

Assignment 2

by Yizhen Ma

FILE	25400_YIZHEN_MA_ASSIGNMENT_2_316598_1287483856.PDF (827.52K)		
TIME SUBMITTED	18-JAN-2021 04:57PM (UTC+0000)	WORD COUNT	5239
SUBMISSION ID	142581558	CHARACTER COUNT	28996

Facial Expression Recognition with ResNet

Yizhen Ma(2006703)^{1,2}

1.University of Wales Trinity St David, Swansea, Britain; 2.Wuhan University of Technology, Wuhan, China

Abstract: We present a method that take advantage of DNNs and CNNs for facial expression recognition task. Based on the given facial expression dataset, we employ some data preprocessing techniques and obtain a subset of 4 expression classes. After reviewing several mainstream neural networks, we select ResNet which is relatively suitable for this task. To evaluate the performance of our model, we employ some common evaluation metrics for classification task, including classification accuracy and confusion matrix. Our method achieves 85% top-1 accuracy in identifying specific expression, and achieves nearly 68% top-1 accuracy on average. Further experimental results demonstrate that our method can solve the problem of facial expression recognition to a certain extent.

Key words: face expression recognition; DNNs and CNNs ; ResNet

1. Introduction

Facial expressions serve as emotional indicators projected from the appearance of humans and other animals. Facial expressions play an important role in social occasions. They can not only convey a person's mental outlook, but also reflect a person's feelings about the current situation. Facial expression identification has been widely used in various fields, industrial scenarios such as polygraph detection, user experience detection, and academic scenarios such as face reenactment, etc.

In 1971, Ekman and Friesen[8] made a groundbreaking contribution to facial expression recognition. They explore the 6 basic facial expressions such as sad, fear and disgust and determined the classes of recognition object. They proposed a facial expression database which established the basis for systematically study of facial expression, even the facial changes corresponding to each expression are detailly described.

In 1978, Suwa et al.[9] made an initial attempt of expression recognition researched on the scene of facial video animation. In their research, a set of automated analysis methods is proposed for facial expression image sequences.

Beginning in the 1990s, K. Mase and A. Pentland[10] used optical flow to determine the main direction of muscle movement. After using the proposed optical flow method for facial expression recognition, automatic facial expression recognition entered a new era.

In this paper, we explore utilizing CNNs to solve the problem of facial expression recognition. The structure of this paper is as follows, In the first section, we review the key development of CNNs and DNNs, based on the research content, we proposed a method utilizing CNNs and DNNs to recognize face expression, as discussed in the second section. In the third section, we evaluate our model utilizing some common metrics for classification task. In the last section, we comment our final model and suggests possible future direction of improvements.

2. Related work

Lecun et al. first proposed LeNet[1] to solve the problem of handwritten digit recognition, which serves as the originator of the CNN network structure and offer a definition of CNN network structure for the first time, their contribution lies in defining the basic paradigm of CNN, i.e. a CNN is composed of several convolutional layers, pooling layers and fully connected layers and also two core ideas of convolution layer, that is local connection (also introducing the concept of receptive field) and weight sharing. After that, Krizhevsky et al. designed a network called

AlexNet[2] to solve the image classification problem on ImageNet[3] dataset, the typical feature of AlexNet is replacing tanh activation function with relu activation function which effectively solve the problem of gradient vanishing, they also adopt dropout technique to avoid network to be over-fitting. Simonyan and Zisserman[4] presented VGG series model, which explore further deeper neural network architecture. In their work, they innovatively proposed using multiple small convolution kernels instead large convolution kernels, they points out this can reduce the amount of network parameters and also increasing the nonlinear expression ability of the network due to the usage of multiple small kernels. In split of a series of small convolution kernels reduce the amount of model parameters to a certain extent, the overall size of the VGG series model is still maintained at a very high level. During that period, many researches believe the most direct way to improve the expressive ability of neural networks is to increase the size of the model[5], but such method leads to two problems: the larger the model is, the easier it is to be over-fitting; the larger the model is, the greater demand of computing resources is required, which is unacceptable in reality. The fact is, when a neural network reach a certain depth, even if we deepen the layers of it, the fitting ability of the network will not be improved, and the result is even worse compared with shallow network. He et al. proposed ResNet[6] which make use of residual block to solve the degradation problem in very deep neural network, the difference between residual block and traditional convolution block is the effective use of shortcut connection which makes deep neural network much easier to optimize and converge, the ResNet architecture can be expanded to much deep architectures such as ResNet-18, ResNet-50, and ResNet-152, these models enhance the expression ability of network while ensure the network will not degenerate.

