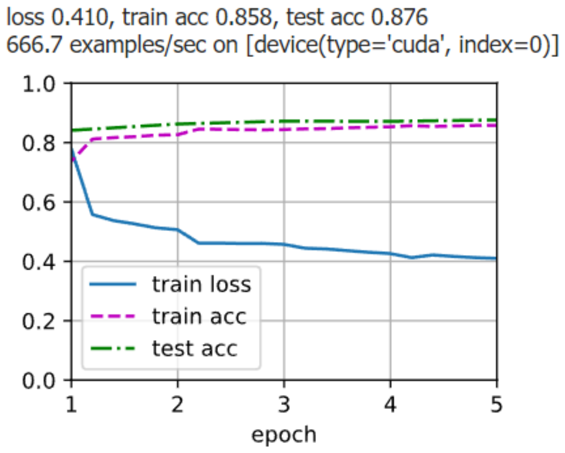


Source: https://www.researchgate.net/figure/Simplified-illustration-of-the-AlexNet-architecture_fig2_329790469

Create a model and set parameters

Due to AlexNet architecture, we should set parameter with a little bit lower than ResNet because it does not quite complex and have many layers. So, we just start with learning rate at 5e-5 so it gives us train accuracy at 87.7% and test accuracy at 82.5% but we want to maximize the performance of AlexNet so we add a greater number of epochs to see the model can increase accuracy or not. We change number of epochs from 5 to 10 epochs and use the same learning rate. Moreover, we did not add speed in learning rate with any features or classifier. Then we start training.

AlexNet Accuracy



```
loss 0.410, train acc 0.858, test acc 0.876
666.7 examples/sec on [device(type='cuda', index=0)]
```

Conclusion

Because Alexnet does not use multiple layers of smaller filters, like VGG-16, that make the model more detailed, or an identity map that helps gradient propagation due to ResNet backpropagation errors. AlexNet does not as good as all previous models in term of accuracy, but AlexNet's prediction time is lower (due to the complexity of the layer) with the training time.

Task 2: Conclusion

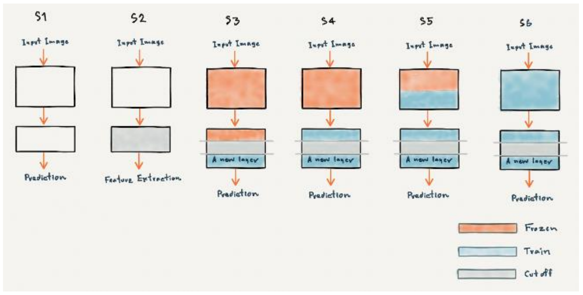
From training all three models, it turned out that ResNet18 had the best results in almost all respects: Train Accuracy, Test Accuracy, Loss Accuracy. Although, in terms of time for training; AlexNet has the lowest time for training compared to all model, but it gives us worse in term of accuracy than ResNet18. Moreover, we use Test set of MNIST data (dataset which model have never seen) to make sure that model can work in any of situation and it’s the right way to compare all model. Note that, we use Cross Entropy Loss to measure the error of model which varies according to Train accuracy; if Training accuracy increases, Training loss will decrease so we can conclude that the model which has the lowest Training loss and highest Test accuracy is the best model for MNIST dataset in Task2 (ResNet18).

Task 3

Task 3: Investigate the learnt CNN features from the last convolution layer and one intermediate layer.

We selected ResNet18 as the best model from Task 2. We must extract CNN features by Transfer learning method. Transfer learning is a technique that reduces the time spent training deep learning models by taking the weight of the model trained with the dataset in one task (Pre-trained Model) to use again instead of having to train from the beginning and we will use only focused layer (last and intermediate layer).

