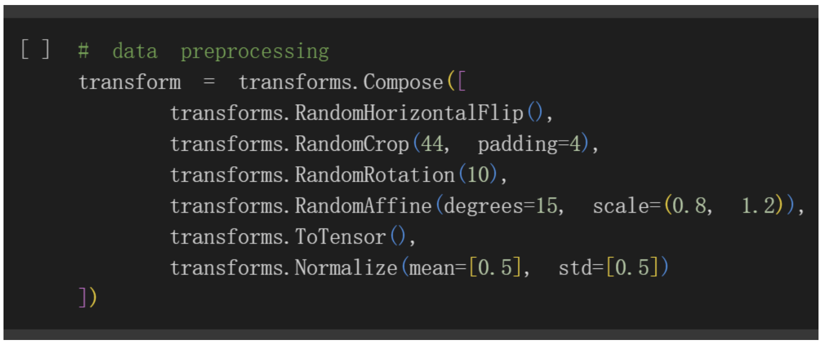
NYCU Introduction to Machine Learning, Homework 4

[111550113], [謝詠晴]

Part. 1, Kaggle (70% [50% comes from the competition]):

(10%) Implementation Details

1) **Environment:**
    a) Google Colab
    b) Hardware: T4 GPU & A100 GPU
    c) Python version: 3.10.12
2) **Introduction:**
    I experimented with three different models for my task. Initially, I used the ResNet50 architecture with the PyTorch framework. However, the accuracy showed no significant improvement, prompting me to explore alternative approaches. During my search, I discovered a Kaggle notebook (ref) that implemented the VGG19 model, achieving a validation accuracy of 0.68—higher than my previous result with ResNet50. After adopting the VGG19 architecture, I observed a slight improvement in accuracy, though it was not substantial. Finally, I tried a modified VGG model based on the paper *Facial Emotion Recognition: State of the Art Performance on FER2013* (ref), which reported an accuracy of 73.2% with fine-tuning. Following its structure, I achieved 61.5% accuracy on the public test dataset and 72.3% on my validation dataset.
3) **How to inference:**
    Since I used Google Colab, the inference steps are simple. First create a “ML” folder in your Google drive and then zip the “data” folder provided by TA and upload to the drive folder. Next, download the model weights I provided in the link and upload it to the same drive folder. Next, open the “inference.ipynb” then follow the instructions, and finally you can get the csv file in your drive folder.
4) **Resnet50**
    a) inference model weights filename: [resnet_no_validate.pth]
    b) augmentation:
    I applied data augmentation to the training dataset to enhance its generalization capability. The augmentations include horizontal flipping, image rotation, and affine transformations. Additionally, the data was normalized to a range of [-1, 1].



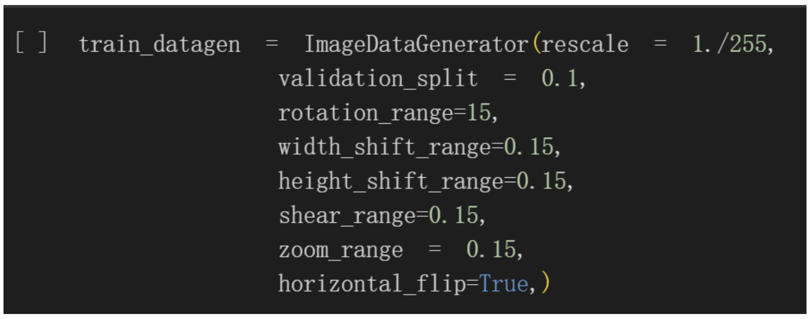
```
[ ]  # data preprocessing
     transform = transforms.Compose([
             transforms.RandomHorizontalFlip(),
             transforms.RandomCrop(44, padding=4),
             transforms.RandomRotation(10),
             transforms.RandomAffine(degrees=15, scale=(0.8, 1.2)),
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.5], std=[0.5])
     ])
```

c) train without validation dataset:
Initially, I split the training dataset into an 8:2 ratio to create a validation dataset. After evaluating the training results, I gained a general understanding of the model's performance. I then considered whether providing more data for training might improve the accuracy. Therefore, I decided to train the model without a validation dataset, using the entire training dataset and selecting the model from the final epoch. The results showed an improvement in accuracy on the test data.