3. Method

3.1 dataset

We obtain an image dataset of different facial expressions, namely angry, disgust, fear, happy, neutral, sad and surprise. Our task is to utilizing these images to train a deep neural network and make prediction of facial expression of any face image.

According to the requirements of the task, we select 4 types of expression images(“angry”, “fear”, “happy”, “sad”) to train our neural network. The reason for choosing these four classes are: Above all, we did not choose the class of the “disgust” because it only contain about 400 face images, which far from sufficient to train a deep neural networks, at the same time, the number of images in this class is too less compared with other categories(averagely 4000 images). We do not choose the “neutral” class because the definition of neutral expression is kind of ambiguous which may make an impact on the performance of our model. For the reason of abandon “surprise” class, we suppose the images of this class share many similarities with other chosen classes such as “happy” and “fear”, many of face images in these classes all contain the characteristic of an open mouth, which may also influence the classification performance of our model. In summary, we choose images of four classes, namely “angry”, “fear”, “happy”, “sad” because they have similar amounts of data and have their independent image characteristics. Since all input image has already been converted to grayscale and the resolution of each image is relatively small(48×48), we do not need to do additional preprocessing steps for each image such as graying and image resizing, except for normalization of the training data, since human faces occupy most of the image in the dataset, so other preprocessing techniques such as face recognition and image cropping are also needless.

We notice there exists a few bad data of these classes, but due to it is too much

time-consuming to find and pretty hard to identify all of them, and also because the amount of bad data is relatively small, so it will not have too much impact on the performance of our model. So we didn't do other special treatment for these bad data. Another thing worth mentioned is that the original dataset only divide images in two group, which are "training" and "testing", so it does not meet the requirement of this assignment. We divide the training data(about 4000 per class) at the ratio of 3:1, which lead to 3000 expression images for training and 1000 images for validation per class.

3.2 implementation details

We adopt the concept of transfer learning and choose ResNet as the backbone of our network which also serves as the image feature extractor. ResNet was trained on large image classification task whose task is quite similar to our job, through which will provide a good start point of our model , and at the same time, it has proved the ability to extract structural information from images. The difference is that we replace the last fully connected layer in original ResNet to adapt to our specific task and train on our expression dataset, which needs to be explained is that we do not freeze the parameters of ResNet backbone during training stage because we suppose the feature extraction process of facial expression classification and object recognition is different, so what we do is fine-tuning the model to adapt to our specific task. For comparison, we select 3 pretrained version of ResNet, namely ResNet-18, ResNet-34 and ResNet-50 as our backbone respectively to test the change in performance caused by the number of layers, due to the fact that all version of ResNet adopt 1×1 convolution layers instead fully connected layers, which make them able to be applied to difference resolution of images. More detailed information and difference of these architectures are showed in Figure 1.

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

Figure 1. architectural design of ResNet series model

As for training, we adopt cross entropy loss as the performance measure since our task is a multi-classification problem. The optimization algorithm is stochastic gradient descent with a momentum of 0.9. We have retained most of the network structure of ResNet, which means all activation function are relu function, all normalization layers are batch normalization and all pooling layers are maxpooling layers. We train our model for 50 epochs respectively, since we utilize the pretrained version of ResNet, it is needless to reinitialize the parameters of our model. The initial learning rate is set to 0.001 and we adopt learning rate decay method to help the network converge better.