Transfer learning Strategies



Source: https://blog.pjjop.org/transfer-learning-with-keras-for-computer-visionapplications/?fbclid=IwAR3Vjj403Pm9GWzNA2s49ljqyS1bNXLRR5HM2hvlhGh6AciflyffjzK8zC4

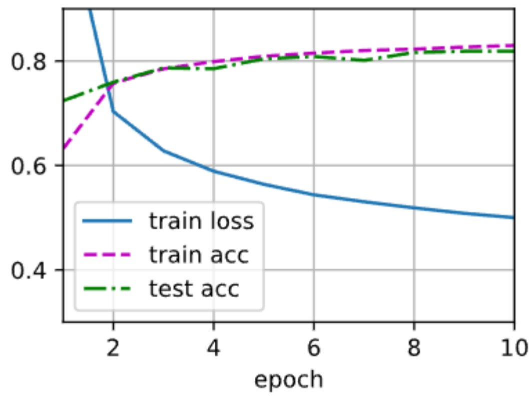
According to visualization. In S1 we train normally (Task 1, Task 2), then in S2 we cut off the top layer to apply the feature, then in S3, S4, S5 (Task 3) we freeze some layers and train some layers. Note that a new layer in visualization is SoftMax regression for training which it can provide the output as probability and calculate the Negative Log Likelihood as a Cross Entropy Loss. So, if you see in the previous task, we add Cross Entropy Loss in all models which mean we add SoftMax regression to all so it will be easy to decide which model is better but in after this session, we will add SoftMax in last layer to correct the task and use the new activation function in term of criterion.

Task3.1

Train simple models (e.g., SoftMax regression, k-nearest neighbor method) on 1) the original input

In the first step, we will put the original input of MNIST dataset, we do not have to put SoftMax regression because as previously mentioned that Cross Entropy Loss include SoftMax operation and SoftMax regression is a FC (fully connected layer), so we add FC with (784,10) which 784 mean dimensional and 10 mean channels. We add μ with 0.1307 and σ with 0.3081 to initialize the wights then we start training

(Original input with SoftMax)



Conclusion

From Accuracy, it seems not to be quite good in term of accuracy due to the original input which is not pass to any of layer and Training loss seem to be higher than input with pretrained model so that's mean original input with SoftMax does not the good way to prediction.

Task 3.2

Train simple models (e.g., SoftMax regression, k-nearest neighbor method) on the CNN features of the last convolution layer

In the second step, researcher will follow S3, freeze every layer (layer1, 2, 3, conv1, bn, fully connected layer) so required gradient of selected layer will not sent to the next layer we do not need to freeze ReLU and Max pooling because both does not contain parameter then unfreeze only last convolutional layer (layer4[0,1]) or layer before pooling layer and fully connected layer. Then we add Log SoftMax and NLLLoss for SoftMax regression. Visualization below illustrates trainable parameters of layer4 and non-trainable. Note that last convolutional layer has parameters more than previous layer, so we get 8,398,858 parameters for training and we just add SoftMax layer to the end of model and set dimension to 1 so the probabilities for each sample will sum to 1 then train it (SoftMax has 0 parameters).

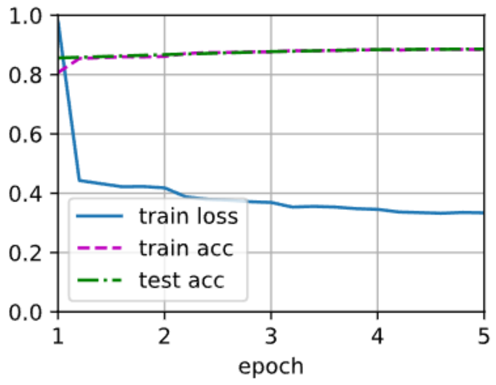
```
================================================================
Total params: 11,181,642
Trainable params: 8,398,858
Non-trainable params: 2,782,784
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 62.79
Params size (MB): 42.65
Estimated Total Size (MB): 106.01
----------------------------------------------------------------
```

After that we set the parameters same as Task 3.1 because we want to compare them which is not the same technique in Task 2 that we set parameters by gradually decrease them until we get the best results. So, we will use Batch size at 256 and Learning rate at 5e-4 then we start predicting.