Model backbone (e.g., VGG16, VGG19, Custom, etc)	Resnet50
Number of model parameters	23,522,375
batch size	32
learning rate	0.001
num of epochs	20
Optimizer	SGD

5) VGG19

a) reference: Facial Emotion Recognition | VGG19 - FER2013
b) inference model weights filename: [VGG_no_validate_origin.keras]
c) I mainly follow the notebook instructions and do some modification to fit my datasets.
d) I split the training dataset into a 9:1 ratio to create a validation dataset first. And then I apply the same strategy as Resnet: without validation set.
e) augmentation: I made some operations like rescale pixels to [0, 1] and rotate, zoom, horizontal flip the images.



```
[ ]  train_datagen  =  ImageDataGenerator(rescale  =  1./255,
                              validation_split  =  0.1,
                              rotation_range=15,
                              width_shift_range=0.15,
                              height_shift_range=0.15,
                              shear_range=0.15,
                              zoom_range  =  0.15,
                              horizontal_flip=True,)
```

f) Transfer learning strategy: The output from the second-to-last layer of the pre-trained model was used as input features. A Global Average Pooling layer was added to compress spatial features into a single value per feature map, reducing parameter count and mitigating overfitting.Finally, a fully connected layer (Dense) was added as the output layer, using a softmax activation function for multi-class classification.
g) I also applied an early stopping strategy when testing with validation dataset.

Model backbone (e.g., VGG16, VGG19, Custom, etc)	VGG19
Number of model parameters	60,083,927
batch size	32
learning rate	0.0001
num of epochs	25
Optimizer	Adam

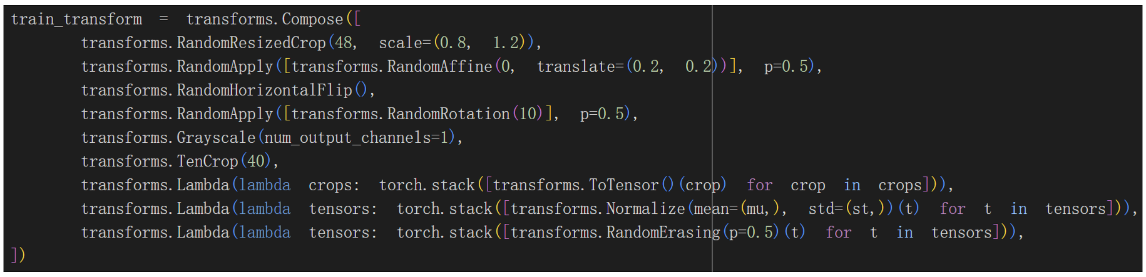
6) **VGG from paper**

a) reference: [paper] [GitHub]

b) inference model weights filename: [epoch_96]

i) This file also stores log information, so in the inference.ipynb, I wrote a further step *net.load_state_dict(checkpoint["params"]).*

c) Augmentation: is also applied in this method such as random cropping, rotation, translation, horizontal flipping. The most different part is the TenCrop transformation. It crops original images into 10 images of size 40x40.



```
train_transform = transforms.Compose([
        transforms.RandomResizedCrop(48, scale=(0.8, 1.2)),
        transforms.RandomApply([transforms.RandomAffine(0, translate=(0.2, 0.2))], p=0.5),
        transforms.RandomHorizontalFlip(),
        transforms.RandomApply([transforms.RandomRotation(10)], p=0.5),
        transforms.Grayscale(num_output_channels=1),
        transforms.TenCrop(40),
        transforms.Lambda(lambda crops: torch.stack([transforms.ToTensor()(crop) for crop in crops])),
        transforms.Lambda(lambda tensors: torch.stack([transforms.Normalize(mean=(mu,), std=(st,))(t) for t in tensors])),
        transforms.Lambda(lambda tensors: torch.stack([transforms.RandomErasing(p=0.5)(t) for t in tensors])),
])
```

d) In the model architecture, The convolutional layers are followed by batch normalization, ReLU activations, max-pooling. This method also applies dropout to prevent overfitting.

e) Using GradScaler for mixed precision training, enabling faster computations.

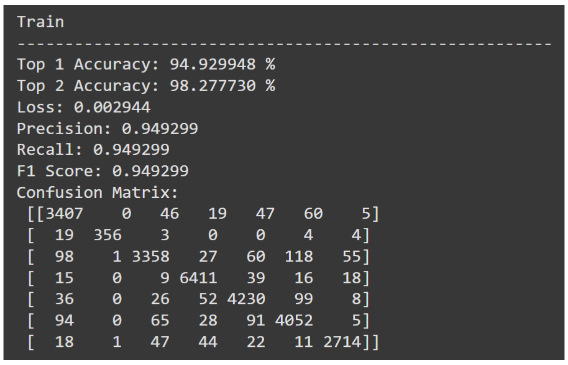
Model backbone (e.g., VGG16, VGG19, Custom, etc)	VGG19
Number of model parameters	29,890,759
batch size	64
start learning rate	0.01
num of epochs	100
Optimizer	SGD with Nesterov acceleration

(10%) Experimental Results

In this section, I focus on the experimental results using the VGG architecture from the paper.