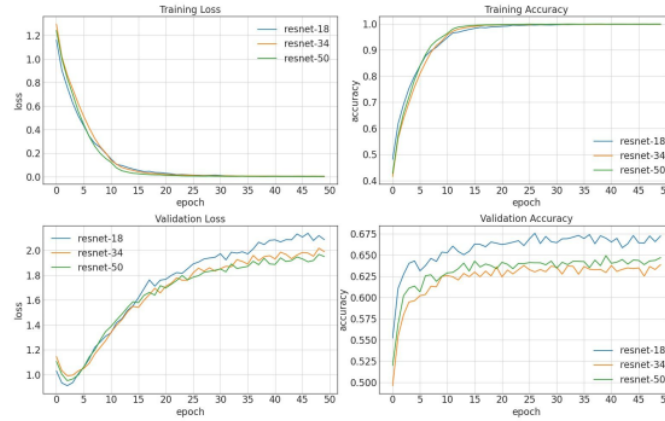


Figure 2. classification performance of different ResNet model

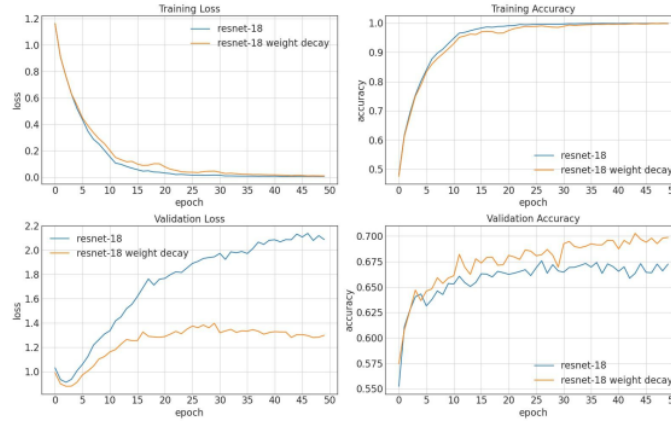


Figure 3. before and after adding a regularization term, the performance of our model

As showed in Figure 2, our model converge very fast, after about 15 epochs, the performance of our model tend to be stable. Compared with model utilize ResNet-34 and ResNet-50 as backbone, There is no much difference in the convergence speed, the model which based on ResNet-18 converge a little faster, but it achieve higher accuracy throughout the training process. We suppose this may be caused by over-fitting, the network with more layers owns more powerful expressive ability, at the same time, more easier to be over-fitting. We observed a phenomenon that with the number of epochs increases, the validation error of the network increases but the accuracy is still improving, according to[7], this may caused by over-fitting or inconsistent training and validation data distribution(or the existence of bad data), that is, in the later stage of training, the predicted results tend to be extreme, so that a small number of wrongly predicted samples dominate the loss, but at the same time a small number of samples do not affect the overall validation accuracy. In order to judge the influence of these two factors, namely bad data or over-fitting on the result, we add a L2 regularization term in the loss function and adopt ResNet-18 for testing, the result can be seen in Figure 3. From the graph, we can draw the conclusion that the existence of bad data do have an impact on the validation stage of our model since after regularization, the validation error still maintain on a relatively high level, and

over-fitting also an important factor of this, by adding a regularization term in the loss function, we achieve a lower validation loss and a high validation accuracy which improve the performance of our model.

4. Evaluation

We evaluate our model which based on ResNet-50 as the backbone with a L2 regularization term. We select two methods to evaluate our model.

4.1 classification report

The first method is using classification report in sklearn. It contains four metrics to evaluate the classification result of our model, namely precision, recall, f1-score and support.

Table 1. classification report of testing

	precision	recall	f1-score	support
angry	0.597374179	0.56993737	0.583333333	958
fear	0.596933187	0.532226563	0.562725865	1024
happy	0.85331144	0.878804961	0.865870592	1774
sad	0.575982209	0.623095429	0.598613251	1247
accuracy	0.684989007	0.684989007	0.684989007	0.684989007
macro avg	0.655900254	0.65101608	0.65263576	5003
weighted avg	0.682704049	0.684989007	0.68310809	5003

From Table 1, we can draw the conclusion that, for “happy” class, we achieve a very high classification accuracy at 85%, while for other classes, our model achieve relatively good results, the average accuracy is nearly 60% and the total accuracy achieve 68% for our model. Compared with the best accuracy of our model at validation stage, which is nearly 70%, this indicates that our model owns sufficient generalization ability.