ResNet18 Accuracy

(Last Convolutional layer with SoftMax)

```
loss 0.334, train acc 0.885, test acc 0.886
806.9 examples/sec on [device(type='cuda', index=0)]
```



Conclusion

From result, note that after freeze all layer, if we compare in term of accuracy with Task 1, it illustrates that it has nearly accuracy values and give a slightly inferior value. But if we compare to Task 3.1, it gives much batter in accuracy and Training loss which mean just last convolution layer can greatly increase the accuracy of predicting image and note that Training loss is only negligible which mean it has highly performance of prediction and if we look at original input with SoftMax regression, it give us high value of Training loss that mean it is not good as training with last convolutional layer in prediction with correct image.

Time (last convolutional layer at 1st training)
12 Minutes

Task 3.3

Train simple models (e.g., SoftMax regression, k-nearest neighbor method) on the CNN features of the intermediate convolutional layer

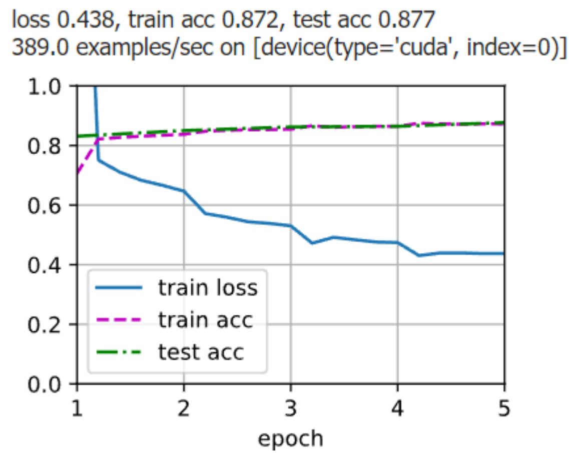
In last step, researcher will follow Task 3.2 but change last convolutional layer to intermediate convolutional layer (layer3). We hypothesize that the accuracy of training with intermediate layer will not and time for prediction is no longer than last convolutional layer training because note that parameters of intermediate layer are lower than compared to Task 3.2 (2,099,712).

```
================================================================
Total params: 11,689,512
Trainable params: 2,099,712
Non-trainable params: 9,589,800
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 62.81
Params size (MB): 44.59
Estimated Total Size (MB): 107.97
----------------------------------------------------------------
```

We use all parameters as the same of Task 3.2 because we want to know the performance of last convolutional layer compare with Task 3.1, Task 3.2 (Batch size=256, Number of epochs = 5, learning rate = 5e-4)

ResNet18 Accuracy

(Intermediate Convolutional layer with SoftMax)



```
loss 0.438, train acc 0.872, test acc 0.877
389.0 examples/sec on [device(type='cuda', index=0)]
```

Conclusion

For the result, we reject hypothesis because the accuracy of training in the intermediate convolutional layer is in criteria that exceeded expectations and require more time for training than Task 3.2. Eventhough, it has good in accuracy but note that, in term of training loss, it has high loss compared to last convolutional layer's training. The reason is that intermediate layer require less parameter than original ouput and last convolutional layer so it mean that traning with intermediate layer has more error and cannot be the choice of classify image in terms of accuracy and time for traning. However if we compare to original input with softmax, training with intermediate convolution layer give the better results in term of accuracy but for Training loss, it seems to close together with Task 3.1 which mean both Task3.1 and Task 3.3 are not good as Task 3.2 and should not be the one of choice for prediction.

Time (last convolutional layer at 1st training)	Time (intermediate convolutional layer at 1st training)
12 Minutes	22 Minutes

Task 3 : Conclusion

We have already know the highly power of Transfer learning techniques. It can reduce the time for training and give the satisfactory results, although the accuracy is slightly different from Fine tuning. But note that the matter of time is one of important factor of Machine Learnning. Imagine the pretrained model is not ResNet18 but it is VGG-16 or ResNet50 which both of them have more number of parameters than ResNet18 and have more complexity in layer, Transfer learning techniques will help a lot and moreover if we use tranfer learning with ResNet50, it can give more satisfy accuracy and training loss than ResNet18 due to more paramters values in every layer.