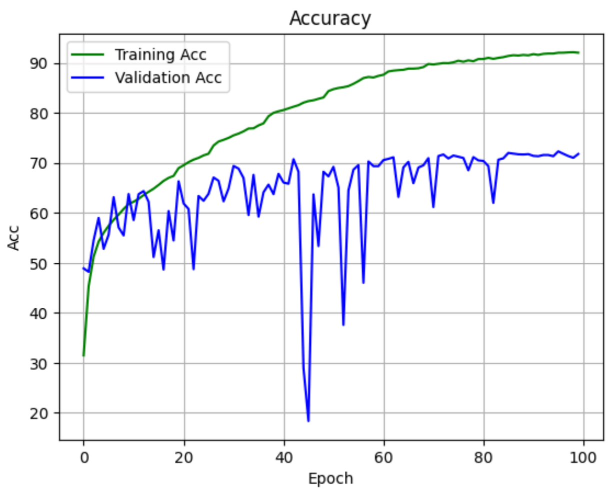
1) Evaluation metrics

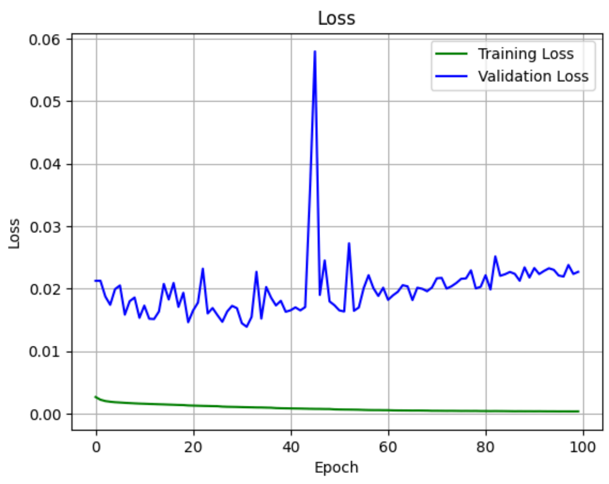
The accuracy obtained during evaluation is slightly different from that recorded during training. This is because the model is loaded and evaluated in evaluation mode after the training process. So the model might already see these training datasets and validation datasets, in turn having a high accuracy. This evaluation is more suitable for testing data but I don’t have the true label of them.



```
Train
--------------------------------------------------------
Top 1 Accuracy: 94.929948 %
Top 2 Accuracy: 98.277730 %
Loss: 0.002944
Precision: 0.949299
Recall: 0.949299
F1 Score: 0.949299
Confusion Matrix:
 [[3407    0   46   19   47   60    5]
 [  19  356    3    0    0    4    4]
 [  98    1 3358   27   60  118   55]
 [  15    0    9 6411   39   16   18]
 [  36    0   26   52 4230   99    8]
 [  94    0   65   28   91 4052    5]
 [  18    1   47   44   22   11 2714]]
```

2) Learning Curves of 100 epochs VGG_paper

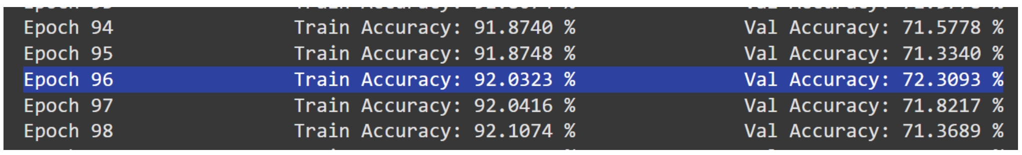




3) Ablation Study

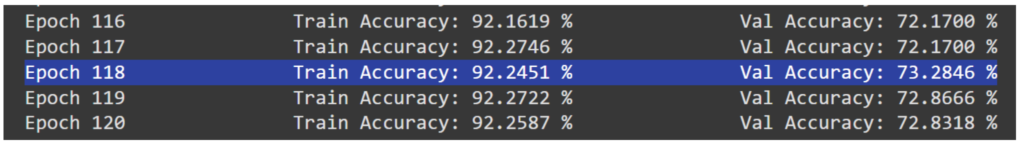
a) Test data accuracy is not as high as validation accuracy:
In this study, the referenced paper achieved over 70% accuracy after training for 300 epochs. However, the dataset used in my experiments differs from theirs. Specifically, my validation set is a subset of the training data, split for evaluation purposes. Interestingly, while my validation results were consistently strong, the performance on the test set did not meet expectations.

b) 300 epochs:
I conducted experiments training for 300 epochs and selected models based on the highest validation accuracy. However, this approach did not lead to better test performance. I didn't finish the 300 epochs run because I think the accuracy will not increase.