4.2 confusion matrix

confusion matrix usually used as a method to visualize the accuracy of classification tasks, it describe the classification result with the ground-truth value and predicted value.

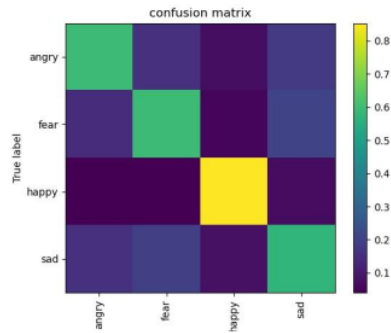


Figure 4. confusion matrix of testing

As show in figure 4, our model achieve a very high accuracy of facial expression “happy” at almost 80%, but achieve a relatively low accuracy of other expressions for example “sad” at nearly 60%. Compared between the result of “sad” and “fear”, we find that they are easily mismatch by our model. We empirically believe that these two expressions share a certain similarity of facial features, which leads to our model not being able to distinguish well. In

addition, for “happy” class, we believe it has a higher degree of classification recognition compared with other expressions, or features that are much easier to recognize.

5. Conclusion

We introduced a face expression recognition method based on ResNet. We compared between different ResNet based architecture and choose a relatively better architecture to complete our task. Some evaluation methods show that our model can finish the task of facial expression recognition to some extent.

Although our method has achieve a relatively good result on the task of facial expression recognition, there are still some problems need to be solved. First, there are significant differences in classification accuracy for different expression, For example, the accuracy of “happy” expression is much higher than others, the major reason of this may be that the levels of difficulty of different expressions are different. Since expressions are our subjectively definition, for deep learning models, the complexity of different expressions in images is very different. Therefore, it is difficult for the model to learn a unified feature representation for face expressions. Another problem need to be considered is that the performance of our model is not good enough, this may caused by the great differences among our training and test images, including identity differences, head pose differences, etc. This increases the difficulty of learning a unified model to recognize these expressions. In order to achieve better accuracy, we need to eliminate the influence of these differences. Ways that can be considered include: training on a data set of several specific persons with multiple expressions; using disentanglement methods to eliminate the identity information of the characters in the image, while only retrain expression-related information; use other normalization methods to transfer face images of different pose to frontal images, and then perform expression recognition.

References

- [1] Lecun Y , Bottou L . Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278-2324.
- [2] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.
- [3] Deng J , Dong W , Socher R , et al. ImageNet: A large-scale hierarchical image database[C]// IEEE Conference on Computer Vision & Pattern Recognition. IEEE, 2009.
- [4] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.
- [5] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.
- [6] He K , Zhang X , Ren S , et al. Deep Residual Learning for Image Recognition[C]// IEEE Conference on Computer Vision & Pattern Recognition. IEEE Computer Society, 2016.
- [7] Ishida T, Yamane I, Sakai T, et al. Do We Need Zero Training Loss After Achieving Zero Training Error?[J]. arXiv preprint arXiv:2002.08709, 2020.
- [8] Ekman P , Friesen W V , Ellsworth P . - Emotion in the Human Face[J]. Emotion in the Human Face, 1972:181 – 187.
- [9] Sown M. A preliminary note on pattern recognition of facial emotional expression[C]//The 4th International Joint Conferences on Pattern Recognition, 1978. 1978.
- [10] Clarkson B, Mase K, Pentland A. Recognizing user context via wearable sensors[C]//Digest of papers. Fourth international symposium on wearable computers. IEEE, 2000: 69-75.