References

[1] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. 2022. “Dive into Deep Learning Release 0.17.5.” [page87-291] https://d2l.ai/d2l-en-pytorch.pdf

[2] Nuttachot Promrit. 2014. “Transfer Learning with Keras for Computer Vision Applications.”

https://blog.pjjop.org/transfer-learning-with-keras-for-computer-vision-applications/?fbclid=IwAR2vkxZKBAt57qLDeA5dQZSnsSFveYBJA9StPim4tKJCF8nAlVXpqmIvmc

[3] Aditi Rastogi. Mar 4, 2022. “ResNet50.” Dev Genius. https://blog.devgenius.io/resnet50-6b42934db431

[4] Azel Daniel. Sep 24, 2020. “Understanding AlexNet: A Detailed Walkthrough.” Towards Data Science. https://towardsdatascience.com/understanding-alexnet-a-detailed-walkthrough-20cd68a490aa

[5] Gaurav Singhal. May 5, 2020. “Transfer Learning with ResNet in PyTorch” https://www.pluralsight.com/guides/introduction-to-resnet

[6] Senjian An. 2022. “Machine Learning Lecture Slides and Practical Code”

[7] Pytorch. 2017. “MODELS AND PRE-TRAINED WEIGHTS” https://pytorch.org/vision/master/models.html

[8] Bijay Kumar. April 13, 2022. “Pytorch MNIST Tutorial” https://pythonguides.com/pytorch-mnist/#:~:text=PyTorch%20mnist%20is%20large%20data%20that%20is%20used,model%20and%20getting%20the%20accuracy%20of%20the%20model.

[9] Stack Exchange. Feb 25, 2019 “How mean and deviation come out with MNIST dataset?” https://datascience.stackexchange.com/questions/46228/how-mean-and-deviation-come-out-with-mnist-dataset?fbclid=IwAR2QAFWsC0vtHfIYvhbaH9zdwXufZSAfovDOnNzQFRFzBBYAYH1KZtAOOkg

[10] SUJAN DUTTA. 2019. “transfer learning CNN.” Kaggle. https://www.kaggle.com/code/orion99/transfer-learning-cnn/notebook?fbclid=IwAR2UztSndWhCqYHXzsujAnznzTtAfUpiw6t7SAcXr3dviEH9w7hZLj3gJVM

[11] Aqeel Anwar. Jun 7, 2019. “Difference between AlexNet, VGGNet, ResNet, and Inception” Towards Data Science. https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

Appendix

```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets,
transforms
import matplotlib.pyplot as plt
from torch import optim
import numpy as np
%matplotlib inline
import os
from d2l import torch as d2l
import numpy as np
import pandas as pd
from torchsummary import summary
```