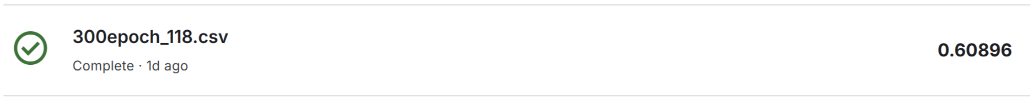
```
Epoch 94        Train Accuracy: 91.8740 %        Val Accuracy: 71.5778 %
Epoch 95        Train Accuracy: 91.8748 %        Val Accuracy: 71.3340 %
Epoch 96        Train Accuracy: 92.0323 %        Val Accuracy: 72.3093 %
Epoch 97        Train Accuracy: 92.0416 %        Val Accuracy: 71.8217 %
Epoch 98        Train Accuracy: 92.1074 %        Val Accuracy: 71.3689 %
```

[100 epochs]



```
Epoch 116       Train Accuracy: 92.1619 %        Val Accuracy: 72.1700 %
Epoch 117       Train Accuracy: 92.2746 %        Val Accuracy: 72.1700 %
Epoch 118       Train Accuracy: 92.2451 %        Val Accuracy: 73.2846 %
Epoch 119       Train Accuracy: 92.2722 %        Val Accuracy: 72.8666 %
Epoch 120       Train Accuracy: 92.2587 %        Val Accuracy: 72.8318 %
```

[300 epochs]



300epoch_118.csv
Complete · 1d ago
0.60896

c) Select which epoch's model:
This suggests that directly selecting models based on validation accuracy may not always guarantee optimal performance on unseen test data. Adjusting the model selection strategy might be necessary in this context.

d) Without validation (more training data):
It is worth noting that in cases where no validation set is used, both ResNet and VGG models showed improved test accuracy. However, without a validation set, the entire training process lacked a clear criterion for model selection. Thus, I chose to retain the validation set in my workflow to ensure a systematic approach to selecting models.

e) Finetune:
The paper mentioned a fine-tuning step to further improve performance. I followed their fine-tuning procedure, but my results showed a performance drop instead. I suspect this might be due to an error in my implementation of the fine-tuning process. Because during my training process, my train loss is always 0.0005. Further investigation is required to identify the root cause and ensure the procedure aligns with the original implementation.

Part. 2, Questions (30%):

1. (10%) Explain the support vector in SVM and the slack variable in Soft-margin SVM. Please provide a precise and concise answer. (each in two sentences)
    1) Support Vector: Data points that are closest to the hyperplane in SVM, and they determine the position and orientation of the hyperplane to maximize the margin.
    2) Slack variable: It allows some misclassification of some data points to increase flexibility and is controlled by a regularization parameter C.

2. (10%) In training an SVM, how do the parameter C and the hyperparameters of the kernel function (e.g., γ for the RBF kernel) affect the model's performance? Please explain their roles and describe how to choose these parameters to achieve good performance.
    1) Parameter C:
       Its role is to control the balance between maximizing the margin and meanwhile minimizing classification errors.
    2) Kernel Hyperparameters:
        a) Take γ for the RBF kernel for example, it controls the influence of a single training data point on the decision boundary.
        b) Take degree d in the polynomial kernel for example, it determines the complexity of mapping the original data to a higher-dimensional feature space. Higher degrees allow the model to capture more complex relationships but may also lead to overfitting.
    3) How to choose parameters:
        a) You can choose a smaller C for noisy data to allow more misclassification, which may generalize better to unseen data. On the other hand, you can choose a larger C if your priority is to correctly classify training data.
        b) Choosing a smaller γ will result in a smoother decision boundary, making the model more tolerant of noise in the data.
        c) You can use techniques like grid search to test different combinations of parameters and evaluate their performance by cross-validation.

3. (10%) SVM is often more accurate than Logistic Regression. Please compare SVM and Logistic Regression in handling outliers.
    1) SVM can handle outliers more effectively using the soft-margin method, which incorporates slack variables and the parameter CCC. By adjusting CCC, we can control the model's tolerance for margin violations, thereby reducing the impact of outliers.
    2) However, logistic regression is more sensitive to outliers due to its log-loss function. Outliers can disproportionately affect the loss, leading the model to be biased. To mitigate this, regularization techniques such as L1 or L2 regularization can be employed to reduce the impact of outliers.