Appendix

main.py

```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

from torch.utils.data import DataLoader
from tensorboardX import SummaryWriter

# plt.ion()

import ssl

# visualize a few images
def imshow(input: torch.Tensor, title=None):
    input_np = input.numpy().transpose((1, 2, 0))
    # unnormalize data
    mean = np.array([0.5, 0.5, 0.5])
    std = np.array([0.5, 0.5, 0.5])

    input_np = std * input_np + mean
    input_np = np.clip(input_np, 0, 1)

    plt.imshow(input_np)
    plt.axis('off')
    if title is not None:
        plt.title(title)
    # plt.pause(0.001) or
    plt.savefig('data/sample.jpg')

def summary_model(model):
```

```

print('----- Networks initialized -----')
num_params = 0
for param in model.parameters():
    # numel(): Returns the total number of elements in the input tensor
    num_params += param.numel()
print('[Network] Total number of parameters : %.3fM' % (num_params / 1e6))
print('-----')

ssl._create_default_https_context = ssl._create_unverified_context

writer = SummaryWriter('runs/train_resnet50_weight_decay_summary_1')

# 1.data process
# data augment and normalization for training
data_transfroms = {
    'train': transforms.Compose([
        # data augment
        # since the image is too small, it's not suitable to use any crop method
        transforms.RandomHorizontalFlip(),
        # normalization
        # totensor:[0,255] -> [0,1]
        transforms.ToTensor(),
        # [0,1]->[-1,1]
        transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ]),
    'val': transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ]),
    'test': transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])
}

# dataloaders
data_dir = "./data"
stages = ['train', 'val', 'test']
train_set = datasets.ImageFolder(os.path.join(data_dir, 'train'), data_transfroms['train'])
test_set = datasets.ImageFolder(os.path.join(data_dir, 'test'), data_transfroms['test'])
class_names = train_set.classes

```

```

# since the original splits don't contain a validate dataset,
# we split a val dataset from trainset at the ratio of 3:1
train_set_len = int(len(train_set) * 0.75)
val_set_len = len(train_set) - train_set_len

train_set, val_set = torch.utils.data.random_split(train_set, [train_set_len, val_set_len])
image_datasets = {'train': train_set, 'test': test_set, 'val': val_set}

dataset_sizes = {x: len(image_datasets[x]) for x in stages}
dataloaders = {x: DataLoader(image_datasets[x], batch_size=64, shuffle=True, num_workers=8)
                for x in stages}

# dataset summary
# print('dataset sizes:', dataset_sizes)
#
#   train_label_statistic,   val_label_statistic,   test_label_statistic   =   [[0   for   _   in
range(len(class_names))] for _ in
#
#                                     range(3)]
#   inputs_train,   inputs_val,   inputs_test   =   image_datasets['train'],   image_datasets['val'],
image_datasets['test']
#
# for input in inputs_train:
#     label = input[1]
#     train_label_statistic[label] += 1
# for input in inputs_val:
#     label = input[1]
#     val_label_statistic[label] += 1
# for input in inputs_test:
#     label = input[1]
#     test_label_statistic[label] += 1
#
# train_set_summary = {x: train_label_statistic[class_names.index(x)] for x in class_names}
# val_set_summary = {x: val_label_statistic[class_names.index(x)] for x in class_names}
# test_set_summary = {x: test_label_statistic[class_names.index(x)] for x in class_names}
#
# print('-----dataset summary-----')
# print('train set distribution:', train_set_summary)
# print('val set distribution:', val_set_summary)
# print('test set distribution:', test_set_summary)

# show sample images
# inputs, classes = next(iter(dataloaders['train']))

```

```

# out = torchvision.utils.make_grid(inputs)
# imshow(out, title=[class_names[x] for x in classes])

def train_model(model: nn.Module, criterion, optimizer, scheduler, num_epochs=20):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                # set model to training mode
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                # forward
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    # [4]:[1,1,1,1] label:[1,0,0,0]
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels.long())

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

```

```

        if phase == 'train':
            # 对 lr 进行调整
            scheduler.step()

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

        print('{} loss : {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))

        # tensorboard
        writer.add_scalar('{} loss'.format(phase), epoch_loss, epoch)
        writer.add_scalar('{} acc'.format(phase), epoch_acc, epoch)

        # deep copy the model
        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60
    ))

    print('Best val Acc : {:.4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model

# train
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print('run on device:', device)

model_ft = models.resnet50(pretrained=True)

num_fts = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_fts, len(class_names))

# model summary
summary_model(model_ft)

model_ft = model_ft.to(device)