ResNet18(Training from scratch)

```
class Residual(nn.Module):  #@save
    """The Residual block of ResNe
t."""
```

```
    def __init__(self, input_chann
els, num_channels,
                 use_1x1conv=False
, strides=1):
        super().__init__()
        self.conv1 = nn.Conv2d(inp
ut_channels, num_channels,
                               ker
nel_size=3, padding=1, stride=stri
des)
        self.conv2 = nn.Conv2d(num
_channels, num_channels,
                               ker
nel_size=3, padding=1)
        if use_1x1conv:
            self.conv3 = nn.Conv2d
(input_channels, num_channels,

 kernel_size=1, stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.BatchNorm2d(
num_channels)
        self.bn2 = nn.BatchNorm2d(
num_channels)

    def forward(self, X):
        Y = F.relu(self.bn1(self.c
onv1(X)))
        Y = self.bn2(self.conv2(Y)
)
        if self.conv3:
            X = self.conv3(X)
        Y += X
        return F.relu(Y)
b1 = nn.Sequential(nn.Conv2d(1, 64
, kernel_size=7, stride=2, padding
=3),
                   nn.BatchNorm2d(
64), nn.ReLU(),
                   nn.MaxPool2d(ke
rnel_size=3, stride=2, padding=1))
def resnet_block(input_channels, n
um_channels, num_residuals,
                 first_block=False
):
    blk = []
    for i in range(num_residuals):
        if i == 0 and not first_bl
ock:
            blk.append(Residual(in
put_channels, num_channels,
                                us
e_1x1conv=True, strides=2))
        else:
            blk.append(Residual(nu
m_channels, num_channels))
    return blk
b2 = nn.Sequential(*resnet_block(6
4, 64, 2, first_block=True))
b3 = nn.Sequential(*resnet_block(6
4, 128, 2))
b4 = nn.Sequential(*resnet_block(1
28, 256, 2))
b5 = nn.Sequential(*resnet_block(2
56, 512, 2))
net = nn.Sequential(b1, b2, b3, b4
, b5, nn.AdaptiveAvgPool2d((1,1)),
 nn.Flatten(), nn.Linear(512, 10))

X = torch.rand(size=(1, 1, 224, 22
4))
for layer in net:
    X = layer(X)
    print(layer.__class__.__name__
,'output shape:\t', X.shape)
```

```
lr, num_epochs, batch_size = 0.05,
 5, 256
train_iter, test_iter = d2l.load_d
ata_fashion_mnist(batch_size, resi
ze=96)
d2l.train_ch6(net, train_iter, tes
t_iter, num_epochs, lr, d2l.try_gp
u())
```

ResNet18(Fine-Tuning)

```
resnet18= torchvision.models.resne
t18(pretrained=True)
resnet18.fc = nn.Linear(resnet18.f
c.in_features, 10)
nn.init.xavier_uniform_(resnet18.f
c.weight);
X = torch.rand(size=(224, 224))
X.unsqueeze_(0)
X=X.repeat(3,1,1)
X.shape
normalize = torchvision.transforms
.Normalize(
   [0.1307], [0.3081])
train_augs = torchvision.transform
s.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])

test_augs = torchvision.transforms
.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])
trainset =torchvision.datasets.MNI
ST('~/.pytorch/MNIST_data/', train
=True, transform=train_augs, downl
oad=True)
testset=torchvision.datasets.MNIST
('~/.pytorch/MNIST_data/', train=F
alse, transform=test_augs, downloa
d=True)
trainloader = torch.utils.data.Dat
aLoader(trainset, batch_size=128,
shuffle=True)
testloader = torch.utils.data.Data
Loader(testset, batch_size=128, sh
uffle=False)
def train_fine_tuning(resnet18, le
arning_rate, batch_size=128, num_e
pochs=5,
                      param_group=
True):
    devices = d2l.try_all_gpus()
    loss = nn.CrossEntropyLoss(red
uction="none")
    if param_group:
        params_1x = [param for nam
e, param in resnet18.named_paramet
ers()
             if name not in ["fc.w
eight", "fc.bias"]]
        trainer = torch.optim.SGD(
[{'params': params_1x},

 {'params': resnet18.fc.parameters
(),

  'lr': learning_rate * 10}],
                                lr
=learning_rate, weight_decay=0.001
)
    else:
        trainer = torch.optim.SGD(
resnet18.parameters(), lr=learning
_rate,

weight_decay=0.001)
    d2l.train_ch13(resnet18, train
loader, testloader, loss, trainer,
 num_epochs,devices)
Fine_tuning =train_fine_tuning(res
net18, 5e-5)
```

ResNet50

```
resnet50 = torchvision.models.resn
et50(pretrained=True)
resnet50.fc = nn.Linear(resnet50.f
c.in_features, 10)
```

```
nn.init.xavier_uniform_(resnet50.f
c.weight);
X = torch.rand(size=(224, 224))
X.unsqueeze_(0)
X=X.repeat(3,1,1)
X.shape
normalize = torchvision.transforms
.Normalize(
   [0.1307], [0.3081])
train_augs = torchvision.transform
s.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])

test_augs = torchvision.transforms
.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])
trainset =torchvision.datasets.MNI
ST('~/.pytorch/MNIST_data/', train
=True, transform=train_augs, downl
oad=True)
testset=torchvision.datasets.MNIST
('~/.pytorch/MNIST_data/', train=F
alse, transform=test_augs, downloa
d=True)
trainloader = torch.utils.data.Dat
aLoader(trainset, batch_size=128,
shuffle=True)
testloader = torch.utils.data.Data
Loader(testset, batch_size=128, sh
uffle=False)
def train_fine_tuning(resnet50 , l
earning_rate, batch_size=128, num_
epochs=5,
                      param_group=
True):
    devices = d2l.try_all_gpus()
    loss = nn.CrossEntropyLoss(red
uction="none")
    if param_group:
        params_1x = [param for nam
e, param in resnet50 .named_parame
ters()
             if name not in ["fc.w
eight", "fc.bias"]]
        trainer = torch.optim.SGD(
[{'params': params_1x},

 {'params': resnet50 .fc.parameter
s(),

  'lr': learning_rate * 10}],
                                lr
=learning_rate, weight_decay=0.001
)
    else:
        trainer = torch.optim.SGD(
resnet50 .parameters(), lr=learnin
g_rate,

weight_decay=0.001)
    d2l.train_ch13(resnet50 , trai
nloader, testloader, loss, trainer
, num_epochs,devices)
train_fine_tuning(resnet50 , 5e-5)
```

AlexNet

```
alex_net = torchvision.models.alex
       net(pretrained=True)

alex_net.classifier[6]=nn.Linear(4
096,10)
X = torch.rand(size=(224, 224))
X.unsqueeze_(0)
X=X.repeat(3,1,1)
X.shape
normalize = torchvision.transforms
.Normalize(
   [0.1307], [0.3081])
```

```
train_augs = torchvision.transform
s.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])
test_augs = torchvision.transforms
.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])
trainset =torchvision.datasets.MNI
ST('~/.pytorch/MNIST_data/', train
=True, transform=train_augs, downl
oad=True)
testset=torchvision.datasets.MNIST
('~/.pytorch/MNIST_data/', train=F
alse, transform=test_augs, downloa
d=True)
trainloader = torch.utils.data.Dat
aLoader(trainset, batch_size=128,
shuffle=True)
testloader = torch.utils.data.Data
Loader(testset, batch_size=128, sh
uffle=False)
def train_fine_tuning(alex_net, le
arning_rate, batch_size=128, num_e
pochs=10,):
    devices = d2l.try_all_gpus()
    loss = nn.CrossEntropyLoss(red
uction="none")
    trainer = torch.optim.SGD(alex
_net.parameters(), lr=learning_rat
e,

weight_decay=0.001)
    d2l.train_ch13(alex_net, train
loader, testloader, loss, trainer,
 num_epochs,
                   devices)
```

```
train_fine_tuning(alex_net, 5e-5)
```

Original input (Softmax)

```
batch_size = 256
train_iter, test_iter = d2l.load_d
ata_fashion_mnist(batch_size)
net = nn.Sequential(nn.Flatten(),
nn.Linear(784, 10))

def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.normal_(m.weight,
mean = 0.1307, std=0.3081)

net.apply(init_weights);
loss = nn.CrossEntropyLoss(reducti
on='none')
trainer = torch.optim.SGD(net.para
meters(), lr=0.1)
num_epochs = 10
d2l.train_ch3(net, train_iter, tes
t_iter, loss, num_epochs, trainer)
```