```

```

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_ft.parameters(), lr=1e-3, momentum=0.9, weight_decay=0.01)

# learning rate decay by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.8)

model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler, 50)
model_name = 'best_model_ft_resnet50_weight_decay.pkl'
torch.save(model_ft, os.path.join('checkpoints', model_name))
print('model save to ', os.path.join('checkpoints', model_name))

# evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import pandas

def plot_confusion_matrix(cm, labels_name, title):
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest')
    plt.title(title)
    plt.colorbar()
    num_local = np.array(range(len(labels_name)))
    plt.xticks(num_local, labels_name, rotation=90)
    plt.yticks(num_local, labels_name)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig('output/confusion_matrix.jpg')

model_ft = torch.load(os.path.join('checkpoints', model_name))
model_ft.eval()

with torch.no_grad():
    all_preds = torch.Tensor([]).to(device)
    all_labels = torch.Tensor([]).to(device)
    for batch in dataloaders['test']:
        images, labels = batch
        images = images.to(device)
        labels = labels.to(device)
        preds = model_ft(images)
        all_preds = torch.cat((all_preds, preds), dim=0)
        all_labels = torch.cat((all_labels, labels), dim=0)

```

```
all_preds = np.argmax(all_preds.cpu().numpy(), 1)
all_labels = all_labels.cpu().numpy()
cm = confusion_matrix(all_preds, all_labels, )

labels_name = list(image_datasets['test'].class_to_idx.keys())
plot_confusion_matrix(cm, labels_name, "confusion matrix")

report = classification_report(all_labels, all_preds, target_names=labels_name, output_dict=True)
print(report)

df = pandas.DataFrame(report).transpose()
df.to_excel('./output/report.xlsx', sheet_name='Sheet1')
```

visualization.py

```
import torch
from tensorboard.backend.event_processing import event_accumulator
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

def visulization_model_selection():
    warnings.filterwarnings(action='once')

    # load event data
    event_resnet18 = event_accumulator.EventAccumulator(
        'runs/train_resnet18_summary_1/events.out.tfevents.1610951504.ANIMT-SERVER')
    event_resnet34 = event_accumulator.EventAccumulator(
        'runs/train_resnet34_summary_1/events.out.tfevents.1610767644.ANIMT-SERVER')
    event_resnet50 = event_accumulator.EventAccumulator(
        'runs/train_resnet50_summary_1/events.out.tfevents.1610768692.ANIMT-SERVER')
    event_resnet18.Reload()
    event_resnet34.Reload()
    event_resnet50.Reload()

    # ['train_loss', 'train_acc', 'val_loss', 'val_acc']

    train_loss_resnet18 = event_resnet18.scalars.Items('train_loss')
    train_loss_resnet34 = event_resnet34.scalars.Items('train_loss')
    train_loss_resnet50 = event_resnet50.scalars.Items('train_loss')

    train_acc_resnet18 = event_resnet18.scalars.Items('train_acc')
    train_acc_resnet34 = event_resnet34.scalars.Items('train_acc')
    train_acc_resnet50 = event_resnet50.scalars.Items('train_acc')

    val_loss_resnet18 = event_resnet18.scalars.Items('val_loss')
    val_loss_resnet34 = event_resnet34.scalars.Items('val_loss')
    val_loss_resnet50 = event_resnet50.scalars.Items('val_loss')

    val_acc_resnet18 = event_resnet18.scalars.Items('val_acc')
    val_acc_resnet34 = event_resnet34.scalars.Items('val_acc')
    val_acc_resnet50 = event_resnet50.scalars.Items('val_acc')

    # matplotlib
    # set the  fontsize and some other elements
    large = 22;
```

```

med = 16;
small = 12
params = {'axes.titlesize': large,
          'legend.fontsize': med,
          'figure.figsize': (16, 10),
          'axes.labelsize': med,
          'axes.titlesize': med,
          'xtick.labelsize': med,
          'ytick.labelsize': med,
          'figure.titlesize': large}

plt.rcParams.update(params)
plt.style.use('seaborn-whitegrid')
sns.set_style("white")

# training loss
x = [i.step for i in train_loss_resnet18]
y_train_loss_resnet18 = [i.value for i in train_loss_resnet18]
y_train_loss_resnet34 = [i.value for i in train_loss_resnet34]
y_train_loss_resnet50 = [i.value for i in train_loss_resnet50]

plt.subplot(2,2,1)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(x, y_train_loss_resnet18, label='resnet-18')
plt.plot(x, y_train_loss_resnet34, label='resnet-34')
plt.plot(x, y_train_loss_resnet50, label='resnet-50')
plt.title("Training Loss")
plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')

# training accuracy
x = [i.step for i in train_acc_resnet18]
y_train_acc_resnet18 = [i.value for i in train_acc_resnet18]
y_train_acc_resnet34 = [i.value for i in train_acc_resnet34]
y_train_acc_resnet50 = [i.value for i in train_acc_resnet50]

plt.subplot(2,2,2)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(x, y_train_acc_resnet18, label='resnet-18')