Last convolutional layer

```
finetune_net = torchvision.models.
resnet18(pretrained=True)
for param in finetune_net.layer1.p
arameters():
    param.requires_grad = False
for param in finetune_net.layer2.p
arameters():
    param.requires_grad = False
for param in finetune_net.layer3.p
arameters():
    param.requires_grad = False
for param in finetune_net.conv1.pa
rameters():
    param.requires_grad = False
for param in finetune_net.bn1.para
meters():
    param.requires_grad = False
for param in finetune_net.fc.param
eters():
    param.requires_grad = False
```

```
net=nn.Sequential(finetune_net,nn.
LogSoftmax(1))
def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.normal_(m.weight,
mean = 0.1307, std=0.3081)
net.apply(init_weights);
net.cuda()
summary(net,(3,224,224))
normalize = torchvision.transforms
.Normalize(
   [0.1307], [0.3081])
train_augs = torchvision.transform
s.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])

test_augs = torchvision.transforms
.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])
trainset =torchvision.datasets.MNI
ST('~/.pytorch/MNIST_data/', train
=True, transform=train_augs, downl
oad=True)
testset=torchvision.datasets.MNIST
('~/.pytorch/MNIST_data/', train=F
alse, transform=test_augs, downloa
d=True)
trainloader = torch.utils.data.Dat
aLoader(trainset, batch_size=256,
shuffle=True)
testloader = torch.utils.data.Data
Loader(testset, batch_size=256, sh
uffle=False)
```

```
def train_fine_tuning(net, learnin
g_rate, batch_size=256, num_epochs
=5,
                      param_group=
True):
    devices = d2l.try_all_gpus()
    loss = nn.NLLLoss(reduction="n
one")
    trainer = torch.optim.SGD(net.
parameters(), lr=learning_rate,

weight_decay=0.001)
    d2l.train_ch13(net, trainloade
r, testloader, loss, trainer, num_
epochs,
                   devices)
train_fine_tuning(net, 5e-4)
```

Intermediate convolutional layer

```
finetune_net = torchvision.models.
resnet18(pretrained=True)
for param in finetune_net.layer1.p
arameters():
    param.requires_grad = False
for param in finetune_net.layer2.p
arameters():
    param.requires_grad = False
for param in finetune_net.layer4.p
arameters():
    param.requires_grad = False
for param in finetune_net.conv1.pa
rameters():
    param.requires_grad = False
for param in finetune_net.bn1.para
meters():
    param.requires_grad = False
for param in finetune_net.fc.param
eters():
    param.requires_grad = False
net1=nn.Sequential(finetune_net,nn
.LogSoftmax(1))
def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.normal_(m.weight,
mean = 0.1307, std=0.3081)
net1.apply(init_weights);
net1.cuda()
summary(net1,(3,224,224))
summary(net1,(3,224,224))
normalize = torchvision.transforms
.Normalize(
   [0.1307], [0.3081])
train_augs = torchvision.transform
s.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])

test_augs = torchvision.transforms
.Compose([
    torchvision.transforms.RandomR
esizedCrop(224),
    torchvision.transforms.ToTenso
r(),torchvision.transforms.Lambda(
lambda X:X.repeat(3,1,1)),
    normalize])
trainset =torchvision.datasets.MNI
ST('~/.pytorch/MNIST_data/', train
=True, transform=train_augs, downl
oad=True)
testset=torchvision.datasets.MNIST
('~/.pytorch/MNIST_data/', train=F
alse, transform=test_augs, downloa
d=True)
trainloader = torch.utils.data.Dat
aLoader(trainset, batch_size=256,
shuffle=True)
testloader = torch.utils.data.Data
Loader(testset, batch_size=256, sh
uffle=False)
def train_fine_tuning(net1, learni
ng_rate, batch_size=256, num_epoch
s=5,
                      param_group=
True):
    devices = d2l.try_all_gpus()
    loss = nn.NLLLoss(reduction="n
one")
    trainer = torch.optim.SGD(net1
.parameters(), lr=learning_rate,

weight_decay=0.001)
    d2l.train_ch13(net1, trainload
er, testloader, loss, trainer, num
_epochs,
                   devices)
train_fine_tuning(net1, 5e-4)
```