```

```

plt.plot(x, y_train_acc_resnet34, label='resnet-34')
plt.plot(x, y_train_acc_resnet50, label='resnet-50')
plt.title('Training Accuracy')
plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('accuracy')

# validation loss
x = [i.step for i in val_loss_resnet18]
y_val_loss_resnet18 = [i.value for i in val_loss_resnet18]
y_val_loss_resnet34 = [i.value for i in val_loss_resnet34]
y_val_loss_resnet50 = [i.value for i in val_loss_resnet50]

plt.subplot(2,2,3)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(x, y_val_loss_resnet18, label='resnet-18')
plt.plot(x, y_val_loss_resnet34, label='resnet-34')
plt.plot(x, y_val_loss_resnet50, label='resnet-50')
plt.title('Validation Loss')
plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')

# validation accuracy
x = [i.step for i in val_acc_resnet18]
y_val_acc_resnet18 = [i.value for i in val_acc_resnet18]
y_val_acc_resnet34 = [i.value for i in val_acc_resnet34]
y_val_acc_resnet50 = [i.value for i in val_acc_resnet50]

plt.subplot(2,2,4)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(x, y_val_acc_resnet18, label='resnet-18')
plt.plot(x, y_val_acc_resnet34, label='resnet-34')
plt.plot(x, y_val_acc_resnet50, label='resnet-50')
plt.title('Validation Accuracy')
plt.grid()
plt.legend()
plt.xlabel('epoch')

```

```

plt.ylabel('accuracy')

plt.tight_layout()
plt.savefig("output/visualze.jpg")

def visulization_with_regulization():
    event_resnet18 = event_accumulator.EventAccumulator(
        'runs/train_resnet18_summary_1/events.out.tfevents.1610951504.ANIMT-SERVER')
    event_resnet18_weight_decay = event_accumulator.EventAccumulator(
        'runs/train_resnet18_weight_decay_summary_1/events.out.tfevents.1610969564.ANIMT-SERVE
R')

    event_resnet18.Reload()
    event_resnet18_weight_decay.Reload()

    train_loss_resnet18 = event_resnet18.scalars.Items('train_loss')
    train_loss_resnet18_weight_decay = event_resnet18_weight_decay.scalars.Items('train_loss')

    train_acc_resnet18 = event_resnet18.scalars.Items('train_acc')
    train_acc_resnet18_weight_decay = event_resnet18_weight_decay.scalars.Items('train_acc')

    val_loss_resnet18 = event_resnet18.scalars.Items('val_loss')
    val_loss_resnet18_weight_decay = event_resnet18_weight_decay.scalars.Items('val_loss')

    val_acc_resnet18 = event_resnet18.scalars.Items('val_acc')
    val_acc_resnet18_weight_decay = event_resnet18_weight_decay.scalars.Items('val_acc')

    x = [i.step for i in train_acc_resnet18]
    y_train_acc_resnet18 = [i.value for i in train_acc_resnet18]
    y_train_acc_resnet18_weight_decay = [i.value for i in train_acc_resnet18_weight_decay]

    # matplotlib
    # set the  fontsize and some other elements
    large = 22;
    med = 16;
    small = 12
    params = {'axes.titlesize': large,
              'legend.fontsize': med,
              'figure.figsize': (16, 10),
              'axes.labelsize': med,
              'axes.titlesize': med,
              'xtick.labelsize': med,
              'ytick.labelsize': med,

```

```

        'figure.titlesize': large}
plt.rcParams.update(params)
plt.style.use('seaborn-whitegrid')
sns.set_style("white")

# training loss
x = [i.step for i in train_loss_resnet18]
y_train_loss_resnet18 = [i.value for i in train_loss_resnet18]
y_train_loss_resnet18_weight_decay = [i.value for i in train_loss_resnet18_weight_decay]

plt.subplot(2, 2, 1)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(x, y_train_loss_resnet18, label='resnet-18')
plt.plot(x, y_train_loss_resnet18_weight_decay, label='resnet-18 weight decay')
plt.title("Training Loss")
plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')

# training accuracy
plt.subplot(2, 2, 2)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(x, y_train_acc_resnet18, label='resnet-18')
plt.plot(x, y_train_acc_resnet18_weight_decay, label='resnet-18 weight decay')
plt.title("Training Accuracy")
plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('accuracy')

# validation loss
x = [i.step for i in val_loss_resnet18]
y_val_loss_resnet18 = [i.value for i in val_loss_resnet18]
y_val_loss_resnet18_weight_decay = [i.value for i in val_loss_resnet18_weight_decay]

plt.subplot(2, 2, 3)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)

```

```

plt.plot(x, y_val_loss_resnet18, label='resnet-18')
plt.plot(x, y_val_loss_resnet18_weight_decay, label='resnet-18 weight decay')
plt.title('Validation Loss')
plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')

# validation accuracy
x = [i.step for i in val_acc_resnet18]
y_val_acc_resnet18 = [i.value for i in val_acc_resnet18]
y_val_acc_resnet18_weight_decay = [i.value for i in val_acc_resnet18_weight_decay]

plt.subplot(2, 2, 4)
x_major_locator = plt.MultipleLocator(5)
ax = plt.gca()
ax.xaxis.set_major_locator(x_major_locator)
plt.plot(x, y_val_acc_resnet18, label='resnet-18')
plt.plot(x, y_val_acc_resnet18_weight_decay, label='resnet-18 weight decay')
plt.title('Validation Accuracy')
plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('accuracy')

plt.tight_layout()
plt.savefig("output/visualze_regulation.jpg")

# visulization_with_regulization()

```

Assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

85/100

PAGE 1

PAGE 2



Comment 1

Good



Comment 2

ok



Comment 3

Hopefully this will make the problem an easier one to solve.

PAGE 3



Comment 4

no comment



Comment 5

As expected. Well done.

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

GRADING FORM: SMU GENERIC_GJ

YIZHEN MA

85

RESEARCH-INFORMED LITERATURE



A very good review of related work. Well done.
Refs generally appropriate - it would be good to
one or two more recent relevant papers.

7.5

KNOWLEDGE AND UNDERSTANDING



A very good understanding of essential topics demonstrated here. Well done. Code produced works as required and also generated suitable evaluations/reports.

12.5

ANALYSIS



Good analysis of dataset. Well done. The affect of mislabelled images mentioned - could have been more detailed here, perhaps. Results for various architectures generated and detailed - excellent.

26

PRACTICAL APPLICATION AND DEP.



Code produced works as required and also generated suitable evaluations and reports. Several experiments on various architectures undertaken.

35

SKILLS FOR PROFESSIONAL PRAC.



A very well written and structured paper. Refs used appropriately.

4